

---

Eine Dissertation der Universität Regensburg:

# System Administration Training in the Virtual Unix Lab

*An e-learning system with diagnosis via a domain specific language as  
base for an architecture for tutorial assistance and user adaption*

---

Autor: Hubert Feyrer <hubert@feyrer.de>

Erstbetreuer: Prof. Dr. Rainer Hammwöhner

Zweitbetreuer: Prof. Dr. Christian Wolff

Eingereicht am: 25. Januar 2008

Mündliche Prüfung: 11. November 2008



# Preface

Development of the basic training system was funded as part of the HWP-project of the German government, the system was implemented at the Computer Science department of the University of Applied Sciences (Fachhochschule, FH) Regensburg, Germany.

As the basic system used in this work was developed in German language, usage examples, exercises and screenshots are in that language. This work as a whole is written in English to match the intended audience. The male gender is used throughout the text for consistency and simplicity.

## Acknowledgements

I'd like to express words of gratitude to Prof. Jürgen Sauer as my longstanding mentor; Prof. Dr. Rainer Hammwöhner the principal advisor of this work, and Prof. Dr. Christian Wolff as co-principal advisor; The department of Computer Science at the University of Applied Sciences of Regensburg, especially the officiating dean Prof. Dr. Kucera and the former dean Prof. Dr. Schicker, kindly supported the work, and last but not least I'd like to thank my parents and friends for all their support.

Further thanks go to the NetBSD, R and PostgreSQL projects for their great and free software; the Virtual Unix Lab beta-tester Holger Amann, Sonja Öttl, Günter Schwarz, Holger Nösekabel, Stefan Zimmermann, DaNiel Ettle, and Michael Jobst; the students of the IST semester in the summer semesters from 2004 to 2007, as well as to my proofreaders Verena Bäumlner, Andreas Fassel, Sabine Salzl, Stefan Schumacher, Günter Schwarz, Matthew Sporleder, and Gabriele Steinberger.

Without them, this work would not exist in its current form today.

Hubert Feyrer  
Regensburg, November 24th, 2008



# Abstract

This work covers training of system administration by introducing a system called the Virtual Unix Lab, and illustrates advanced topics based on it. The work is divided into three parts.

In the first part, the goals of the Virtual Unix Lab is illustrated and compared to related works, followed by observations about education of system administration. General learning theories are observed and compared to an existing lecture on system administration, showing that there is demand for practical exercises in advanced topics.

The second part describes how diagnosis of the Virtual Unix Lab exercise results and feedback to the user are realized with the help of a domain specific language. After observing the fundamentals of domain specific languages, the design of diagnosis and feedback to the learner is presented, the Verification Unit Domain Specific Language (VUDSL) is described, and architecture and implementation within the existing Virtual Unix Lab are shown. An evaluation of the system was performed and shows that repeated exercises show improved performance of the students, and that the system is regarded as useful by students in general.

The third part adds tutoring and user adaption. Based on the fundamentals of tutoring and user adaption, an architecture for a tutoring component for the Virtual Unix Lab based on an overlay architecture is described. Aspects discussed include on-line diagnosis, feedback, assistance to the user, considerations for the user model, and impact on the user interface. User adaption is based on the user model built by the tutoring component. It observes structural and longitudinal consistency, and provides personalized feedback to the student. An architecture is described that fits in the overall Virtual Unix Lab architecture, and possible extensions for the VUDSL used for diagnosis and feedback are proposed.



# Contents (short)

## I Introduction

1 Problem domain and goal of the Virtual Unix Lab	3
2 Related works	11
3 Education of system administration	23

## II Diagnosis and feedback with a domain specific language

4 Basic design of the Virtual Unix Lab	53
5 Introduction of domain specific languages	69
6 Architecture and implementation of diagnosis and feedback with a domain specific language	81
7 Evaluation of the Virtual Unix Lab	129

## III Tutoring and user adaption

8 Introduction of tutoring and user adaption	179
9 Design of tutoring and user adaption	211

---

<b>10 Architecture of tutoring</b>	<b>221</b>
<b>11 Architecture of user adaption</b>	<b>247</b>
<b>12 Conclusion</b>	<b>271</b>
<b>List of figures</b>	<b>272</b>
<b>List of tables</b>	<b>278</b>
<b>Bibliography</b>	<b>280</b>
<b>A Example exercise components</b>	<b>311</b>
<b>B Database structure</b>	<b>345</b>
<b>C Evaluation data and code</b>	<b>351</b>
<b>D A theory of bugs — attempt of a reconstructive approach</b>	<b>383</b>
<b>E Analysis of exercises under tutorial and adaptive aspects</b>	<b>387</b>



# Contents

## I Introduction

<b>1</b>	<b>Problem domain and goal of the Virtual Unix Lab</b>	<b>3</b>
1.1	Problem domain of the Virtual Unix Lab . . . . .	3
1.2	The goal of the Virtual Unix Lab . . . . .	6
1.3	How this book is organized . . . . .	8
<b>2</b>	<b>Related works</b>	<b>11</b>
2.1	Computer science education . . . . .	11
2.2	System administration education . . . . .	12
2.3	Training systems for system administration . . . . .	13
2.3.1	Systems focused on education . . . . .	13
2.3.2	Systems focused on deployment . . . . .	14
2.3.3	Systems offering user-level access . . . . .	15
2.4	Domain specific languages . . . . .	16
2.5	Result verification, diagnosis and feedback . . . . .	17
2.6	Tutoring systems in Unix education . . . . .	18
2.7	Adaptive systems in Unix education . . . . .	19
2.8	Other virtual labs . . . . .	20
2.9	Virtualization & emulation . . . . .	20

---

<b>3</b>	<b>Education of system administration</b>	<b>23</b>
3.1	Fundamentals of education . . . . .	23
3.1.1	Psychology and learning theory . . . . .	23
3.1.2	Didactic realization, instruction theory and instructional design	27
3.1.3	Dimension of implementation and adaption . . . . .	31
3.1.4	Alternative learning-theoretical approaches . . . . .	32
3.1.5	Education – ideal progression and tools . . . . .	34
3.2	The “System Administration” class . . . . .	35
3.2.1	History and target audience . . . . .	35
3.2.2	Current curriculum . . . . .	36
3.2.3	Course layout . . . . .	40
3.2.4	Didactic instruments . . . . .	43
3.3	Analysis of the current situation . . . . .	46
3.4	Future directions . . . . .	48
<b>II Diagnosis and feedback with a domain specific language</b>		
<b>4</b>	<b>Basic design of the Virtual Unix Lab</b>	<b>53</b>
4.1	A user-level walkthrough of the Virtual Unix Lab . . . . .	53
4.2	Hardware and network setup of the Virtual Unix Lab . . . . .	64
4.3	Software components of the Virtual Unix Lab . . . . .	65
<b>5</b>	<b>Introduction of domain specific languages</b>	<b>69</b>
5.1	Classification of languages . . . . .	69
5.2	Attributes of domain specific languages . . . . .	71

---

5.3	Design patterns . . . . .	72
5.4	Choosing an implementation languages . . . . .	77
<b>6</b>	<b>Architecture and implementation of diagnosis and feedback with a domain specific language</b>	<b>81</b>
6.1	Requirements of exercise verification . . . . .	81
6.2	Roadmap of implementation . . . . .	83
6.2.1	Stepwise refinement . . . . .	83
6.2.2	Exercise phases . . . . .	84
6.2.3	What and how to verify . . . . .	85
6.3	Step 0: Basic design . . . . .	86
6.4	Step I: Instructions and checks not coupled . . . . .	87
6.4.1	Components . . . . .	87
6.4.2	Integration and interaction . . . . .	92
6.4.3	Summary and suggested improvements . . . . .	97
6.5	Step II: Instructions and checks coupled . . . . .	97
6.5.1	Improved check primitives . . . . .	97
6.5.2	Coupling of exercise text and checks . . . . .	102
6.5.2.1	Options . . . . .	103
6.5.2.2	Data structure representation . . . . .	105
6.5.2.3	Forming a domain specific language . . . . .	106
6.5.3	Giving feedback . . . . .	107
6.5.4	Creating a system front-end with check scripts . . . . .	110
6.5.5	Integration and interaction . . . . .	114
6.5.6	Summary of step II . . . . .	122
6.6	The Verification Unit Domain Specific Language (VUDSL) . . . . .	122

---

6.7	Conclusion of diagnosis and feedback with a domain specific language	124
6.8	Future Perspectives	125
<b>7</b>	<b>Evaluation of the Virtual Unix Lab</b>	<b>129</b>
7.1	What to evaluate	129
7.2	Analysis of data gathered during student exercises	131
7.2.1	Methodology of the data analysis	131
7.2.2	Number of exercises taken and repeated	132
7.2.3	Performance of repeated exercises	133
7.2.4	Results of selected exercise topics	136
7.2.5	Exercise duration	146
7.2.6	Exercise time	149
7.2.7	Summary	150
7.3	Analysis of the user questionnaire	155
7.3.1	Methodology of the questionnaire analysis	155
7.3.1.1	Aspects evaluated by the questionnaire	156
7.3.1.2	Design and implementation of the questionnaire	156
7.3.1.3	Evaluation methods	157
7.3.2	Evaluation of user acceptance	158
7.3.2.1	Questionnaire results	158
7.3.2.2	Interpretation of the questionnaire results	159
7.3.3	Evaluation of the course of exercises	159
7.3.3.1	Questionnaire results	159
7.3.3.2	Interpretation of the questionnaire results	160
7.3.4	Evaluation of the use of learning material	161
7.3.4.1	Impact of learning materials in general	161

7.3.4.2	Impact of learning materials during Virtual Unix Lab exercises . . . . .	163
7.3.4.3	Impact of the “SA” lecture for exercises in the Virtual Unix Lab . . . . .	165
7.3.4.4	Impact of the “SA” lecture notes for exercises in the Virtual Unix Lab . . . . .	166
7.3.4.5	Interpretation of the questionnaire results . . . . .	167
7.3.5	Evaluation of the target audience . . . . .	168
7.3.5.1	Questionnaire results . . . . .	169
7.3.5.2	Interpretation of the questionnaire results . . . . .	170
7.3.6	Summary . . . . .	170
7.4	Other aspects to evaluate . . . . .	172
7.5	Conclusion of the evaluation . . . . .	175

### III Tutoring and user adaption

<b>8</b>	<b>Introduction of tutoring and user adaption</b>	<b>179</b>
8.1	Fundamentals of tutoring . . . . .	179
8.1.1	Approaching tutoring . . . . .	180
8.1.2	The teaching model . . . . .	181
8.1.2.1	Teaching and didactic operations . . . . .	182
8.1.2.2	Methods for plan recognition and assistance . . . . .	184
8.1.2.2.1	Classical approaches . . . . .	184
8.1.2.2.2	Cognitive approach . . . . .	185
8.1.2.2.3	Linguistic approach . . . . .	186
8.1.2.2.4	Artificial intelligence . . . . .	186
8.1.2.2.5	Semantic networks and ontologies . . . . .	188

8.1.2.2.6	Frames and scripts . . . . .	189
8.1.2.2.7	Bayesian networks . . . . .	190
8.1.2.3	Choosing a method . . . . .	190
8.1.3	The domain model . . . . .	191
8.1.4	The user model . . . . .	192
8.1.4.1	Theories of bugs . . . . .	193
8.1.4.2	Viewpoints . . . . .	194
8.1.4.3	Diagnosis . . . . .	195
8.1.4.3.1	Behavioral diagnosis . . . . .	196
8.1.4.3.2	Epistemic diagnosis . . . . .	197
8.1.4.3.2.1	Direct assignment of credit and blame	197
8.1.4.3.2.2	Structural consistency . . . . .	199
8.1.4.3.2.3	Longitudinal consistency . . . . .	199
8.1.4.3.3	Diagnostic data . . . . .	200
8.1.4.4	Feedback . . . . .	201
8.1.5	The user interface . . . . .	202
8.2	Fundamentals of user adaption . . . . .	203
8.2.1	The meaning of context . . . . .	207
8.2.2	Adaptive services and multiple agents . . . . .	208
8.2.3	Modeling techniques . . . . .	208
8.2.4	Adaptive axes . . . . .	210
<b>9</b>	<b>Design of tutoring and user adaption</b>	<b>211</b>
9.1	Goals of tutoring and user adaption . . . . .	211
9.2	Methodology of tutoring and user adaption . . . . .	212
9.3	The domain model . . . . .	212

---

9.3.1	Content decomposition . . . . .	213
9.3.2	Considerations for a theory of bugs . . . . .	215
9.3.2.1	Adjusting the domain model . . . . .	216
9.3.2.2	Analyzing existing exercise data . . . . .	216
9.3.2.3	Results and conclusion . . . . .	217
9.4	Software architecture . . . . .	219
<b>10</b>	<b>Architecture of tutoring</b>	<b>221</b>
10.1	Establishing the teaching model . . . . .	221
10.1.1	Selection criteria . . . . .	222
10.1.2	Classical approaches with overlay architecture . . . . .	222
10.1.3	Cognitive approach . . . . .	223
10.1.4	Linguistic approach . . . . .	224
10.1.5	Artificial Intelligence based approach . . . . .	225
10.1.6	Semantic networks and ontologies . . . . .	227
10.1.7	Frames and scripts . . . . .	228
10.1.8	Bayesian networks . . . . .	229
10.1.9	Comparison . . . . .	230
10.2	Using model tracing for diagnosis during the exercise . . . . .	232
10.3	Investigating on-line diagnosis . . . . .	232
10.4	Giving feedback and assistance . . . . .	235
10.4.1	Goal . . . . .	235
10.4.2	Assumptions . . . . .	236
10.4.3	Challenges . . . . .	236
10.4.4	Realization . . . . .	237
10.4.4.1	Contents . . . . .	237

---

10.4.4.2	Form of feedback . . . . .	238
10.4.5	Impact on organization of exercises and learning material . . .	239
10.5	Considerations for the user model . . . . .	240
10.6	Impact on the user interface . . . . .	241
10.6.1	Communication channels . . . . .	242
10.6.2	Analysis of the current user interface . . . . .	242
10.6.3	Blending information into the web-based user-interface . . . .	245
10.7	Summary . . . . .	245
<b>11</b>	<b>Architecture of user adaption</b>	<b>247</b>
11.1	Establishing and maintaining the user model . . . . .	247
11.1.1	Initialization . . . . .	248
11.1.2	Clustering . . . . .	249
11.1.3	Observed data . . . . .	249
11.1.4	Updating the user model . . . . .	249
11.1.5	Accommodating plan recognition . . . . .	250
11.2	Adaptive axes . . . . .	251
11.3	Structural consistency . . . . .	252
11.3.1	Observing exercise velocity . . . . .	252
11.3.2	Observing mastered skills . . . . .	253
11.3.3	Observing help requests . . . . .	253
11.3.4	Adjusting the user model . . . . .	254
11.3.5	A metric for evaluation . . . . .	255
11.4	Longitudinal consistency . . . . .	256
11.4.1	Assumptions and methodology . . . . .	256
11.4.2	Descriptive analysis . . . . .	257



---

11.4.2.1	Interpolation vs. more data . . . . .	257
11.4.2.2	Detecting speed changes . . . . .	257
11.4.2.3	Observations for repeated exercises . . . . .	257
11.4.2.4	Speed and acceleration of progress . . . . .	258
11.4.2.5	Data model and storage . . . . .	258
11.4.2.6	Drawing conclusions from speed and acceleration . . . . .	258
11.4.3	Indicative analysis . . . . .	259
11.5	Personalizing feedback . . . . .	260
11.5.1	Adjusting of help contents . . . . .	260
11.5.2	Handling non-standard exercise progress . . . . .	261
11.5.3	Adjusting the system . . . . .	261
11.5.4	Preventing abuse of the help system . . . . .	262
11.6	Extending the VUDSL for user adaption . . . . .	262
11.6.1	VUDSL extensions for structural consistency . . . . .	263
11.6.2	VUDSL extensions for longitudinal consistency . . . . .	266
11.6.3	VUDSL extensions for personalized feedback . . . . .	267
11.6.4	Other VUDSL extensions . . . . .	267
11.7	Summary . . . . .	270
<b>12</b>	<b>Conclusion</b>	<b>271</b>
	<b>List of figures</b>	<b>272</b>
	<b>List of tables</b>	<b>278</b>
	<b>Bibliography</b>	<b>280</b>
<b>A</b>	<b>Example exercise components</b>	<b>311</b>

---

A.1	Exercise texts for users . . . . .	311
A.1.1	Network Information System (NIS) exercise . . . . .	311
A.1.2	Network File System (NFS) exercise . . . . .	313
A.2	Exercises including text and check data . . . . .	314
A.2.1	Network Information System (NIS) exercise . . . . .	314
A.2.2	Network File System (NFS) exercise . . . . .	318
A.3	The VUDSL processor: uebung2db . . . . .	321
A.4	Complete lists of checks used in exercises . . . . .	325
A.4.1	Network Information System (NIS) exercise . . . . .	325
A.4.2	Network File System (NFS) exercise . . . . .	326
A.5	List of check scripts and parameters . . . . .	327
A.6	Selected check scripts . . . . .	329
A.6.1	Step I . . . . .	329
A.6.1.1	netbsd-check-finger.sh . . . . .	329
A.6.1.2	netbsd-check-masterpw.sh . . . . .	330
A.6.1.3	netbsd-check-pkginstalled.sh . . . . .	330
A.6.1.4	netbsd-check-pw.pl . . . . .	330
A.6.1.5	netbsd-check-usershell2.sh . . . . .	331
A.6.1.6	check-program-output . . . . .	331
A.6.2	Step II . . . . .	333
A.6.2.1	admin-check-clearharddisk . . . . .	333
A.6.2.2	admin-check-makeimage . . . . .	333
A.6.2.3	check-file-contents . . . . .	336
A.6.2.4	unix-check-user-exists . . . . .	337
A.6.2.5	unix-check-user-shell . . . . .	338

---

A.6.2.6	unix-check-user-password . . . . .	340
A.6.2.7	unix-check-process-running . . . . .	341
A.6.2.8	netbsd-check-rcvar-set . . . . .	342
<b>B</b>	<b>Database structure</b>	<b>345</b>
B.1	Table: benutzer . . . . .	345
B.2	Table: rechner . . . . .	345
B.3	Table: images . . . . .	346
B.4	Table: uebungen . . . . .	346
B.4.1	Definition . . . . .	346
B.4.2	Example records . . . . .	346
B.5	Table: uebung_setup . . . . .	347
B.6	Table: uebungs_checks . . . . .	347
B.6.1	Definition . . . . .	347
B.6.2	Example records . . . . .	348
B.7	Table: buchungen . . . . .	348
B.7.1	Definition . . . . .	348
B.7.2	Example records . . . . .	348
B.8	Table: ergebnis_checks . . . . .	349
B.8.1	Definition . . . . .	349
B.8.2	Example records . . . . .	350
<b>C</b>	<b>Evaluation data and code</b>	<b>351</b>
C.1	Questionnaire: questions — raw format . . . . .	351
C.2	Questionnaire: questions and results . . . . .	353
C.3	Exercise results: selected SQL queries and results . . . . .	371

<b>D A theory of bugs — attempt of a reconstructive approach</b>	<b>383</b>
<b>E Analysis of exercises under tutorial and adaptive aspects</b>	<b>387</b>

# **Part I**

## **Introduction**



# Chapter 1

## Problem domain and goal of the Virtual Unix Lab

This work is about education of system administration. With the increasing complexity of today's IT systems and their related management, corresponding education becomes more and more important. This work describes the Virtual Unix Lab, which is an interactive course system that supports electronic learning (e-learning) for that purpose.

The basic implementation of the Virtual Unix Lab as framework for performing practical exercises for Unix system administration was created during the "Praktikum Unix-Cluster-Setup" project as part of the "Hochschul-Wissenschafts-Projekt" (HWP) of the German ministry of education and research (Bundesministerium für Bildung und Forschung, BMBF). The system was designed to consist of several components<sup>1</sup>, and most of the implementation was done as diploma thesis at the University of Applied Sciences Regensburg, see [Zimmermann, 2003]. The result of the project was not fully functional, and this work puts a focus on those missing components – diagnosis, feedback, tutoring and user adaption.

This chapter outlines the problem domain in which the following work is performed in, including a brief description of underlying terms and related working areas. The second section identifies the goal of the Virtual Unix Lab and how it will be reached.

### 1.1 Problem domain of the Virtual Unix Lab

This work describes the Virtual Unix Lab under aspects of computer science and information science. This section lists aspects that are related to the problem domain.

---

<sup>1</sup> [Feyrer, 2004c]

**What is “e-learning”?** Teaching can be seen as knowledge communication. The goal of knowledge communication is to improve knowledge in a student through learning. The process of knowledge communication can be enhanced by means of electronic communication, which converges into the term “electronic learning” or in short, “e-learning.”<sup>1</sup> The Virtual Unix Lab system introduced here borders both the “knowledge” and the “communication” part in that it defines what and how to teach.

**What does “virtual” mean?** The term “virtual” has several meanings. In the educational environment, separation of space is meant, decoupling the location of the student from that of the teacher by having them meet in a “virtual classroom”. The Virtual Unix Lab provides such a separation, which contrasts a real lab in which a student has to go to for all interaction with the system to happen.

A different approach to the term would be by using virtual machines to realize the lab environment. This is not on focus here, but a possible future extension as suggested by some of the related works outlined in chapter 2.

**Why system administration?** System administration is an area where many students that graduate in computer science find employment. Following [Hubwieser, 2000, pp. 63], the human role in the management of information systems is not only setup and maintenance of systems, but also to obtain and provide information on the system status and setup.

No fixed curriculum exists for education in the area of system administration in the large. The topic of system administration itself is bordering on many major technical and administrative topics shown in figure 1.2, of which each one is taught well to students: operating systems, network management, systems and software engineering, security, and law. System administration itself requires comprehensive thinking, combining of known and documented components, mental transfer and application of expert knowledge. Furthermore, problem solving strategies are required, as components do *not* work as expected or documented at many times.

This existing situation, plus personal interest in system administration and related areas, led to work on the Virtual Unix Lab, and this work.

**Why Unix?** Besides Microsoft Windows, a number of operating systems that can be found today are grouped under the term “Unix.” Originating from AT&T in 1969, there are many Unix flavours now, of which Solaris, NetBSD and Linux are just a few examples, see [Lévénez, 2007] for a complete overview. Several years of experience in administrating various Unix derivatives, esp. Sun’s Solaris, personal work on NetBSD, a successor of BSD Unix, and work on the g4u project have influenced the work described here. The influence affects realization of the Virtual Unix Lab on one side, and the contents on the Virtual Unix Lab on the other side.

---

<sup>1</sup> [Kuhlen and Laisiepen, 2004] pp. 469



Tabelle 5. Befehle zum Erstellen eines Aliasnamens auf der Rückschleifeinheit (lo0) für den Dispatcher

AIX	<code>ifconfig lo0 alias Cluster-Adresse netmask Netzmaske</code>
HP-UX	<code>ifconfig lo0 Cluster-Adresse</code>
Linux	<code>ifconfig lo1 Cluster-Adresse netmask 0.0.0.0 up</code>
OS/2	<code>ifconfig lo Cluster-Adresse</code>
Solaris	<code>ifconfig lo0:1 Cluster-Adresse 127.0.0.1 up</code>
Windows NT	<ol style="list-style-type: none"> <li>1. Klicken Sie auf <b>Start</b> und dann auf <b>Einstellungen</b>.</li> <li>2. Klicken Sie auf <b>Systemsteuerung</b> und dann doppelt auf <b>Netzwerk</b>.</li> <li>3. Fügen Sie den MS Loopback Adapter Driver hinzu (falls noch nicht erfolgt). <ol style="list-style-type: none"> <li>a. Klicken Sie im Fenster <b>Netzwerk</b> auf <b>Netzwerkarten</b>.</li> <li>b. Wählen Sie <b>MS Loopback Adapter</b> aus und klicken Sie dann auf <b>OK</b>.</li> <li>c. Legen Sie die Installations-CD oder -disketten ein, wenn Sie dazu aufgefordert werden.</li> <li>d. Klicken Sie im Fenster <b>Netzwerk</b> auf <b>Protokolle</b>.</li> <li>e. Wählen Sie <b>TCP/IP-Protokoll</b> aus und klicken Sie dann auf <b>Eigenschaften</b>.</li> <li>f. Wählen Sie <b>MS Loopback Adapter</b> aus und klicken Sie dann auf <b>OK</b>.</li> </ol> </li> <li>4. Setzen Sie die Rückschleifeadresse auf die Cluster-Adresse. Akzeptieren Sie die standardmäßige Teilnetzmaske (255.0.0.0) und geben Sie keine Gateway-Adresse ein.</li> </ol> <p>Anmerkung: Der MS Loopback Driver wird möglicherweise erst dann in der TCP/IP-Konfiguration angezeigt, wenn Sie die Netzwerkeinstellungen verlassen und erneut aufrufen.</p>

Figure 1.1: Instructions for the command line and a graphical user interface. Image source: [Emzy Bilder Galerie, 2007]

While Unix-derivatives can be managed via graphical user interfaces (GUI), system administration usually happens via a command line interface and an assorted set of command line tools. This has proven to be easier for documentation and learning, as the steps to perform a certain configuration step shown in figure 1.1 documents: The command is similar for the five Unix systems, but Windows as a system that requires use of a GUI for administration needs a lot more documentation to perform the same single step.

The text-based nature of Unix is considered a bonus when documenting and learning to use and administrate a Unix-based operating system<sup>1,2</sup>. Also, as Unix systems are compatible among each other, and as many are available as Open Source, their operation can be verified as part of the tutoring and adaption process. The Unix system interfaces of have remained stable over many years, which provides enduring benefits and return of investment for learners.

Finally, the increasing popularity of Linux as a Unix-like operating system, and the demand for knowledge and education on those systems is another reason to focus on Unix systems. At the same time, the system design of the Virtual Unix Lab is kept flexible enough to also accommodate exercises in an heterogeneous environment, or even one that consists only of machines running Microsoft Windows.

**Why information science?** Information science acts in the triangle of science, information technology and man. It uses theories from parts of social sciences and humanities on one side, and engineering sciences on the other side. Topics that information science is related to include linguistics, philosophy, computer science, communication science, psychology, economics, law, politics and sociol-

<sup>1</sup> [Norman, 2007]

<sup>2</sup> [Hall, 2007]

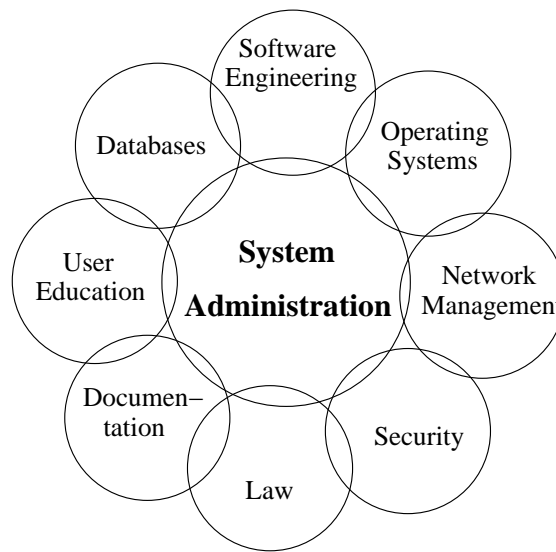


Figure 1.2: Topics related to system administration

ogy. As such, it is an interdisciplinary science<sup>1</sup>, figure 1.3 illustrates some of its working areas.

A number of areas which are touched within this work include e-learning, education science, human machine interfacing, tutoring systems, user adaption, linguistics, knowledge management and information processing.

Both system administration and information science are topics with many facets, which are combined here to solve the lack of education in former area. This follows the paradigm described in [Dagdilelis and Satratzemi, 1999] that teaching of technical topics also needs to give attention to didactics, not only technology.

## 1.2 The goal of the Virtual Unix Lab

Didactical analysis of the “System Administration” class in chapter 3 identifies a lack of interactive, hands-on exercises. The goal of the Virtual Unix Lab is to provide a system that allows students to do practical exercises in system administration, with an emphasis on cluster management.

The following key items are important for reaching this goal:

<sup>1</sup> [Kuhlen and Laisiepen, 2004] pp. 5

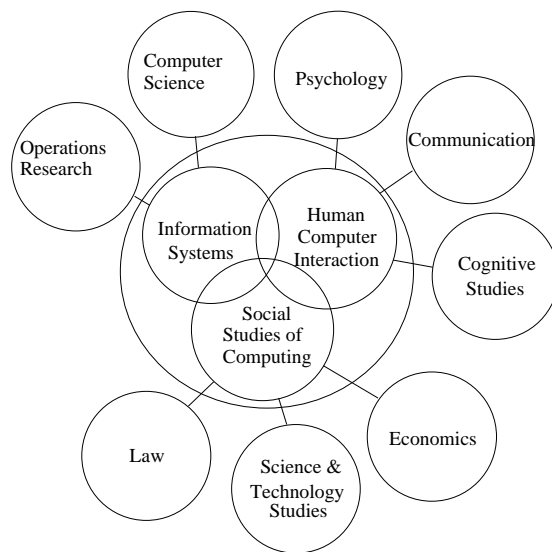


Figure 1.3: Topics of information science. Image source: [Cornell University, 2007]

- An interactive course environment for access
- Exercises with full access to lab machines, including system privileges
- Diagnosis via verification and analysis of exercise results
- Elaborated feedback on the exercise results
- A tutorial component to assist learning students
- User adaption to accommodate the system do students

From an instructional design point of view, the Virtual Unix Lab provides a “transaction shell” component in the sense of Merrill’s “Component Display Theory”<sup>1</sup>, where the component can be understood as a mutual, synchronous exchange of information between the learner and the learning system, thus allowing case-based learning. The Virtual Unix Lab acts like the simulations used by Kuyper<sup>2</sup> and Schulmeister<sup>3</sup>, with the added improvements of result verification, elaborated feedback<sup>4</sup>, a tutoring component, and user adaption. Following Hubwieser, the Virtual Unix Lab acts as medium

<sup>1</sup> [Merrill, 1983] pp. 279

<sup>2</sup> [Kuyper, 1998] p. 51

<sup>3</sup> [Schulmeister, 2007] pp. 351

<sup>4</sup> [Schulmeister, 2007] pp. 104

to support teaching the role of the system administrator for an information system: installing and maintaining the system as well as obtaining and displaying information<sup>1</sup>.

The key feature of the Virtual Unix Lab is that exercises happen on real systems with all possible errors and configurations, not on a simulated system with restricted functionality. Analysis of exercise results and feedback are also performed on those real systems, which sets a number of demands for the result verification process.

Other training systems have tried to provide assistance for Unix systems, too. One notable example is the Berkeley Unix Consultant project, of which its author stated that their “goals were not strictly technological, we did not feel that it was necessary [...] to produce a product that could actually be used in a real-world setting.”<sup>2</sup> In contrast to that, the Virtual Unix Lab has a clear goal of being able to be used in a real-world exercises, which already is done for the system described so far, in spite of being not strictly technological in this work. While the architecture definitions for the tutoring and user adaption do not include practical realization, they are still designed with this goal to eventually implement them.

The Virtual Unix Lab is intended to provide an adaptive tutoring system, not a learning management system. Learning management systems and their tasks and possibilities are described in chapter 3, [Yacef, 2004, p. 344], [Bruns and Gajewski, 2002, p. 16]. Embedding an adaptive web based educational system like the Virtual Unix Lab into a learning management system requires corresponding interfaces. These interfaces support describing of exercises for selecting and composing courses. This is important, as learning management platforms are not monolithic, closed platforms (like e.g. Web-CT and Blackboard) but more and more consist of open architectures with components that can be freely used, like uPortal, OKI, CampusSource and OpenUSS<sup>3</sup>. As no single interface can be considered as established at this point, integration of the Virtual Unix Lab into a learning management system is not covered in this work.

### 1.3 How this book is organized

The following chapters show how the goal of the Virtual Unix Lab is reached. They are separated into three parts.

In the first part, chapter 1 defines the goals of Virtual Unix Lab in this work, followed by an illustration of related works in chapter 2. Education of system administration is observed in chapter 3 as background its relationship to the learning system that the Virtual Unix Lab provides.

The second part covers diagnosis of the Virtual Unix Lab exercise results and feed-

---

<sup>1</sup> [Hubwieser, 2000] p. 39, pp. 63

<sup>2</sup> [Wilensky et al., 1988] p. 36

<sup>3</sup> [Nodenot et al., 2004] pp. 94

back to the user with the help of a domain specific language. The overall design of the system is outlined in chapter 4, chapter 5 covers the fundamentals of domain specific languages, and chapter 6 illustrates architecture and implementation within the existing Virtual Unix Lab in detail. The resulting system was evaluated as described in chapter 7.

The third part adds tutoring and user adaption to the basic Virtual Unix Lab system. Related fundamentals are covered in chapter 8, and the overall design is outlined in chapter 9. An architecture for a tutoring component for the Virtual Unix Lab is described in chapter 10. Based on tutoring, an architecture of a user adaptive component is described in chapter 11.

Chapter 12 draws conclusions from the work on exercise result verification, domain specific languages, tutoring, and user adaption, and gives future perspectives.

A number of appendices support the above chapters by giving example exercise components in appendix A and illustrating the database structure of the Virtual Unix Lab in appendix B. Appendix C lists data and program code used during evaluation. Appendix D shows the data that was used to attempt a reconstructive approach for a theory of bugs in system administration, and appendix E gives details on an analysis of exercises under tutorial and adaptive aspects.



## Chapter 2

### Related works

This chapter shows works that are related to the Virtual Unix Lab in some way. It illustrates works related to Computer science education in general, and System administration education in particular. A number of training systems for system administration and related topics are introduced, which leads to an observation of the status quo on Domain Specific Languages in the domain of operating systems, to what extent result verification is available in training systems, and the availability and state of tutoring systems and adaptive systems in Unix education. Other virtual labs that cover related topics are considered next, closing with a brief overview of the status of virtualization and emulation.

#### 2.1 Computer science education

The Virtual Unix Lab's application domain is in the wider field of computer science education. Related discussion – especially in Europe – is more focused on the didactics of computer science. Related keywords are “Computer Science Education” (CSE) and “Didaktik der Informatik” (DDI). A list of universities in Europe that do research and teaching in that field is listed in table 2.1, similar lists for the USA can be found at the websites of the ACM Special Interest Group for Computer Science Education (SIGCSE) at [SIGCSE, 2007], and of the Computer Science Teacher Association (CSTA) at [CSTA, 2007].

Care should be taken that the level of computer science education is often focused on the level of highschool/K12 rather than the scientific level of universities or colleges. Furthermore, computer science education is often performed as part of a general education of teachers for e.g. math and physics in the corresponding departments, as they use the computer as tool. A difference between this focus and the one of computer science education in computer science departments can be expected.

University	Homepage
Uni Antwerpen	<a href="http://www.ua.ac.be/main.aspx?c=.VAKBESE2005&amp;n=28907">http://www.ua.ac.be/main.aspx?c=.VAKBESE2005&amp;n=28907</a>
Uni Athen	<a href="http://www.di.uoa.gr/en/research_act.php?id=5">http://www.di.uoa.gr/en/research_act.php?id=5</a>
Uni Bayreuth	<a href="http://did.inf.uni-bayreuth.de/">http://did.inf.uni-bayreuth.de/</a>
FU Berlin	<a href="http://www.inf.fu-berlin.de/inst/ag-ddi/">http://www.inf.fu-berlin.de/inst/ag-ddi/</a>
Uni Dortmund	<a href="http://ddi.cs.uni-dortmund.de/">http://ddi.cs.uni-dortmund.de/</a>
TU Dresden	<a href="http://dil.inf.tu-dresden.de/">http://dil.inf.tu-dresden.de/</a>
Uni Erlangen	<a href="http://ddi.informatik.uni-erlangen.de/">http://ddi.informatik.uni-erlangen.de/</a>
Uni Frankfurt	<a href="http://www.informatik.uni-frankfurt.de/~poloczek/">http://www.informatik.uni-frankfurt.de/~poloczek/</a>
Uni Jena	<a href="http://www.uni-jena.de/Didaktik_der_Informatik.html">http://www.uni-jena.de/Didaktik_der_Informatik.html</a>
TU München	<a href="http://ddi.in.tum.de/">http://ddi.in.tum.de/</a>
Uni Münster	<a href="http://ddi.uni-muenster.de/">http://ddi.uni-muenster.de/</a>
Uni Paderborn	<a href="http://ddi.uni-paderborn.de/">http://ddi.uni-paderborn.de/</a>
Uni Passau	<a href="http://lehramt.fmi.uni-passau.de/informatik/">http://lehramt.fmi.uni-passau.de/informatik/</a>
Uni Potsdam	<a href="http://ddi.cs.uni-potsdam.de/">http://ddi.cs.uni-potsdam.de/</a>
ETH Zürich	<a href="http://www.inf.ethz.ch/education/courses/#dida">http://www.inf.ethz.ch/education/courses/#dida</a>

Table 2.1: Education of computer science at European universities [cited 2007-08-16]

Both SIGCSE and CSTA offer cooperation in the area of computer science education, a comparable chapter is available in the German Gesellschaft für Informatik (GI) [GI, 2007]. Related publications that cover computer science education esp. from the didactic side include [Hubwieser, 2000], [Humbert, 2006], and [Schubert and Schwill, 2004].

## 2.2 System administration education

After graduation, many students of computer science and related technical subjects like math and physics, find work in the area of system administration. Yet, system administration education at large is not common as part of computer science education today. Instead of focusing on system administration, a number of topics touch it from different angles as shown in figure 1.2, including operating systems, network management, databases, and management of information security (MIS)<sup>1,2,3,4</sup>. This section gives a number of pointers to ongoing research and education for system administration.

When looking at the list of universities that offer special courses on system administration in table 2.2, it is obvious that more entries with “FH” exist, i.e. schools that are focused more on practice than on theory. This emphasizes the point that system administration is mostly used as an add-on when performing education and research

<sup>1</sup> [Corbesero, 2003]

<sup>2</sup> [Adams and Erickson, 2001]

<sup>3</sup> [Mata-Toledo and Reyes-Garcia, 2002]

<sup>4</sup> [Yang, 2001]



University	Homepage
FH Augsburg	<a href="http://www.fh-augsburg.de/informatik/vorlesungen/unix/index_i.html">http://www.fh-augsburg.de/informatik/vorlesungen/unix/index_i.html</a>
FU Berlin	<a href="http://www.mi.fu-berlin.de/kvv/?veranstaltung=279">http://www.mi.fu-berlin.de/kvv/?veranstaltung=279</a>
Uni Bielefeld	<a href="http://www.rvs.uni-bielefeld.de/lecture/SysAdmin/">http://www.rvs.uni-bielefeld.de/lecture/SysAdmin/</a>
TU Chemnitz	<a href="http://www.tu-chemnitz.de/urz/lehre/psa/">http://www.tu-chemnitz.de/urz/lehre/psa/</a>
FH Hagenberg	<a href="http://cms.fh-hagenberg.at/_studienplan/1_0/sam/">http://cms.fh-hagenberg.at/_studienplan/1_0/sam/</a>
FH Isny	<a href="http://www.misc.st23.org/sysadmin/">http://www.misc.st23.org/sysadmin/</a>
Uni Mainz	<a href="http://www.zdv.uni-mainz.de/ak-sys/ak-sys-index.html">http://www.zdv.uni-mainz.de/ak-sys/ak-sys-index.html</a>
Uni Muenster	<a href="http://www.uni-muenster.de/ZIV/Lehre/2007_Wintersemester/kse2.html">http://www.uni-muenster.de/ZIV/Lehre/2007_Wintersemester/kse2.html</a>
FH Regensburg	<a href="http://www.feyrer.de/SA/">http://www.feyrer.de/SA/</a>

Table 2.2: Education of system administration at universities [cited 2007-08-16]

on other topics, rather than being a separate topic on its own.

A few theoretical considerations about teaching system administration can be found in [Burgess, 2000], example course material for teaching system administration are available in [Corbesero, 2003], [Campbell and Cohen, 2005] and [Feyrer, 2005].

Finally, the USENIX Special Interest Group for Sysadmins (SAGE) focuses on system administration from various angles, including education and professional development. An – unfortunately somewhat dated – overview is available in [Kuncicky and Wynn, 1998].

## 2.3 Training systems for system administration

After outlining the general state of computer science and system administration education, this section gives an overview of training systems. The choice is split into three parts, of which the first one describes systems that come closest to the Virtual Unix Lab due to their focus on education. The second part describes systems that are also available for training, but where the focus is more on the technical side of the system, including setup, deployment and access. Last, a few systems are introduced that allow training many Unix and system administration skills by offering user-level access, with no special emphasis on education, training, or feedback.

### 2.3.1 Systems focused on education

The following systems offer training for system administration and related topics as discussed above, and thus come closest to the Virtual Unix Lab:

- The Tele-Lab “IT-Security” offers automatic setup of machines for security ex-

ercises. No verification on the results of those exercises is performed, and no feedback is given to the student, though – this is left to the student taking part in the exercise. More information is available in [Hu et al., 2004].

- The TU Chemnitz “Root-lab” offers similar automated setup, and allows giving courses on topics that require modifications on the operating system and network configuration level. No support support for evaluation and feedback is available again. More information on use the system can be found at [Root-Lab, 2007b], an overview of the available hardware is available at [Root-Lab, 2007a], and details on the setup of the system are described in [Heinichen et al., 2007].
- The Remote Laboratory Emulation System (RLES) described in [Border, 2007] uses virtual machines to provide a training environment for changes on the system-level. Again, the system does not offer feedback to the student.
- LiveFire Labs offer a Unix system administration course with remote access to their lab. Again, no mention of feedback is given. Information about the LiveFire Labs can be found at [LiveFire Labs, 2007b], the system administration course is described at [LiveFire Labs, 2007c] and details on their Internet Lab can be found at [LiveFire Labs, 2007a].

### 2.3.2 Systems focused on deployment

The following systems support installation and deployment of various operating systems to a number of real and virtual machines, to perform training and research in the areas of operating systems, networking and related topics:

- The Emulabs project is “a network testbed, giving researchers a wide range of environments in which to develop, debug, and evaluate their systems. The name Emulab refers both to a facility and to a software system.”<sup>1</sup> Facilities offered include emulated computer systems with a choice of operating systems, 802.11 and mobile wireless networks as well as software-defined radio and sensor networks.

More information on the Emulab project is available in [Anderson et al., 2006], [Lepreau, 2006], and [Eide et al., 2006]. A list of other Emulab testbeds is available at [Emulab, 2007b].

- The openQRM project provides “an open source systems management platform that automates enterprise data centers and keeps them running.”<sup>2</sup> In a data center environment, the number of systems is always growing, and automation of setup, installation and esp. maintenance is needed to assist system administrators from manually repeating error-prone tasks. The openQRM system offers help in those

---

<sup>1</sup> [Emulab, 2007a]

<sup>2</sup> [openQRM, 2007]

Software	Homepage
Acronis True Image	<a href="http://www.acronis.com/homecomputing/products/trueimage/">http://www.acronis.com/homecomputing/products/trueimage/</a>
g4l	<a href="http://sourceforge.net/projects/g4l">http://sourceforge.net/projects/g4l</a>
g4u	<a href="http://www.feyrer.de/g4u/">http://www.feyrer.de/g4u/</a>
Norton Ghost	<a href="http://www.symantec.com/ghost">http://www.symantec.com/ghost</a>
Paragon Drive Backup	<a href="http://www.drive-backup.com/home/personal/">http://www.drive-backup.com/home/personal/</a>
Symantec DriveImage	<a href="http://www.symantec.com/">http://www.symantec.com/</a>
YAGI	<a href="http://dan.deam.org/yagi.php">http://dan.deam.org/yagi.php</a>

Table 2.3: Harddisk image cloning software [cited 2007-08-18]

areas, for both heterogeneous x86 PCs and virtual machines. See the openQRM homepage at [openQRM, 2007] for more information.

None of those systems offer facilities to evaluate status of the setup, and compare it to some goals that are defined in an learning environment. They can still serve as base for such a system, e.g. the deployment subsystem of the Virtual Unix Lab could benefit from work of those projects.

Besides those fully integrated systems, a number of low-level software products are available that help in cloning systems by replication of harddrives, which may be useful when implementing a similar system, see table 2.3.

### 2.3.3 Systems offering user-level access

From the number of operating systems available today, some are better fit for operation and administration from remote systems than others. While Microsoft Windows systems allow some remote access, Unix systems of any flavour – Linux, Solaris, NetBSD, and all others<sup>1</sup> – can be fully used over the network.

This section outlines a number of systems that offer remote access for using the systems without admin privileges. No admin privileges means that the systems do not require fresh setup, and many areas of system administration can be learned without changing the system, so they are considered an important resource:

- The “Virtual Unix Lab” of the University of Cyprus provides “machines stuck in a dark room, where users can access from other terminal rooms via telnet, rsh or x-sessions.”<sup>2</sup> Documentation on the lab is only available in greek, see [University of Cypria, Department of Computer Science, 2007a]. Some more general information is available in english language at [University of Cypria, Department of Computer Science, 2007b].

<sup>1</sup> [Lévénez, 2007]

<sup>2</sup> [Zoulas, 2007]

Similar labs that are accessible for practicing to students either for local or remote access can be found at other universities, too.

- There are a number of commercial and free public access Unix systems, e.g. by the Super Dimension Fortress<sup>1</sup>, Panix<sup>2</sup>, Solaria<sup>3</sup>, and Nixsys<sup>4</sup>.
- Various vendors and non-profit organizations have setup machines for remote access so users and developers can experience the hardware and/or specific features of the operating system. Examples would be Intel's cooperation with the Linux Foundation on the Open Source Lab<sup>5</sup>, the SourceForge shell service<sup>6</sup>, and HP's TestDrive program<sup>7</sup>.

## 2.4 Domain specific languages

This section gives an overview of two areas in which domain specific languages are employed: generation of system setups, and verification of system status.

The following domain specific languages that are used in creating system setups:

- Cfengine can be used to describe setup for one or many systems, including configuration parameters for the operating system and networking. It can create configuration for the systems, depending on their exact operating system, environment, and other constraints. For more information on cfengine, see [Burgess, 1995] and [Burgess and Frisch, 2007].
- Puppet is intended to be a successor to cfengine. It addresses some of the shortcomings of cfengine in the areas of the configuration language, portability, and support community. Information on Puppet can be found at [Reductive Labs, 2007b], a comparison between Puppet and cfengine is available at [Reductive Labs, 2007a].
- In [Zheng et al., 2007, pp. 219], the authors approach the problem of misconfiguration for Internet services. They propose a software infrastructure that eliminates misconfiguration by defining their own scripting language, configuration file templates, communicating runtime monitors, and heuristic algorithms to detect dependencies between configuration parameters and select ideal configurations. The scripting language they use is a domain specific language for their area of application.

---

<sup>1</sup> [Super Dimension Fortress, 2007]

<sup>2</sup> [Public Access Networks Corporation, 2007]

<sup>3</sup> [sol.net Network Services, 2007]

<sup>4</sup> [Nixsys, 2007]

<sup>5</sup> [The Linux Foundation, 2007]

<sup>6</sup> [SourceForge, 2007]

<sup>7</sup> [Hewlett Packard, 2007]

- [Madhavapeddy et al., 2007, pp. 101] describes an OCAML-based approach to generate network and application layer protocols, ranging from Ethernet to SSH and BGP.
- [Narain, 2005] also describes model finding in network configurations, defining the desired configuration and its properties via a domain specific description language. A similar approach is taken for validation of network configurations in the Emulab project as described in [Anderson et al., 2006].

The named works have a strong focus on network configuration. Applying standard interfaces to components in system management will allow to also apply those mechanisms in system administration eventually, and to create system configuration automatically.

Besides system setup, analyzing, troubleshooting and debugging are vital areas that system administrators need to be trained in. Similar, systems analyzing and evaluating an administrator's performance have to evaluate the system state. Currently, no system like the VUDSL is widely deployed, but a number of domain specific languages exist that perform evaluation for related areas:

- GNU autoconf is used by software programmers to determine in what environment the program will be compiled. Attributes of the environment include the operating system, installed software packages, places (paths) in which to look for various programs, and many more. The system itself is based on the m4 macro processor. See [Elliston et al., 2000] for more information.
- Perl's t/TEST framework is used to implement unit tests for Perl modules and other software written in the Perl programming language. For more information see the Perl Test(3) module at CPAN::Test and the related modules in the "See also" section there.
- Nessus' "Network Attack Scripting Language" (NASL) allows to write programs that automate penetration testing. The programs test for known issues in local and network services under security aspects, and report any problems found. An introduction of the Nessus system can found in [Dhanjani, 2004], a reference manual of the NASL language can be found in [Beale and Rogers, 2007, pp. 363, 423].

## 2.5 Result verification, diagnosis and feedback

In learning systems, feedback to the student is considered important. To provide that, the student's actions and/or their cause need to be observed, and a diagnostic process will lead to feedback to the student. A number of systems today offer an environment

in which students can experiment in complex areas, but diagnosis and feedback is mostly left to the student.

In some systems, the process of giving feedback means to tell the teacher if he did a good job, but this is not what is meant here.

The following list reflects the state of result verification after modification of a learning system:

- Moodle provides a full-featured learning management system that can be used to provide learning material to students, facilitate communication between students and teachers, and offer tests of the students' knowledge. Unfortunately, the tests are either simple multiple-choice tests, or tightly coupled to the subject module, so no general verification of exercise results is available. More information can be found in [Rice, 2006], [Cole, 2005], and on the Moodle homepage at [Moodle, 2007].
- A set of changes that need to be determined on systems is within the security area, to detect break-ins performed either manually or automatically by some worm or virus. In general, the detection routines of every virus scanner can be observed here. The matter comprises problems from linguistics, pattern matching and automata theory. An overview of the area can be found in [Patcha and Park, 2007], implementation and application examples are given in [Tucek et al., 2007, pp. 115], [Kolter and Maloof, 2006] and [Zhang et al., 2007].

## 2.6 Tutoring systems in Unix education

The related works observed so far focus strongly on the domain of system administration. When widening that focus, a wealth of projects can be found that offer tutoring for use of Unix systems in general. I.e. instead of administration, emphasis is on use of the system from a user's point of view, including tasks like file handling and editing. Noteworthy projects in this area include:

- The Berkeley Unix Consultant (UC) was a research project that was never intended to be used in practice, see chapter 1. Various aspects of the tutoring system are described in [Chin, 1983] and [Wilensky et al., 1988].
- The AQUA project described in [Quilici et al., 1986] and [Quilici, 2000] also provides a Unix Advisor that observes neophyte users' behavior, infers plans, and detects misconceptions.
- TNT, the talking Tutor'n'Trainer, is a system for teaching the use of interactive computer systems, focusing on the Unix "vi" editor. See [Nakatani et al., 1986].

- COMFOHELP is an adaptive help system that supports the COMFOTEX graphical text processing program, which is available on some Unix systems. COMFOHELP works by observing user actions, determining the user's plan, and assisting him in reaching that goal. Details can be found in [Krause et al., 1993].
- AutoBash is an assistant that tries to analyze a user's input into a system by both looking at the commands typed as well as system calls made for interactive programs. It tries to infer a plan, detect any false approaches, takes wrong steps back and perform the right operations to get the user to his goal. The system is described in [Su et al., 2007].
- The NAGLICE system introduced in [Manaris and Pritchard, 1993] and [Manaris et al., 1994] describe development of a natural language interface to the Unix operating system.
- The GOETHE project described in [Heyer et al., 1990] is a natural language system focusing on knowledge representation and semantics in the complex domain of the Unix operating system. Focus of the work is on plan recognition via a frame-based approach.
- The "Yucca-\*" project is a successor to a number of projects, and it focuses on natural language interaction and plan recognition in complex environments. "Complex" in that context means constructs like Unix shell pipes ("command1 | command1"), which – when compared to the domain of system administration – puts this project into perspective. See [Hegner, 2000].

## 2.7 Adaptive systems in Unix education

In the context of research on the Unix operating system's user interface, some of the tutoring systems were extended to provide adaption to the user. Here is a selection of related works:

- Menix is an adaptive user interface that presents a limited set of Unix commands to a user. The commands presented are selected based on a predefined level of information for the user, which in term is determined from the user's past interaction with the Unix system. See [Chauvin, 1991] for more information.
- [Tyler and Treu, 1989] describes an interface architecture to provide an adaptive task-specific context for the user.
- Other systems that focus on tutoring and that were mentioned in the previous section also grew extensions for adaption, see e.g. the GOETHE and TNT projects, and [Chin, 1986] for user modeling in the Berkeley Unix Consultant.

## 2.8 Other virtual labs

Virtual labs are becoming popular for many areas of application, to decouple time and physical presence of students from the lab hours and rooms of traditional labs. While the above sections have shown that the supply for system administration and its related topics is scarce, there are still a number of projects that are noteworthy in related areas. Aspects like general handling, user interfacing, presence of learning material, and other aspects can be learned from them:

- The Laboratory of Communication Technologies of the University of Applied Sciences Regensburg, Germany, offers a virtual lab in cooperation with the Virtuelle Hochschule Bayern (VHB). The lab allows practicing wireless and wired networks, switch and router setup, offers automatic setup of the exercise components, and access to the exercise systems via VNC. Verification of exercise results is part of the tasks of the students, and as such performed by the students. See [Fachhochschule Regensburg, 2007] for more information.
- The “Virtuelles Informatik-Labor” (VILAB) of the FernUniversität Hagen, Germany, allows practice of various topics related to computer science: programming, neural networks, databases, and knowledge based systems. The system is designed to give adaptive feedback as described in [Lütticke and Helbig, 2004, pp. 443], more information on the system can be found at [FernUniversität Hagen, 2007].
- The “Verbund Virtuelles Labor” (VVL) is a collaboration of various universities from Baden-Württemberg, Germany, to make virtual labs available for a number of topics, including robotics, lab engineering, measurement engineering, 2D and 3D graphics, and others. See the homepage at [Virtuelle Hochschule Baden-Württemberg, 2007] for more information.

Lists of further simulations and virtual labs can be found in [Ma and Nickerson, 2006], [Kopp and Michl, 2000], and [Bundesministerium für Bildung und Forschung, 2004].

## 2.9 Virtualization & emulation

The Virtual Unix Lab got its name by providing a “virtual” lab environment, i.e. one where the place at which the student takes the exercise is de-coupled from the real lab, exercise time is not bound to any lab opening hours, and which students can access from anywhere and at any time.

No virtualization techniques are currently used for the implementation of the Virtual Unix Lab, and there is a lot of potential in that area, as recent publications show, e.g.



[Guruprasad et al., 2005], [Vollrath and Jenkins, 2004], and [Adams and Laverell, 2005]. A comparison of various technologies related to virtualization was done by the Emulab project and can be found in [Hibler et al., 2004].

For the purpose of further extensions of the Virtual Unix Lab to use virtual machines instead of real ones, an overview of available solutions for virtualization and emulation of various systems as of this writing are listed in table 2.4.

Besides virtualization, a number of other technologies that may prove useful for future works on the Virtual Unix Lab. Given the goal of providing several operating systems, and not focusing on one system, Solaris Zones<sup>1</sup>, FreeBSD Jails<sup>2,3</sup>, and UserMode Linux<sup>4,5</sup> may be of interest.

The topics covered in this work – verification of exercise results, tutoring and user adaption – are not influenced whether virtualization is used or not, though.

---

<sup>1</sup> [Sun Microsystems, 2007]

<sup>2</sup> [Kamp and Watson, 2007]

<sup>3</sup> [The FreeBSD Documentation Project, 2007] Chapter 15: Jails

<sup>4</sup> [User Mode Linux, 2007]

<sup>5</sup> [Dike, 2006]

Software	Homepage
Ardi Executor	<a href="http://www.ardi.com/executor.php">http://www.ardi.com/executor.php</a>
Basilisk II	<a href="http://basilisk.cebix.net/">http://basilisk.cebix.net/</a>
bochs	<a href="http://bochs.sourceforge.net/">http://bochs.sourceforge.net/</a>
CoLinux	<a href="http://www.colinux.org/">http://www.colinux.org/</a>
dosbox	<a href="http://dosbox.sf.net/">http://dosbox.sf.net/</a>
FAUmachine	<a href="http://www.faumachine.org/">http://www.faumachine.org/</a>
gxemul	<a href="http://gavare.se/gxemul/">http://gavare.se/gxemul/</a>
JPC	<a href="http://www.physics.ox.ac.uk/jpc/">http://www.physics.ox.ac.uk/jpc/</a>
LilyVM	<a href="http://lilyvm.sourceforge.net/">http://lilyvm.sourceforge.net/</a>
Microsoft Virtual Server	<a href="http://www.microsoft.com/virtualserver/">http://www.microsoft.com/virtualserver/</a>
Parallels	<a href="http://www.parallels.com/">http://www.parallels.com/</a>
PearPC	<a href="http://pearpc.sourceforge.net/">http://pearpc.sourceforge.net/</a>
qemu	<a href="http://www.qemu.org/">http://www.qemu.org/</a>
Serenity Virtual Station	<a href="http://www.serenityvirtual.com/">http://www.serenityvirtual.com/</a>
SIMH	<a href="http://simh.trailing-edge.com/">http://simh.trailing-edge.com/</a>
SkyEye	<a href="http://www.skyeye.org/">http://www.skyeye.org/</a>
VirtualBox	<a href="http://www.virtualbox.org/">http://www.virtualbox.org/</a>
VirtualIron	<a href="http://www.virtualiron.com/">http://www.virtualiron.com/</a>
VirtualPC <sup>1</sup>	<a href="http://www.microsoft.com/virtualpc/">http://www.microsoft.com/virtualpc/</a>
Virtuozzo	<a href="http://www.sw-soft.com/en/products/virtuozzo/">http://www.sw-soft.com/en/products/virtuozzo/</a>
VMWare	<a href="http://www.VMware.com/">http://www.VMware.com/</a>
WABI	<a href="http://docs.sun.com/app/docs/doc/802-6306/">http://docs.sun.com/app/docs/doc/802-6306/</a>
Xen	<a href="http://www.cl.cam.ac.uk/research/srg/netos/xen/">http://www.cl.cam.ac.uk/research/srg/netos/xen/</a>

Table 2.4: Virtualization and emulation software [cited 2007-08-16]

# Chapter 3

## Education of system administration

This chapter illustrates didactics and education of system administration. It introduces theories of didactics, then applies them to an existing class on system administration. An analysis of that situation leads to future directions for the domain of teaching system administration.

### 3.1 Fundamentals of education

There are several aspects to instructional research which will be investigated in this chapter. First, psychology and learning theory explain how acquisition of new information works in the human mind. Second, didactic realization is explained by instruction theory, which describes how to model information so that it is best fit for one of several learning theories. Third, instructional design determines how to prepare teaching material to fit for instructional and learning theories<sup>1</sup>. After looking at the various theories, dimensions of implementation and adaption will be discussed, followed by a look at alternative learning-theoretical approaches to consider. This defines an ideal progression of education, and an optimal set of tools that is applied to an existing course on system administration.

#### 3.1.1 Psychology and learning theory

The process of human learning can be approached from two sides: philosophy and psychology. From the philosophical side, epistemology gives a view on knowledge and learning with its impact on teaching<sup>2</sup>. On the other side, psychology has recognized

---

<sup>1</sup> [Kuyper, 1998] p. 49

<sup>2</sup> [Hammer and Elby, 2000] p. 2

the relation of human learning with its subject early, and has developed theories of learning, which in turn emerged into theories of instruction and instructional designs<sup>1</sup>.

This section gives an overview of the various aspects of teaching, starting with the psychological learning theories and moving on the possible realizations.

**Behaviorism:** Early learning theories go back to Iwan Pawlow, who observed “conditional reflexes” in his famous “drooling-dog” experiment<sup>2</sup>, and Edward Thorndike, who found laws about learning by trial and error, by experimenting with cats<sup>3</sup>. John Watson and Burrhus Skinner picked up their works, and while Watson coined the term “behaviorist” in his article “Psychology as the Behaviorist Views it”<sup>4</sup>, Skinner made experiments with operant conditioning of pigeons<sup>5</sup>. Skinner was also influenced by Sidney Pressey’s “testing and learning” machines<sup>6,7</sup>, and in collaboration with James Holland he worked on a “teaching machine”<sup>8</sup>, which led him to define the term and concepts of “programmed teaching.”<sup>9</sup>

Based on these fundamentals, Norbert Wiener’s theories of cybernetics<sup>10</sup>, and Helmar Frank’s “cybernetic pedagogy”<sup>11</sup>, it was hoped that teaching could be automated with the aid of machines (computers), so that human learning can be guided in a better way<sup>12</sup>.

The focus of the behavioristic approach to teaching is to supply information to the learner, give him time to understand, then ask questions on the subject taught, and give feedback based on the quality of the reaction. The whole process can be seen in figure 3.1<sup>13,14</sup>.

By repeating this sequence, simple tasks can be trained efficiently, as Pawlow and others have shown. The same method works for training humans as well. A few examples on how to design instructions to fit behavioristic learning will be introduced in section 3.1.2.

The behavioristic learning theory is most appropriate for small learning steps. Bigger learning goals have to be split into several smaller goals, which are usually presented in a sequential manner<sup>15</sup>.

---

<sup>1</sup> [Kuyper, 1998] p. 49

<sup>2</sup> [Pawlow, 1972] pp. 203

<sup>3</sup> [Thorndike, 1911]

<sup>4</sup> [Watson, 1913] pp. 158

<sup>5</sup> [Skinner, 1947] p. 168ff

<sup>6</sup> [Pressey, 1926]

<sup>7</sup> [Pressey, 1927]

<sup>8</sup> [Holland and Skinner, 1961] p. V

<sup>9</sup> [Skinner, 1968]

<sup>10</sup> [Wiener, 1948] pp. 11

<sup>11</sup> [Frank, 1969]

<sup>12</sup> [Seidel and Lipsmeier, 1989] p. 32

<sup>13</sup> [Nösekabel, 2005] p. 6, Figure 2

<sup>14</sup> [Kerres, 1998] p. 46

<sup>15</sup> [Tulodziecki, 2000] pp. 57

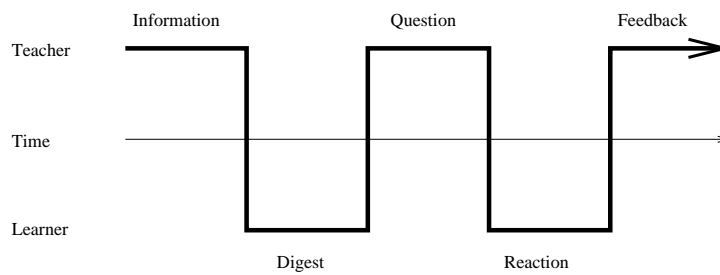


Figure 3.1: Behavioristic approach of teaching. Image Source: [Kerres, 1998, p. 46]

Critics of behaviorism point out that the approach does not consider the individual nature of human beings enough, e.g. Watson described that there is “no dividing line between man and brute.”<sup>1</sup> This led to development of metacognition and cognitivism as learning theories<sup>2</sup>.

**Cognitivism:** The concept of cognitivism goes back to the early days of the 20th century, notable names are Jean Piaget, Edward Tolman, Jerome Bruners and Wolfgang Köhler<sup>3</sup>.

The idea in cognitivism is to view the learner as an individual, which is able to process external stimulus on his own, and do more than just react to it. As such, the learner behaves as an interactive receiver of messages that contains news and knowledge in the sense of Shannon and Weaver’s communication theory, and messages can be carried in various media<sup>4</sup>. Learning is considered a creative process of problem solving<sup>5</sup>, and Piaget proposed that the learner adapts to the problem domain and solves it by using assimilation and accommodation<sup>6</sup>. In this context, accommodation and assimilation mean to adjust the learned cognitive concepts to new environments, and to match new external objects and conditions to the individual’s internal structure by modifying the existing cognitive structures<sup>7</sup>. Cognitive development happens through both external influence by learning material, and internal influence by the learner’s existing cognitive structures. The learner’s “knowledge” is considered to be the sum of all patterns of recognition, understanding and processing available to the individual, including its environment<sup>8</sup>. A number of ways to model instructions after constructivist approaches will be illustrated in section 3.1.2.

The cognitivist learning theory is best used for complex subjects that go be-

<sup>1</sup> [Watson, 1913] p. 158

<sup>2</sup> [Seidel and Lipsmeier, 1989] pp. 36

<sup>3</sup> [Seidel and Lipsmeier, 1989] p. 26

<sup>4</sup> [Shannon and Weaver, 1949] pp. 31

<sup>5</sup> [Seidel and Lipsmeier, 1989] p. 26

<sup>6</sup> [Piaget, 1967] pp. 7

<sup>7</sup> [Schulmeister, 2007] p. 67

<sup>8</sup> [Tulodziecki, 2000] p. 58

yond pure factual knowledge. Depending on the learning goals, constructivistic approaches can be used to<sup>1</sup>:

- Determine which kind of knowledge structures to build up.  
Cognitive theories can not only be used for learning simple facts (declarative knowledge), but also rules (procedural knowledge) and concepts (contextual knowledge).
- Determine how knowledge is stored in the brain.  
Various theories can be applied here as well, for example by giving the topic taught a context with other subjects that it can associate with (theory of meaning structures), explaining concepts both in words and non-verbal (dual encoding theory) or trying to analyze all structural and functional components (theory of mental models).
- Determine if specific topics or general strategies should be learned.  
Specific topics such as system administration include various areas of science which provide an intellectual challenge. In contrast, it is also possible to teach general development aspects, e.g. social or moral considerations.

Leaving the possibilities that the constructivist learning theory offers aside, the basic concept is still based on the interaction between external medial presentation and internal processing, just as in behaviorism.

**Constructivism:** The constructivist learning theory is based on works by a number of philosophers, most notably Jean Piaget. The central thesis is that cognition is construction and interpretation, and that objective, subject-independent learning and understanding is not possible<sup>2</sup>. As such, it goes one step further than cognitive theory: constructivism emphasizes the “individual” components like experience and the way of thinking first found in cognitivism even more, to a point where it does not include any external instructions to the learner. Instead, the idea of constructivism is to act freely in an environment, and construct new knowledge from existing knowledge and interpretations of feedback given to various actions in an act of recognition. This act is individual to each learner.

Due to this subjective nature, there is no “best” way of teaching in constructivism. Instead, learning happens by actively dealing with tasks that provides a context to the learning process, and that make acquired knowledge context-bound or “situated.” This approach also prevents “inert knowledge”, i.e. knowledge that was once learned, but cannot be applied in a given situation as there is no mental connection between the context given by the situation and the knowledge needed to be applied<sup>3</sup>.

During the learning process, knowledge is created dynamically and is not stored in a fixed way. As a consequence, knowledge cannot be passed on without

<sup>1</sup> [Tulodziecki, 2000] pp. 58

<sup>2</sup> [Bruns and Gajewski, 2002] p. 14

<sup>3</sup> [Bruns and Gajewski, 2002] p. 15

repeating the same learning process in the receiving learner, who has to reconstruct that knowledge<sup>1</sup>.

The “creation” of knowledge can also be improved by encouraging communication between students and a teacher or in a learning group among themselves, which allows changing role and perspective. That way, the classical roles between teacher and student are not sharply defined any more, and it becomes clear that social interaction between learners is an important part of constructivism<sup>2</sup>.

There are several approaches to model “instruction” (put into quotes here as there is no concept of instruction in constructivism). Among them are the concept of cognitive apprenticeship<sup>3</sup>, knowledge communities<sup>4</sup>, and cognitive tools<sup>5</sup>. Some of these will be discussed in section 3.1.2.

As a summary, constructivistic approaches are best fit for approaching complex subjects and learning goals, as it goes far beyond the cognitive teach-review cycle. There are downsides though, which become obvious when looking at the didactic realization.

This section introduced three fundamental learning theories with some of their basic ideas. None of the theories is ideal for teaching every subject – some are better fit for simple, introductory topics, while others are better fit for advanced topics. This needs to be considered when approaching the didactic realization of a teaching system, which is what the next section covers.

### 3.1.2 Didactic realization, instruction theory and instructional design

The psychological and pedagogical foundations given on learning theories need to be applied to create learning systems, which is covered in instruction theory and in instructional design. Starting from the three learning theories introduced in the previous section, some methods for realizing them will be introduced here.

This section only gives an overview on the methods needed to approach system administration. Related introductory texts on instruction theory and design can be found in [Eikenbusch and Leuders, 2004, pp. 153] and [Wiggins, 1989], an in-depth coverage of the topic can be found in [Gagné, 1967], [Gagné and Briggs, 1974], [Richey, 1986], [Reigeluth, 1983], and [Schulmeister, 2007].

**Behaviorism:** The “instruction paradigm” provides a realization of the behavioristic learning theory. It assigns the learner a passive but nonetheless important role of

---

<sup>1</sup> [Schulmeister, 2007] pp. 67

<sup>2</sup> [Bruns and Gajewski, 2002] pp. 15

<sup>3</sup> [Schulmeister, 2007] pp. 75

<sup>4</sup> [Schulmeister, 2007] pp. 76

<sup>5</sup> [Schulmeister, 2007] pp. 79, 315

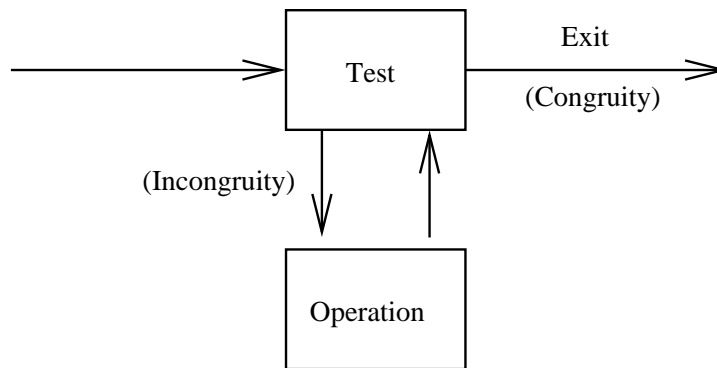


Figure 3.2: The TOTE model. Image source: [Miller et al., 1960, p. 26]

receiving and processing information and instruction given by a tutor, teacher, or a teaching program. After each learning unit, feedback is provided before moving on to the next unit<sup>1</sup>. This kind of instruction is also known as “drill & practice” due to its main components. It provides the basic concept behind programmed teaching<sup>2</sup>.

Big lectures are split into small learning atoms by experts on the subject. It is possible to teach the learning atoms in several ways, either by classroom teaching, by providing it in book form, or via a computer program. In all but the first case, the learner can decide the speed of progresses on his own<sup>3</sup>.

With the aid of computer programs, it is even possible to deny progress to later learning units until prior ones are mastered successfully. A system to assert “success” is needed in that case. This schema is reflected in the TOTE-model developed by Miller, Galanter and Bram in 1960, which consists of four phases as illustrated in figure 3.2: Test, Operation, Test and Exit. First, a condition is tested, and unless it is satisfied, a learning operation has to happen. This is repeated until success of the test is indicated by congruity, leading to an exit of the procedure<sup>4,5</sup>.

**Cognitivism:** As an alternative to the instruction paradigm, the “problem solving paradigm” corresponds to realization of a cognitivistic learning theory. The central idea there is to provide an environment where learners can search their own challenges within an open learning environment, or solve given problems with no clear description of how to solve them. That way, learners are encouraged to use their existing knowledge and the tools and information available in the

<sup>1</sup> [Bruns and Gajewski, 2002] p. 32

<sup>2</sup> [Seidel and Lipsmeier, 1989] p. 40

<sup>3</sup> [Kerres, 1998] p. 49

<sup>4</sup> [Miller et al., 1960]

<sup>5</sup> [Seidel and Lipsmeier, 1989] p. 28



learning environment to construct new knowledge<sup>1</sup>.

Environments that encourage this kind of learning are those for “explorative learning” as described in [Bruner, 1961], and as a special case the microworlds described by Seymour Papert. A microworld in this context means a small (“micro”) environment (“world”) with a fixed set of rules, within which a given task should be solved. Probably the best-known microworld includes the “logo” programming language. Logo allows teaching procedures, interaction and list processing on one side, but as it also provides a facility to move a turtle across the screen in a way described by the user, it can also be used to teach basic concepts of computer science and programming<sup>2</sup>.

In general, several types of tasks can be requested from the learner, depending on the kind of knowledge that he should build up. On the one hand, correlations can best be learned by predicting the behavior of the system when changing various parameters. On the other hand, the task can be to explain which parameters need changing to achieve a certain condition of the system, and problems can be solved by choosing the proper conditions and changes for a given effect<sup>3</sup>.

**Constructivism:** Verbalizing as described in the prior paragraphs helps the learner order vague concepts that are mentally present and internally connected to a subject into a clear form needed for communication. Learning in dialog with a teacher, a tutor, or with other learners in a learning group is good for more than just working on a given task. Due to the changes in role and position required for participants of a learning group, it goes beyond the given task, and encourages constructing new knowledge. Tools found useful for supporting communication can be divided into synchronous and asynchronous groups<sup>4</sup>:

- **Asynchronous communication tools** include email, discussion forums and electronic bulletin board systems, user galleries, and facilities for giving feedback on existing material.
- **Synchronous communication tools** include video- and audio-conferencing, application sharing, interactive whiteboards, chat, and instant messaging,

Similar to cognitivism, the approach taken in constructivism is to follow the “problem solving paradigm.”<sup>5</sup> But given the basic idea behind constructivism that no instructions are given at all, the learning environment needs to be much more flexible. It has to provide a framework to create reflections of learned knowledge in various media, communicate and cooperate with others, and construct new media from existing one<sup>6</sup>.

---

<sup>1</sup> [Bruns and Gajewski, 2002] p. 32

<sup>2</sup> [Papert, 1982] p. 152

<sup>3</sup> [Kuyper, 1998] p. 53

<sup>4</sup> [Bruns and Gajewski, 2002] pp. 48

<sup>5</sup> [Bruns and Gajewski, 2002] p. 32

<sup>6</sup> [Bruns and Gajewski, 2002] p. 16

To allow learners free navigation in a wealth of information, a display of information that connects all information with everything related to it is necessary. In effect, this means building a hypertext or even hypermedia structure as can be found (to some extent) on the World Wide Web today<sup>1</sup>. Such a tightly interconnected system would allow references to other resources of information, as can be found in electronic libraries, discussion groups like the Usenet etc., and it would also accommodate a wide variety of media formats, including (hyper-linked) text, images, audio and video<sup>2</sup>.

Prospective learners can apply a number of problem solving strategies, e.g. depth first, width first, hill-climbing and means-end analysis. All of them require a model of the problem domain, though. These techniques can be used for plan recognition and automation of assistance as discussed in section 8.1.2.2.

Methodical elements to provide these media and various way for accessing them include hyperlinked content, guided tours, a document pool, encyclopedia, interactive exercises, business games and map exercises, simulations, online tests, movies and help-functions for all these facilities<sup>3</sup>.

Besides offering an environment for communication, collaboration, navigation and construction, learning systems like microworlds, simulations and intelligent tutoring systems (ITS) offer interaction within an environment for creating new knowledge from various views on new and existing knowledge gained through them. A number of projects realizing these ideas are described in [Schulmeister, 2007, pp. 321, 351, 171] and [Schulmeister, 2002, pp. 16, 178, 241].

Another approach to realize constructivistic learning theories is by not dealing with a particular subject either directly or via some (possibly simulated) interface, but by talking about it. For this approach, a teacher or tutor is needed to ask questions that the learner answers. The most well-known form of this is known as the “Socratic dialogue.” By considering all aspects of a certain topic, unknown areas will be discovered, and relationship to existing knowledge can be used to build up new mental connections, and thus knowledge, through interaction with a guiding instance<sup>4</sup>.

This section covered methods which can be employed to realize various learning theories. There is a variety of options to choose from, and the effects will also vary widely. There are various levels at which these methods can be integrated into teaching environments, which is covered in the next section.

---

<sup>1</sup> [Schulmeister, 2007] p. 77

<sup>2</sup> [Schulmeister, 2007] p. 22

<sup>3</sup> [Bruns and Gajewski, 2002] pp. 44

<sup>4</sup> [Bruns and Gajewski, 2002] p. 31

### 3.1.3 Dimension of implementation and adaption

In 1929, Edward Thorndike and Arthur Gates pondered “If, by a miracle of mechanical ingenuity, a book could be so arranged that only to him who had done what was directed on page one would page two become visible, and so on, much that now requires personal instruction could be managed by print.”<sup>1</sup> An early prototype of such a book was built in Alan Kay’s “DynaBook” project<sup>2,3</sup>. Looking at this idea from today’s perspective, it is obvious that one would use a computer to construct a “book” with these constraints.

Offering a guided tour through a book is only one of several methodical forms for teaching. The spectrum ranges from pure classroom teaching as performed in the current “System Administration” class as described in section 3.2, over a mixture between presence teaching with virtual components to pure virtual teaching as e.g. offered by the “Virtuelle Hochschule Bayern” (VHB)<sup>4</sup> and others<sup>5</sup>. When employing methods for online learning, various degrees exist. Examples include self-paced online learning, collaborative online learning (tele-tutoring), and live online learning (tele-teaching)<sup>6</sup>. The central entity here is the “learning environment”, in which teaching and learning happens<sup>7</sup>. Depending on the type of education and the learning theory applied, various methodical communicative elements can be used, as described in section 3.1.2.

One component of the learning environment not covered yet is the “feedback” given to learners, which means the reaction of the learning platform to attempts on solving a task given to the learner<sup>8</sup>. While existing platforms often use multiple choice texts and gaps in a text to fill in, all these test forms train basic behavioristic learning instead of real understanding of concepts<sup>9</sup>. On one hand side, more advanced concepts like interactive maps, images, or feedback on a given scenario that the learner was asked to create are rarely found, even if these advanced ways for evaluation and feedback are more appropriate for the concepts taught via realizations of cognitivistic and constructivistic learning approaches<sup>10</sup>. On the other hand, systems implementing these methods like simulations or microworlds often do not include any components for evaluation and feedback at all<sup>11</sup>.

Comparing learning theories in general<sup>12</sup>, their realization, and virtual, computer based

---

<sup>1</sup> [Holland and Skinner, 1961] p. V

<sup>2</sup> [Kay, 1972]

<sup>3</sup> [Ryan, 1991]

<sup>4</sup> [Virtuelle Hochschule Bayern, 2001]

<sup>5</sup> [Schulmeister, 2002] pp. 228

<sup>6</sup> [Bruns and Gajewski, 2002] pp. 39

<sup>7</sup> [Schulmeister, 2002] pp. 6

<sup>8</sup> [Schulmeister, 2007] pp. 104

<sup>9</sup> [Seidel and Lipsmeier, 1989] pp. 53

<sup>10</sup> [Schulmeister, 2002] p. 154

<sup>11</sup> [Schulmeister, 2002] p. 223

<sup>12</sup> [Schuman, 2007]

ones in particular<sup>1</sup>, one comes to the conclusion that approaches following constructivistic learning theories are best, but that their realization is just as hard. As a consequence, most of the realizations found so far are incomplete, non-working or otherwise insufficient<sup>2</sup>.

Two conclusions can be drawn from this: First, not all subjects can be taught by virtual teaching<sup>3</sup>, and second, realization of constructivistic approaches, especially ones which defeat the instructional components of the learning process, may not be the best. There is room for alternative learning-theoretical approaches<sup>4,5</sup>, which are described in the next section.

### 3.1.4 Alternative learning-theoretical approaches

There are ups and downs to the various learning theories and the instruction designs resulting from them, as discussed in the previous section. Recognizing this, a number of approaches have been suggested that take a pragmatic position between cognitivist and constructivistic approaches<sup>6</sup>. The instructional design of the 2nd generation combines elements of constructivism, like explorative learning and communication, with elements of cognitivism, which intends to add new components into the learner's existing knowledge structures, intending an integration of new contents<sup>7</sup>. The learner should not be made a reactive entity, but rather made to act proactively. Proactive learning concepts allow a change of the pedagogical situation towards choices for the learner, making room for own arrangements and self-organisation of the learning process<sup>8</sup>.

Merrill laid the fundamentals in instruction design with his "Second Generation Instructional Design" (ID<sub>2</sub>). ID<sub>2</sub> tries to overcome the limitations of what Merrill calls the "First Generation Instructional Design (ID<sub>1</sub>)" by integrating sets of knowledge and skills, producing pedagogical guidelines, selecting and sequencing instructional transaction sets, and esp. integrating phases of instructional design<sup>9</sup>. The following components are part of ID<sub>2</sub><sup>10</sup>:

1. A theoretical base that organizes knowledge about instructional design and defines methodology for performing instructional design.

---

<sup>1</sup> [Schulmeister, 2002] p. 223

<sup>2</sup> [Schulmeister, 2007] pp. 218

<sup>3</sup> [Schulmeister, 2002] p. 160

<sup>4</sup> [Schulmeister, 2007] p. 109

<sup>5</sup> [Merrill et al., 1991] pp. 3

<sup>6</sup> [Tulodziecki, 2000] pp. 59

<sup>7</sup> [Bruns and Gajewski, 2002] p. 17

<sup>8</sup> [Weidenmann, 1993] pp. 11

<sup>9</sup> [Merrill et al., 1991] p. 9

<sup>10</sup> [Merrill et al., 1991] p. 10

2. A knowledge base for domain knowledge, for making instructional decisions.
3. A series of intelligent computer-based design tools for knowledge analysis and acquisition, strategy analysis, transaction generation, and configuration.
4. A collection of mini-experts with small knowledge bases for one or more instructional design decisions each.
5. A library of instructional transactions, with interfaces to add new transactions.
6. An online intelligent advisor program that dynamically customizes the instruction during delivery, based on a mixed-initiative dialog with the student.

The interfaces mentioned in item 5 are important in learning systems that incorporate many sources of teaching materials, teachers and topics taught. For the present discussion, the topic of interfaces and formats of meta-data for easy exchange are beyond the scope; More information can be found in [Schulmeister, 2002, pp. 202, 207]. The advisor program and dynamic customisations of instruction will be addressed later when discussing tutoring systems, personalisation and user-adaptive systems.

Other approaches to address the named problems can be found in the concepts of situated cognition and situated learning.

Situated cognition assumes that thinking and learning are bound to a certain context in which knowledge is learned. This context is defined by the learning environment, which also names and defines the goals that should be learned, as learning is most efficient when the goals are known<sup>1</sup>.

Situated learning goes into more detail. It emphasizes the fact that an individual's learning performance is not only affected by the content presented and his internal learning processes, but also by the context in which the learning material is presented. Real world examples are considered important, so the acquired knowledge and problems solving methods can be applied<sup>2</sup>. At the same time, a variety of examples should be used to achieve decontextualisation. Following the concept of situated cognition, situated learning is employed in a learning environment which gives contextual information as well as instructions on goals to achieve. Situated learning provides methods to reach the given goals, and also emphasizes social interaction to facilitate elaboration and reflection<sup>3</sup>.

There are a number of approaches to realize situated learning. Choices in implementation include the degree to which virtualization should be employed (i.e. whether a "teacher" is present either in real or in the form of a computer program), if there is a guide, and how strong didactic embedding is. The approaches range from a teacher/student relationship in "Cognitive Apprenticeship" over learning in groups in

---

<sup>1</sup> [Schulmeister, 2007] p. 70

<sup>2</sup> [Lave and Wenger, 1991] pp. 32

<sup>3</sup> [Mandl et al., 1994] p. 170

“Knowledge Communities” to using cognition-promoting tools with the “Cognitive Tools” theory<sup>1</sup>. Most of these choices use multimedia technology to various degrees<sup>2</sup>.

The difference in situated learning to a pure constructivist approach is that individuals are both guided in what they should learn, as well as being provided with an environment that promotes solution of the given problems. The difference between situated learning and cognitivistic learning approach is that more emphasis is put on the learning context and elaboration in the former, not only on acknowledging the (internal) individual character of the learner, but also providing more (external) context for learning. As such, situated learning can be placed between cognitivistic and constructivistic teaching approaches.

This section covered various alternative approaches to learning theory and instruction design. Using a synthesis of the “classical” approaches and their derived forms, a set of powerful teaching tools is available, and it is possible to achieve teaching methods that are considered ideal in traditional education, as outlined in the next section.

### 3.1.5 Education – ideal progression and tools

A number of structures for instructional design of lectures have been proposed<sup>3</sup>. An ideal course of teaching is considered to consist of the following steps<sup>4</sup>:

1. A collection of assignments, collecting and discussing spontaneous ideas for solving.
2. Defining learning goals, and discussing their meaning.
3. Communication about proceeding towards these goals.
4. Acquiring fundamentals needed to solve the assignment.
5. Putting the assignment into effect.
6. Comparing various solutions, and summarizing what has been learned.
7. Introducing and working on domain specific assignments.
8. Discussing the knowledge learned, and the way it was learned.

So far, discussion has named a number of instruments to realize various learning theories and instructional designs. A number of these instruments can be used to shift the focus from the result of the learning process to the learning process itself<sup>5</sup>:

---

<sup>1</sup> [Schulmeister, 2007] p. 75

<sup>2</sup> [Mandl et al., 1994] pp. 171

<sup>3</sup> [Clark, 2000]

<sup>4</sup> [Tulodziecki, 2000] pp. 62

<sup>5</sup> [Schulmeister, 2007] pp. 73

- Empowering learning environments, to promote creativity.
- Games to increase motivation.
- Cognitive tools, to promote understanding and representation of cognitive processes.
- Tools to support writing and reasoning.
- Programs to support reflection of the mental processes of the learner.

The above lists are guidelines for realizing learning environments. This section has discussed the learning theories, instruction designs resulting from them, and circumstances in which to use one over another or a mixture of several approaches, to gain a maximum benefit from all approaches.

## 3.2 The “System Administration” class

Discussion was kept on a theoretical level in the previous section. This section looks at the existing class on “System Administration” (SA) as taught at the University of Applied Sciences Regensburg for several years now. That class is considered equal to classes on the same topic given at other universities, see section 2.2. The goal is to outline history and target audience of the existing class, describe the contents of the current curriculum, and discuss the didactic instruments used so far. More details on the existing class on system administration can be found in [Feyrer, 2007a].

### 3.2.1 History and target audience

The “System Administration” class is offered to students of computer science at the University of Applied Sciences Regensburg. It was started as an elective course for students in their advanced study period, i.e. the 7th or 8th semester, by Prof. Jürgen Sauer in 1994, and held until 1998. Since 1999, the course was given by Dipl.-Inf. Hubert Feyrer. Starting in 2003, the course was added as mandatory for all students. This discussion only covers the class in its mandatory form as it is given since 2003.

The target audience of the “System Administration” course are students of general computer science (“Allgemeine Informatik”) in the advanced study period, usually in their 5th semester. Volunteer students from technical computer science (“Technische Informatik”) or commercial information technology (“Wirtschaftsinformatik”) are allowed to participate and take the course as elective course.

### 3.2.2 Current curriculum

The course consists of two lectures and one lab exercise per week, with the lectures and the lab exercises being 90 minute each. For lab exercises, the students are split into two groups due to lack of sufficient working places. On average, a course consists of 40 students, resulting in two groups of 20 students.

Given other focuses in the curriculum, students usually have little Unix knowledge. Some understanding of the basic operating system and networking principles are available from corresponding courses, but experience in using the Unix operating system, its commands, as well as concepts for automating tasks can not be expected from all students. As such, a part of the lecture introduces basic commands and concepts of the Unix operating system, focusing on the latter under the light of system administrative tasks.

The following topics are covered in the lecture and accompanying lab exercises<sup>1</sup>:

**0. Introduction:** The introduction of the class gives a small historical overview of the past and a description of the class's overall goals<sup>2</sup>.

**1. Historical Overview:** This section illustrates the history of Unix, starting with AT&T and going to BSD and the various systems derived from it.

Exercises compare various Unix systems using the "Rosetta Stone for Unix"<sup>3</sup>, and look at descriptions of commands in standards like POSIX and the Single Unix Specification<sup>4,5</sup>.

**2. Login process, process correlation:** As an introduction, the classical login process is discussed, including processes involved. Further concepts like signals, job control, and general handling of documentation under Unix is discussed<sup>6</sup>.

**3. User commands (standalone and for shell programming):** Assuming that only few students have a sound Unix background, basic Unix commands are discussed which are useful both when used alone as well as when used in shell programming. Areas covered include managing files and directories, permissions and access control in a multiuser environment, text processing, and using regular expressions<sup>7</sup>.

**4. Information about the system:** To properly administrate and tune a system, it is essential to know as much data about the system's state as possible. This section gives related commands, output usually found and how to interpret it. The

---

<sup>1</sup> [Feyrer, 2007e]

<sup>2</sup> [Feyrer, 2007e] "Vorwort"

<sup>3</sup> [Hamilton, 2007]

<sup>4</sup> [The Open Group, 2004]

<sup>5</sup> [Feyrer, 2007e] "Historischer Überblick"

<sup>6</sup> [Feyrer, 2007e] "Login Prozeß, Prozeßzusammenhänge"

<sup>7</sup> [Feyrer, 2007e] "Hilfsprogramme (Standalone und für Shell-Programmierung)"



areas covered are processes, signals, users, installed software, operating system version, kernel, terminals, remote machines, swap-space, process accounting, filesystems, disk quotas, device-handling and harddisks<sup>1</sup>.

**5. Shell programming:** Assuming a basic understanding of a Unix system, this chapter introduces shell programming using the Bourne shell (`/bin/sh`) to automate recurring tasks. Topics include redirection of input and output, expansion of wildcards, shell and environment variable, quotes, control structures, and shell functions<sup>2</sup>.

**6. Application of shell scripts: booting and shutdown:** Students have been introduced to all the features that are available in shell programming. This section shows an application of shell programming by observing the system’s startup mechanism, which is usually realized as a set of shell scripts. The approach of letting students read existing code written by experts, instead of writing their own, is intended to show solutions for common problems and also practice reading and understanding the flow of code and data. This section introduces general booting of systems and outlines the System V “init”-system. Attributes of the init-system discussed include runlevels, concept and functionality of start- and stop-scripts, and their layout in the filesystem. After the System V “init” system, alternative approaches for disabling/enabling of services and determining the order in which to start services are discussed.

Exercises for this section are mostly of analytical nature, as changing the system’s boot system to gain experience would require system privileges. Those cannot be handed out for practical reasons described in section 3.3. As a result, exercises include analyzing the existing startup systems found on Solaris, SuSE Linux and NetBSD<sup>3</sup>.

**7. Networking:** Students in the 5th semester visit the “Data Communications” lecture in parallel with the system administration lecture, so a basic understanding of networking and TCP/IP basics can be assumed, and concepts of addressing, routing and name services are only repeated briefly. Building upon these, the network model of Unix is explained, again covering various implementations with an emphasis on Solaris, but also Linux and NetBSD. With the network model understood, the next steps covered are how to configure the system and name resolving for basic TCP/IP networking. Following this introduction, three topics are picked up that are considered important when managing clusters of workstations: setup of public key authentication in the Secure Shell (ssh), the Network File System (NFS) and Network Information System (NIS) clients and servers.

For practical exercises, the same situation as described above in “Booting and shutdown” applies: for maximum learning effect, system privileges would be

<sup>1</sup> [Feyrer, 2007e] “Informationen über das System”

<sup>2</sup> [Feyrer, 2007e] “Shellprogrammierung”

<sup>3</sup> [Feyrer, 2007e] “Anwendung von Shellsripten: Hoch- und Runterfahren des Systems”

needed, but they cannot be handed out to students for the named reasons. As such, exercises mostly consist of the analysis of existing systems<sup>1</sup>.

**8. The X Window System:** An application of networking is the X Window System, which is the graphical subsystem used on Unix. The interesting fact here is that the X Window System itself is network transparent, i.e. an application can run on one machine and display its graphical output on another machine. The lecture starts with some basic concepts like the client/server architecture, addressing displays, and simple access control. Redirection of applications across the network is covered next, using the mechanisms provided by the X Window System and the secure shell, followed by the startup process of the X Window System with processes and files involved. Last, the functionality of a “Window Manager” is explained in the context of a demonstration the KDE desktop environment.

Lab exercises for the X Window System build a graphical environment step by step from single components, starting with window managers, placing client windows next, followed by tuning application look and feel<sup>2</sup>.

**9. Security:** Security is considered important today, mostly requiring system administrators to take appropriate measures to establish secure systems. The curriculum of computer science includes special lectures on security to give basic understanding and methods. The system administration lecture approaches “security” from the practical side by discussing what kinds of problems may exist, including host and network security, showing that a number of problems have equal origins. The lecture closes by pointing at various sources of information, from full disclosure over general security lists to vendor provided information to assist in securing systems.

Student exercises start by a briefing on the legal situation of computer security, and that any security holes found should be reported to the system administrator immediately. The lab systems should then be analyzed and monitored for the various classes of security problems discussed, followed by finding special system services that may get exploited<sup>3</sup>.

**10. Practical Extraction and Report Language - Perl:** A language found often in system administration environments is Perl. The introduction given to Perl covers the difference from other programming languages in data types, input/output and control structures. Features presented include processing of regular expressions, arrays, lists, stacks, and hash tables. The Perl programming language’s built in functions, creating one’s own functions, using existing modules, and an overview of all existing modules round up the introduction to Perl.

Exercises for Perl include programming tasks that handle lists and associative arrays, analysis of web server logfiles and scanning and sorting of mailboxes<sup>4</sup>.

---

<sup>1</sup> [Feyrer, 2007e] “Networking”

<sup>2</sup> [Feyrer, 2007e] “Das X Window System”

<sup>3</sup> [Feyrer, 2007e] “Security”

<sup>4</sup> [Feyrer, 2007e] “Practical Extraction and Report Language - Perl”

**11. User management:** This section approaches user management by repeating the related concepts in Unix, including user databases, password encryption, home-directories, dot-files, quotas, and site-specific setup steps. The tools used for user management at the University of Applied Sciences Regensburg’s computer science faculty are then introduced to show an approach of large scale user handling – the computer science department has an average of 1.000 students which have access to various Unix machines. Besides showing students how to realize user management in Perl, it gives students a chance to learn from existing Perl code.

Unfortunately, exercises are restricted again, as the students can not work with system administrator privileges. As such, the exercises consist of examining various operating systems’ tools via their documentation and to the extent possible with normal user privileges<sup>1</sup>.

**12. Software management:** After describing system operations and user management, handling application software is the third big topic covered in the “System Administration” lecture. This section introduces the software architecture found on operating systems including separation into “operating system” and “applications.” The historical development that led to the various models and components is explained, followed by handling of precompiled binary software. Software management tools covered include those of Solaris, Linux Systems that use the RedHat Package Management (RPM) system, and NetBSD.

Exercises for software management involve software installation as a “normal” user. After getting familiar with various package systems, the meta data used by these systems is investigated, and dependencies between software packages are analyzed<sup>2</sup>.

**13. Backups:** The last section of the lecture covers backing up data. Topics discussed include media, various concepts of backups from file based to whole filesystems, and data compression. An overview of integrated solutions including commercial backup systems closes the chapter.

Lacking not only access to systems on a filesystem/harddisk base, but also backup hardware and enterprise solutions for performing backups, practical exercises are kept on the base of backing up single files and directories<sup>3</sup>.

This section has described the various topics covered in the “System Administration” lecture with a focus on the contents taught in class and the lab exercises students are expected to do. The next section will give more information on the overall layout of the course and the reasons behind it.

<sup>1</sup> [Feyrer, 2007e] “Benutzerverwaltung”

<sup>2</sup> [Feyrer, 2007e] “Software-Management”

<sup>3</sup> [Feyrer, 2007e] “Datensicherung”

### 3.2.3 Course layout

After the previous section discussed the contents of the single lectures in detail, this section illustrates the overall building blocks of the “System Administration” class, and how they are arranged to reach the goal of the lecture. Furthermore, it covers how the building blocks reflect the change demanded in learning strategies, ranging from behavioristic learning for the fundamentals to cognitivistic learning for the more advanced topics.

The class has two goals:

1. General understanding of Unix, and that there is not “the one Unix system”, but several implementations that differ in various details – some minor, some major.
2. System management of large scale clusters, with system, user and software management as well as procedures to setup the necessary network infrastructure

The first goal aims at giving students a general understanding of the Unix operating systems, assuming they are not familiar with the concepts to use and/or administrate such a system. Throughout the lecture, exercises are given to show differences in platforms using the hardware and operating systems available to students at the computer science and computing center’s department of the University of Applied Sciences Regensburg. The main system that the lecture is based on is Sun Microsystems’ “Solaris” operating system as incarnation of a System V system, other systems discussed throughout the class and exercises are “SuSE Linux” as representatin of the Linux family of operating systems, and “NetBSD” for the BSDs.

The second goal gives a direction for the contents of the lecture. “System Administration” itself is a wide field, and the goal of administrating a cluster of workstations is considered worthwhile. The various steps needed for this goal are difficult to learn e.g. in a self-teaching home-environment, while other topics like setup of mail, DHCP, DNS, Web and Samba servers may be easy to learn and practice.

For the further discussion, here is a list of the topics covered during the lecture, presented in detail in section 3.2.2. Short names are given for further reference:

Section	Title	Short name
0.	Introduction	-
1.	Historical Overview	-
2.	Login process, process correlation	-
3.	User commands (standalone and for shell programming)	UserCmds
4.	Information about the system	SysInfo
5.	Shell programming	ShellProg
6.	Application of shell scripts: booting and shutdown	Booting
7.	Networking	Network
8.	The X Window System	X
9.	Security	Security
10.	Practical Extraction and Report Language	Perl
11.	User management	UserMgmt
12.	Software management	SWMgmt
13.	Backups	Backup

When analyzing the structure of these topics, some act as fundamentals to others. Figure 3.3 displays “Cluster Management” as the main topic of the class, and illustrates the relations between the various topics discussed.

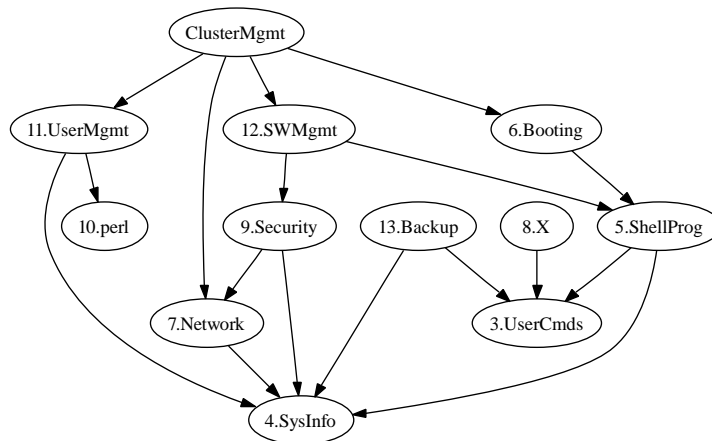


Figure 3.3: Structure of the “System Administration” lecture

The topics shown in figure 3.3 can be divided into three groups as shown in figure 3.4:

- User Management
- System Operations
- System Startup

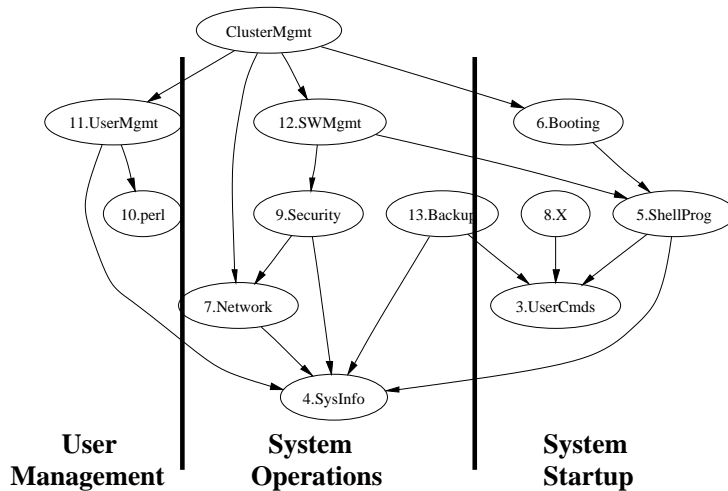


Figure 3.4: Thematic groups in the “System Administration” lecture

The “User Management” group on the left of figure 3.4 assumes an understanding of the Perl programming language, and it also requires understanding of commands from “System Information, for the areas of user databases, and how to handle them. The “Systems Startup” group on the right asks for an understanding of shell programming, which in turn uses a variety of user and system specific commands to determine information like which services to start. Finally, the middle group of “System Operations” is a loosely coupled collection of topic that cover software management, security and networking, which again build up on the information derived from the system as well as various user commands.

Examining the discussion of the groups, it becomes obvious that each group can be divided into various levels according to the difficulty or how advanced the topic is, i.e.

- Basic
- Advanced
- High-level

Figure 3.5 illustrates this separation. The basics upon which all other topics rely are user commands, commands to determine information about the system, understanding of networking concepts, and related configuration. The Perl programming language listed as “advanced” here could be in the “basic” category here too. Advanced topic are backups, the X window system, shell programming and security. Using all these basic and advanced topics, the high-level goals of user and software management as

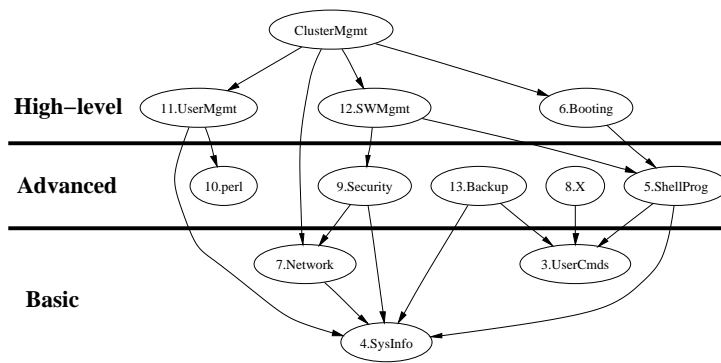


Figure 3.5: Levels of difficulty in the “System Administration” lecture

well as booting of the system (which is important for system configuration esp. in large scale environments) can be realized. In turn, these are the foundation needed for management of large clusters of Unix workstations and servers, which is the ultimate goal of the “System Administration” lecture.

As not all of the fundamentals of such a complex topic can be explained at once, the lectures have been split both horizontally into various levels of fundamentals building one upon another (see figure 3.5), and vertically to separate topical groups that can be separated to build logical units (see figure 3.4). The result is a collection of single topics, as presented in section 3.2.2.

### 3.2.4 Didactic instruments

Besides the learning goals presented in the previous section, there are a number of didactic instruments that are used in various parts of the lecture and in lab exercises that are presented in this section, grouped by what part of the course they are used in.

**Lecture:** The “System Administration” class consists of two lectures and one lab exercise per week. Each lecture takes 90 minutes, the lab exercise takes 90 minutes, too. During lectures, a laptop and a video projector are used to present slides. Occasionally, examples are developed on the blackboard to illustrate examples that are not immediately clear from the slides.

The teacher’s notebook is used to display examples for commands, procedures and to show example outputs of various systems. Either the local laptop is used, or a remote system is accessed via the ethernet connection available to the teacher in each classroom.

A number of books covering various topics from the lecture are passed around

in the first lecture<sup>1</sup>, but students are not required to read all or parts of them - all the lecture material is present in the lecture slides and in online lecture notes.

The history of Unix is illustrated by a printout of a graph displaying all historical Unix releases so far, printed on many sheets of paper and glued together<sup>2</sup>

**Online lecture notes:** The online lecture notes are identical to the slide presented during the lecture. Each student has online access to the lecture notes so he can print them in advance, bring them to the lecture and make personal notes if needed. If a situation is found to be described suboptimally during class, the notes are updated after the lecture to clarify the situation

Lecture notes are available at <http://www.feyrer.de/SA/> as a set of HTML files<sup>3</sup>.

**Examples:** The lecture notes include examples for many situations that may arise in the topics described. Examples include a description of the situation, the exact command name to input and example output. That way, students are not required to sit in front of a computer to learn what a command does, but can do so from the online lecture notes' examples only. Examples are designed to be as complete as live demonstrations for the purpose of learning with no machines at hand, which is esp. important as the lecture tries to give examples of many different machines and operating systems, many of which are not widely available. Figure 3.6 shows an example found in the online lecture script.

**Live Demonstrations:** System administrative procedures consisting of a number of steps are demonstrated live. For them, a detailed description of the context in which the demonstration happens is given, including machine hardware and operating system, goal of the demonstration, and an outline of the conceptual steps. This is followed by a description of commands and tools used. The next step is an interpretation of the output and other effects resulting from the demonstration steps, as well as an analysis and description of the system after the demonstration, with retrospect on how each step affected the system.

**Analysis of existing systems:** Strong emphasis on the multi-platform property of Unix is given in the entire "System Administration" course. This is supported by many examples and exercises that are intended to be ran on multiple different systems, to learn the properties of single systems as well as differences. That way, students can infer concepts commonly found on many Unix systems as well as others that are only found on single systems, e.g. as discussed during the "Networking" or "System startup" sections.

For that purpose, a number of machines are available: Solaris/x86, Linux/i386 and NetBSD/i386. Possible systems for future demonstrations may include SGI machines running Irix, IBM machines running AIX, and Sun machines running

---

<sup>1</sup> [Feyrer, 2007e] "Literatur"

<sup>2</sup> [Lévénez, 2007] .

<sup>3</sup> [Feyrer, 2007e]



```

rfhinf032% cat /proc/stat
cpu 11284627 358168 713256 46962094
cpu0 11284627 358168 713256 46962094
page 1270776 2846120
swap 78 416
...

• iostat:
rfhpc6317% iostat 1
          tty          cmdk0
tin tout kps tps serv kps tps serv kps tps serv kps tps serv us sy wt id
0 302 1 0 6 0 1 0 0 0 0 0 0 0 1347 0 0 12 87
0 234 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 13 86
0 80 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 13 87
...

• vmstat:
rfhpc6317% vmstat 1
procs memory
r b w swap free re mf pi po fr de sr cd s0 s1 -- in sy cs us sy id
0 0 0 692472 202748 0 3 0 0 0 0 0 0 0 1 0 0 164 407 271 0 0 99
0 0 0 679504 189588 0 6 0 0 0 0 0 0 1 0 0 158 199 251 0 1 99
0 0 0 679504 189588 0 0 0 0 0 0 0 0 1 0 0 162 141 242 0 0 100
...

• psrinfo (Solaris):
e3500# psrinfo
cpu0: SUNW,UltraSPARC-II (upaid 6 impl 0x11 ver 0x20 clock 336 MHz)
cpu1: SUNW,UltraSPARC-II (upaid 7 impl 0x11 ver 0x20 clock 336 MHz)
cpu2: SUNW,UltraSPARC-II (upaid 10 impl 0x11 ver 0x20 clock 336 MHz)
cpu3: SUNW,UltraSPARC-II (upaid 11 impl 0x11 ver 0x20 clock 336 MHz)
cpu4: SUNW,UltraSPARC-II (upaid 14 impl 0x11 ver 0x20 clock 336 MHz)
cpu5: SUNW,UltraSPARC-II (upaid 15 impl 0x11 ver 0x20 clock 336 MHz)

```

Figure 3.6: Examples help learning without a computer

Solaris/sparc. Students have logins on all these systems thanks to the NIS/NFS infrastructure of the computer science department.

**Lab exercises for hands-on learning:** Each section of the online lecture notes contains suggested exercises that students are expected to work on during lab time. The lab for these exercises consisted of 15 PCs running a dual boot of Solaris/x86 and Windows 2000 (of which the latter is little to not used for the exercises), and four PCs running NetBSD. During lab exercises, machines are reserved for the students to do the exercises. No solutions to the exercises are published, to motivate students to come up with their own solutions. Personal experiences show that when handing out solutions for the mixture of exercises, students prefer to just read the solution (or learn it by heart), and not go through all the steps to learn the aspects of the topics discussed.

**Attended tutorials:** During lab exercises, a tutor is present for answering questions about the working environment, machines, operating systems, their configuration, the exercise in question, and any related questions. Questions can be answered during lab exercise time; As students are free to do the exercises outside the lab exercise time, they can contact the teacher via email (and in some rare occasions via IRC chat) to ask questions. Students are encouraged to use the lab exercise time for asking questions, though.

The instruments described here are currently in use in the “System Administration” class given at the University of Applied Sciences Regensburg. The instruments have

been the same for the past few years that the lecture is given, with only some minor variety on machines and operating systems that students have access to. A number of alternative instruments could be used in theory, some of which are discussed in the next sections.

### 3.3 Analysis of the current situation

This section analyzes the “System Administration” lecture discussed in section 3.2 under the light of the learning theories and didactic fundamentals given in section 3.1. Taking the suggestions for ideal progression and tools for a lecture given in section 3.1.5 into account, a list of measures can be identified to improve student orientation and learning performance of students over the current form of the lecture:

- Define goals at the start of the semester.  
Besides setting general, system/distribution-independent understanding of Unix as a goal, cluster management should be mentioned explicitly.
- Give a better overview of the way how the given goals are reached.  
This needs change in two places: First, give an overview over all chapters at the start of the semester, similar to the overview given in section 3.2.3. Second, at the start of each chapter, outline the contents that will be presented.

These changes can be accomplished easily in future incarnations of the “System Administration” lecture.

Another problem is more difficult to solve, though: In advanced topics, practical exercises are indispensable, which can be seen from the description of the current lecture in section 3.2.2 and the “wishlist” of alternative instruments given in section 3.4. Merrill supports this by stating that “much new scientific knowledge is dynamic in character and cannot be understood without a more active representation and student involvement.”<sup>1</sup> The approaches of situated learning and related cognitive concepts introduced in section 3.1.4 support this, and Hubwieser also asks for modelling and simulation to be part of the educational principle, and not part of the lecture contents<sup>2</sup>.

The course starts out with basic topics that can be easily learned without practical exercises, by merely looking at examples and descriptions. However, more advanced sections need practical exercises for understanding. The gradual move from behavioristic learning theories for basic topics to cognitivist learning theories for advanced topics also shows that these two forms are not the only ones needed to fulfill all the needed requirements, and that mixed forms like illustrated in section 3.1.4 are needed.

---

<sup>1</sup> [Merrill et al., 1991] p. 7

<sup>2</sup> [Hubwieser, 2000] p. 69

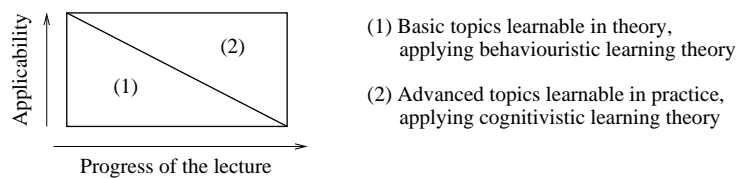


Figure 3.7: Change in learning paradigm with advancing level

Figure 3.7 illustrates the correlation between basic and advanced topics: Basics introduced at the start (left) of the lecture can be learned without practical exercises, applying behavioristic learning theory only. But the more advanced the lecture gets, both in time and in level of topics discussed, the more practical exercises are needed for understanding and learning, applying cognitivistic learning theories.

Basic topics can be learned through simple behavioristic learning methods like drill-and-practice in theoretical manners. But with increasing level of difficulty, practical exercises following cognitivistic or constructivistic models are required. For the “big” topics that build the goals of the “System Administration” lecture, it is not sufficient to cover them on a theoretical level. Instead, practical exercises with full system privileges are mandatory for throughout understanding.

An operational problem regarding practical exercises with administrative privileges to access the system configuration level is present in the lab setup used at the University of Applied Sciences of Regensburg: If a student used administrative privileges during an exercise, the machine’s state is not known after the exercise. In order to assure proper operation of the lab machines for future exercises, the machine would have to be re-installed. Re-installing the systems is not an option, unfortunately, due to time constraints and the lack of human resources.

In summary, there is a need for practical exercises applying cognitivistic learning methods in the “System Administration” lecture, esp. with system privileges. Currently, a lack of manpower and resources to setup and re-install machines prevent this. A possible solution would be to build a virtual environment that allows practising the real goals of the course where only theoretical coverage of these topics is possible so far. This approach is also suggested by [Adams and Laverell, 2005].

The implementation of this “Virtual Unix Lab is more demanding, and will be covered in the remaining chapters of this work.

### 3.4 Future directions

There are a number of didactic instruments that are not currently used in the existing course, but that may be useful for future improvements. None of them breaks any new grounds in education of computer science or system administration, and they may be found in other courses on system administration and related topics. Still, they are considered worthwhile:

- Solutions to (selected) exercises could be handed out, including a description of the solution. Providing well-written examples esp. for the programming parts like Bourne and Perl programming may give students a better idea of how to use certain constructs.
- Access to more (different) machines with more operating systems would be helpful for better understanding of system attributes. Useful machines to name are SGI machines running Irix, Sun SPARC- and AMD-based machines running Solaris as well as a set of PC machines running different Linux distributions like RedHat, Gentoo, Debian and Slackware (besides the SuSE already available). This would allow to analyze setups even without root privileges. For example, the NIS and NFS client setup could be derived from such machines.

The problems involved here are the cost of hardware and operating systems on one side, and maintenance of machines on the other side, which the computer science department cannot provide currently.

- At some points, contents could concentrate on using available GUI tools like SuSE's "yast", Solaris' "admintool", or the Solaris Management Console, instead of configuration files and command line tools. These tools could be used if it was ensured that students understood the underlying concepts properly, e.g. they would be more appropriate in an "Advanced System Administration" class instead of a class teaching basics, like the current "System Administration" lecture.
- Section 3.2.2 pointed out that many topics cannot be practiced properly, due to the lack of machines which can be accessed with system administrator privileges. While a number of workstations are available for all students – 15 running SuSE Linux, 15 running Solaris/x86, 4 running NetBSD – none of them are available for practicing system administrative tasks, as all of them are public machines that other students need to use too. Handing out system administrator privileges on these machines would require re-installation of the machine after the exercise, as the system's state would not be known, and could not be trusted for public services.

Examples of exercises where exclusive access to hardware would be useful include:

- Setup of various operating systems, and related initial configuration to get the systems to a predefined state
- Setup of various client/server scenarios, e.g. mail, POP, IMAP, spam filtering, DNS, DHCP, FTP, SSH, Samba, NFS, NIS and many others<sup>1</sup>.
- Troubleshooting scenarios where systems are setup to misbehave in one way or another. Students would be expected to identify and solve the problems.

Due to the lack of manpower, this re-installation cannot be performed after each lecture, and as there are no machines dedicated for system administration training, no practical exercises are currently offered for many areas that need these privileges.

- Besides machines and operating systems, access to other hardware components would be useful for practicing some of the basic setup and operations principles. Those components could include network components like cabling, hubs and switches as well as hardware for backup, such as tape drives of various technologies like AIT or DLT, and maybe some external disks and RAID arrays. Again, this is not possible or available at this time due to financial constraints.
- For describing certain setup or troubleshooting situations, it would be useful to have machines available in exactly such a situation as described, which students then could pick up for further practicing, realizing an “Anchored Instruction” approach<sup>2</sup>.

Besides the lack of hardware resources, such a setup would need a lot of preparation to define the systems to be in a specific state, and even more effort to backup and restore exactly that situation for later replay by all students. While this would be very useful for troubleshooting setups, it is again not possible due to lack of manpower, machines and money.

- Right now, the whole “System Administration” class is centered around classroom teaching where students are expected to be present. While students are free to take the lab exercises outside of the lab hours, it is recommended to take them when the teacher is in to get optimal feedback on questions and problems.

Moving the lecture into a “virtual” environment, where students decide on the process themselves would be possible due to the availability of the online lecture notes. Moving the lab exercises into a completely virtual environment with no teacher present would be more demanding. Interactive, tutoring and adaptive components would be needed to help students in situations where teachers can look over their shoulder today, and react to the situations they see<sup>3</sup>.

---

<sup>1</sup> [Ernst, 2004]

<sup>2</sup> [Mandl et al., 1994] p. 171 and 173

<sup>3</sup> [Bruns and Gajewski, 2002] pp. 22

This list of further instruments that could be used in the “System Administration” lecture is by no means complete, but it illustrates a number of approaches that could be used to improve teaching.

## **Part II**

### **Diagnosis and feedback with a domain specific language**





# Chapter 4

## Basic design of the Virtual Unix Lab

This chapter gives an overview of the Virtual Unix Lab that was developed during the “Praktikum Unix-Cluster-Setup” project from a user perspective. The system presented here is used as a foundation for the following works. Key design components, the hardware and network setup are introduced briefly here. See [Feyrer, 2004c] and [Zimmermann, 2003] for a more detailed description.

### 4.1 A user-level walkthrough of the Virtual Unix Lab

The Virtual Unix Lab basically has two user modes, one for regular users (students), and one for administrators (teachers). A brief overview of the student’s perspective is given in this section to get an overview of the system. The administrative view is discussed in detail in [Feyrer, 2004a] and in the following chapters.

This tour through the user area of the Virtual Unix Lab covers login and account creation, booking an exercise, taking an exercise and retrieving feedback afterwards. The walkthrough consists of a number of screenshots displaying the web based user interface that the Virtual Unix Lab presents in the order that a student using the system would see:

1. Access to the user interface of the Virtual Unix Lab is through a web browser, which allows accessing all facilities provided, except performing exercises themselves (see below). Language of the user interface is German (only) right now – internationalisation is on the list of items to do in the future.

When accessing the webpage, the first thing students encounter is a mask to login as displayed in figure 4.1.

2. If a student does not have a login yet, he can create a new login (“Profil”) using the form displayed in figure 4.2.

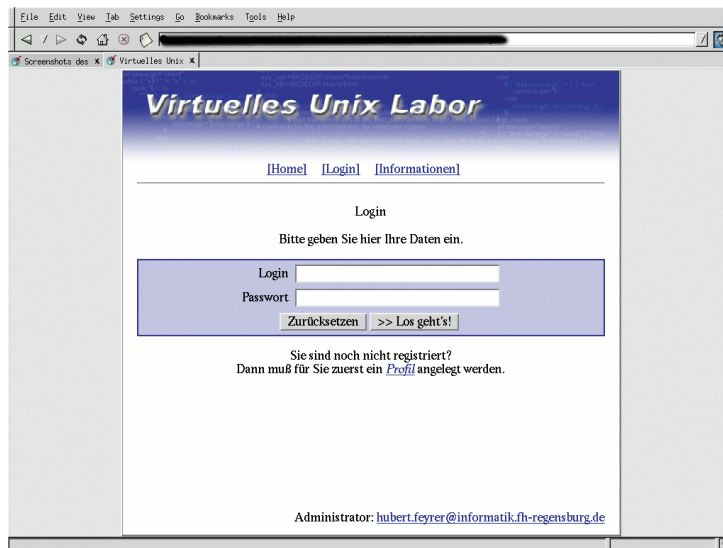


Figure 4.1: Logging into the Virtual Unix Lab

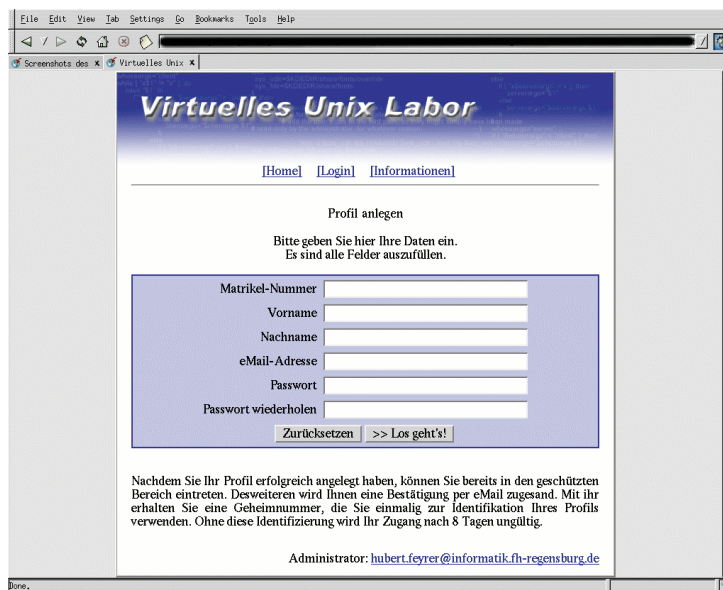


Figure 4.2: Entering data for a new login

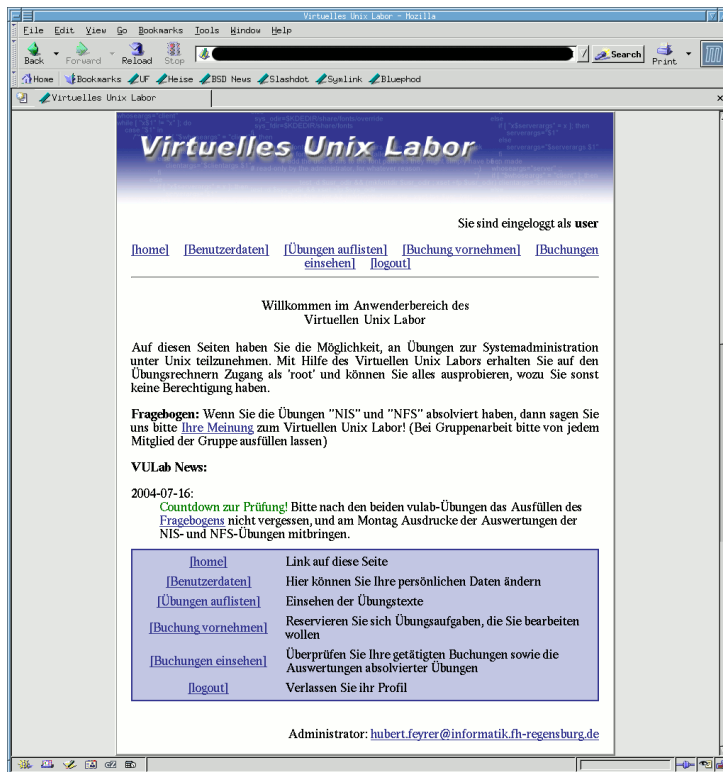


Figure 4.3: Welcome to the Virtual Unix Lab

The student will have to give his student ID number (“Matrikel-Nummer”), first and last name, an email address where he can be reached and a password (twice). Upon registration, an email will be sent to the given email address. The email contains an authentication token that the user has to enter to permanently enable his account. Accounts not enabled that way will be deleted after 7 days. This allows instant access to the lab, but ensures that people provide at least a valid email address if they want to keep using the lab.

- After successful login into the Virtual Unix Lab, the welcome screen shown in figure 4.3 is displayed, and users can choose from several actions they want to do: Update their user settings (“Benutzerdaten”), get a list of available exercises (“Übungen auflisten”), book an exercise for a certain time and date (“Buchung vornehmen”), get a list of past and future exercises, delete future exercises and retrieve feedback on past ones (“Buchungen einsehen”) as well as logout of the web site.
- Next, an exercise can be booked. This is done by selecting the “Übung buchen”

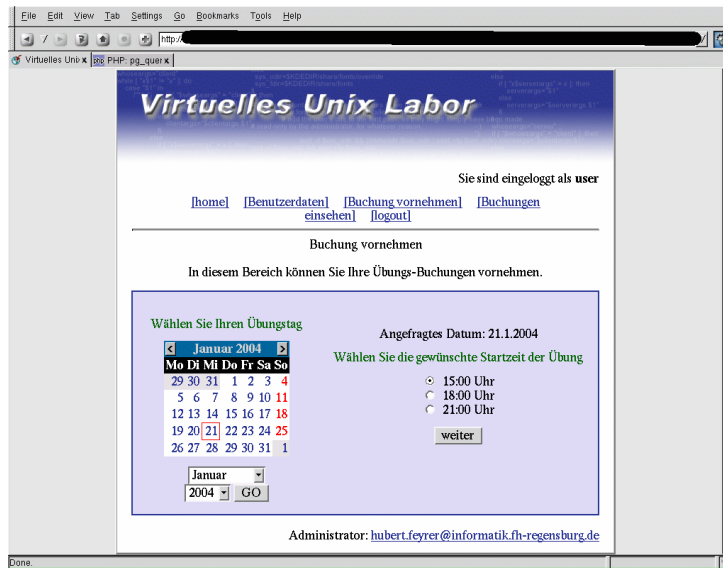


Figure 4.4: Booking an exercise: selecting date and time

menu item. The first step in booking an exercise consists of deciding at which date and time to take the exercise, which is displayed in figure 4.4.

Exercises are available in three-hour intervals (1.5 hours for the exercise, plus about one hour for preparation of the lab machines and some time for postprocessing). Slots already booked by other users are not displayed. In the screenshot, some exercises are not available because of this.

5. After deciding on the date and time for the exercise, the next step is to choose which actual exercise to take. The exercise name, description and duration are displayed, and the user has to decide for one as displayed in figure 4.5.
6. After selecting date, time, and which course to take, a final confirmation shown in figure 4.6 has to be made before the exercise is booked.
7. The exercise is booked at this point, and the system will know when to prepare the lab machines for the exercises by using an `at(1)` job.

The student can walk away and prepare for the exercise. Like for a school test, he should come back to the lab a few minutes before the selected time of the exercises and log in again as shown in figure 4.1.

8. After the user has logged in again, the system will tell him that an exercise was prepared, and that he can already start to prepare the exercise by following the provided link (“bitte *hier* klicken” in red text) as displayed in figure 4.7.

The screenshot shows a web browser window displaying the 'Virtuelles Unix Labor' interface. The page title is 'Virtuelles Unix Labor'. The user is logged in as 'user'. The interface includes a navigation menu with links for 'home', 'Benutzerdaten', 'Buchung vornehmen', 'Buchungen einsehen', and 'logout'. The main content area is titled 'Buchung vornehmen' and contains the text: 'In diesem Bereich können Sie Ihre Übungs-Buchungen vornehmen. Wählen Sie die gewünschte Übung für 21.1.2004 15:00 Uhr'. Below this is a search bar labeled 'Stichwort-Übungs-Suche:' with a 'Suchen' button. The search results show 'Vorhandene Übungen: 7' and a list of exercises. The exercises are listed in a table with columns for 'Kurzbezeichnung', 'Bezeichnung', 'Dauer', and 'wiederholbar'. The exercises are: 'apache' (Aufsetzen eines Apache Servers, 01:00, ja), 'netbsd' (NetBSD konfigurieren, 01:30, ja), 'nfs' (Aufsetzen von NFS Client und Server, 01:30, ja), 'nis' (Aufsetzen von NIS Client und Server, 01:30, ja), 'pruefung' (Verwalten von Benutzern mit Hilfe von NIS, 01:00, nein), 'pruefung2' (Einrichten eines Apache Servers mit SSL, 01:00, nein), and 'solaris' (Solaris konfigurieren, 01:30, ja). The page also includes 'zurück' and 'weiter' buttons and the administrator's email address: 'Administrator: hubert.feyrer@informatik.fh-regensburg.de'.

Kurzbezeichnung	Bezeichnung	Dauer	wiederholbar
<input type="radio"/> apache	Aufsetzen eines Apache Servers	01:00	ja
<input type="radio"/> netbsd	NetBSD konfigurieren	01:30	ja
<input type="radio"/> nfs	Aufsetzen von NFS Client und Server	01:30	ja
<input type="radio"/> nis	Aufsetzen von NIS Client und Server	01:30	ja
<input type="radio"/> pruefung	Verwalten von Benutzern mit Hilfe von NIS	01:00	nein
<input type="radio"/> pruefung2	Einrichten eines Apache Servers mit SSL	01:00	nein
<input type="radio"/> solaris	Solaris konfigurieren	01:30	ja

Figure 4.5: Booking an exercise: selecting the exercise

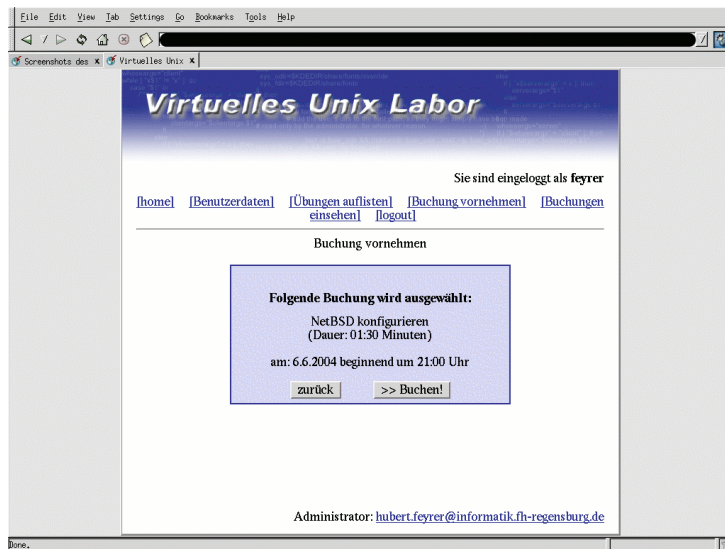


Figure 4.6: Booking an exercise: confirmation

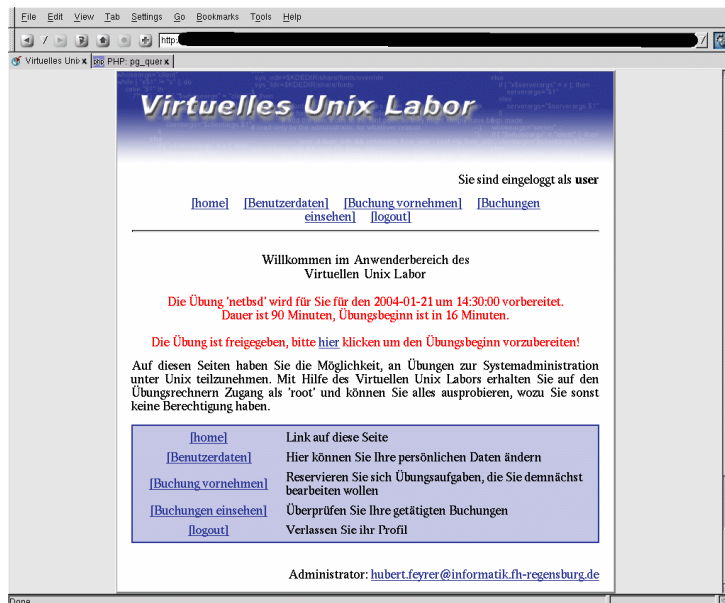


Figure 4.7: An exercise is prepared and waiting

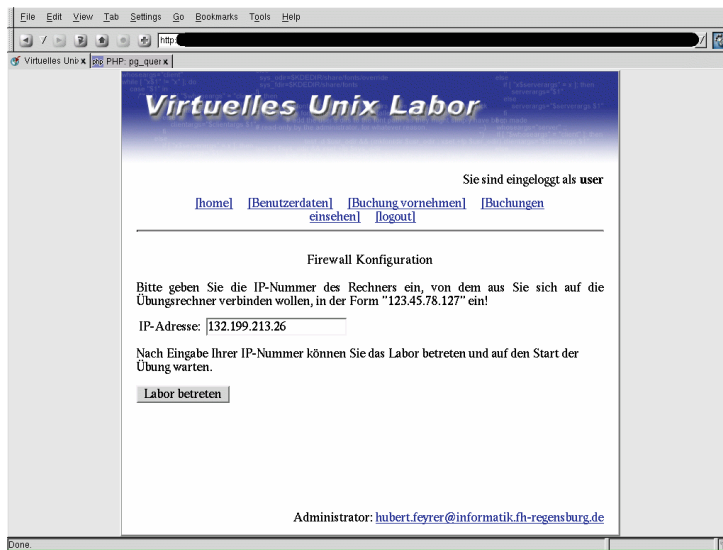


Figure 4.8: Configuring access to the lab machines

9. Before starting the exercise, the student has to enter the IP address of the machine from which he wants to access the lab machines. This process is shown in figure 4.8. The IP address will be used to configure the firewall when the exercise actually starts, to restrict access so other students cannot disturb the exercise.
10. Just as in a real test, the student can enter the lab website and sit down, but the test will not start until the specified time. In a real lab test, this would be when the teacher hands out the questions. In the Virtual Unix Lab, the student has to wait for the start of the exercise too, as displayed in figure 4.9.
11. When exercise time is reached, the firewall protecting the lab systems will be opened to allow (only) the student to access the lab systems, and the exercise text will be displayed as shown in figure 4.10.

The text is the same as the one provided for looking at before the exercise, so students can prepare properly. There are few additions to the text, though. First, a link with help for accessing the lab systems is placed under the exercise text, so students not yet familiar with the lab can learn how to access the lab machines, giving proper syntax for telnet, ftp and ssh. Below this link, the time remaining for the exercise is printed on the lower left (“Verbleibende Zeit”). If the user decides to finish the exercise before the time runs out, he can press the “Fertig!” (done) button.

12. Separate terminal windows have to be opened to access the lab machines and

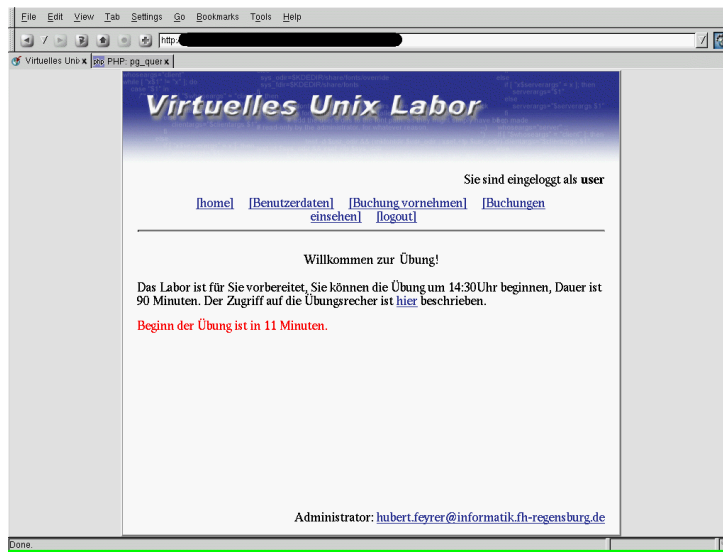


Figure 4.9: Waiting for start of exercise time

perform the tasks requested in the exercise text. Figure 4.11 shows access to a Solaris/sparc (left terminal window) and NetBSD/sparc (right terminal window) system.

Each lab system offers a “normal” user account as well as one with system administrator (root) privileges. The corresponding passwords are given in the instructions on how to access the lab machines.

The student can solve the given task by any measures he finds appropriate, using the full administrative privileges he has available. If one of the lab machines has to be rebooted, this can be done as with any remotely administrated machine.

13. After the exercise has ended – either by timeout, or because the student pushed the “Fertig!”-button – the system will revoke access to the lab systems by re-enabling the firewall. It then prints a message that the exercise is over, and that feedback on the exercise can be retrieved from the database within a few minutes as shown in figure 4.12.

The lab systems are analyzed in the background by a number of scripts. These scripts know what configuration steps are necessary for successful performance of the exercise, and will report their findings in the database for later retrieval.

14. Later, students can retrieve feedback on an individual exercise by selecting “Buchungen einsehen” from the main menu. They will see the exercise text, comments on what checks were done (green text), and if the particular task was done successfully (“OK”) or not (“Nein”). See figure 4.13 for an example.



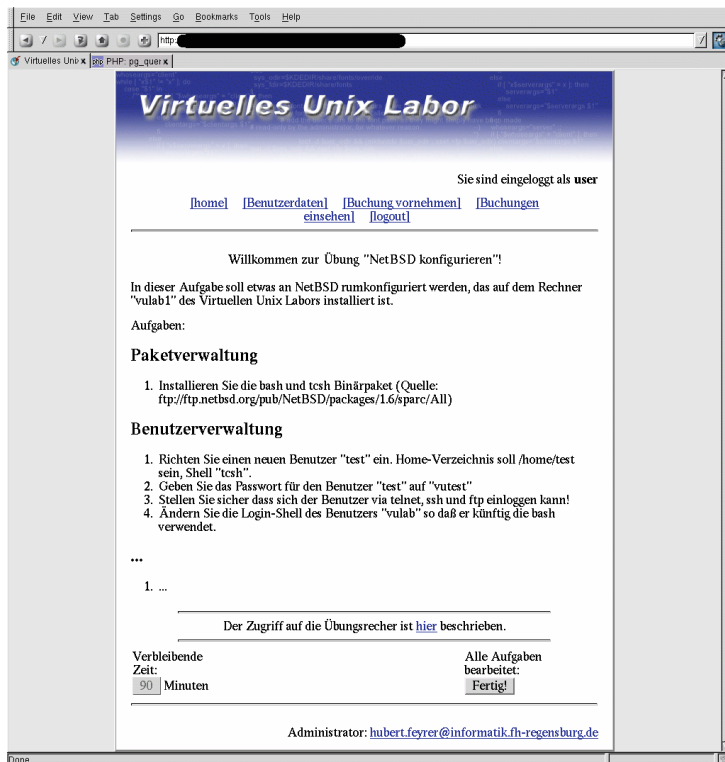


Figure 4.10: Display of the exercise text

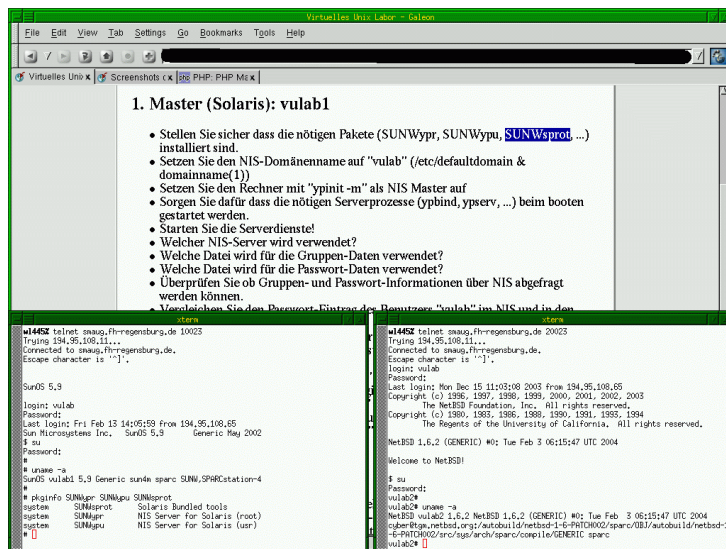


Figure 4.11: Logging into lab machines for the exercise

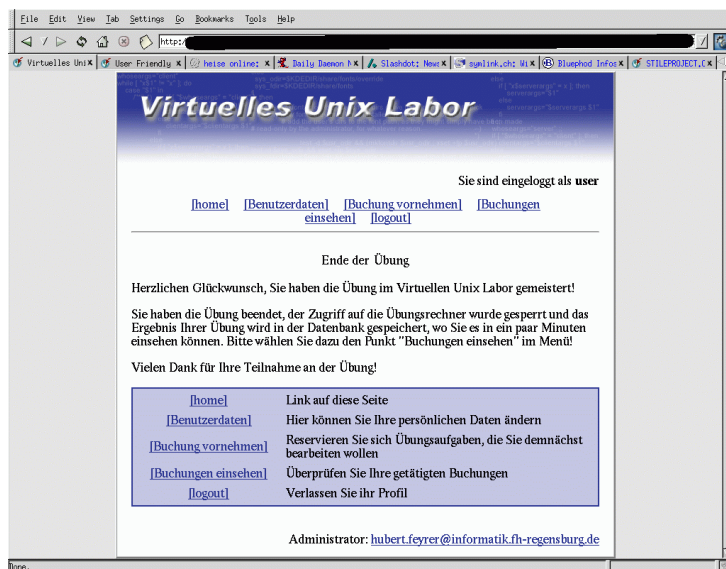


Figure 4.12: End of exercise

File Edit View Tab Settings Go Bookmarks Tools Help

virtuelles.uni.k... PHP: pg\_query

## Virtuelles Unix Labor

Sie sind eingeloggt als user

[\[home\]](#) [\[Benutzerdaten\]](#) [\[Buchung vornehmen\]](#) [\[Buchungen einsehen\]](#) [\[logout\]](#)

---

Auswertung der Übung "NetBSD konfigurieren"

Die Übung "NetBSD konfigurieren" (Buchungs-ID #77) fand am 2004-01-21 von 12:00:00 bis 13:23:37 statt und dauerte damit 83 von max. 90 Minuten. Die Übung wurde von der IP-Nummer 132.199.213.26 aus absolviert.

Es folgt die genaue Auswertung der einzelnen Teilaufgaben:

---

In dieser Aufgabe soll etwas an NetBSD rumkonfiguriert werden, das auf dem Rechner "vulab1" des Virtuellen Unix Labors installiert ist.

Aufgaben:

### Paketverwaltung

1. Installieren Sie die bash und tsh Binärpaket (Quelle: <ftp://ftp.netbsd.org/pub/NetBSD/packages/1.6/sparc/All>)
 

Pakete installiert? (pkg\_info -e) Nein

### Benutzerverwaltung

1. Richten Sie einen neuen Benutzer "test" ein. Home-Verzeichnis soll /home/test sein, Shell "tsh".
 

"test" finger(1)bar? OK  
 Korrektes Home-Verzeichnis? (finger, test -d) OK  
 Shell richtig gesetzt? (finger) OK  
 Eintrag in /etc/master.passwd? OK
2. Geben Sie das Passwort für den Benutzer "test" auf "vutest"
 

Passwort richtig gesetzt? (getpwnam(3), crypt(3)) OK
3. Stellen Sie sicher dass sich der Benutzer via telnet, ssh und ftp einloggen kann!
4. Ändern Sie die Login-Shell des Benutzers "vulab" so daß er künftig die bash verwendet.
 

Login-Shell vulab? (chfn/chsh, finger) Nein

...

1. ...

---

Anzahl Teilübungen: 7  
 Davon bestanden: 5 (72%)

---

<a href="#">[home]</a>	Link auf diese Seite
<a href="#">[Benutzerdaten]</a>	Hier können Sie Ihre persönlichen Daten ändern
<a href="#">[Buchung vornehmen]</a>	Reservieren Sie sich Übungsaufgaben, die Sie demnächst bearbeiten wollen
<a href="#">[Buchungen einsehen]</a>	Überprüfen Sie Ihre getätigten Buchungen
<a href="#">[logout]</a>	Verlassen Sie ihr Profil

Administrator: [hubert.feyrer@informatik.fh-regensburg.de](mailto:hubert.feyrer@informatik.fh-regensburg.de)

Done

Figure 4.13: Feedback on an exercise taken



Figure 4.14: The initial implementation of the Virtual Unix Lab

This overview illustrates the basic mode of operation a user can perform in the Virtual Unix Lab. The next section will describe the hardware components and the network setup that the Virtual Unix Lab was composed of.

## 4.2 Hardware and network setup of the Virtual Unix Lab

This section describes the hardware setup used in the Virtual Unix Lab, some of the possible alternatives, and why there were (not) used.

Despite its name, the current implementation of the Virtual Unix Lab uses real machines for performing the exercises on. During the initial design phase of the project in 2001/2002, virtual machines were slowly starting to become available in widespread use and were considered a useful alternative<sup>1</sup>. Due to budget limitations, no hardware was available to run virtual machines, and so “real” hardware was chosen for the current instance. A possible future goal is replacing the real machines with virtual ones. As such, keeping an eye on developments in that area was always considered of importance during the whole project’s lifecycle so far, even if that aspect is not realized yet – see chapter 2 for relevant work in that area.

Another design issue was that the production network should not be influenced. Due to that, the lab machines were put on an extra network behind the Virtual Unix Lab control machine, through which all access has to go. While keeping the lab machines from doing any evil on the production network, the added benefit is that access to the lab machines can be controlled tightly. Using the firewall’s port forwarding, it allows access to the lab machines only to those users who have booked an exercise previously.

Figure 4.15 illustrates the setup of the Virtual Unix Lab’s network and the network services that are available. Figure 4.14 shows a photo of the machine setup as it was made initially.

Initially, the control machine of the Virtual Unix Lab – shown in the left half of figure 4.14 and in the center of figure 4.15 – ran on a Sun SPARCstation 5 with a 85MHz CPU, 192 MB RAM and three external SCSI disk. An additional SBus ethernet card was added for connecting the internal lab, the machine ran NetBSD 1.6.2/sparc for

---

<sup>1</sup> [Pratt and Zelkowitz, 2001] pp. 57

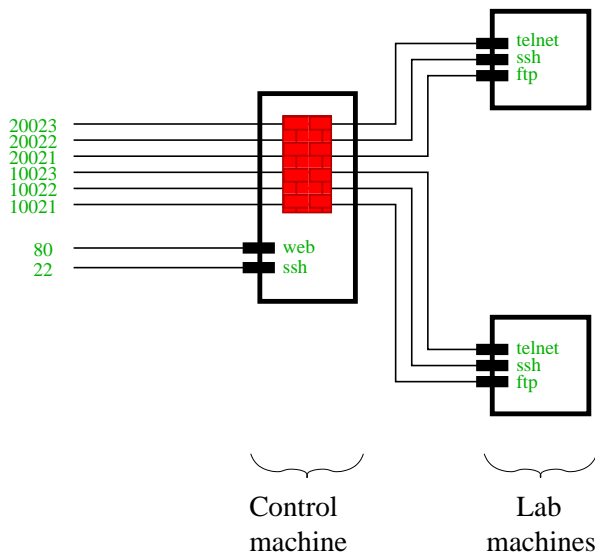


Figure 4.15: Accessing the lab clients

historic reasons. A hardware upgrade was made in 2005 to a Dell PC with a 3.2GHz CPU, 1GB RAM and two 100GB harddisks that are used in a software RAID1 configuration. NetBSD was chosen as operating system again, as it was easy to upgrade, fulfilled all requirements, and experience in its handling was available in-house.

For the lab clients – shown on the right of figures 4.14 and 4.14 – two Sun SPARCstation 4 with 110MHz CPU, 64 MB RAM and 1 GB internal SCSI disk, were used. The machines run NetBSD/sparc or Solaris/sparc, depending on the exercise.

### 4.3 Software components of the Virtual Unix Lab

This section gives an overview on the software components of the Virtual Unix Lab. Figure 4.16 illustrates the components and their relationship, a full overview is available in [Feyrer, 2004c]. A brief discussion of the various components follows:

**User:** The user is not part of the Virtual Unix Lab, but he is the main active component in the system. He provides input and interacts with the system, and is as such considered to be a vital part of the system design. Interaction is done through a web browser for management of exercises, and through a command line interface (ssh, ftp, telnet) during the exercises.

Interaction happens with the “User Management” component.

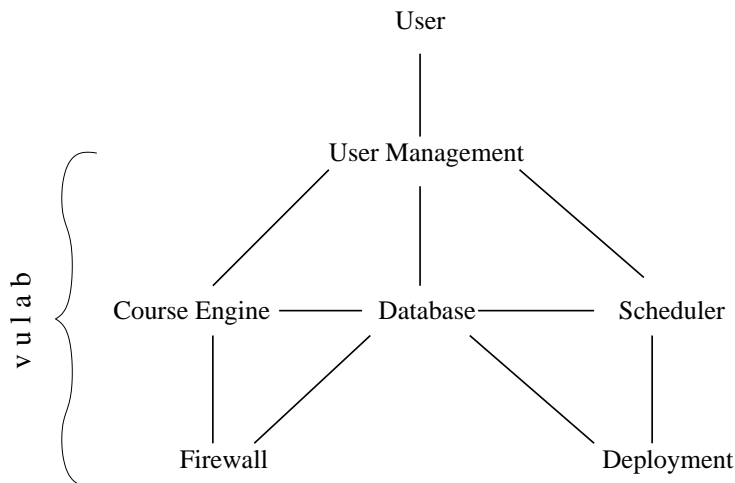


Figure 4.16: Software components of the Virtual Unix Lab

**User Management:** This component acts as a single point of contact towards the user. In cooperation with the Database, Scheduler and Course Engine it performs login procedures, account generation, books exercises and acts as user interface during the exercise. This component is described in [Zimmermann, 2003, pp. 9, 38]. The User Management component is mostly implemented in PHP.

**Course Engine:** After the User Management component has logged in the user, an exercise booked earlier may be ready for taking. If so, the handling of this will be done by the Course Engine: It makes sure that only the specific user has access to the lab machines by configuring the Firewall, waits for the start time of the exercise, and displays the exercise text and time left for the exercise. At the end of an exercise, it verifies the exercise results by analyzing the lab systems and collecting the data that is needed to give feedback to the user on his performance during the exercise.

The Course Engine component is discussed in detail in the following sections, an overview is available at [Feyrer, 2004e]. It is implemented as a mixture of PHP scripts for the user/web frontend parts, and Bourne shell and Perl scripts for the result verification parts.

**Database:** All data collected for the user accounts, exercise setup and deployment, feedback, etc. is stored in a relational database that all other components access. Access of the database happens via SQL from Perl and shell scripts.

The database is implemented with PostgreSQL. Reasons for PostgreSQL over other alternatives, in particular MySQL, are that PostgreSQL is free, and that it ran on the target platform, whereas MySQL did not work. The database is accessed from PHP, Perl and through Bourne shell scripts via the “psql” utility. See [Zimmermann, 2003, pp. 9, 69] for more information.

**Scheduler:** This component has two tasks: First, prepare the lab machines for any exercises that are booked, so they are ready in time. Second, 90 minutes after the exercise's scheduled start, the evaluation process is started, and the exercise is marked as done. If a user takes the exercise and finishes early by clicking on the "Fertig"-button, this is performed earlier, and the corresponding job is cancelled not to run after 90 minutes. The point is, if a user books an exercise but does not take the exercise, it would be marked as 'available' for an infinite time, which is suboptimal. By scheduling the second job, this is prevented.

The Unix at(1) facility that's started by the atrun(8) and cron(8) facility is used to implement the Scheduler component, the tasks to perform are realized as Bourne shell and Perl scripts.

**Firewall:** Access to the lab's exercise machines is controlled by a firewall, to ensure data safety in two ways. It restricts inbound access to the lab machines, and prevents outbound disruption of the production network that the Virtual Unix Lab is hooked up, see below. The firewall is configured in interaction with the Course Engine component.

The firewalling software used is IPfilter<sup>1</sup>, which is part of the NetBSD operating system, and which allows dynamic configuration. For more information see [Feyrer, 2004d].

**Deployment:** Setup of the lab machines is done by rebooting them via network boot (netboot), as described in [Feyrer, 2004f]. The netboot environment allows to access a file server, which provides harddisk images that are then written to the client's harddisk. After another reboot, the client boots from harddisk and is freshly installed. This deployment process is initiated by the Scheduler Component for every lab machine that needs to be setup for a particular exercise, the data for which is taken from the database.

The implementation of this is by performing a netboot of the Sun SPARCstation 4 machines via their OpenBoot PROM, use DHCP and TFTP to load NetBSD as base for deploying the harddisk image<sup>2</sup>. The lab client's harddisk image is loaded via the network file system (NFS) just like the netboot system itself. The design of this was influenced by previous experience from the g4u project<sup>3</sup>. See also [Feyrer, 2004b] for more details about the deployment process.

This section gives some understanding of the overall structure of the Virtual Unix Lab, which will be referred to further in this work from several places.

---

<sup>1</sup> [Reed, 2007]

<sup>2</sup> [The NetBSD Foundation, 2007]

<sup>3</sup> [Feyrer, 2007b]





# Chapter 5

## Introduction of domain specific languages

The Virtual Unix Lab uses a domain specific language to realize diagnosis and feedback in chapter 6. Domain Specific Languages (DSLs) or “Minilanguages” are programming and data description languages that are intended to be used for a special “domain”, a field of application that does not use the full potential of a language<sup>1</sup>.

This chapter classifies programming languages, lists criteria by which to recognize Domain Specific Languages from traditional programming languages, and explains how they are related. An overview of design patterns is given to determine ways to implement DSLs for an application domain, and criteria for the selection process as well as a DSL candidate is introduced.

### 5.1 Classification of languages

There are a number of ways to classify a language, and research on Domain Specific Languages is still ongoing. Judging purely by the term, DSLs are languages that are designed to serve a certain area (domain) of application to which they are specific, and which they intend to serve well.

When comparing programming languages, distinctions can be made according to various attributes and paradigms, e.g. as listed by Finkel<sup>2</sup>, Abelson and Sussman<sup>3</sup>, Hoare<sup>4</sup>,

---

<sup>1</sup> [Raymond, 2003] pp. 183

<sup>2</sup> [Finkel et al., 1995] pp. 1 and 3

<sup>3</sup> [Abelson et al., 1985] pp. 335

<sup>4</sup> [Hoare, 1973]

Aho<sup>1</sup>, Raymond<sup>2</sup> and Pratt and Zelkowitz<sup>3</sup>.

Describing how to design a new language is beyond the scope of this document, guidelines on language design, syntax, semantics and how to write a compiler or interpreter can be found e.g. in [Wirth, 1974], [Wexelblat, 1976], [Hoare, 1973], [Hilfinger, 1981], [Floyd, 1979], [Finkel et al., 1995], [Pratt and Zelkowitz, 2001], and [Aho et al., 2003], with ongoing research being discussed e.g. in the ACM SIGPLAN's "Programming Language Design and Implementation" (PLDI) group<sup>4</sup>.

When writing a computer program, two approaches are possible. One is compiling a source code written in a certain programming language into executable machine code<sup>5</sup>, the other is defining an "evaluator" or "interpreter" that interprets instructions and performs operations<sup>6,7</sup>. Compiling a language is a process that takes some effort once for lexical and syntactical analysis, code optimisation and code generation, but results in a fast executable when ran later, assuming the program is not modified very often. When a program is expected to change often, the overhead of interpreting the source language is not too high, or the program should be written once to be used on several different machine architecture, an interpreted language can be used - Spinellis talks about the distinction between "deep or shallow translation" here<sup>8</sup>. Medvidovic and Rosenblum also note that compiling a program into machine executable code is "simply a special case of architectural refinement" stepping down from a high level "boxes and arrows" design<sup>9</sup>.

Another aspect of current programming language evolution is that due to the ever-increasing processing speed of computers, the runtime overhead of interpreters is getting less and less of an issue<sup>10</sup>, and scripting languages<sup>11</sup> like Perl<sup>12</sup>, Python<sup>13,14</sup>, Ruby<sup>15</sup> and PHP<sup>16</sup> are becoming more and more attractive today, which also benefits any domain specific languages that are based on scripting languages instead of being compiled into machine code.

A recent trend is to use a hybrid approach that compiles code from a high-level language into bytecode, which is then interpreted by a virtual machine instead of a "real"

---

<sup>1</sup> [Aho et al., 2003]

<sup>2</sup> [Raymond, 2003] pp. 183

<sup>3</sup> [Pratt and Zelkowitz, 2001] pp. 19 and pp. 114

<sup>4</sup> [PLDI, 2007]

<sup>5</sup> [Aho et al., 2003]

<sup>6</sup> [Abelson et al., 1985] p. 294

<sup>7</sup> [di Forino, 1969] p. 68

<sup>8</sup> [Spinellis, 2001] p. 96

<sup>9</sup> [Medvidovic and Rosenblum, 1997] pp. 5

<sup>10</sup> [Wirth, 1974] p. 28

<sup>11</sup> [Wikipedia, 2007] "Scripting programming language"

<sup>12</sup> [Wall et al., 1996]

<sup>13</sup> [Rossum and Drake, 2003]

<sup>14</sup> [Rossum and Fred L. Drake, 2003]

<sup>15</sup> [Matsumoto, 2001]

<sup>16</sup> [The PHP Project, 2007]

CPU. The compilation step ensures type safety and optimization, and the virtual machine offers the same platform on every hardware and operating system which which it is available. The result is a trade-off between safety in programming and portability, with minor impact on performance. Languages to name in this area are Java<sup>1</sup> and C#<sup>2</sup>.

Modifying an existing language to adjust it for a special application domain<sup>3</sup> leads to a Domain Specific Language. There are several ways to create a domain specific language, as will be discussed in the next section.

## 5.2 Attributes of domain specific languages

By definition, Domain Specific Languages are limited to a small area of application, and often embedded into a larger system with the goal to reduce the semantic distance between a problem and the program. Spinellis lists the following attributes as specific to DSLs, which set them apart from “normal” languages<sup>4</sup>:

**Concrete expression of domain logic:** Instead of using an existing programming language and overloading it with details from the application domain, the details are put into the DSL. This removes details that are not of interest to the application domain, and programmers can concentrate on issues related to the domain.

**Direct involvement of domain experts** results from the above. As there is no excessive ballast between the application domain and the person with expert knowledge of the application domain, the domain expert can directly model any domain knowledge with no person in between that needs to translate from the application domain into a programming language.

**Expressiveness** is the other result from removing unnecessary parts of a language. What is left is explicitly expressing knowledge of the application domain only, no superfluous code that only exists to support the programming system or application language. Instead, all this meta-knowledge is moved into the DSL’s processing system.

**Runtime efficiency:** Possible interactions between different elements of general purpose languages can have negative impacts on performance, e.g. from type systems and conversion of data between multiple internal formats. Using a DSL focused on the problem can provide optimisation and lead to efficiency here.

**Modest implementation costs:** DSL systems are usually implemented within a larger system, and as such, they can use tools and interfaces already available. Also, as

---

<sup>1</sup> [Gosling and McGilton, 1996]

<sup>2</sup> [ISO 23270, 2006]

<sup>3</sup> [Wirth, 1974] p. 29

<sup>4</sup> [Spinellis, 2001] p. 91

they are directed toward a certain (small) goal and not towards solving a general problem, implementation costs can be kept low by only making them handle that area of application (and possibly hand over remaining tasks to other subsystems or languages).

**Reliability** follows from this immediately, as the language does not intend to be of general purpose. Handling only a small scope can often be easier, if not trivially, verified to be correct.

**Tool support limitations** might be a problem with DSLs, as existing software tools and languages like editors, CASE tools, build systems, debuggers and version control systems may not be prepared to handle a new, unknown language. Ad hoc solutions need to be developed to integrate DSLs into tools to solve that problem.

**Training costs** arise from the fact that system implementers and maintainers as well as domain experts have by definition no prior experience in the new DSL, and thus need training to get familiar with the new language and possibly its integration into the development process and environment.

**Software process integration** can not be expected in currently established software processes. CASE tools and processes usually assume already existing, well known programming languages and are flexible enough to integrate languages that reflect application specific properties rarely, and thus need to be modified, if possible.

**Design experience** is needed for creating a DSL that will actually solve problems instead of creating new ones. There are several guidelines for doing so, as outlined in the next section.

The above list of attributes was compiled by Spinellis<sup>1</sup>, similar findings can be found in [Bentley, 1986]pp. 719 and [Mernik et al., 2005].

### 5.3 Design patterns

In the previous sections, different kinds of programming languages following different paradigms and areas of application were observed. If a language does not fully fit a certain application, it can be changed to fit better. If a new language needs to be created for a certain kind of application, it can often be based on or derived from an existing language. In either way, a Domain Specific Language (DSL) is the result, and we will look at various approaches to do so in this section.

---

<sup>1</sup> [Spinellis, 2001]

Raymond describes three ways to create a domain specific language or “minilanguage”, two “right” ones and a “wrong” one. The two “right” ones are recognizing upfront that an existing language needs to be extended and pushed up to a higher level of abstraction, and noticing that a data file format starts containing complex structures and elements that imply action. The “wrong” approach to add one feature after another to a data or configuration file, as this will lead to an inconsistent language that may be difficult or impossible to verify, or even provide insecure exits to routines not originally intended. A solution to avoid designing a bad language by accident is to know how to do it right<sup>1</sup>. For this reason, the patterns that can be identified when designing a domain specific language are introduced here.

Spinellis uses design patterns to describe ways for constructing programming languages<sup>2</sup>, and he cites Christopher Alexander, who defined design patterns as the relationship between recurring problems and their respective solutions: “Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice”<sup>3</sup>.

Spinellis identifies the following design patterns for domain specific languages<sup>4</sup>:

**Language extension:** If an existing language is mostly fit for a particular application, but lacks some constructs, the language can be extended to support the additional features<sup>5</sup>. Most of the existing language is kept in use for this, including command structure and type system. Only the few constructs that are needed for the DSL are added, usually by a preprocessor that transforms the extended language into the original one without burdening the domain expert with the details of the implementation. Notable examples of this pattern are the original C++ implementation via the “cfront” preprocessor<sup>6</sup>, and the “Rational Fortran” (Ratfor) compiler that provided elements of structured programming for Fortran<sup>7</sup>.

**Piggyback:** If a new language needs to be created due to lack of an existing language that can be extended, it is likely that features like control structures, type system and procedure handling are needed<sup>8</sup> that are already present in an existing programming language and its processing tools (compiler or interpreter). Construction of the new DSL including definition of grammar, syntax and semantics can still happen independent of any existing language, while implementation can be done by using common elements shared with the existing language<sup>9</sup>. Possible ways are to either translate into existing source code of an already-existing

---

<sup>1</sup> [Raymond, 2003] pp. 183

<sup>2</sup> [Spinellis, 2003] pp. 331

<sup>3</sup> [Alexander, 1995] p. X (Foreword)

<sup>4</sup> [Spinellis, 2001]

<sup>5</sup> [Spinellis, 2001] p. 95

<sup>6</sup> [Stroustrup, 1994] pp. 66

<sup>7</sup> [Kernighan, 1975]

<sup>8</sup> [Ledgard, 1971]

<sup>9</sup> [Spinellis, 2001] p. 93

language and let the existing compiler system handle the code, or even to create a compiler-frontend that translates into machine-readable intermediate code that can then use the compiler's optimizer and code generator. The latter approach needs intimate knowledge of the interface between the compiler frontend and backend and the (machine-readable) data format passed between them. Translation using (human-readable) source code is usually found easier for faster progress, less compiler-internals to consider and easier debugging. Notable tools for aiding in a piggyback design are `lex`<sup>1</sup> and `yacc`<sup>2</sup>.

**Language specialization:** At times, it happens that a language is needed for an area of application that should not allow certain constructs, like for example dynamic memory allocation, references to static and/or dynamic memory (pointers), type-free programming (C: “`void *`”) and jumps (`goto`)<sup>3</sup>. Reasons for this may be easier verification and increased security of programs written in the new language. A possible approach is to “remove” the unwanted constructs from an existing language<sup>4</sup>. Examples where major languages had some features removed are `Javalight`<sup>5</sup>, the Automotive “Save Subset” of C<sup>6</sup>, and both `HTML`<sup>7</sup> and `XML`<sup>8</sup> as a special form of `SGML`<sup>9</sup>.

**Lexical Processing:** Due to the limited field of applicability, it is possible to create DSLs by using techniques of simple lexical processing and substitution. Instead of a full, tree-based syntax analysis, lexical hints can be embedded into a language that are used to identify tokens that need special processing, e.g. by adding a special prefix and/or postfix for variables<sup>10</sup>. The form of lexical processing can also be used together with the piggyback approach to translate the DSL into its base language by applying simple lexical translation, and then handing off the result to the base language's processing tools.

Using the technique of lexical processing and substitution reduces implementation costs as it makes creating special languages possible where a full tree-based approach would demand too many resources in knowledge, implementation effort, and effectively time and money. Often, interpreters or rapid prototyping languages are used for implementing the lexical process, which allows design and implementation of the DSL to happen as an iterative process. Tools often

---

<sup>1</sup> [Lesk and Schmidt, 1975]

<sup>2</sup> [Johnson, 1975]

<sup>3</sup> [Wirth, 1974] p. 25

<sup>4</sup> [Spinellis, 2001] p. 95

<sup>5</sup> [Nipkow and von Oheimb, 1998]

<sup>6</sup> [Edwards et al., 1997]

<sup>7</sup> [Berners-Lee et al., 1999]

<sup>8</sup> [Derose, 1997]

<sup>9</sup> [ISO 8879, 1986]

<sup>10</sup> [Spinellis, 2001] pp. 94

found in realizing a DSL this way include sed<sup>1,2</sup>, awk<sup>3,4</sup>, m4<sup>5,6</sup>, the C Preprocessor<sup>7</sup>, Perl<sup>8</sup> and Python<sup>9</sup>. All of these tools offer easy to use ways for lexical processing and substitution, often based on regular expressions<sup>10,11</sup>.

**Data Structure Representation:** It is not always program code that is special to a particular area of application. At times, data needs to be structured for a certain application. Describing it in a form that's close to the application domain and then transforming it from a domain-specific representation to an implementation-specific representation offers all the benefits found in domain specific application languages, like easier use by domain experts and possibility of verification during the transformation process. Spinellis argues that anything beyond initialisation of a simple rectangular array should be represented by a DSL, and the more complicated data becomes by means of interconnection and intercorrelation, the more important consistency, automated consistency checking, and validation of input are<sup>12</sup>. No matter what the complexity is, keeping the representation in the application domain allows transforming it into various internal ways along with choosing an optimal internal representation by e.g. replacing linear lists with trees or hash tables<sup>13</sup>. Prominent examples where this approach is used is the transformation of lists in Perl as well as many Lisp and Prolog dialects into more efficient internal representations at runtime, as well as the internal tables used by lexical analyzers and parsers created by lex<sup>14</sup> and yacc<sup>15</sup>.

**Source-to-Source Transformation:** As already mentioned, there are several approaches when creating a DSL. Doing simple lexical substitution is one way, full lexical and syntactical analysis, constructing an internal tree based on grammar, doing possible optimisation and generating code is another. Spinellis calls it the difference between "shallow or deep translation."<sup>16</sup> As described above at the "Piggyback" pattern, the latter approach is possible, but not easy in terms of implementation costs, testing, debugging, verification and knowledge needed. When using this pattern, the goal is to transform the DSL into an existing source language that can then be processed by the existing compiler, optimizer and de-

---

<sup>1</sup> [Dougherty and Robbins, 1997]

<sup>2</sup> [The Open Group, 2004] Base Specifications Issue 6: "sed - stream editor"

<sup>3</sup> [Aho et al., 1988]

<sup>4</sup> [The Open Group, 2004] Base Spec. Issue 6: "awk - pattern scanning and processing language"

<sup>5</sup> [Kernighan and Ritchie, 1994]

<sup>6</sup> [The Open Group, 2004] Base Specifications Issue 6: "m4 - macro processor"

<sup>7</sup> [Kernighan and Ritchie, 1988] pp. 192

<sup>8</sup> [Wall et al., 1996]

<sup>9</sup> [Dougherty and Robbins, 1997]

<sup>10</sup> [Friedl, 1997]

<sup>11</sup> [The Open Group, 2004] Base Specifications Issue 6: "9. Regular Expressions"

<sup>12</sup> [Spinellis, 2001] pp. 96

<sup>13</sup> [Ledgard, 1971] pp. 134

<sup>14</sup> [Lesk and Schmidt, 1975]

<sup>15</sup> [Johnson, 1975]

<sup>16</sup> [Spinellis, 2001] p. 96

bugger<sup>1</sup>. Another advantage is that the output of the DSL “compile” (transformation) process is still human-readable, which makes verifying and debugging much easier. An example for this pattern can be found in the PIC picture language and its use as a target language for the CHEM language preprocessor<sup>2</sup>.

**Pipeline:** Domain specific languages by definition intend to do a small job well. When several similar tasks need to be performed, extending a language is one thing, splitting it into two separate tools with each tool only covering its area of excellence and depending on other tools to do the remaining work is another option<sup>3</sup>. This is the principle of “pipelining” languages and tool – feed a source language in on one end, get it processed by one language and tool, then pass its output on to the next one after possibly rewriting the input. At the end, the result is influenced by all tools that processed the input<sup>4</sup>. This approach encourages splitting up a language into several smaller parts, with each one having the benefits of a DSL. Assembling the single parts can then happen by using facilities found on many modern operating systems, like the command line processors and utilities that can be found on Unix systems<sup>5,6</sup>. Examples that use the pipelining pattern are the PBMplus image manipulation tools<sup>7</sup> and the “troff” set of typesetting tools<sup>8</sup>. The latter come with specialized tools and languages for handling equations, tables, references, pictures and derivations like directed graphs, chemical structures, that are all transformed back into the basic format before being processed eventually<sup>9</sup>.

**System Front-End:** In large systems that have several ways to access and configure internal objects, e.g. using either graphical user interfaces, programming libraries or command line options, it is useful to provide a DSL that allows users to perform these actions. This leads to a declarative, maintainable, organized and open-ended mechanism for accessing these areas. When exposing settings and objects via some variables and functions of a DSL, it may be useful to remove code manipulating these settings and objects from the original system, and rewrite them in the DSL for simplicity of implementation, prevention of code redundancy, and easier maintenance. Besides simplifying a system, inventing a DSL leads to other benefits like making the system extendable via the DSL (e.g. via some plugins or loadable scripts), the DSL provides a common language for its users, and it also allows third parties to supply products based on the interface provided by the DSL<sup>10</sup>.

Existing languages that are used for customizing and adapting large software

---

<sup>1</sup> [Spinellis, 2001] p. 96

<sup>2</sup> [Bentley, 1986] pp. 716

<sup>3</sup> [Spinellis, 2001] p. 95

<sup>4</sup> [Bentley, 1986] pp. 712

<sup>5</sup> [Salus, 1994] pp. 50

<sup>6</sup> [Meunier, 1995]

<sup>7</sup> [Poskanzer, 2007]

<sup>8</sup> [Ossanna and Kernighan, 1976]

<sup>9</sup> [Bentley, 1986] pp. 716

<sup>10</sup> [Spinellis, 2001] pp. 97



products include Lisp for the Emacs<sup>1,2</sup> editor and AutoCAD<sup>3</sup>, Microsoft's Application Basic<sup>4</sup> for Microsoft's Office suite and ABAP<sup>5</sup> for the SAP ERP system. Numerous small DSLs exist for many of the tools found on Unix systems, including mail readers, shells and graphical application<sup>6</sup>.

The above list contains single patterns that can be employed on existing general and domain specific languages alone or in combination to create new DSLs with the goal that the new language is better suited to the area of application. A similar list can be found in [Mernik et al., 2005, pp. 320].

Whether an existing language is better suited for applying any of the patterns or for implementing a translator for a DSL depends on the language's characteristics, which are illustrated in the next section.

## 5.4 Choosing an implementation languages

When looking at a language for use in a DSL creation process, an existing programming language may be used in either of two positions<sup>7</sup>:

- Use as a **base language** by extending an existing language for a new DSL. See section 5.3 for a discussion of possible design patterns that can be employed for this task.
- Use as an **implementation language** for the translator (compiler) or evaluator (interpreter) of the new language. Various requirements for this are listed in [Spinellis, 2001], among them are presence of lexical methods and in-depth knowledge of interfaces between compiler frontends and backends for possible reuse of compiler backends.

Other attributes that are considered important here are:

- **Integration layer**: At which level can a DSL process be installed<sup>8</sup>? Source level is one possibility, using existing interfaces between compiler frontend and

---

<sup>1</sup> [Chassell, 2004]

<sup>2</sup> [Glickstein, 2004]

<sup>3</sup> [Rawls and Hagen, 1998]

<sup>4</sup> [Boctor, 1999]

<sup>5</sup> [Keller and Krüger, 2001]

<sup>6</sup> [Raymond, 2003] pp. 183

<sup>7</sup> [Bentley, 1986] pp. 717

<sup>8</sup> [Spinellis, 2001] p. 94

backend or between bytecode compiler and bytecode interpreter may be possible, but depends on accessibility and documentation of that interface. This may be implementation specific.

- **Compiler or Interpreter:** The fact whether a resulting executable depends on a certain machine architecture but does not need an exhaustive runtime system – as in a compiler scenario – or if a program can run on many platforms but needs an appropriate runtime system – usually in the form of an interpreter – may be less interesting from the design point of view of a DSL, but when using the result later, it may very well be a limitation<sup>1</sup>, and needs to be considered early in the design process, see “platform-availability” below. Also, scalability limits may be encountered, depending on the approach chosen here<sup>2</sup>.
- **Platform-availability** can be a limiting factor in existing projects, as DSLs have to work in the environment they are designed for, and can not dictate that environment per definition. The question is if an implementation is available on the platform – hardware and operating system – of choice. When looking at operating systems, mostly Microsoft Windows and Unix based systems are of interest today, where “Unix” includes all POSIX-compliant flavours from both commercial vendors like Sun’s Solaris, IBM’s AIX, Apple’s Mac OS X, and HP’s HP/UX as well as free systems like Linux and NetBSD. The language in question may be part of the base operating system, being available either through commercial vendors or as freeware.

A programming language that fulfills the above criteria was needed for the creation of a domain specific language in chapters 6 and on. The choice fell on the Perl programming language for the following reasons:

- The Perl programming language contains features of C, sed, awk and the Bourne shell<sup>3</sup>. It offers control structures needed for structured programming, Perl version 5 and later also offers the option of object oriented programming, which is not imposed upon the programmer by following the Perl mantra: “There is more than one way to do it.” Data representation in Perl programs is mostly strings, with implicit data conversion for numerical context. Advanced data structures are available as lists and hash tables, more complex structures can be realized with the OOP framework.
- Perl itself works as a interpreter – internally, the source code is compiled into a bytecode that is then interpreted, but there is no easy to use interface available for accessing the bytecode to modify it or feed created bytecode to the execution backend, and extend Perl that way. An interface for embedding Perl language support into existing programs is available though, and the fact that Perl code is interpreted gives it platform independence.

<sup>1</sup> [Spinellis and Guruprasad, 1997] p. 2

<sup>2</sup> [Spinellis and Guruprasad, 1997] p. 8

<sup>3</sup> [Wall et al., 2000]

- In contrast to C, C++ and Java, Perl provides built-in support for handling regular expressions, and combined with its strong string processing model, this makes Perl an ideal choice for an implementation language for a DSLs, and there are indeed many examples of this<sup>1,2,3,4</sup>. While handcrafting lexical analysis is easy using Perl's built in regular expression feature, analysing syntax can be handed off to tools like `py`<sup>5</sup> if needed.
- Platform availability for Perl is excellent, for both operating systems and hardware platforms covered. Perl is written in C using portable system interfaces, which ensures that it works on all platforms that provide POSIX compatibility - initially being developed on Unix, Perl has been ported to Microsoft Windows and also many more platforms. The Perl source code is freely available<sup>6</sup>, and a big collection of routines and modules is available in the Comprehensive Perl Archive Network (CPAN)<sup>7</sup>.

The overview given for domain specific languages, their associated design patterns and the choice of an implementation languages will be applied to the architecture and implementation of diagnosis and feedback in the Virtual Unix Lab in section 6. The next chapter illustrates the related design.

---

<sup>1</sup> [Spinellis, 2007]

<sup>2</sup> [Spinellis and Gritzalis, 2000]

<sup>3</sup> [Ramming, 1997]

<sup>4</sup> [Ball, 1999]

<sup>5</sup> [py, 2007]

<sup>6</sup> [CPAN, 2007] "Perl Source Code"

<sup>7</sup> [CPAN, 2007] "Perl Modules"



## Chapter 6

# Architecture and implementation of diagnosis and feedback with a domain specific language

A major task of the Virtual Unix Lab is to perform diagnosis and verification of the exercise results, and provide feedback to the student. This chapter discusses verification of exercises results performed in the Virtual Unix Lab by using Domain Specific Language (DSL) techniques. The basics of DSLs were discussed in section 5, and the requirements, design, and implementation for the Virtual Unix Lab will be described in this chapter.

A more in-depth description of the implementation including many technical details can be found in [Feyrer, 2007d].

### 6.1 Requirements of exercise verification

There are a number of requirements tied to the result verification framework, that will be discussed in this section. This includes portability of verification checks, an efficient verification interface to the lab systems, integration of verification into exercise-design, and storing results for evaluation purpose.

**Portability of checks:** The ultimate goal of the Virtual Unix Lab is not to be specific to Unix only, but to also offer exercises for other, non-Unix(like) systems like Microsoft Windows. While the Virtual Unix Lab engine will remain on one machine using whatever platform, it has to be flexible enough to execute code for verification purpose on many systems.

An exercise consists of several individual task. Successful performance of each

task can be verified by testing one or more system settings on the lab machine. Verification of a single setting is done via a so-called “check-script”, a small code fragment that inspects only that fact, and that returns either “true” to indicate that the setting was in favour of the exercise’s goal, or “false” indicating that the setting was not tuned properly to solve the task. The exact range of checks to perform is specific to the exercise and lab system(s) as described in the next sections.

A requirement of the verification mechanism is to be independent of machine architecture and operating system where possible: The first implementation of the Virtual Unix Lab used two Sun SPARCstations running NetBSD and Solaris as lab machines, but the goal was to also include PCs running Linux and Windows. For this, the check-scripts had to be general enough to check one aspect on as many systems as possible. Aspects that were specific to a certain hardware or operating system were still possible, and runnable only on one system then, but the general goal was portability of the checks.

*Example:* A script that checks if a file is present should be usable on as many systems as possible. A C program will only run on one (CPU, operating system) combination, so an interpreter was required to run on the system and take more abstract commands.

**Efficient interface to the lab systems:** A requirement tightly coupled to portability is to have a way to run the check scripts on the lab machines in an efficient way. There are several ways across different operating systems that had to be evaluated, and the goal was to find one method that was common to most systems.

*Example:* Be able to have one script that checks if a certain file was present when the target system to verify runs either Unix or Windows.

A second requirement of the remote execution system was that it is fast. Only one setting was checked by each check-script call, and a number of calls were needed to acquire the full state of the lab system to give an overview of the overall success or failure of the system.

*Example:* Doing a ssh-call to a 75MHz SPARCstation 4 is quite slow thanks to the cryptographic methods used by ssh. rsh is much better in this regard<sup>1,2</sup>.

The third and last requirement for the interface system to the lab machines was passing back the check-result. As passing of complex data is not easily possible, a simple boolean value indicating success or failure was chosen as the result, which was passed back to the calling system.

*Example:* The above-mentioned script that checks if a file is present should say if it is there or not. Other checks are made for file contents or other files.

**Integration into exercise-design:** The first approach for the Virtual Unix Lab exercise design was to have exercise texts separated from verification of the exercise

---

<sup>1</sup> [Feyrer, 2001] “Beschreibung der Berechnungsvorgänge”

<sup>2</sup> [Schaumann, 2004] p. 146

goals (see sections 6.3 and 6.4 below). It quickly became obvious that having a connection between a part of an exercise's text and the corresponding aspects to test to see if that part was successful was helpful both for designing the exercise as well as to provide feedback to the user later.

The requirements for this connection between the exercise text and the verification checks had to be easy to realize, but also to be flexible enough to allow storing the result for later evaluation.

**Storing results:** The results of checks need to be stored for later evaluation. For example, to compare the performance of all students on a certain exercise or part of it, how an individual student performs on a class of exercises, or to identify special areas where a lack of knowledge exists and should be filled by better education. Other applications would be a tutoring component that could act based on the results of earlier exercises of an individual or all students, or a system that may adapt to the user, again based on results of earlier exercises of an individual or all students. See chapters 10 for tutoring extensions to the Virtual Unix Lab, and chapter 11 for a discussion of user adaption.

In summary, it is desirable to store information on the granularity of the check-level, and associate each check result with a context identified by the student, and the exercise that the result was performed in.

## 6.2 Roadmap of implementation

### 6.2.1 Stepwise refinement

Implementing verification of exercise results in the Virtual Unix Lab consists of several design and implementation steps described below, displaying an evolutionary design<sup>1,2</sup>, which is common for complex learning systems today<sup>3</sup>.

Cocke and Schwartz suggest the following four steps to express complex functions like the verification of the exercises' results<sup>4</sup>:

1. Find a set of "stereotypes", i.e. functional abstractions, that cover as much of the field to investigate as possible.
2. Analyze the stereotypes for repetitive patterns and develop a framework to call them, including any possible parameters.

---

<sup>1</sup> [Wirth, 1974] p. 29

<sup>2</sup> [Hoare, 1973] pp. 25

<sup>3</sup> [Kölle, 2007] pp. 139

<sup>4</sup> [Cocke and Schwartz, 1970] pp. 10

3. Design a “language processor” (interpreter) or compiler that understands the overall structure.
4. When steps 1-3 are implemented properly, a domain specific language and processor will be available to efficiently implement verification of results in the Virtual Unix Lab.

The steps involved in the design and implementation of the verification of exercise results in the Virtual Unix Lab do not follow the scheme described above in an exact way. To allow early testing, a simple abstraction of stereotypes with a framework for creating and calling them was chosen and implemented first, which was refined in later steps. As such, the system evolved in a rapid approach, following methods from agile programming and extreme programming, with special emphasis to ensure a test driven development (TDD)<sup>1,2,3</sup>.

### **6.2.2 Exercise phases**

The following phases of the exercise are involved in the result verification process. They are covered for each of the design steps:

**Preparation:** Describes the procedure to create a new exercise. Items involved are the text of the exercise that is presented to to the student during the “Exercise” phase, and the program code to perform the actual verification called in the “Verification” phase.

**Exercise:** after the initial preparation, the text of the exercise is presented. Possible preprocessing that is done at runtime instead of during the preparation phase is described here.

**Verification:** After the student has ended the exercise, the system performs the verification steps by running the code to check what was done and what was not, and stores the results in the database.

**Feedback:** After the verification has stored the results of the various checks in the database, methods for feedback will access the data. Ranging from simple evaluation of one student’s performance in one particular exercise (“This is how you performed in the preceding exercise”) to more sophisticated analysis involving several students and/or exercises.

Providing feedback is discussed in section 6.5.3.

---

<sup>1</sup> [Cunningham, 2001]

<sup>2</sup> [Beck, 1999]

<sup>3</sup> [Beck, 2002]



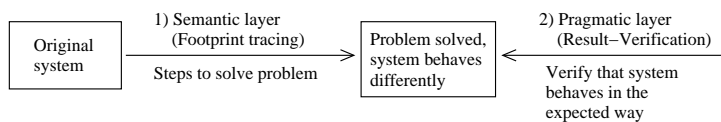


Figure 6.1: Verifying on the semantic and pragmatic layer

### 6.2.3 What and how to verify

Following Morris' theory of signs, there are several semiotic layers on which signs can be interpreted. Such signs can be actions and/or states in a computer system in general, or during verification of exercise results in particular. The signs can be interpreted on the syntactic, semantic and pragmatic layer<sup>1</sup>:

**Syntactic layer:** Interaction with the systems happens through single mouse strokes, keys and mouse buttons pressed. Intercepting all these interaction “events” is possible, and the “higher-level” actions initiated by any number of these “low-level” actions can be determined with some effort. This is useful for finding how basic interaction with the system is performed, e.g. if many commands are mistyped and corrected or if many menus are searched before finding and selecting the required item.

**Semantic layer:** There are several logical steps involved to solve a problem, for example each consisting of one or several commands being run or items in graphical user interfaces being clicked on. The execution of each of these logical steps can be verified, and testing on the semantic layer means to verify if any of these pre-defined steps were performed properly (“Footprint-Tracing”). This process is also known as causality tracking<sup>2</sup>. Examples are software packages installed, files created, entries made to configuration files etc.

**Pragmatic layer:** Ignoring the way *how* a problem was solved, the system can be checked to determine if the requested result was reached or not, i.e. if it behaves in a different way as if the problem in question was not solved (“Result-Verification”). Examples here are verification if a certain service like web, mail or file service runs, or if a configuration problem no longer exists.

In the Virtual Unix Lab, no testing is performed on the syntactical layer, as this is difficult to realize. The higher levels are more appropriate to determine outcome of the exercises. Figure 6.1 illustrates how verification on the semantic and pragmatic layers can be performed. See also the discussion of diagnostic data in section 8.1.4.3.3, methods for plan recognition in section 8.1.2.2 and on-line diagnosis in section 10.3.

In summary, besides the *what* to verify there is also a *how* to verify:

<sup>1</sup> [Morris, 1938] pp. 20

<sup>2</sup> [Alvisi et al., 2002]

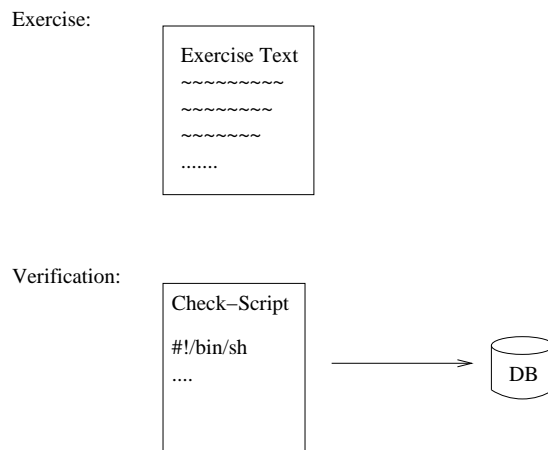


Figure 6.2: Step 0: Separate exercise text and verification check script

**What to verify** depends on the *tasks* that are to be solved as part of the exercise, and also of the goals of the exercise that the student should learn.

**How to verify** depends on the *system environment* that the exercise defines, as for some systems doing one way or the other may be easier.

### 6.3 Step 0: Basic design

An exercise in the Virtual Unix Lab consists of two parts, the exercise text and the corresponding check scripts. The exercise text is presented as a web page to the student, which describes all the tasks to perform. The check scripts are ran when the lab exercise is over, either when time runs out, or when the user clicks on the “Fertig”-button. The results found by the check scripts are stored in the database for later analysis and evaluation. Figure 6.2 illustrates the basic idea.

The verification process itself is controlled by the Virtual Unix Lab’s Course Engine, which is omitted from figure 6.2 to make the basic flow of information clearer.

The scheme displayed here was never implemented in any of the design and implementation steps of the Virtual Unix Lab, as a single, monolithic check script to do all the verification steps does not allow passing the result back in an easy way. Also, code re-usability would have been more difficult.

The following sections describe steps that were actually implemented in the Virtual Unix Lab, and they address the flaws mentioned in this basic design.

## 6.4 Step I: Instructions and checks not coupled

This section describes the first design step of the result verification architecture implemented in the Virtual Unix Lab. A description of the design with the key components their integration is given, followed by a discussion of possible improvements for the second design iteration in step II.

### 6.4.1 Components

The Virtual Unix Lab result verification architecture consists of a number of key components that reflect the exercise and which interact in certain ways. This section introduces these components.

**Exercise text:** In the first incarnation of the Virtual Unix Lab’s result verification architecture, the exercise text was stored in plain HTML text and displayed at exercise time. The HTML text was embedded into a larger document that gave the usual web layout, a display of the time remaining and a button to indicate that all tasks were completed and the exercise was finished early.

See figure 6.3 for an example of the plain exercise, figure 6.4 shows the text rendered in a HTML browser. The full texts of the exercises presented to the students can be found in appendix A.1.

**Check-scripts:** They run either on of the lab machines or an “outside” machine to check if an aspect of the exercise was performed successfully or not. The whole exercise consists of a number of checks to verify all parts of the exercise.

Following Cocke and Schwartz’ “stereotype” paradigm, check scripts are abstractions to map complex verification operations expressed in an arbitrary language (usually a Perl or Bourne shell script) into abstract primitives that perform their pre-defined task, and report success or failure upon completion<sup>1</sup>.

To determine which primitives are needed for a certain task, getting an overview of the problem area in question e.g. as expressed in [Ernst, 2004] was a good first step. Looking at the possible areas that the Virtual Unix Lab would be used for, various groups can be identified for which check primitives will be needed:

- **Networking:** Example primitives could verify configurations and settings, conformance to specifications like RFCs, proper network throughput, a list of open or closed ports, and standard replies to various network protocols.
- **Operating systems:** Checking would be for type, version, system platform, installation of system and application software, files, processes and other topics covered e.g. in standards like POSIX and SUSv3<sup>2</sup>.

<sup>1</sup> [Cocke and Schwartz, 1970] pp. 6

<sup>2</sup> [The Open Group, 2004]

```
feyrer@rfhpc8317:~/home3/bedienst/feyrer/work/vulab/code/public
rfhpc8317% pwd
/net/rfhs8012/home3/bedienst/feyrer/work/vulab/code/public.html
rfhpc8317% cat texte/netbsd.html
<!-- $Id: netbsd.php,v 1.14 2004/02/23 15:40:15 feyrer Exp $ -->

In dieser Aufgabe soll etwas an NetBSD rumkonfiguriert werden, das auf
dem Rechner "vulab1" des Virtuellen Unix Labors installiert ist.
<p>

Aufgaben:
<p>

<h2> Paketverwaltung </h2>
<ol>
<li> Installieren Sie die bash und tcsh Binärpaket (Quelle:
ftp://ftp.netbsd.org/pub/NetBSD/packages/1.6/sparc/All)
</li>

<h2> Benutzerverwaltung </h2>
<ol>
<li> Richten Sie einen neuen Benutzer "test" ein. Home-Verzeichnis
soll /home/test sein, Shell "tcsh".
<li> Setzen Sie das Passwort für den Benutzer "test" auf "vutest"
<li> Stellen Sie sicher dass sich der Benutzer via telnet, ssh und ftp
einloggen kann!
<li> Ändern Sie die Login-Shell des Benutzers "vulab" so daß er
künftig die bash verwendet.
</li>

<h2> ... </h2>
<ol>
<li> ...
</li>
rfhpc8317% █
```

Figure 6.3: Exercise text with no associated checks, in plain ASCII

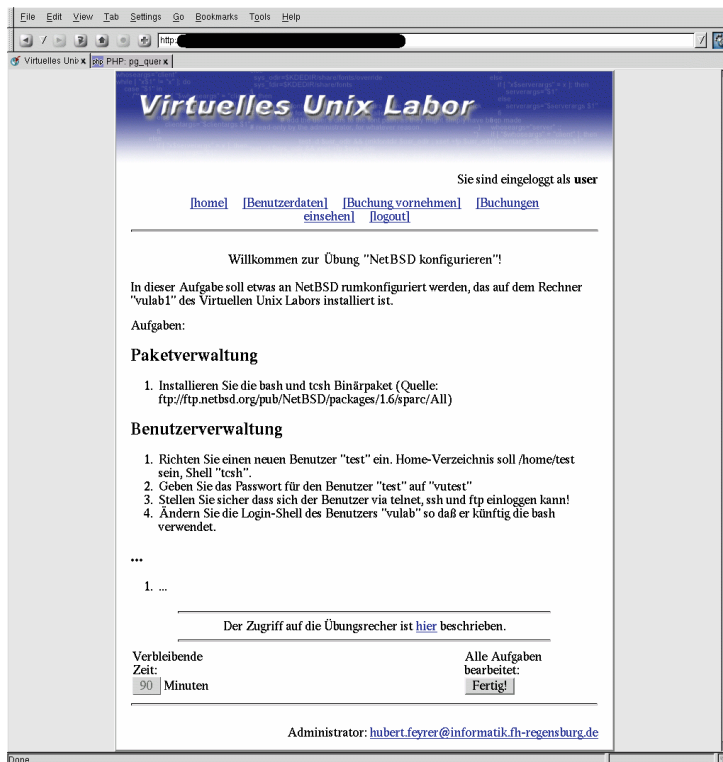


Figure 6.4: Exercise text with no associated checks, rendered in web browser

- **System interfaces:** Testing presence of facilities needed for certain tasks like file locking interfaces, thread availability, determining which interface to use e.g. for system installation and service configuration, packaging systems, startup system, etc.

Depending on the scenario and system environment that should be examined, primitives could be grouped according to their availability and scope:

- Primitives for **all systems**, e.g. TCP/IP networking, basic file attributes.
- Primitives for a **group of systems**, e.g. Unix/POSIX like systems and their specific services, or any Microsoft Windows version.
- Primitives for a **single system** only, e.g. Microsoft Windows in a specific version (95, 98, ME, 2000, XP), Novell, BeOS as well as specific Unix systems (Solaris, NetBSD) or Linux distributions (SuSE, Red Hat, Gentoo, Mandrake).

In the Virtual Unix Lab, the two exercises “Network Information Service” (NIS) and “Network File System” (NFS) were examined closer. A list of needed check primitives was identified and realized for each exercise.

For NIS, an overall number of 43 items to test was identified, see appendix A.4.1 for a full list. Some notable examples of checks needed for NIS are:

- Check if `domainname(1)` and `/etc/defaultdomain` are set on both machines.
- Do files like `/var/yp/Makefile`, `/var/yp/passwd`, `/var/yp-/binding/vulab/ypservers` and `/var/yp/passwd.time` exist on the NIS server?
- Does `ypwhich(1)` return the correct NIS server?
- Do `passwd-`, `host-` and `group-NIS-maps` contain the expected data?
- Is `/etc/nsswitch.conf` properly set up to search `passwd`, `group` and `host-information` in NIS?
- Is `/etc/rc.conf` setup to start `rpcbind` and `ypbind` on NetBSD?
- Are home directory and shell of the “ypuser” user set according to the exercise text on the NIS server?
- Is account information for the “ypuser” user provided properly via NIS?
- Is “ypuser” member of the group “benutzer”?
- Can the host “tab” be pinged (assuming the name is resolved via NIS)?
- Are `tsh` and/or `bash` installed on Solaris and/or NetBSD?

For NFS, 36 items to test were identified, see appendix A.4.2 for a full list. Among them:

- Is the `/usr/homes` filesystem exported properly in the file `/etc/dfs/dfstab` on the (Solaris) NFS server?
- Do the NFS server processes `rpcbind`, `mountd`, `nfsd`, `statd`, `lockd` run on the NFS server?
- Do the NFS client processes `rpcbind`, `rpc.lockd`, `rpc.statd` run on the (NetBSD) NFS client?
- Can the remote filesystem be mounted on the client manually and via `/etc/fstab`?
- Is the filesystem exported so clients can access it with root privileges?
- Does the user “nfsuser” exist on both machines, and does he own his home directory `/usr/homes/nfsuser` on both machines?
- Are files created on one machine accessible and owned on the other one?
- Are `tsh` and/or `bash` installed on Solaris and/or NetBSD?

**Database with web-interface to define checks:** Check scripts are stored on the Virtual Unix Lab master machine, which installs the lab machines, runs the check scripts for result verification, and also the web frontend for course management. All data on exercises is stored in various tables of the database, and the “`uebungs_checks`” table describes the connection between an exercise and a check. It uses the following information from appendix B.6:

- A unique identifier for the exercise, e.g. “nis”, “nfs”, ...
- Filename of the check-script, e.g. `check-domainname-set`
- Which machine to run the check-script on, e.g. `VULAB1`, `VULAB2` or `LOCALHOST` for the Virtual Unix Lab master machine
- A description of what the check-script does, to be printed when giving the user feedback about the exercise’s result, e.g. “Was `domainname(1)` set properly?”

Besides the data on which check to run (and on what lab machine), there was another table (“`ergebnis_checks`”) in the database that describes the checks’ results. Basically the check associated with the exercise and a boolean “success” value is stored for evaluation and feedback purpose.

Section 6.4.2, appendix A.1.1, the “preparation” in figure 6.5 and the “verification” phase in figure 6.7 contain more details on the database and the web frontend.

**Result verification engine:** This is the part where all the components are tied together: exercises, the checks as stored in the database, evaluation of the checks and storing the results. All this is done by the script `uebung_auswerten` which is described in more detail in the “verification” part of section 6.4.2.

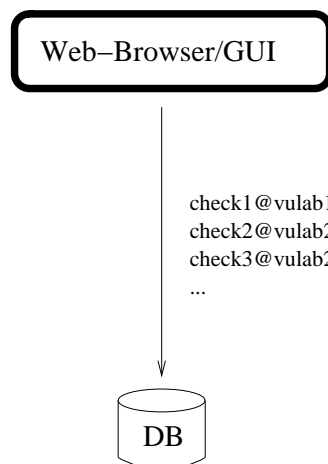


Figure 6.5: Step I: Preparation

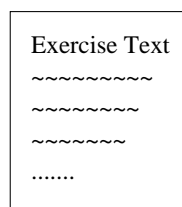


Figure 6.6: Step I: Exercise

## 6.4.2 Integration and interaction

This section describes how the components of the first implementation of the Virtual Unix Lab result verification architecture that were introduced in the previous section are integrated, and how they interact with each other. Figures 6.5, 6.6, and 6.7 give an overview of the exercise phases that were introduced in section 6.2.2, and which are involved in the process of result verification.

**Preparation:** Preparation of an exercise in the first implementation consisted of several parts:

1. Define the general parameters as of the exercise using the web interface shown in figures 6.8 to 6.10. Parameters include name and description of the exercise, duration, preparation- and post-processing time, name of the exercise file, and which harddisk image to use for installing each of the lab machines.



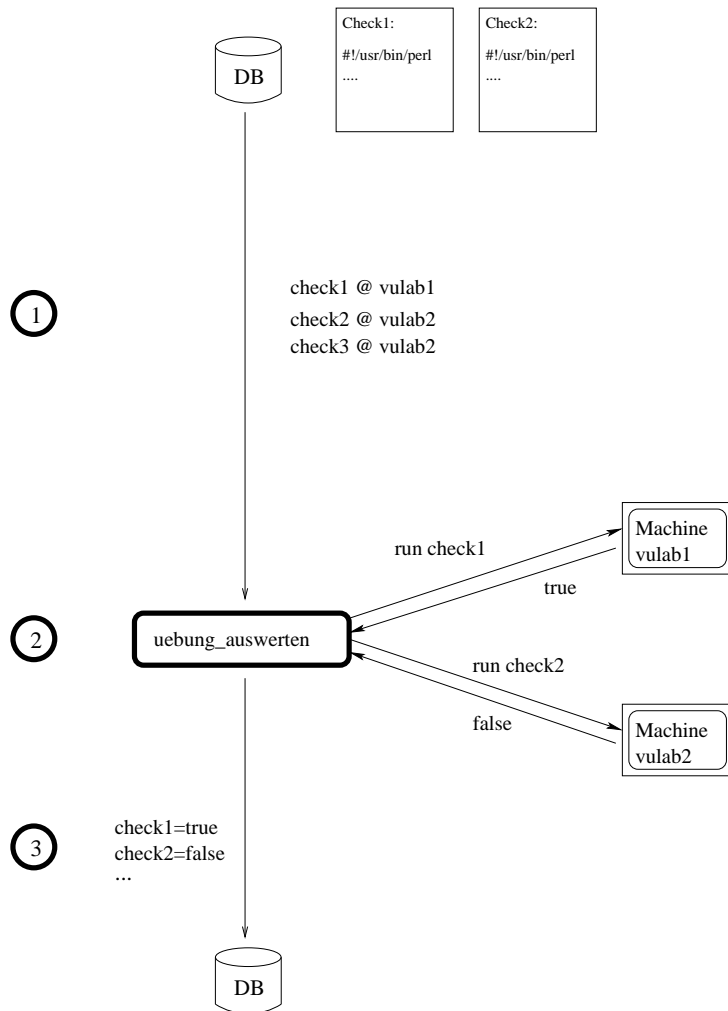


Figure 6.7: Step I: Verification

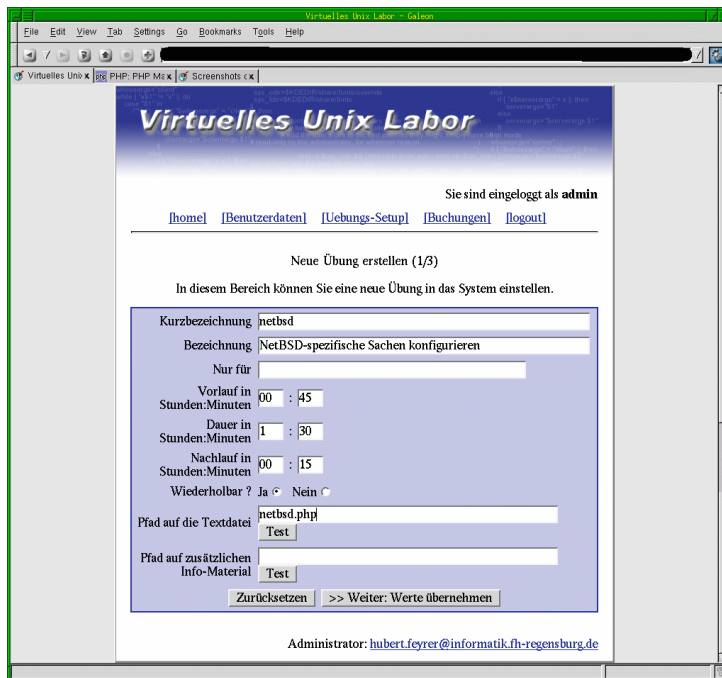


Figure 6.8: Defining an exercise, step 1: general properties

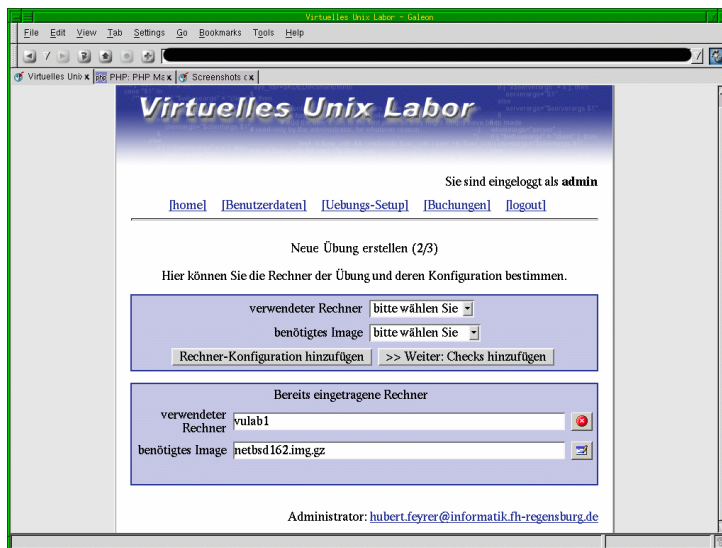


Figure 6.9: Defining an exercise, step 2: which image to deploy on which lab machine

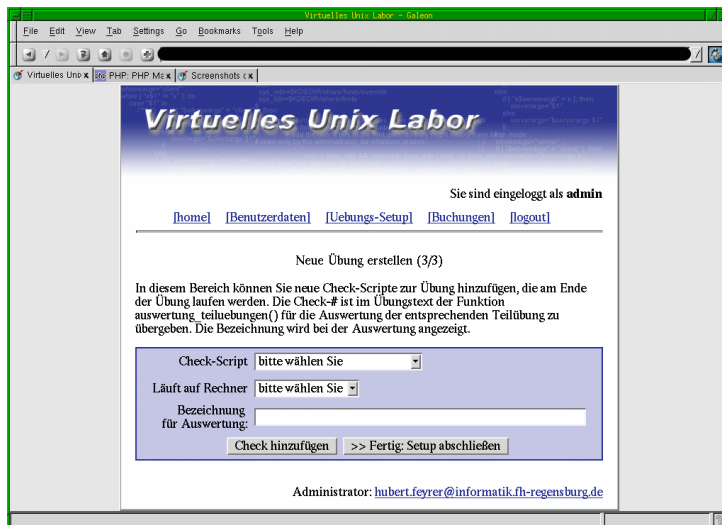


Figure 6.10: Defining an exercise, step 3: what checks to run on which machine

2. Write the exercise text as a HTML file, which is stored in the `public_html/texte` directory of the Virtual Unix Lab HTML code.
3. Determine which checks are needed to verify if the exercise is completed successfully, and write the corresponding check-scripts.

Which checks are needed depended on the task to be performed, as dictated by the exercise text and the way chosen to check if the task was performed correctly, see section 6.2.3. Also, if an already existing check handled a similar task, copying the corresponding check script and adjusting just one or two parameters usually gave a working shell script.

*Example:* The script to check if the shell of the “nisuser” from the NIS exercise existed was derived from the check that did the same for the “nfsuser” from the NFS exercise.

It became clear quickly that a way to parametrize the check scripts was needed, but this had to wait for the second implementation step of the Virtual Unix Lab, see chapter 9.

4. Associate the checks with the exercise in the database, using the web frontend.

After the check scripts were put into the `/vulab` directory on the master machine, the web frontend could be used to create entries in the database that described which checks to run on which machine after an exercise was completed, see section 6.4.1.

Although data entry via the web frontend is not very difficult, using the web frontend to enter 30-40 checks was tedious. This point was addressed in the second implementation, see chapter 9.

**Exercise:** After the exercise was written and stored on the Virtual Unix Lab master machine, nothing needs to be done at runtime. The PHP script that displayed the exercise text read the exercise text file, added HTML header and footer, and displayed it to the student.

**Verification:** At the end of the exercise, the verification process is started. The three steps involved in the verification process are illustrated in figures 6.5, 6.6, and 6.7:

1. Determine which checks to run on which lab machine.  
The database contained all the information about the checks that were part of the exercise's verification step, and what check script to run on which lab machine.
2. Get the lab machine to run the check script, and collect the result.  
The check script needed to be run either on the Virtual Unix Lab master machine or on one of the lab machines, as described in the database. Running the check on the master machine was easy, running it on the remote machine required the script to be transported to the remote machine first, then executed (by running it through the right script interpreter), and collecting the result of the test afterwards.
3. Store the check's result in the database.  
Following the requirements, the result was then stored in the database's "ergebnis\_checks" table for later retrieval, evaluation, and feedback.

**Feedback:** No feedback on the results stored in the database was realized for the first implementation of the Virtual Unix Lab, neither for users to query their individual results, nor for teachers to get an overview of the overall performance for each test. During the design of exercises it became clear that the combination of exercise in one file and definition of checks in the database was too hard to maintain when exercises needed adjusting during their design phase. As a result no real exercises were done on the first implementation of the Virtual Unix Lab that yielded any values to analyze.

Another major reason for moving towards the second implementation of the Virtual Unix Lab was that for analysis of individual exercises and giving feedback to students, doing so in the context of the single tasks of the exercise is much clearer than giving feedback without that context.

*Example:* An exercise consists of two tasks, A and B. The result of task A was verified by checks 1, 2 and 3, and task B was verified by checks 4 and 5. The first implementation of the Virtual Unix Lab did not allow giving feedback for checks 1–3 associated to task A and checks 4–5 associated to task B, but only a list of checks 1–5 and their results, without saying which check was associated to which task. This was of little use for users who wanted to learn from their errors. Figure 6.14 illustrates this.

### 6.4.3 Summary and suggested improvements

This section looks at what has been achieved in step I, and suggests improvements for step II. In the context of Domain Specific Languages, stereotypes have been identified and implemented via check scripts. They were a good starting point to continue. Initially, they were inflexible, as different scripts were required to verify the results of different but similar tasks. Flexibility was improved by providing a set of check scripts for basic operations, plus adding a parameter passing mechanism.

Beyond stereotypes for result verification, no real “language” has been defined at this point that embeds the stereotypes as activators for the specified actions. In the context of the Virtual Unix Lab, this became obvious as there was no direct connection between the exercise text and the verification steps carried out by the check scripts. The loose coupling of exercise text and checks was not enough for giving detailed feedback as described in the “Analysis” part of section 6.4.2. This can be improved by embedding activators for the check scripts into the exercise text. With a close coupling between exercise text and checks, it is possible to give feedback on parts of the exercise, telling the student which parts of the exercise were solved successfully, and which were not.

All the above points are addressed in step II of the Virtual Unix Lab.

## 6.5 Step II: Instructions and checks coupled

This section describes the version of the Virtual Unix Lab that addresses the issues identified in step I, and that was used for evaluation in section 7. This section describes how check scripts were improved, and how coupling of exercise text and checks was achieved by creating a domain specific language. This also allows to give elaborated feedback, and it allowed creating a system front-end with check primitives. For these aspects, integration and interaction with the existing system are illustrated.

### 6.5.1 Improved check primitives

The check scripts that implement the verification primitives were improved in several ways. To be of more general use, the checks were implemented using one common language and framework for all scripts. Common tasks were identified and expressed in more generic scripts, which were taught how to handle parameters, to accommodate the scripts to the specific tests. The following items describe the changes in detail.

**Rewrite all check scripts in Perl:** Implementing check scripts in the Bourne shell is fast if the shell supports the task intended to be performed, or if there is an

external program to do the verification and report back success or failure. While this is possible for a number of tests, the shell does not provide internal methods to test system specific items. Relying on external programs is problematic as these commands may or may not be present between different systems, they may be named differently, or have different calling conventions<sup>1</sup>.

An alternative to avoid those problems with portability and interoperability is to use another language for implementing the check primitives. Section 5.4 discusses possible options, and with the given requirements, Perl is a good candidate: it is available on many platforms, and also provides many operating system specific interfaces.

For step II of the Virtual Unix Lab, all check scripts were rewritten in Perl. At the same time, their names were changed to indicate their scope of applicability, i.e. if they can be used on all systems, on Unix systems, or only on specific Unix(like) systems, by giving them common filename prefixes. See appendix A.5 for examples.

**Extend check scripts to handle parameters:** Check scripts often verify similar results. For the first implementation, scripts were often copied, and similar items – filenames, text patterns, etc. – were changed. Sometimes the components likely to be changed were even put into internal variables that were set at the start of the script, and that were the only parts of the scripts that needed changing. While it was an improvement that not the whole script had to be understood when deriving a new test from an existing one, copying the script was still necessary with all the drawbacks. Those drawback included the need to understand internals of the Virtual Unix Lab, and redundant maintenance effort when the core part of the check script had to be changed, e.g. for feature improvements or bug fixing. As such, it quickly became clear that a method to pass these parameters to the check script would be of benefit.

Passing parameters into a check script involved several of the Virtual Unix Lab components: Besides the check scripts that needed changes to accept parameters, the parameters had to be stored in the database. The web based interface to store and edit the check data in the database had to be adjusted, and an interface had to be defined to pass the parameters from the database to the scripts when running it. Furthermore, an interface was introduced to query a check script for its purpose and the parameters it supported.

To realize this, the following changes were made:

- **Check scripts:** All check scripts were changed to use Perl as the only language, as described above. At the same time, a framework was introduced to make passing and querying parameters more easy.
- **Database:** The `uebungs_checks` table (see appendix B.6) was extended by a “parameter” field to store an arbitrary string that can then be processed lexically into single parameters and arguments.

---

<sup>1</sup> [Mayer, 2001] pp. 24

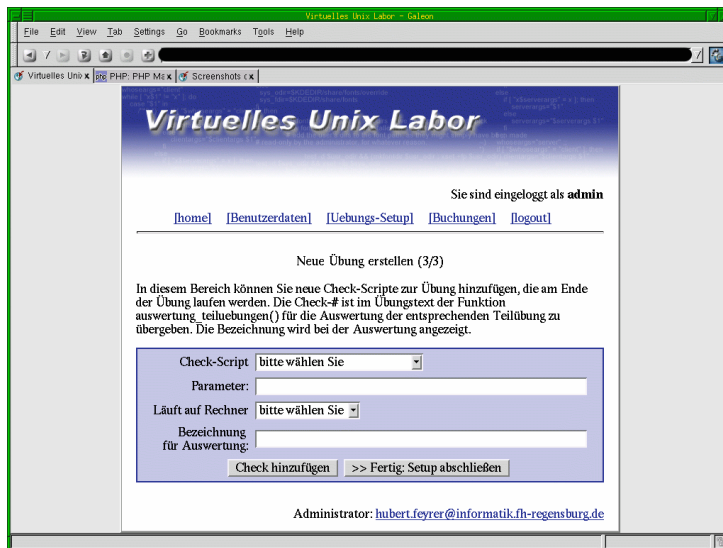


Figure 6.11: Extended web interface to enter parameters for check script

- **Web interface to database:** The web interface realized in step I was extended by a field for a string of parameters that was stored into the database as described above. See figure 6.11 for a screenshot.
- **Parameter-passing interface:** As the check script is ran either on the Virtual Unix Lab master machine or copied to one of the lab machines and executed there, the parameters need to be passed. From several possible ways (passing on as command line arguments, via a file descriptor like standard input, or as environment variable), passing as environment variable was chosen.
- **Interface to query possible parameters:**

Check scripts were extended by a querying interface to retrieve their general purpose, a list of possible parameters, and their description. This information is displayed in the web-based user interface for entering and changing check scripts. Figure 6.12 shows a choice of available check scripts (read from the harddisk's /vulab directory), a description the parameter(s) of the `unix-check-user-shell` check script is shown in figure 6.13.

**Improvements of check scripts:** The check scripts used in step I of the Virtual Unix Lab only tested one aspect of the system. Testing two similar aspects following the same concept required two separate shell scripts. Following the description of Cocke and Schwartz, the check scripts were improved to provide “indicative subpatterns” to be embedded into exercise texts. These “subpatterns” provide the “contextually implied information”, i.e. they act as a collection of subroutines

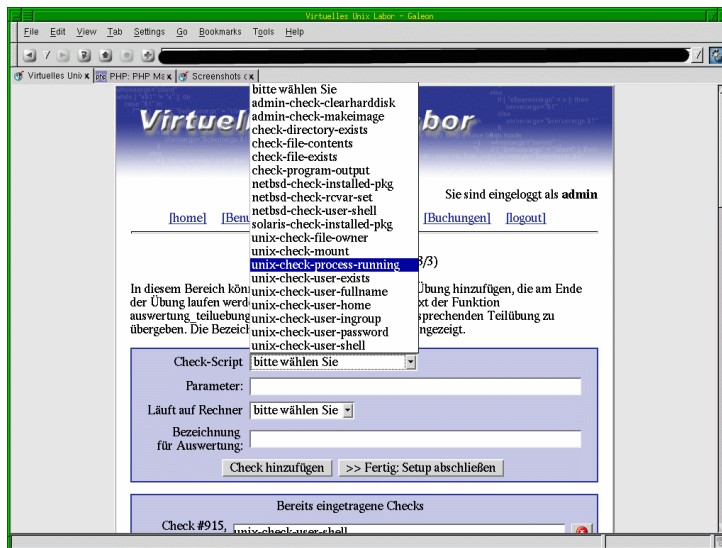


Figure 6.12: Listing existing checks

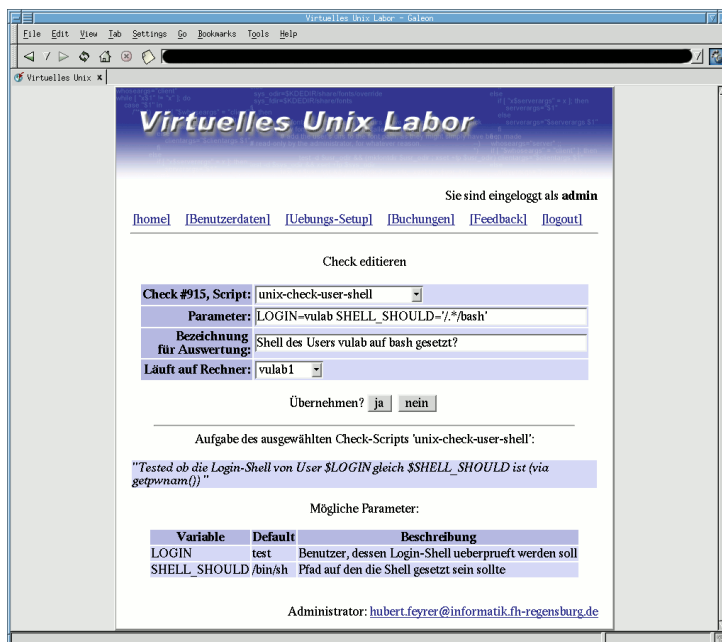


Figure 6.13: Possible parameters of a check script, and their description



that can be called for specific check tasks when needed, the context is defined by parameters<sup>1</sup>.

Improvements made were:

- Output of programs and contents of files were first compared against fixed values, the final check scripts then compared against patterns directly.

*Example:* The following transition was made from a check script that checked a program's output to determine the contents of a file into one that checked the contents of a given file. In effect, the scope of the script was thus narrowed, and the complexity of parameters was reduced.

Before:

Script: `check-program-output` (see appendix A.6.1.6)

Parameters:

```
PROGRAM='grep "^PWDIR.*.**/var/yp" /var/yp/Makefile | wc -l'
OUTPUT_SHOULD=1
```

After:

Script: `check-file-contents` (see appendix A.6.2.3)

Parameters:

```
FILE=/var/yp/Makefile
CONTENTS_SHOULD="'^PWDIR.*.**/var/yp"'
```

- Specific checks were implemented to test various attributes of user accounts, e.g.
  - if a user account exists at all<sup>2</sup>
  - if a user has a certain login shell<sup>3</sup>
  - if a user has a certain password<sup>4</sup>
- Output of the same program was compared against different values. Passing these values by parameters made it possible to use the same check script to test various aspects in different exercises, by passing different parameters.

*Example:* The script `unix-check-process-running` (see appendix A.6.2.7) was designed to take a process name as parameter, and check if the named process is running. The following database query shows the places where this is used:

```
vulab=> select distinct uebung_id, bezeichnung, parameter
vulab-> from uebungs_checks
vulab-> where script='unix-check-process-running';
  uebung_id | bezeichnung | parameter
-----+-----+-----
  nfs      | Lauft lockd? | PROCESS=lockd
```

<sup>1</sup> [Cocke and Schwartz, 1970] pp. 10

<sup>2</sup> See the `unix-check-user-exists` script in appendix A.6.2.4

<sup>3</sup> See the `unix-check-user-shell` script in appendix A.6.2.5

<sup>4</sup> See the `unix-check-user-password` script in appendix A.6.2.6

```

nfs      | Läuft mountd? | PROCESS=mountd
nfs      | Läuft nfsd?   | PROCESS=nfsd
nfs      | Läuft rpc.lockd? | PROCESS=rpc.lockd
nfs      | Läuft rpc.statd? | PROCESS=rpc.statd
nfs      | Läuft rpcbind? | PROCESS=rpcbind
nfs      | Läuft statd?   | PROCESS=statd
nis      | rpcbind läuft? | PROCESS=rpcbind
nis      | ypbind läuft?  | PROCESS=ypbind
(9 rows)

```

- Checks that queried specific system databases by using a program and checking its output were changed to get the fields to query only, and return if the field is set.

*Example:* Checks existed to test if a certain service was started in the boot process of NetBSD, as defined in the `/etc/rc.conf` file. They were implemented by looking for a certain pattern in that file. These checks were replaced by a script that only specified the field to query, and that either returned success or failure, whether the field was set or not.

Before:

Script: `check-file-contents` (see appendix A.6.2.3)

Parameters:

```

FILE=/etc/rc.conf
CONTENT_SHOULD='^rc_configured.*.*[Yy][Ee][Ss]'

```

After:

Script: `netbsd-check-rcvar-set` (see appendix A.6.2.8)

Parameters:

```

RCVAR='rc_configured'

```

Similar checks that are specific to one operating system can be written e.g. for the Irix startup system's database or the Windows registry.

As a summary, the initial set of task-specific check scripts was changed into a set of check scripts that are more general. Parameters can be given to the scripts to specify which aspects of the specific subsystem to examine closer.

## 6.5.2 Coupling of exercise text and checks

The example at the end of section 6.4.2 illustrates one of the problems in step I of the Virtual Unix Lab: Giving useful feedback after an exercise was not possible. Exercise text and check scripts were completely separated, and as a result it was not possible to tell the student which of the exercise's tasks were solved successfully, and which were not. To give detailed feedback on each task of the exercise, an association needs to be made between the textual description of a specific task of the exercise, and the check(s) that verify the results of that task.

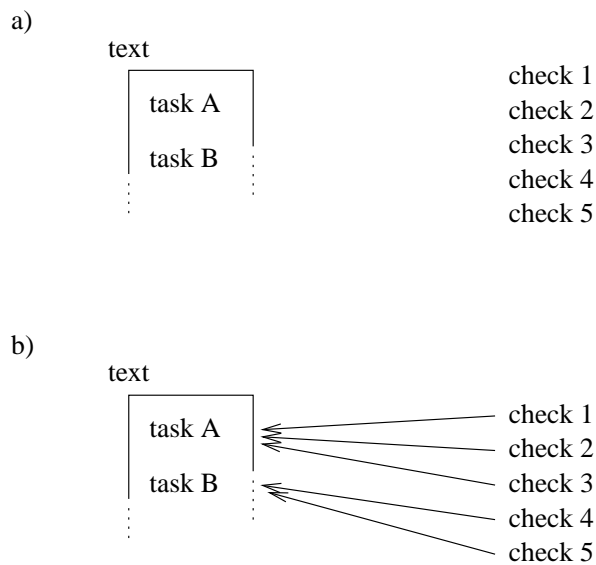


Figure 6.14: Exercise text and checks: a) uncoupled in step I, b) coupled in step II

Figure 6.14 a) illustrates the uncoupled exercise text and checks used in step I of the Virtual Unix Lab, while figure 6.14 b) shows the coupling realized in step II.

### 6.5.2.1 Options

Several ways for coupling exercise text and checks are possible:

- Split the exercise text into single tasks, associate the tasks with the exercise, and the checks with tasks:

exercise ← task ← checks

instead of the

exercise ← checks

association used so far, in effect splitting a “big” exercise into several smaller ones. Instead of writing one HTML file for the whole exercise text, several files would be needed, one per task.

The maintenance impact of this was considered to be too much of an issue to realize this solution.

- Write check code inline into exercise text, i.e. mix the HTML/PHP code used to describe tasks with code to check the results.

There are a number of problems involved in this solution. First, putting the code for the tests directly into the exercise text would (re)create redundancy of code that was removed with the work to use generic check scripts. Second, an association between the result of the check and the check must still be made to give feedback for a particular exercise to the user and say if the exercise was solved successful or not. For this association, an identifier is still required that maps between the check and its result. Third, as the code needs to be executed either on the master or one of the lab machines, extracting the code from the exercise text would be necessary, either at test time without storing the verification code anywhere, or with extracting it once and storing it into a place that would need management. This would make the whole exercise handling complex and complicated.

- The last alternative for coupling exercise text and checks is a hybrid version of the above solutions: The exercise text is augmented with “calls” or “activators” to check scripts. The check script is stored separately, and the result of its execution is stored in the database for later retrieval using a unique identifier for the (booked exercise, check number) combination.

The advantages of this approach is that the code doing the verification itself can be abstracted in the check scripts as described above. Results can be stored and retrieved for feedback by giving them a unique identifier for the check and one for the booked exercise, and the existing verification engine can be used unchanged.

From these three approaches, the last one was chosen to implement coupling of exercise text and checks for step II of the Virtual Unix Lab. Reasons for this decision are that it allows to reuse existing code from the check scripts and the result verification engine, is has a low overhead on maintenance, and that it keeps the abstraction between check script stereotypes and their implementation.

The information needed for a single check to run are the check script name, any possible parameters, which host to run it on as well as a textual description what the check does, for giving user feedback. Also, an identifier for the particular check is needed to identify the result of the particular check for a particular exercise taken.

In step I, the main tasks for creating an exercise were writing of the exercise text and associating checks with the exercise using the web frontend. Using the web interface for a large number of checks (43 for the NIS exercise, 36 for the NFS exercise) was not practical, and it also split the exercises’ parts into two places. This split made it hard to maintain a full overview over a particular exercise and all the associated checks, and as a result exercise maintenance was hard. To solve this problem, integrating check data into the exercise was chosen following the “Data Structure Representation” pattern for domain specific languages described in section 5.3, with an additional twist.

### 6.5.2.2 Data structure representation

So far, the per-check data – check script name, parameters, which host to run on and a textual description for the user feedback – was stored in the `uebungs_checks` table, filled in using the web interface. The idea for improvement was to place this data into the exercise text, to keep check data near the exercise text.

That is, the idea was to have something like this in the exercise text:

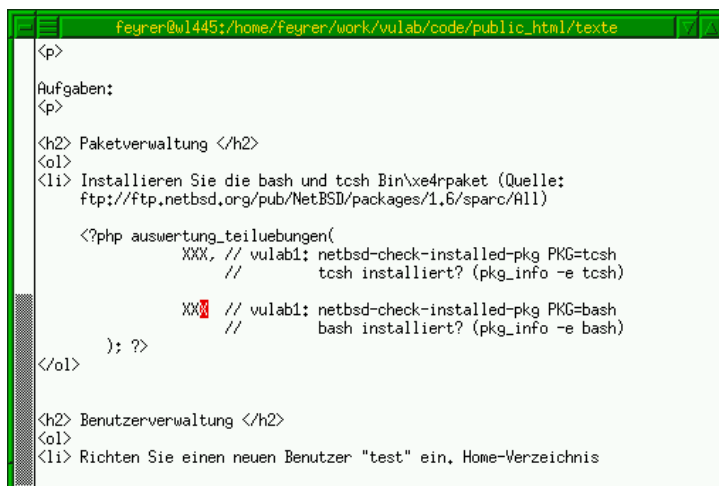
```
1. Perform some task on host vulabl with parameters x and y.
  // Check 1: run check-task-done (no parameters) on host vulabl
  //           Feedback: ``Was the task performed successfully?``
  // Check 2: run check-task-parm (PARM=x) on host vulabl
  //           Feedback: ``Does the task use parameter x?``
  // Check 3: run check-task-parm (PARM=y) on host vulabl
  //           Feedback: ``Does the task use parameter y?``
```

This example first describes the task to the user in textual form, then contains some comments to indicate the check data. It uses “//” as an indicator for the processing engine to not include the check data when displaying the exercise text to the user. While preventing the displaying of the check data was possible by using HTML or PHP comments, there were two problems given with this approach. First, how to extract the check data for running when the exercise is over, and second, how to display the feedback to the user.

The first problem was solved by a processor that realizes the “data structure representation” pattern from section 5.3. This processor extracts the check data from the exercise text and stores it into the database’s `uebungs_checks` table. The check data can be stored as comments in the exercise text, and then get extracted into the database, where it can be edited using the existing web interface, and executed by using the existing result verification engine as described in section 6.4. Effectively, using a processor to extract the data from the exercise text into the database prevented a need to change the whole result verification engine. It still allowed using the existing web interface to edit checks when needed, and most important, keeping all the data for an exercise in one file. This was considered a key item for keeping maintenance of exercises manageable. The processor is described in more detail in section 6.5.5.

What is not described so far is how feedback for the user is given after an exercise was taken, which leads to the second problem and the “twist” mentioned before.

To give feedback after the exercise, hiding the check data as passive comments in the exercise text as displayed in the above example is not possible. An active component needs to remain, which acts as an activator to display the comment stored for feedback purpose (“Was the task performed successfully?”, ...) as well as an indicator for success or failure of the exercise, retrieved from the database. A unique identifier for each check was needed, and the check ID stored in the “`check_id`” field already present



```
Feyrer@w1445:/home/feyrer/work/vulab/code/public_html/texte
<p>
Aufgaben:
<p>
<h2> Paketverwaltung </h2>
<ol>
<li> Installieren Sie die bash und tcsh Bin\xe4rpaket (Quelle:
ftp://ftp.netbsd.org/pub/NetBSD/packages/1.6/sparc/All)

  <?php auswertung_teiluebungen(
      XXX, // vulab1: netbsd-check-installed-pkg PKG=tcsh
          // tcsh installiert? (pkg_info -e tcsh)

      XX, // vulab1: netbsd-check-installed-pkg PKG=bash
          // bash installiert? (pkg_info -e bash)
  ): ?>
</ol>

<h2> Benutzerverwaltung </h2>
<ol>
<li> Richten Sie einen neuen Benutzer "test" ein. Home-Verzeichnis
```

Figure 6.15: Example exercise text with check data

in the `uebungs_checks` table fits this purpose. Finally, deciding whether to display feedback or not was handed over to a PHP function, which can be defined to hide the feedback before and during the exercise, and show it when the user requests feedback after the exercise was taken. So as a twist, feedback is given or not depending on a single function definition, which is different before/while and after the exercise, see section 6.5.3.

The final exercise design consists of the exercise text being written in HTML text with PHP functions included that control printing of evaluation as well as feedback text stored as PHP comment. This gives the data for the associated check, and is stored in the database by the `uebung2db` processor.

An example exercise text is displayed in figure 6.15, appendix A.2 lists full exercise texts for the NIS and NFS exercises as used for evaluation of the Virtual Unix Lab described in chapter 7.

### 6.5.2.3 Forming a domain specific language

Given the above design, exercise texts contain a number of details. First, the checks, its parameters, and which lab machine to run it on are extracted from comments in the exercise text, and stored into the database by the `uebung2db` processor. This realizes the “data structure representation” pattern described in section 5.3 and by Spinellis<sup>1</sup>. Second, information on the check script names are given in the comments, which

<sup>1</sup> [Spinellis, 2001] p. 96

acts as calls to functions. The functions are defined by the check scripts as discussed in section 6.5.1. Last, the exercise text is augmented with PHP function calls that give feedback depending on the context that the functions are called in. This forms a domain specific language which is further referred to as “Verification Unit Domain Specific Language” (VUDSL).

Details on the PHP functions as well as the feedback they allow is given in section 6.5.3, and a summary of the Verification Unit Domain Specific Language can be found in section 6.6.

### 6.5.3 Giving feedback

Giving proper feedback on what tasks of an exercise where solved successfully and which were not was one of the primary design goals of step II of the Virtual Unix Lab. Given the exercise design described in the previous section, it was easy to realize giving different feedback to single users and teachers.

Key elements for giving feedback are the PHP functions embedded into the exercise text as shown in figure 6.15. They controll what checks are printed if feedback is requested:

- `auswertung_ueberschrift()`
- `auswertung_teiluebungen()`
- `auswertung_zusammenfassung()`

For presenting the exercise text to the user when previewing and while taking the exercise, these functions do not print anything at all.

To give feedback for a user after the exercise, these functions are defined differently. While `auswertung_ueberschrift()` and `auswertung_zusammenfassung()` give general information including a header and footer for the exercise, the main work is done by `auswertung_teiluebungen()`. The function takes a list of check IDs, and it retrieves and prints the corresponding textual description of the check (from the “bezeichnung” field of the `uebungs_checks` table) as well as the result of the check (stored in the “erfolg” field of the `ergebnis_checks` table). Figure 6.16 shows a screenshot of feedback for a single user.

While single users are only interested in their own performance, teachers are interested in statistics on how the whole study group performed. Step II of the Virtual Unix Lab allows users with admin privileges to retrieve such information for all users. Using the named PHP functions, a routine can be added for `auswertung_teiluebungen()` to print the following information in addition to “normal” user feedback:

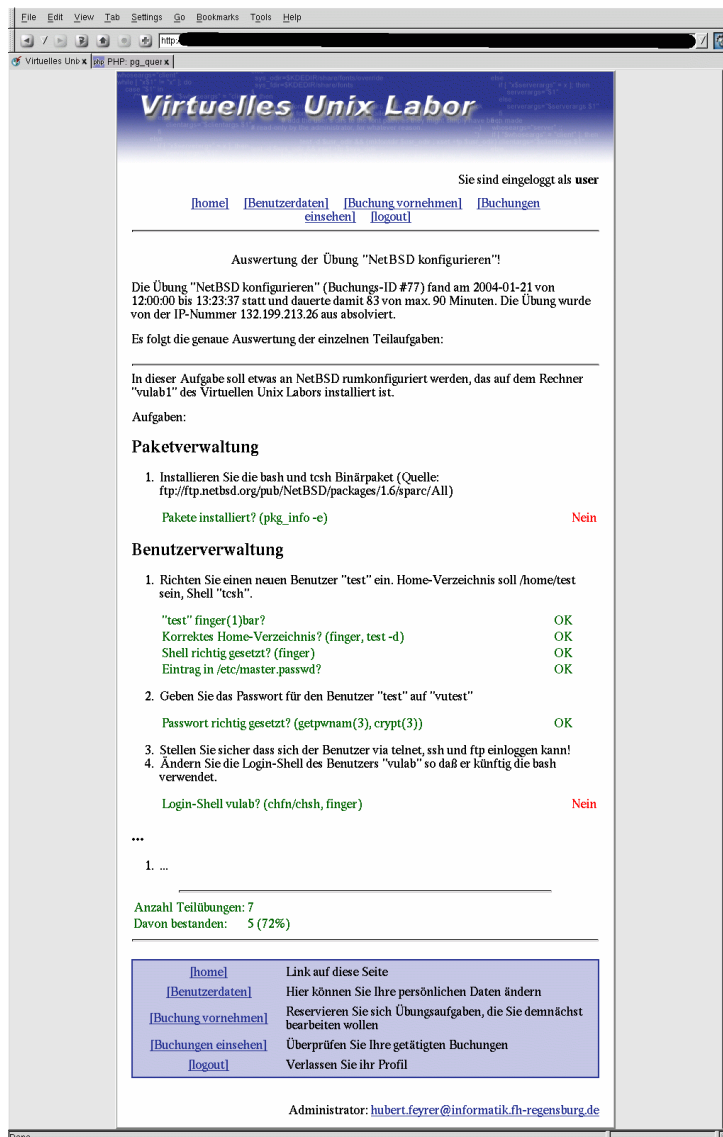


Figure 6.16: Giving feedback on an exercise for a single user



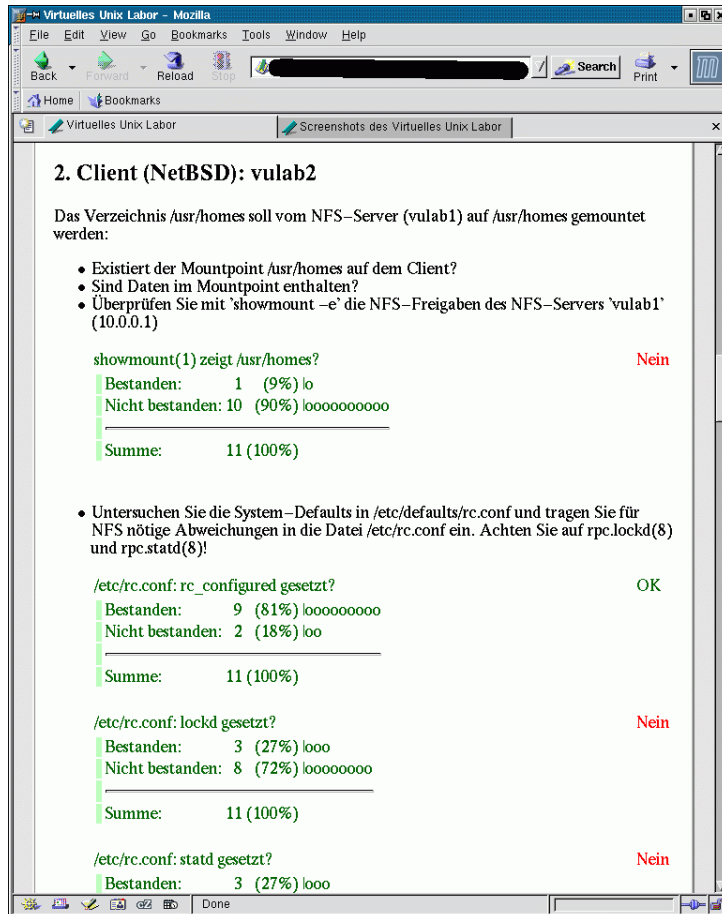


Figure 6.17: Giving teacher/admin feedback for all users which took an exercise

- The number of students who solved the task successfully, both by count and percentage
- The number of students who did not solve the task successfully, again given by count and percentage.
- The count of students who took the exercise, and either succeeded or failed.
- A bar of “o”s is printed besides each result, with one “o” representing one student. This allows to get a quick visual overview of the results.

Figure 6.17 displays an example of feedback on how all users performed on a certain exercise, with all the data mentioned above.

Many other ideas for giving feedback to single users and esp. analysis of performance of whole groups of students are possible. For a first overview, the ones implemented in step II of the Virtual Unix Lab were considered adequate.

### **6.5.4 Creating a system front-end with check scripts**

Besides exercise texts, the other important part of an exercise is the machine setup provided for an exercises that users start with. This machine setup is stored in form of harddisk images that are written to the lab machines’ harddisks before the exercise starts as described in section 4.3. Sometimes a harddisk image needs to be updated or newly created. The process to update/create an image is to first install an existing image or install the machine from CDROM, then store the machine’s harddisk image to an image file that can then be used for exercises as shown in figure 6.9.

Before step II, the process of creating or updating a harddisk image was done manually by first preventing any exercises from being taken for some time (by disabling logins in the Virtual Unix Lab), then – when updating an existing image – issuing the command to deploy an existing image manually, or installing a machine from CDROM and configure it so that its configuration can be used for the exercise in mind. After that, the machine had to be shut down and netbooted. From the netboot environment, the harddisk image was taken and written to the Virtual Unix Lab master machine via NFS. After entering the newly created image file into the `images` table with an appropriate SQL statement, the new/updated image was ready to be used in newly created exercises.

The process where these manual steps were used for updating the NetBSD client image from NetBSD 1.5.2 to NetBSD 1.6.

This process is tedious, prone to errors, and many internal details of the Virtual Unix Lab need to be known. Looking closer at the process, most of these steps can be done automatically easily though: The normal exercise system can be used to book a

certain “admin-type” exercise. It will prevent users from interrupting the process, and also install a predefined image on the lab machine for updating (if wanted). Normal users are prevented from booking the exercise by entering the administrator’s login name into the “Nur für” (only for) field so only the named administrator can book the admin-type exercise. The exercise text file has to exist, but can be empty. See figure 6.18 for an example setup.

When the exercise time arrives, the machine will be prepared, and instead of doing a predefined exercise, the administrator changes the client machine as needed.

The verification of the exercise results consists of two special check scripts that will care to do the postprocessing, `admin-check-clearharddisk`<sup>1</sup> and `admin-check-make-image`<sup>2</sup>. The first script cleans up any unused space on the lab machine’s harddisk and prepares it to be better compressible<sup>3</sup> while the second script does all the real work of shutting down the lab machine, taking precautions so a netboot will create a harddisk image in a given file, perform the netboot, wait until the image file is created, and storing the newly created image’s filename in the `images` table. Image 6.20 shows an example setup in the web user interface.

Figures 6.18, 6.19, and 6.20 show the steps of an example exercise setup that was used to update the Solaris image for some minor changes. Important items to notice are:

- The exercise can only be booked by one user, “admin”, as that login name is entered in the “Nur für” (only for, see image 6.18 field of the first mask
- The exercise text in the “Pfad zur Textdatei” (path to exercise text file, see image 6.18 must exist so it can be displayed. As it is expected that the admin taking the “exercise” knows what he wants to change, not much text needs to be put there, and the file can be empty as well.
- Both machines have “benötigtes Image” (required harddisk image, see image 6.19 set so that they get Solaris installed. Only one machine will be modified and taken an image from, but the other one will be useful for reference, so both are given the same default installation.
- The `admin-check-clearharddisk` check script is started on the `VULAB1` lab machine to clean up the harddisk before generating an image. This will result in a smaller image, as it is expected that there is unused random data left on the disk which would prevent optimal compression of the harddisk image.
- The `admin-check-makeimage` check script is run on `LOCALHOST`, i.e. the Virtual Unix Lab master machine and not on one of the lab machines. This is required because the script needs to access the netboot area and the database,

---

<sup>1</sup> See appendix A.6.2.1

<sup>2</sup> See appendix A.6.2.2

<sup>3</sup> [Feyrer, 2007b] Section “5.10 Reducing the image size”

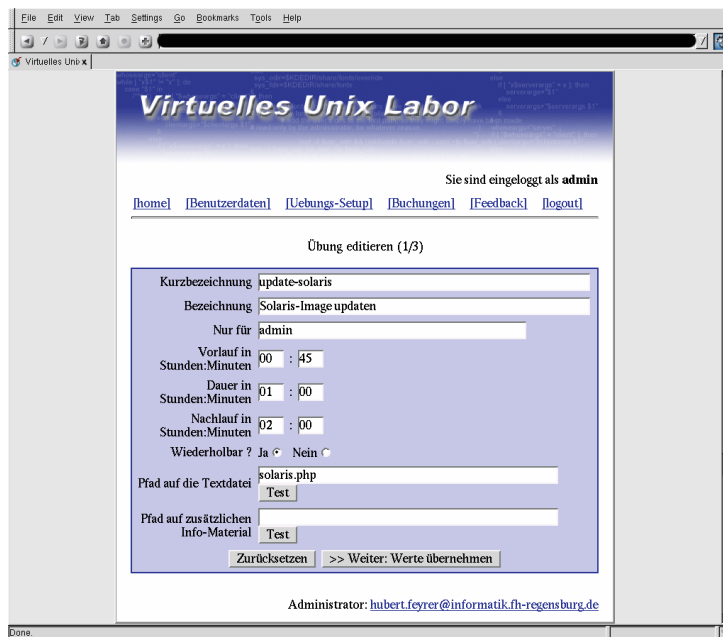


Figure 6.18: Defining an admin-only exercise to update the Solaris image, step 1: only “admin” may book

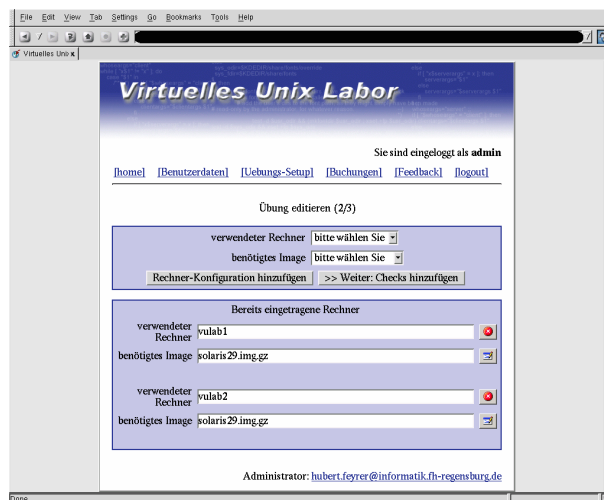


Figure 6.19: Defining an admin-only exercise to update the Solaris image, step 2: Solaris will be preinstalled

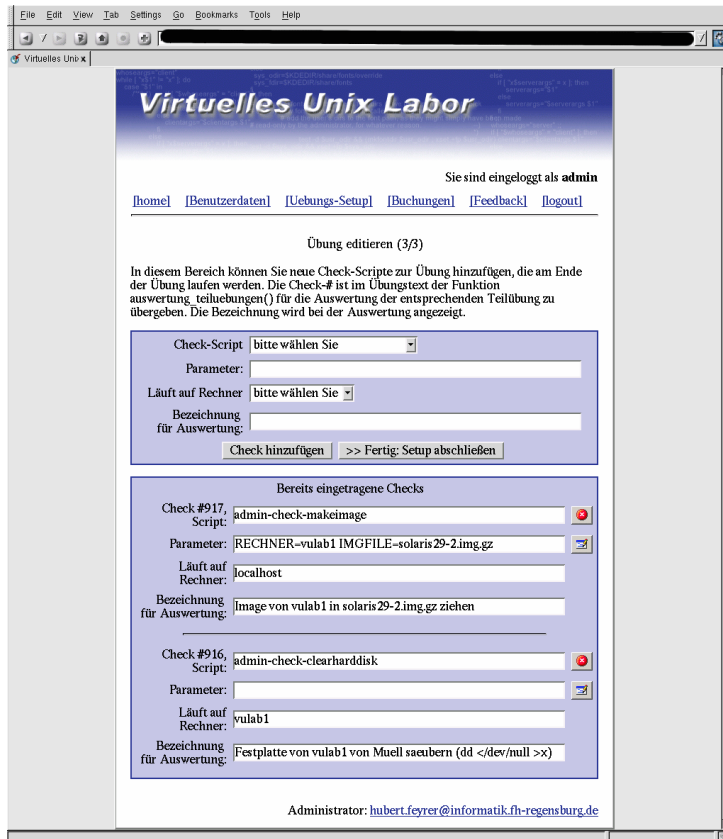


Figure 6.20: Defining an admin-only exercise to update the Solaris image, step 3: the disk will be cleaned and put into an image file after the exercise

which it can not do from a lab machine. Parameters are passed to tell the script from which machine's harddisk the image should be used (`RECHNER=vu-lab1`), and in which file in `/vulab` to store it on the master machine (`IMG-FILE=solaris29-2.img.gz`).

- The check scripts are run in order of their ascending number, which will assure that first the harddisk is cleared before it is put into an image.

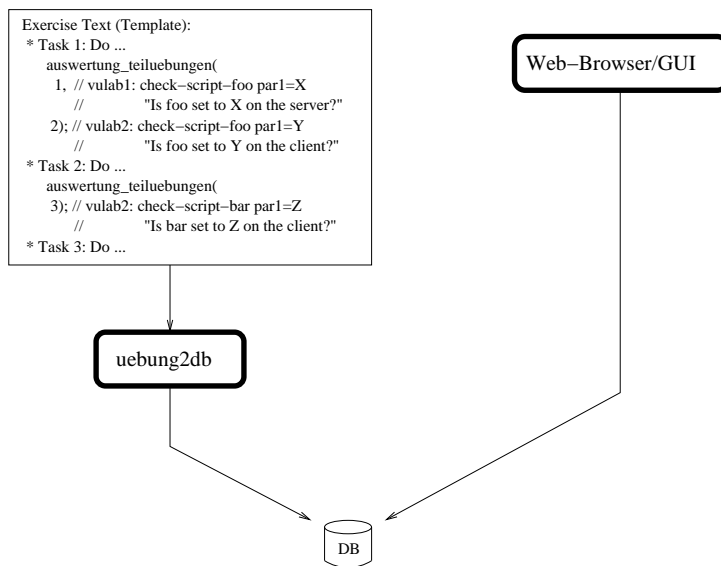
In summary, it was possible to add a high-level interface for modifying an important part of an exercise by adding two special check scripts – a task which required intimate knowledge of the design and implementation of the Virtual Unix Lab before, and which was greatly simplified that way. As such, the newly created facility can be viewed as a system front-end following the corresponding “System Front-End” pattern described in section 5.3 as well as in [Spinellis, 2001, pp. 97] and [Mernik et al., 2005, p. 323].

### **6.5.5 Integration and interaction**

So far, all the major new components and features of step II of the Virtual Unix Lab were described. This section goes into detail how all the components from both step I and step II fit together, and how they interact to provide the result verification architecture of the Virtual Unix Lab. Figures 6.21, 6.22, 6.23, and 6.24 give an overview of the various components involved as well as their interaction.

**Preparation:** Creation of an exercise in step II of the Virtual Unix Lab is similar to step I with a few changes in detail:

- Define the exercise with its general parameters by using the web frontend from step I as displayed in figures 6.8 to 6.10. The definition of check scripts in the third screen (shown in figure 6.10 can be left empty, as the initial set of checks will be derived from the exercise text.
- Write the exercise text in HTML as in step I, and add hints for result verification and giving feedback embedded as comments comes next. See section 6.5.3 for details and examples. As the check-numbers which identify each individual check and which are given to the `auswertung_teiluebungen()` PHP functions are not known when writing the exercise, “xxx” should be put in as a placeholder, which will be filled in automatically later. See figure 6.25 for an example.
- After writing the exercise text, the hints are extracted into the database by running the script `uebung2db` as shown in figure 6.26. Parameters given on the command line are the exercise name, the filename containing the exercise text, and a filename which will contain the exercise with the check-numbers filled in.



check 1: check-script-foo(par1=X) @ vulab1,  
"Is foo set to X on the server?"  
check 2: check-script-foo(par1=Y) @ vulab2,  
"Is foo set to Y on the client?"  
check 3: check-script-bar(par1=Z) @ vulab2,  
"is bar set to Z on the client?"  
...

Figure 6.21: Step II: Preparation

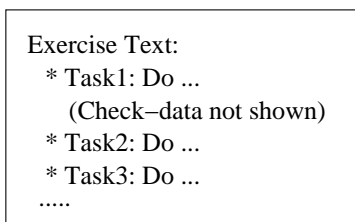


Figure 6.22: Step II: Exercise

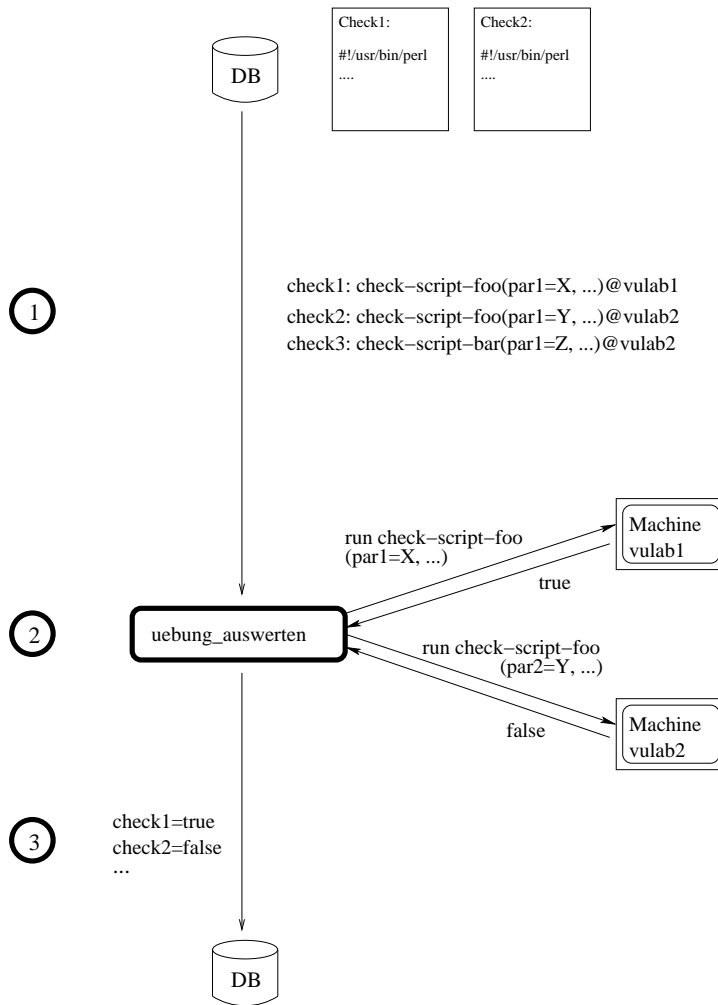


Figure 6.23: Step II: Verification



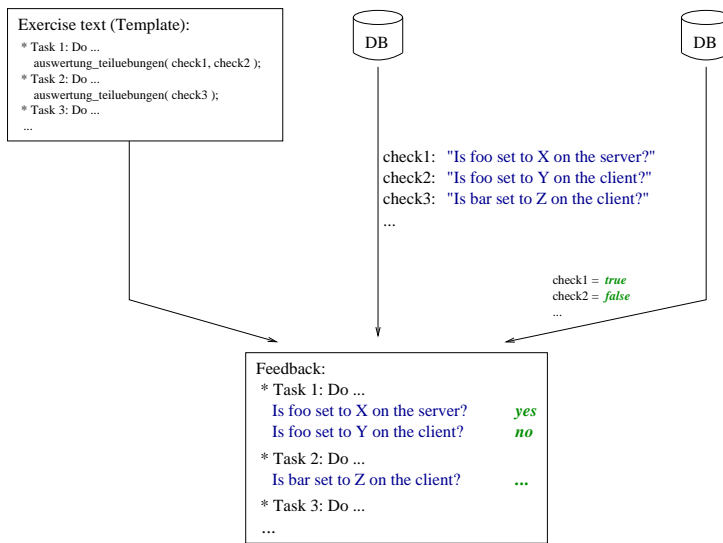


Figure 6.24: Step II: Feedback

At this point, the database's `uebungs_checks` table is filled with the checks from the exercise text's comments, and the second file given to the "uebungs2db" call will contain the original file's contents with the check numbers filled in from the `uebungs_checks` table for the "XXX"s. Figure 6.27 illustrates the difference between the original and the updated exercise text using the `diff(1)` output format.

- After the updated exercise text has been reviewed, it needs to be put into the place where the Virtual Unix Lab expects exercise texts to be stored. The corresponding subdirectory is `public_html/texts`, the file name is given in the web user interface's "Pfad auf die Textdatei" (path to text file) field.

Optionally, the updated exercise text can be committed to a content management system (CMS) like the CVS repository used for development of the Virtual Unix Lab, as shown in figure 6.28.

After these steps – define general properties, write exercise text, extract data from the exercise text into the database, move updated exercise text into place – the exercise is prepared, and it can be used for exercises by students.

**Exercise:** The exercise text is stored in a HTML file with calls to PHP function `auswertung_ueberschrift()`, `auswertung_teiluebungen()` and `auswertung_zusammenfassung()` as described in section 6.5.3. When displaying the text during the exercise, only the plain text and no feedback on success is displayed. This is achieved by signalling the PHP functions to not

```
Feyrer@w1445:/home/feyrer/work/vulab/code/public_html/texte
<p>
Aufgaben:
<p>
<h2> Paketverwaltung </h2>
<ol>
<li> Installieren Sie die bash und tcsh Bin\exe4rpaket (Quelle:
ftp://ftp.netbsd.org/pub/NetBSD/packages/1.6/sparc/All)

<?php auswertung_teiluebungen(
    XXX, // vulab1: netbsd-check-installed-pkg PKG=tcsh
        // tcsh installiert? (pkg_info -e tcsh)

    XX, // vulab1: netbsd-check-installed-pkg PKG=bash
        // bash installiert? (pkg_info -e bash)
): ?>
</ol>

<h2> Benutzerverwaltung </h2>
<ol>
<li> Richten Sie einen neuen Benutzer "test" ein. Home-Verzeichnis
```

Figure 6.25: Preparing an exercise, part 1: Writing exercise text and hints

```
Feyrer@w1445:/home/feyrer/work/vulab/code/public_html/texte
w1445% perl uebung2db -v netbsd netbsd.php n
check_id 908 inserted (1)
check_id 909 inserted (1)
check_id 910 inserted (1)
check_id 911 inserted (1)
check_id 912 inserted (1)
check_id 913 inserted (1)
check_id 914 inserted (1)
check_id 915 inserted (1)
old checks removed from database
w1445% █
```

Figure 6.26: Preparing an exercise, part 2: Extracting hints into database and writing new text with check-numbers for feedback hints

```

feyrer@w1445:/home/feyrer/work/wulab/code/public_html/texte
--- netbsd.php Mon Feb 23 16:39:21 2004
+++ n Mon Feb 23 16:37:58 2004
@@ -1,3 +1,4 @@
+<!-- DB updated by feyrer on Mon Feb 23 16:37:57 MET 2004 from netbsd.php -->
<!-- $Id: netbsd.php,v 1.13 2004/02/19 10:55:52 feyrer Exp $ -->
<?php auswertung_ueberschrift(); ?>
<!-- ----->
@@ -15,10 +16,10 @@
ftp://ftp.netbsd.org/pub/NetBSD/packages/1.6/sparc/All)

<?php auswertung_teiluebungen(
-     XXX, // vulab1: netbsd-check-installed-pkg PKG=tcsh
+     908, // vulab1: netbsd-check-installed-pkg PKG=tcsh
//         tcsh installiert? (pkg_info -e tcsh)

-     XXX, // vulab1: netbsd-check-installed-pkg PKG=bash
+     909, // vulab1: netbsd-check-installed-pkg PKG=bash
//         bash installiert? (pkg_info -e bash)
); ?>
</ol>
@@ -30,23 +31,23 @@
soll /home/test sein, Shell "tcsh".
bute 886

```

Figure 6.27: Preparing an exercise, part 3: Comparing original and updated exercise text

```

feyrer@w1445:/home/feyrer/work/wulab/code/public_html/texte
w1445# mv n netbsd.php
w1445#
w1445#
w1445# cvs ci -m 'Datenbank-Update nach neuen Aufgaben' netbsd.php
Checking in netbsd.php;
/home/feyrer/cvsroot/code/public_html/texte/netbsd.php,v <-- netbsd.php
new revision: 1.14; previous revision: 1.13
done
w1445#

```

Figure 6.28: Preparing an exercise, part 4: Moving the updated exercise into place and saving to the CMS

print any feedback data. The signalling is done by the Virtual Unix Lab framework before it pulls in both the functions' definition and the exercise text.

**Verification:** Verification of the exercise results consists of almost the same procedure as in step I, as illustrated in figures 6.21 to 6.24:

1. Query the database to determine which check scripts to run, what parameters to pass to them, and on which machine to run them.

The additional field "parameter" was added to the `uebungs_checks` table, where the "uebung2db" script stored the parameters for the call of the shell script. This field is retrieved in addition to the data already required in step I.

2. Run the check script with the parameters from the database, and collect the result.

The same procedure is used as in step I, i.e. the script is handed to an interpreter for execution, and the output is collected to see if the check indicated success or failure. Parameters for the scripts are passed as environment variables.

3. Store the check script's result into the database.

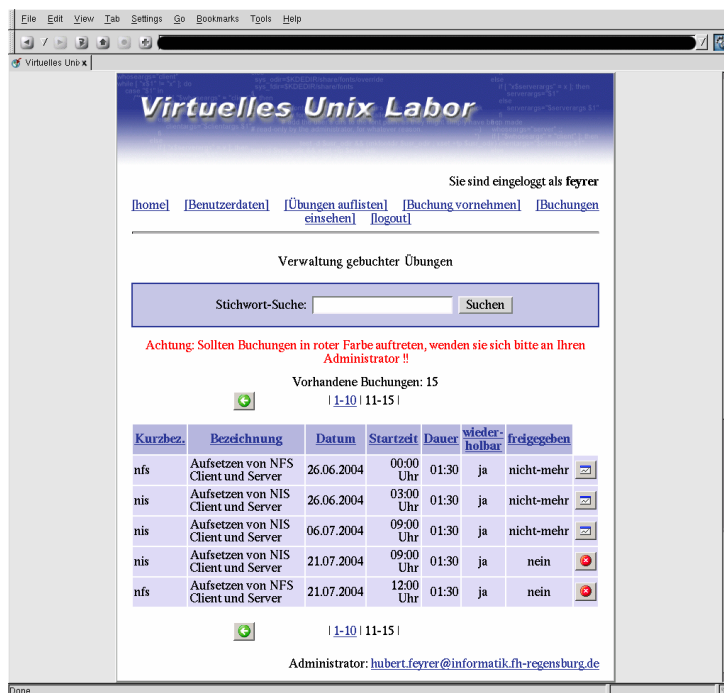
There is no change from step I here. The textual output of the check script is scanned for an indicator of success or failure, and the boolean "erfolg" (success) field of the `ergebnis_checks` table is set accordingly.

**Feedback:** At any time, a list of all booked exercises ever can be retrieved by selecting the "Buchungen einsehen" (view booked exercises) menu item. The list contains both exercises already completed as well as exercises that were booked for future dates, see figure 6.29. Exercises that have already been taken do have a button on their right that can be used to analyze that particular exercise and retrieve feedback it, see figure 6.30 a) for an image of the analyze/feedback-button. If an exercise was booked but has not been prepared and taken yet, it can be cancelled by using the button displayed in figure 6.30 b). This works only until the exercise's preparation time has arrived, which is about 45 minutes before the start time. An exercise that has been prepared will be recorded as such. If a student does not show up for a booked and prepared exercise, the system will notice and keep a record on this.

When an exercise has been completed successfully and the verification of the exercise's result is done, feedback on the exercise can be retrieved by pressing the corresponding button.

As described in section 6.5.3, the Virtual Unix Lab system then displays the exercise text, and runs the embedded PHP functions to show the textual descriptions of the tasks, details on what the checks tested, and if the tests were successful or not.

In detail, the PHP code in the exercise text first calls `auswertung_ueberschrift()` and prints a header with general information about the exercise:



Sie sind eingeloggt als feyrer






[home](#) | [Benutzerdaten](#) | [Übungen auflisten](#) | [Buchung vornehmen](#) | [Buchungen einsehen](#) | [logout](#)

Verwaltung gebuchter Übungen

Stichwort-Suche:  Suchen

Achtung: Sollten Buchungen in roter Farbe auftreten, wenden sie sich bitte an Ihren Administrator !!

Vorhandene Buchungen: 15  
| 1-10 | 11-15 |

Kurzbez.	Bezeichnung	Datum	Startzeit	Dauer	wiederholbar	freigegeben
nfs	Aufsetzen von NFS Client und Server	26.06.2004	00:00 Uhr	01:30	ja	nicht-mehr 
nis	Aufsetzen von NIS Client und Server	26.06.2004	03:00 Uhr	01:30	ja	nicht-mehr 
nis	Aufsetzen von NIS Client und Server	06.07.2004	09:00 Uhr	01:30	ja	nicht-mehr 
nis	Aufsetzen von NIS Client und Server	21.07.2004	09:00 Uhr	01:30	ja	nein 
nfs	Aufsetzen von NFS Client und Server	21.07.2004	12:00 Uhr	01:30	ja	nein 

| 1-10 | 11-15 |

Administrator: [hubert.feyrer@informatik.fh-regensburg.de](mailto:hubert.feyrer@informatik.fh-regensburg.de)

Figure 6.29: The list of booked exercises contains both completed exercises for which feedback can be requested (“freigegeben: nicht-mehr”) as well as uncompleted exercises that have not yet started (“freigegeben: nein”)



Figure 6.30: Buttons for a) retrieving feedback on completed exercises, and b) deleting uncompleted exercise that have not yet started

date and time of start and end, duration in minutes and the IP number from which the exercise was taken. The IP number is the one stored by the firewall configuration when the lab was entered for the exercise, as shown in figure 4.8. The exercise text is displayed next, augmented with calls to the `auswertung_teiluebungen()` function, which does the main job of giving feedback.

The `auswertung_teiluebungen()` function takes a variable number of arguments, each representing a check-number. See figure 6.32 for an example. For each of the check numbers, the function retrieves:

- the textual description of the check as stored in the “bezeichnung” (description) field of the `uebungs_checks` table, and
- the result of the check as stored in the “erfolg” (success) field of the `ergebnis_checks` table.

If the feedback is not requested by a “normal” user of the Virtual Unix Lab but by an administrator, an overview of all students’ performance is shown in addition to the single user’s result as discussed in section 6.5.3.

From the description of these phases, it can be seen that there are a number of small to medium size changes, but that the general design and implementation of the Virtual Unix Lab result verification architecture could have been kept for step II.

### **6.5.6 Summary of step II**

Comparing the improvements intended for step II of the Virtual Unix Lab and the changes made, the conclusion can be drawn that the goals were met within the given requirements. Step II of the Virtual Unix Lab was realized as described here, and used as base for the evaluation in chapter 7. During the implementation and evaluation of step II, a number of possible improvements were identified, which can be addressed in future implementation steps of the Virtual Unix Lab. They will be listed in the conclusions drawn on result verification of exercise results after discussing the resulting Domain Specific Languages in the next section.

## **6.6 The Verification Unit Domain Specific Language (VUDSL)**

This section summarizes the domain specific language defined for the Virtual Unix Lab so far. This DSL was titled and will be referred to as the “Verification Unit Domain Specific Language” (VUDSL). Exercises in the Virtual Unix Lab described consist of three major components:

## 6.6. THE VERIFICATION UNIT DOMAIN SPECIFIC LANGUAGE (VUDSL)

1. Exercise text, which is displayed both during the exercise and also when giving feedback.
2. Data on what aspects of the lab machines to evaluate.
3. Display of feedback on the exercise results as established by the data on what to evaluate from (2.) in the context of the exercise text (1.).

The connection between the data for the actual evaluation (2.), the exercise text (1.) and its display for the results is implemented by a domain specific language that realized the “data structure representation pattern.”

As described in section 5, DSLs are usually not described by a full syntax specification with a context free grammar, but rather as extension of an existing programming language. This is also the case for the VUDSL, which is an extension to PHP that specifies evaluation data. That data is later on stored into the SQL database by the VUDSL processor, and the resulting database IDs are noted in the resulting PHP file.

All “real” data for verification of the exercise part’s verification are kept in PHP comments. A full exercise including all VUDSL statements can be seen in appendix A.2. The examples in figure 6.31 explains the important components of the VUDSL.

Currently there are only two lines of comments, and they are expected to have fixed format:

1. The first line currently contains the name of the lab machine on which a check is performed, the name of the check script as the primitive of the verification language, and any possible parameter that may be needed to further specify the test.
2. The second line of the PHP comments contains the help text that is given as feedback to the student to give him an idea what the actual check did, in addition to telling him if that part of the exercise was mastered successfully or not.

The VUDSL processor `uebung2db` applies lexical analyzing. It extracts the data from the PHP comments and stores them in the SQL database. It also updates/generates IDs for the checks, and updates them in the exercise text as shown in figure 6.27.

```
<?php auswertung_teiluebungen(
    ??? // vulabl: check-program-output PROGRAM=ypwhich OUTPUT_SHOULD='vulabl'
        //      Gibt ypwhich(1) 'vulabl' zurück?
); ?>
```

Figure 6.31: VUDSL example for verifying one aspect of the exercise

```
<?php auswertung_teiluebungen(  
    776, // vulabl: check-file-exists FILE=/var/yp/Makefile  
        //      Existiert /var/yp/Makefile?  
  
    777, // vulabl: check-file-exists FILE=/var/yp/binding/vulab/ypservers  
        //      Existiert /var/yp/binding/vulab/ypservers?  
  
    778 // vulabl: check-file-exists FILE=/var/yp/passwd.time  
        //      Existiert /var/yp/passwd.time?  
  
); ?>
```

Figure 6.32: VUDSL example for verifying multiple aspects of the exercise in one go

A list of all check primitives with their assorted parameters for the first line can be found in appendix A.5. Extensions of the VUDSL to add user adaption in the Virtual Unix Lab are discussed in section 11.6.

## 6.7 Conclusion of diagnosis and feedback with a domain specific language

This section summarizes the results from the steps taken to design and implement verification of exercise results in the Virtual Unix Lab and the “Verification Unit Domain Specific Language”. When viewing result verification in the Virtual Unix Lab under aspects of Domain Specific Languages, the following key items were covered in this chapter:

**Stereotypes** in the form of check scripts for testing single aspects of a system have been identified, a framework for implementation and running those check scripts was defined, scripts were implemented based on the framework, and the overall organisation was improved in an iterative approach. The result is a number of check scripts that can be supplied with parameters to test unique aspects of various systems, while being used as “activators” (function calls) for the tests from a programming language view. The scope of most of the check scripts is to be usable on all systems, but some are tailored towards testing of aspects that are unique to certain operating system implementations only.

**A language** was defined for the domain of combining exercise texts with result verification. This was done by embedding activators for the check scripts into the exercise text, using only special constructs of the PHP language as described by the “Language Specialization” pattern in section 5.3. Consequences of this are:

- Keeping the check-related data in a place that is close to the application domain of result verification, instead of the place where its physical storage is (the database).



- Easier maintenance of exercises, as all the important parts of an exercise – exercise text and data on check script calls – can be stored in a single place.
- The possibility to give feedback on exercises for both single users as well as administrators by coupling exercise texts and checks.

The language only knows about sequences of check script invocations so far. Extensions for selection of alternatives would be a future goal. No need is currently seen in implementing iterations as the third basic building block for programming languages.

**Keeping exercise text and check data in one place** is good for creation and maintenance of exercises. To perform the actual result verification after an exercise and to give feedback, it is easier to access the check-related data using database access routines though. To achieve this, the exercise text contains the check data in the form of the above-mentioned “language”, processed by a simple lexical analyzer which transforms the check data stored in the exercise text and stores it into the database. This transformation from the data representation which is close to the application domain of the exercise to the data representation used in the database realizes the Data Structure Representation pattern described in section 5.3.

**Implement a system front-end** for generating and updating harddisk images for new and updated exercises. This was done by using special check scripts with the result verification architecture. Instead of requiring a lot of details about the implementation of the Virtual Unix Lab for creating or updating an exercise, this special knowledge was moved into two check scripts that can be used for a special kind of exercise. This allows concentrating on the contents of the exercise creation without distraction by technical details. The System Front-End pattern is described in detail in section 5.3, its design and implementation within the Virtual Unix Lab are described in section 6.5.4.

Descriptions of related items and some implementation details were included to illustrate the connections between the key items as well as their integration and interaction, and to show possible areas for future improvements, which are summarized in the next section.

## 6.8 Future Perspectives

While implementing result verification and feedback in step II of the Virtual Unix Lab, a number of items were discovered that may be of interest for its future incarnations, assuming that another design & implementation cycle as described in section 6.2 and as realized in steps I and II of the Virtual Unix Lab will happen. For this new cycle – step III – possible areas of improvement are:

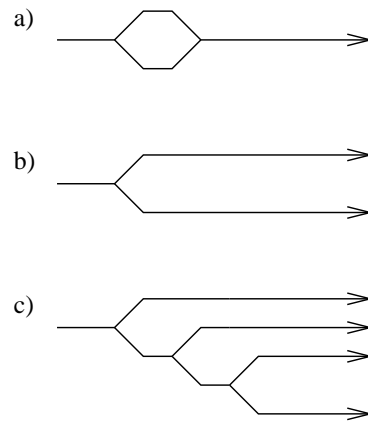


Figure 6.33: Various forms of non-linear exercises

**Non-linear exercises:** For some types of exercises, better control than strictly linear evaluation is required, i.e. to also allow users to choose alternative paths for solving a given problem. Depending on the nature of the decision, it is possible to either have only single (small) alternatives, or whole (big) trees, depending on the nature of the alternative.

A “small” alternative can be a single item that can be solved in various ways, but which will not have an impact on the verification of the further items, as displayed in figure 6.33 a). An example for this would be the setting of the domain name on NetBSD in the NIS exercise, which could either be done in `/etc/rc.conf` or `/etc/defaultdomain`.

“Big” differences decide about further tests, the way they are applied, or the exact state of which object will be checked as shown in figure 6.33 b). This is of concern if the choice of important software components should be left to the user, and not restricted by the exercise text. Components where this would apply are e.g. choice of operating system (Windows vs. NetBSD vs. various Linux distributions), webserver (Apache vs. Internet Information Server) and database software (Oracle vs. MS SQL Server vs. MySQL vs. PostgreSQL).

As the result of a check script is a boolean value and thus can only have two results. To realize multiple choices, a combination of boolean checks has to be used, as shown in figure 6.33 c).

Possible realization of multiple alternatives based on check results could happen during evaluation, by using check results and the language features that are provided by the PHP language, which is already used for evaluation and reporting of feedback. Theoretical foundations can be found in [Witschital, 1990, pp. 18] and [Robberecht, 2007].

**Assessment:** During an early stage, the feedback given to students included a counter

telling that “X out of Y tests were performed successfully.” During the beta test period, this led students to think they should get as high of a “score” as possible. This thinking was wrong for two ways: for one, a number of tests were built into the exercises that were meant to be ok by default, and only to test if the user damaged a crucial default-setting. Also, when alternative exercise paths (as described before) are implemented, many checks (50% for each alternate path) will be wrong. As such, just going for the absolute number of checks performed successfully is not useful – think of a user trying to install both Windows and Unix on the same machine to score both points.

If giving scores like “X out of Y possible points” or “Z% completed successfully” should be realized, work is needed to identify which check results should impact the scoring, and which should not. For alternatives, only one out of two or more alternatives can score a point. Others tests can influence the score directly, either in a positive or negative way, i.e. if the user achieved a goal during the exercise, or if he damaged an important configuration that was working properly by default (“false positive”).

Beyond giving scores, it is possible to give grades for completing of exercises, i.e. excellent ones if all/most of the “important” checks are solved properly, medium ones if there are some errors and bad ones if the goals of the exercise were not met. Establishing criteria on which check would have which kind of impact on the score and grade would need further research. The details could be encoded in the VUDSL, see section 11.6 for possible ways of such extensions. Existing literature on instructional and test design can be used for this enhancement<sup>1</sup>

Creating interfaces and APIs for external assessment tools could be considered as an alternative to realizing assessment in the Virtual Unix Lab.

**Further check script optimization:** Besides the major changes in the way feedback is given to users, some smaller changes can be done to optimize the implementation.

One such change is to extract common code from all the check scripts. Currently, the check scripts define a number of variables to describe purpose of the script, parameters supported, and Perl function that performs the actual test. Besides that, each of the check scripts contains code that evaluates the variables, checks parameters, calls the check function etc., which is the same code for all the check scripts. As only the first part of all the check script differs, and the second one is the same for all, the second half could be moved into a separate file and appended when calling the script, i.e.

```
“cat foo-check-bar common-body | perl”
```

**Check script parameter checking in web frontend:** Each check script can be queried for the parameters it accepts. The VUDSL-processor already uses this to verify check script data before storing it into the database. Another change that has

---

<sup>1</sup> [Eikenbusch and Leuders, 2004] pp. 10

small impact of the functionality of the system but increases stability would be to add a similar verification to the web GUI. That way, simple errors like typos could be caught earlier and easily.

The described items would have impact on both functionality of the Virtual Unix Lab towards users and administrators designing new exercises. Further improvements to performance, reliability and usability of the Virtual Unix Lab can be made in future versions by using the same iterative design and implementation cycle that was used in the first implementation steps described in this chapter.

# Chapter 7

## Evaluation of the Virtual Unix Lab

This chapter observes the Virtual Unix Lab that was described in the previous chapters, and evaluates it under a number of aspects. So far, only teaching of theoretical knowledge was possible for advanced topics in the system administration class held at the University of Applied Sciences Regensburg which, as described in chapter 3. After the system was realized, it is possible to offer practical exercises for those topics, and test practical competence, instead of theoretical knowledge. The Virtual Unix Lab was used to supplement the classroom lecture in the summer semester 2004, and this chapter covers the experiences that were made during its use.

With the work in the area of diagnosis and giving user feedback, the Virtual Unix Lab is complete for practical use. As a consequence, the evaluation can concentrate on the effect of the system as a whole, instead of observing only single components of the system and their efficiency. Evaluation of the whole system and its reflection on the user is considered to be more than the sum of its components.

Many aspects of the components of the Virtual Unix Lab, their use and the system as a whole are not covered in detail here, some of which are discussed briefly in the section 7.4.

### 7.1 What to evaluate

To establish the effect of the Virtual Unix Lab as a whole, the question arises if the Virtual Unix Lab is “useful”, i.e. if the students learned “more” or “better” than without the system.

When observing the impact of a learning tool, an established method for evaluation is to create a controlled testing environment by split a class into two groups. One group uses the platform, and the other group uses an alternative, likely a method that was

used before the new platform was available. After the testing period, the results from both groups are compared.

This approach would be recommended for use with the Virtual Unix Lab, too: Have students attend the “System Administration” lecture, let them participate in the usual lab exercises, but only allow half of the students to use the Virtual Unix Lab in addition. At the end of the semester, both groups would take the same end-of-term paper test. The results of that paper test would be examined for impact by the learning platform.

There are two problems with this approach. The first one is that for small groups, it is possible that all “good” students are in one group, and all “bad” students are in the other group. Increasing the group size would help, but this would require more students than available. The other problem with this approach is that it cannot be performed in a “live” setup with students, as the “System Administration” lecture is a mandatory course at the University of Applied Sciences Regensburg, and allowing part of the students to use a learning material while denying it to others is not possible. Arranging for a separate course outside the normal curriculum was unfortunately not possible due to lack of students and funding for such a venture.

An attempt of comparing existing end of term papers from student groups that did use the Virtual Unix Lab with the results of students that did not use the Virtual Unix Lab was made and described in [Feyrer, 2007c]. Even if the comparisons suggested that the Virtual Unix Lab indeed had a positive effect on student’s performance in the paper tests, a control group is really needed for reliable results.

As a result, data that was gathered during existing exercises in the Virtual Unix Lab is utilized. In particular, the following material is examined for this evaluation:

1. Students were asked to perform two particular exercises in the Virtual Unix Lab. Data was gathered for the full study group, and analyzed in section 7.2.
2. Students were asked to fill out an online questionnaire after their exercises in the Virtual Unix Lab. The results from this questionnaire are analyzed in section 7.3.

Other methods like personal interviews or recording the students’ practices on video would have been possible to obtain information to evaluate in theory. In practice this would have limited students in their free choice of time and place for doing the exercises, thus canceling the “virtual” effect of the Virtual Unix Lab. As a result, those methods were not pursued.

### 7.2 Analysis of data gathered during student exercises

This section analyzes data that was gathered during the Virtual Unix Lab exercises in the summer semester 2004. The analysis covers a number of aspects that have impact on the learning performance.

After a brief introduction of the methodology used, this section first examines exercises that were taken several times by a user, and makes an investigation of the performance of those repeated exercises. Next, it was noted that some basic tasks occur in more than one place in the exercises. An analysis of the performance in similar tasks is made to determine conceptual problems that need better education. Last, the time at which exercises were performed, and the duration of those exercises, was observed.

#### 7.2.1 Methodology of the data analysis

This section describes the methods that are used for the evaluation of the Virtual Unix Lab. It observes data that was gathered during students' exercises, and utilizes visualization techniques to aid in the evaluation process.

For visualization, methods from statistics are used to compare various values with each other. Histograms and box-plots (also known as whisker plots) are used in the following sections<sup>1,2,3</sup>. To also allow visual comparison of median values in box-plots, "notches" are added to indicate the confidence intervals for the median of the distribution. This allows to compare the median of two distributions – if the intervals around two medians do not overlap, they can be considered different with 95% confidence<sup>4</sup>. This method allows to tell which median is "better" (higher or lower, depending on score or grade) by visual inspection of the graph<sup>5</sup>.

The data used for analysis and evaluation is stored in an SQL database, appendix B has details on the database structure. Queries to retrieve data from the SQL database are listed in appendix C.3 and referenced from this section where the data is discussed.

In many cases, SQL is not adequate for analyzing data, and the R program was used for statistical analysis. Export of data from the PostgreSQL database was done by using the "psql" command line tool, which was told to print output unaligned ("`\a`"), use a ";" as record separator ("`\f ;`"), write the results of SQL queries into a file with colon-separated values (CSV; "`\g file.csv`"), and do not include the standard footer ("`\pset footer`") in the output. Import of data into R was performed by reading the CSV file into an R table ("`table=read.csv("file.csv")`").

---

<sup>1</sup> [Tukey, 1977] pp. 39

<sup>2</sup> [Fahrmeir, 2003] pp. 65

<sup>3</sup> [Chambers, 1983] pp. 21

<sup>4</sup> [McGill et al., 1978] pp. 12

<sup>5</sup> [Garrett and Nash, 2001] pp. 12

```

count | uebung_id
-----+-----
    58 | nfs
    71 | nis
     6 | netbsd
(3 rows)

```

Table 7.1: Exercise popularity

To examine “similar” exercise tasks, the checks are examined as identified by their check numbers and the associated data, check scripts name and parameters. Full definition of the checks are contained in the exercise texts for NIS and NFS. See appendix A.2.1 for the NIS exercise text, and appendix A.2.2 for NFS. A brief list of checks including the description that is printed as feedback for users can be found in appendix A.4.1 for the NIS exercise and in appendix A.4.2 for the NFS exercise.

### 7.2.2 Number of exercises taken and repeated

Students were told to perform the NIS and NFS exercises each once at least, with no restrictions on repeating an exercise several times. This section observes how often students really booked exercises. For those students that booked an exercise more than once, any possible differences in performance between the first and last time they repeated an exercise will be observed. The goal is to determine the impact of use of the Virtual Unix Lab here.

The number of total exercises (NIS, NFS, NetBSD) every student took<sup>1</sup> can be seen in results of query 5 in appendix C.3. From the 27 students, one student performed only one exercise (instead of the requested two, NIS and NFS), and three students performed exactly two exercises. A more detailed overview which includes the exact exercises<sup>2</sup> can be seen in results of query 6 in appendix C.3. 19 students performed single exercises only once, but the majority of exercises was taken two or more times, up to a maximum of one student taking the NIS exercise 8 times.

Overall, at the end of the summer semester 2004, 135 exercises were performed by students in the Virtual Unix Lab. Table 7.1 shows how often each of the exercises was chosen<sup>3</sup> – the NIS and NFS exercises were requested to be taken, the “netbsd” exercise was offered to become more familiar with the NetBSD operating system.

When examining the number of distinct students that took exercises, it can be seen

<sup>1</sup> See query 5 in appendix C.3 on page 372

<sup>2</sup> See query 6 in appendix C.3 on page 373

<sup>3</sup> See query 7 in appendix C.3 on page 374



## 7.2. ANALYSIS OF DATA GATHERED DURING STUDENT EXERCISES 133

that 26 students took the NIS exercise<sup>1</sup>, and 27 students took the NFS exercise<sup>2</sup> – The fact that one student only took one of the two exercises that make the difference here was also seen in the number of total exercises that every student took<sup>3</sup>. With these numbers, it can be said that students took the NIS exercise 2.73 times on average<sup>4</sup>, and the NFS exercise was booked 2.15 times on average<sup>5</sup>.

The reason why the NIS exercise was practiced more often than the NFS exercise are not known at this point. While it could be suspected that students may regard NIS as more difficult, and thus wanted to repeat it more often to get all the tasks in the exercise right, there is no evidence for this, at least not from the data observed here. See also section 7.3 for students' opinions on the Virtual Unix Lab.

### 7.2.3 Performance of repeated exercises

With the fact that many students repeated an exercise more than once, a comparison between the various exercise repetitions' results can be made, to see if a difference in performance could be found.

To find the first and last repetition of a certain exercise and a certain user, the corresponding booked exercise IDs ("buchungs\_id") need to be known. IDs are numbers and allocated increasingly for each new exercise that is booked. When looking at the various booked exercise IDs of a user, it can be assumed that his first exercise had the smallest (minimum) ID, and the last exercise had the biggest (maximum) ID.

The other question is how to assess an exercise's performance. For this comparison, the number of successfully performed tasks are counted, without looking at false positives, i.e. tasks that would test as successful per default, but that are tested to see if students broke the configuration for them. A more in-depth look at exercise results for assessing performance is outside the scope of this discussion. See also section 6.8 for further information on establishing assessment.

By combining these data points, an investigation can be made to look at students' results of their first and last exercise of a certain kind (NIS, NFS). This can be seen in results of query 10 in appendix C.3<sup>6</sup>. The list includes the ID ("first\_id", "last\_id") and percental score ("f\_pscore", "l\_pscore") of a student's first and last exercise of a certain kind ("uebung\_id") and student, as well as the difference between the two percental scores ("dpscore").

Looking at all exercises, i.e. NIS and NFS, the average score of the first exercises is

<sup>1</sup> See query 8 in appendix C.3 on page 374

<sup>2</sup> See query 9 in appendix C.3 on page 374

<sup>3</sup> See results of query 5 in appendix C.3 on page 372

<sup>4</sup>  $71 / 26 = 2.73$

<sup>5</sup>  $57 / 27 = 2.15$

<sup>6</sup> See query 10 in appendix C.3 on page 374

44.61% and the average score of the last exercises is 62.73%, i.e. between students' first and last exercise there is an average increase of 18.13%. Figure 7.1 shows the distributions of the first and last exercises of students. The facts that the notches of both plots do not overlap shows that there is a statistically significant improvement between the students' first and last exercise!

Investigating this further, a comparison of the scores of the first and last exercises with each sorted ascending can be seen in figure 7.2. The figure shows that the results of the last exercises are better than the first ones. The figure also suggests a correlation between the first and last exercise, but to investigate this, the scores have to be displayed in pairs of each student's first and last score. This can be seen in figure 7.3 – while the results of the first exercises are still sorted ascending, the scores of the last exercise are printed accordingly. Two results can be seen. First, almost all scores are higher in the last exercise than in the first exercise, and vice versa, which confirms that an improvement in performance was achieved. Second, looking at the last exercises shows that students with low scores on the first result reached more or less the same scores as students who performed average or good in their first exercise, i.e. most gain was made by students who performed bad on their first attempts.

As such, no direct correlation between the first and last scores can get established, which is also confirmed by the Pearson correlation coefficient of 0.4397. Reasons which influence the overall increase in performance here may be that students scored bad scores in the first exercise because the tasks were not clear, not enough information to solve the tasks was available, the environment was not as familiar as in later exercises, or that the feedback provided after the first exercise helped to obtain better scores on later exercises. Exact reasons for the improvement cannot be given here, and are subject of further investigation.

Section 7.2.5 observed that the time needed by students for all exercises was influenced equally by both NIS and NFS. In a similar effort, after observing the results of all exercises together, the scores observed for NIS and NFS exercises will be examined separately next, to determine if the Virtual Unix Lab provided the same gain that was shown for all exercises.

The average percental score of students' first NIS exercise was 39.58%, and the average percental score of the last NIS exercise was 67.65%. This amounts for an average increase of 28.08%. For NFS, the average percental score of the first exercises was 53.96% and 64.96% of the last exercise, i.e. a gain of 11.00%. While the first results were a bit better for NFS than for NIS, the results show that students greatly improved in the NIS exercise, while results for NFS did increase too, but not as much. Figure 7.4 contains the corresponding box plots - the great improvement between the first and last NIS exercise is depicted in 7.4 a), while the lesser, but still existing improvement can be seen in 7.4 b). Both improvements are statistically significant, as shown by the non-overlapping notches in figure 7.4 a) and b).

Comparing the two results, the Virtual Unix Lab had a bigger impact for the NIS exer-

## 7.2. ANALYSIS OF DATA GATHERED DURING STUDENT EXERCISES 135

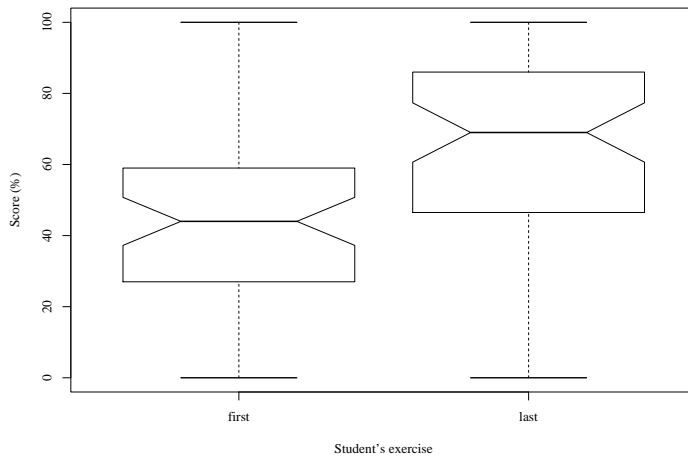


Figure 7.1: Comparison of all scores between students' first and last exercise

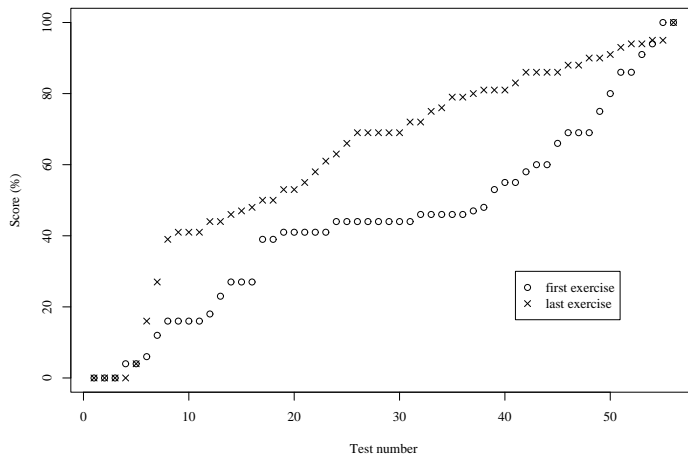


Figure 7.2: Score of all first and last exercises ordered ascending

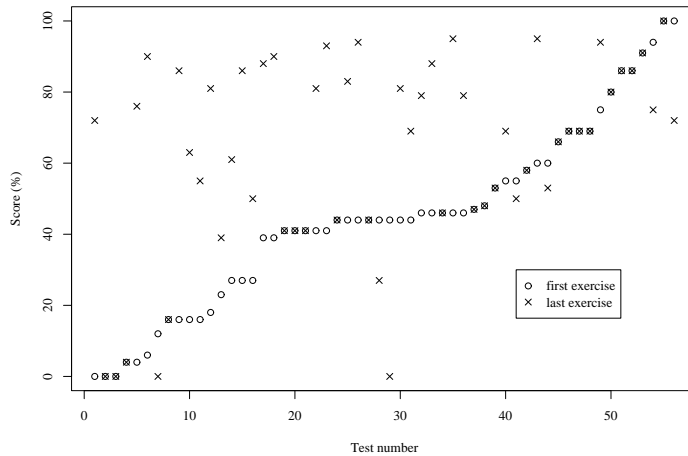


Figure 7.3: Score of all first and last exercises ordered by first exercise

cise than for the NFS exercise. This can also be seen from the distribution of the results in figure 7.5. While most students scored at best average scores in the first NIS exercise in figure 7.5 a), the last exercises were much better, reaching 100%. The NFS results shown in figure 7.5 b) are different here, where good students hardly increased their scores, but less scores below average were reached. Figure 7.6 also confirms that in NIS, most students' scores did improve, while the NFS exercises mostly had an effect on students that performed badly in their first attempts. A direct correlation between "first" and "last" exercises cannot be established for neither NIS nor NFS though, as indicated by the corresponding Pearson correlation coefficients of 0.26 (NIS) and 0.42 (NFS).

In summary, it can be said with statistical significance that the Virtual Unix Lab had a positive effect for both NIS and NFS exercises, and that the gain was most notable for the NIS exercise.

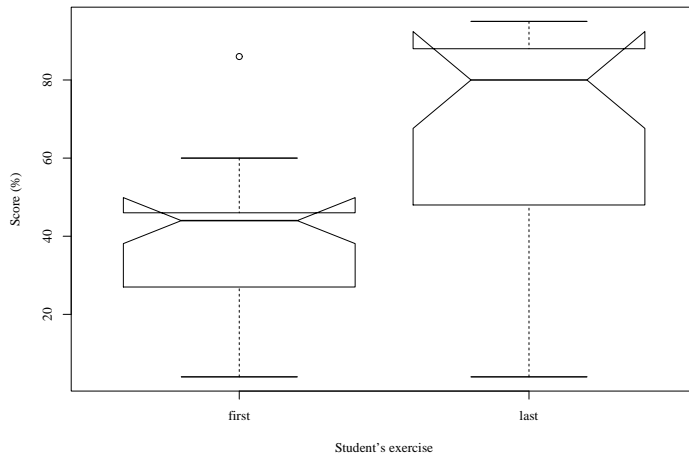
## 7.2.4 Results of selected exercise topics

After observation the impact and benefits of the Virtual Unix Lab, topics where students still have problems are identified next. The intent is to improve their knowledge about these topics, e.g. by discussing them in more detail in class, or by offering special exercises in the Virtual Unix Lab for these topics.

The identification of common topics will be made by observing tests performed at the

## 7.2. ANALYSIS OF DATA GATHERED DURING STUDENT EXERCISES 137

a) NIS:



b) NFS:

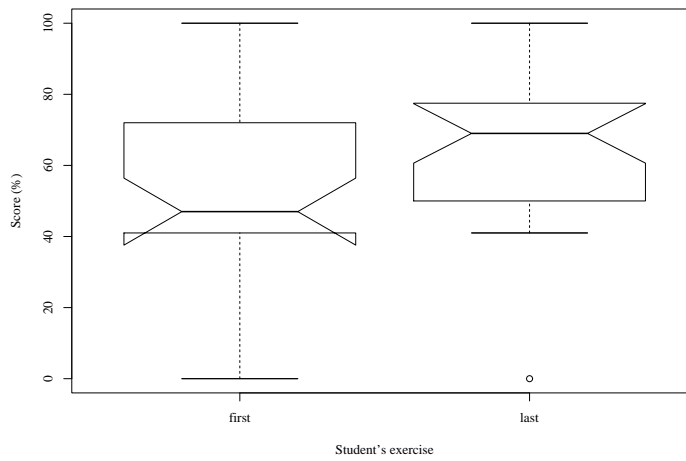
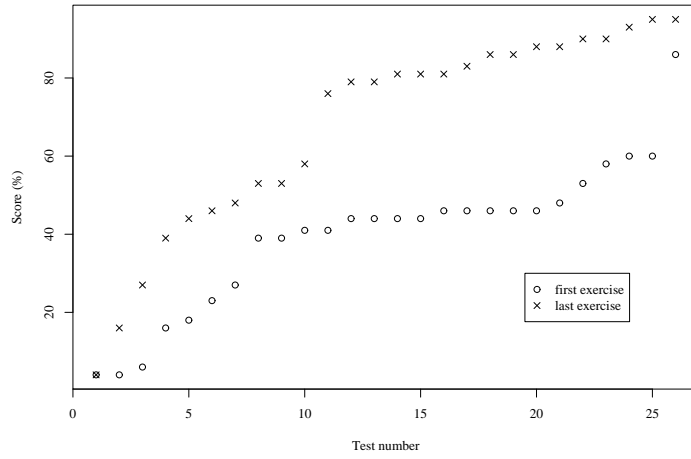


Figure 7.4: Comparison of a) NIS and b) NFS scores between students' first and last exercise

a) NIS:



b) NFS:

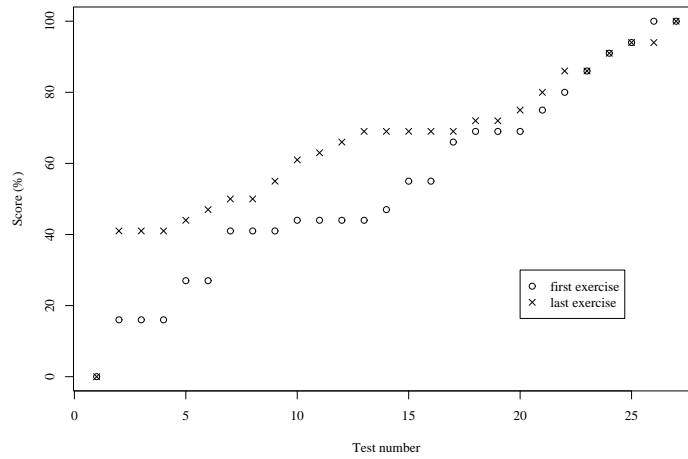
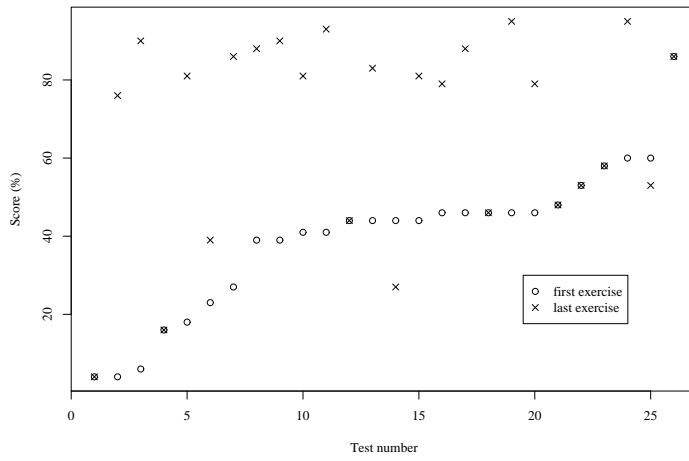


Figure 7.5: Score of first and last exercise ordered ascending for a) NIS and b) NFS exercise

## 7.2. ANALYSIS OF DATA GATHERED DURING STUDENT EXERCISES 139

a) NIS:



b) NFS:

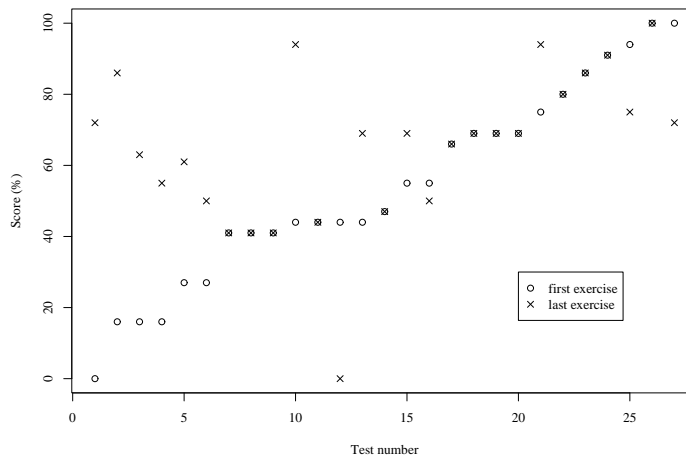


Figure 7.6: Score of first and last exercise ordered by first exercise for a) NIS and b) NFS exercise

count	script
21	check-program-output
10	check-file-contents
10	unix-check-process-running
7	netbsd-check-rcvar-set
6	unix-check-file-owner
5	check-file-exists
4	netbsd-check-installed-pkg
4	solaris-check-installed-pkg
4	unix-check-user-exists
3	check-directory-exists
1	unix-check-user-ingroup
1	unix-check-user-fullname
1	unix-check-user-password
1	unix-check-user-shell
1	unix-check-mount

Table 7.2: Check scripts and their usage in various checks

end of the Virtual Unix Lab. Similar topics are tested by using the same check scripts, and topics that are tested by the same check script are considered as related.

Table 7.2 shows a list of all check scripts in use in the Virtual Unix Lab<sup>1</sup>, and a count in how many places they were used to test for various similar topics by running the same script on different hosts with possibly different operating systems and with different parameters.

To compare various results of a single script, it has to be used in more than one place, obviously - as such, the last five scripts listed in table 7.2 are of limited use in this discussion. The following scripts and their results will be considered in this discussion. Attention must be brought to “false positive” tests here, as they are “true” by default to verify the system is operating properly, and only are “false” if the students break the configuration. For each script, a short description of the tested topic is given, followed by a comparison of the various tests by using boxplots, and a summary is drawn from the results, reflecting on students’ overall performance on the related topic.

`check-program-output`: used 21 times, see figure 7.7. No false positives.

This script tests the output of various programs, e.g. “ypwhich”, “domainname”, “ypcat”, “showmount”, “share”, “mount”, “df”, “cat” and, “ping.” These programs belong to a wide variety of topics, and are thus of little use to make a prediction for a particular topic. The non-uniform results in figure 7.7 confirm this.

<sup>1</sup> See query 11 in appendix C.3 on page 376



## 7.2. ANALYSIS OF DATA GATHERED DURING STUDENT EXERCISES 141

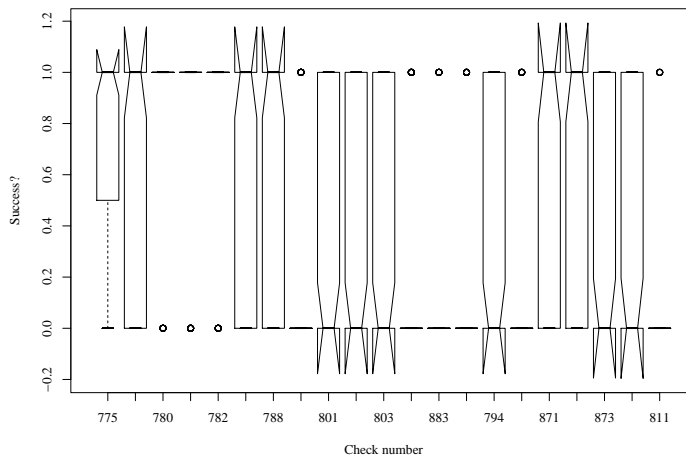


Figure 7.7: Results of check-program-output

`check-file-contents`: used 10 times, see figure 7.8. No false positives.

This script is used to see if files were edited properly, where use of an editor program was required. See results of query 12 in appendix C.3 for a description of the various editing tasks<sup>1</sup>. The results in figure 7.8 show that on average, six of the corresponding tasks were solved successfully while the rest were unresolved. Problems here could be that students were able to use the editor for one task but not another one, or more likely that students did not know what to edit in the first place.

`unix-check-process-running`: used 10 times, see figure 7.9. No false positives.

The test to see if a certain process runs properly was mostly solved successfully by students, as can be seen in figure 7.9. An interesting detail is that the boot system of Solaris needs no special configuration to start processes if a subsystem is configured, while NetBSD needs additional work. While students coped with both operating systems, the tests on systems that ran NetBSD (checks 798, 799, 878, 879, 880; see results of query 13 in appendix C.3) were performed slightly less successfully than on Solaris (checks 865, 867, 868, 869).

The conclusion is that process startup itself is performed properly by most students, but more emphasis could be put on understanding of the NetBSD boot system that is responsible for process startup and its configuration.

<sup>1</sup> See query 12 in appendix C.3 on page 376

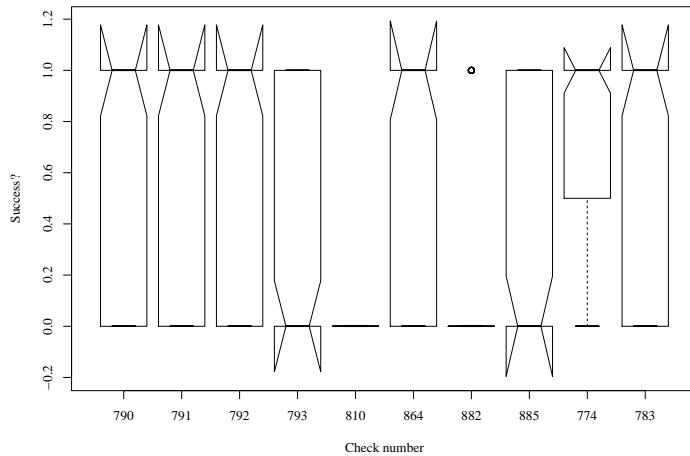


Figure 7.8: Results of check-file-contents

`netbsd-check-rcvar-set`: used 7 times, see figure 7.10. False positives: 795, 874.

This script checks if various services were started properly using the NetBSD startup mechanism. The results shown in 7.10 confirm the findings from the `unix-check-process-running` script above, as the majority of students had problems configuring the needed processes properly. Exceptions seem to be the results of checks 795 and 874, but the topics they test – see if the variable “`rc_configured`” is left at “`yes`” in `/etc/rc.conf`, as shown in results of query 14 in appendix C.3 – is properly configured by default (“false positives”), so they cannot be regarded as successfully solved.

In summary, the request for better education of students in the area of the NetBSD startup system can be repeated from these results.

`unix-check-file-owner`: used 6 times, see figure 7.11. No false positives.

This script verifies permission setting skills with a special focus on a distributed (NFS) environment. The results in figure 7.11 show that there is a lot of room for improvements. An interesting side effect is that the setting of permissions on a local system (checks 890, 892) seems to be easier for students, while the remaining checks test permission setting via NFS, as can be seen in results of query 15 in appendix C.3 (the NFS exercise defines `VULAB1` to be the NFS server with the data on local storage, and `VULAB2` the NFS client).

`check-file-exists`: used 5 times, see figure 7.12. False positive: 870.

## 7.2. ANALYSIS OF DATA GATHERED DURING STUDENT EXERCISES 143

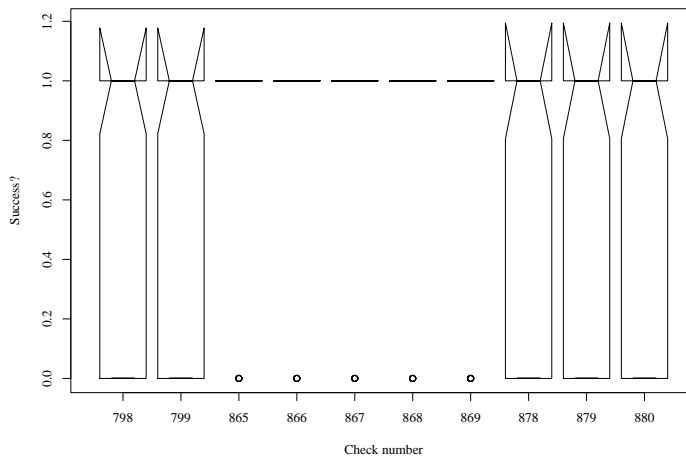


Figure 7.9: Results of unix-check-process-running

This test is used to see if a file exists, usually as a consequence of a user running a certain setup procedure like the NIS “ypinit” command. As the Solaris operating system starts the NFS service by default if the proper NFS configuration is present, check #870 is a false positive<sup>1</sup>. Regardless of that false positive, the results in figure figure 7.12 show that most students succeeded in performing the associated tasks.

`netbsd-check-installed-pkg`: used 4 times, see figure 7.13. No false positives.

Installation of binary packages on NetBSD is tested. to see if users installed either `tchsh` or `bash` as “convenience” shells in any of the exercises<sup>2</sup>. None of these is needed for successfully performing the NIS or NFS exercise, but the possibility is offered to users as an alternative to the less user-friendly default shells.

The results in figure 7.13 show that students either did not feel a need to install those shells, or failed to do so, as very few of them picked up the opportunity of a more convenient command line interface. The reasons for this are not known, and could be work of future investigation.

`solaris-check-installed-pkg`: used 4 times, see figure 7.14. No false positives.

<sup>1</sup> See results of query 16 in appendix C.3 on page 378

<sup>2</sup> See results of query 17 in appendix C.3 on page 378

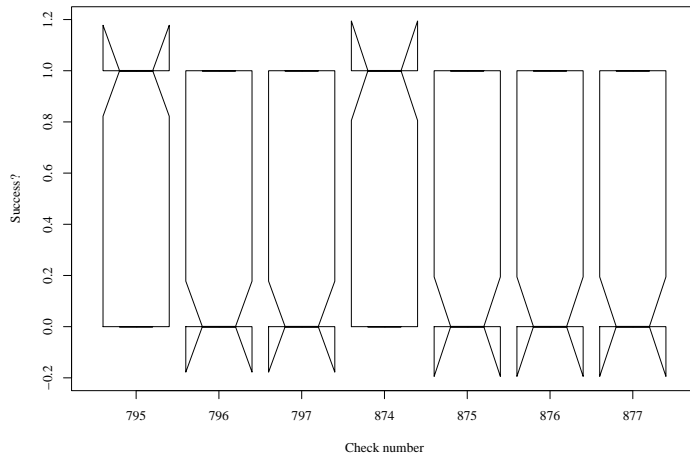


Figure 7.10: Results of netbsd-check-rcvar-set

Just as with the previous check script, this one tests the installation of packages, but this time on the Solaris operating system, which differs in some details from NetBSD. Also, installation of bash and tcsh was offered as convenience again, but not mandatory for neither the NIS nor the NFS exercise<sup>1</sup>.

The results shown in figure 7.14 indicate that most students did not attempt to install any of these packages, but that at least some tried successfully. Whether more students were interested in installing convenience shells on Solaris than on NetBSD is unknown (and rather less likely), but it is could be that it was also easier for students to install the packages on Solaris as they were provided for installation on the system in /cdrom, instead of requiring students to download them from the network, as needed for NetBSD. Whether this was too much effort for students, regarded as plain inconvenient, or if students just did not know how to handle packages properly is not known, but could be subject of future research.

`unix-check-user-exists`: used 4 times, see figure 7.15. No false positives.

One important resource distributed among machines in a NIS and/or NFS environment are user accounts and related data. This test checks if a certain user account was created or is accessible properly, i.e. if students were able to apply the appropriate user management skills that are required properly<sup>2</sup>.

Looking at the distribution of the results in figure 7.15, some improvements of the skills needed to manage user accounts, esp. in distributed environments,

<sup>1</sup> See results of query 18 in appendix C.3 on page 378

<sup>2</sup> See results of query 19 in appendix C.3 on page 378

## 7.2. ANALYSIS OF DATA GATHERED DURING STUDENT EXERCISES 145

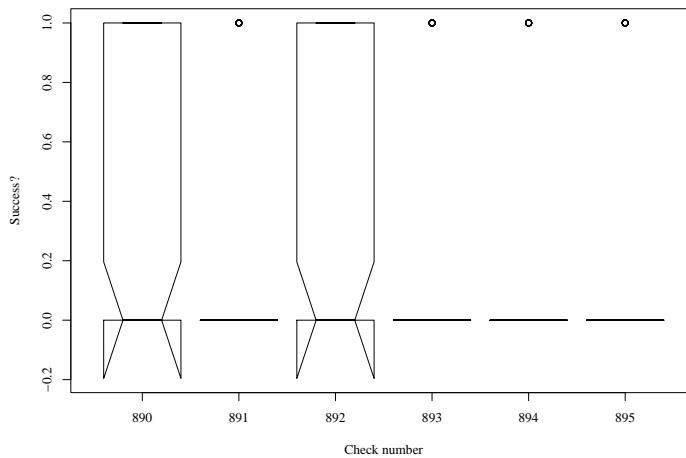


Figure 7.11: Results of unix-check-file-owner

seem required. The fact whether the problem here is on the “user management” or on the “distributed” part cannot be derived from the existing data.

`check-directory-exists`: used 3 times, see figure 7.16. No false positives.

This last check script and its results are related to the previous one: Checking for existence of a directory can be used for a number of applications. Within the Virtual Unix Lab, the primary application is to test if home directories of user accounts are created properly<sup>1</sup>.

The results of this exercise are shown in figure 7.16, they are similar to the ones of the previous check – some success, but definitely more education needs to be done in making sure users understand what the purpose of directory creation is in the area of user management.

After observing the various areas that are covered in the Virtual Unix Lab, it can be said that some topics are handled competently by students, while more education and practice would be appropriate for others. Topics that the students performed good in are changing system settings by editing files, handling of process startup via the Solaris boot system and setting up files for the Network Information System (NIS).

Areas that need further investigation are the process startup via the NetBSD boot system and its configuration in general, user management in general including creation of user accounts and making user data accessible and finally some emphasis of handling software packages on both Solaris and NetBSD.

<sup>1</sup> See results of query 20 in appendix C.3 on page 379

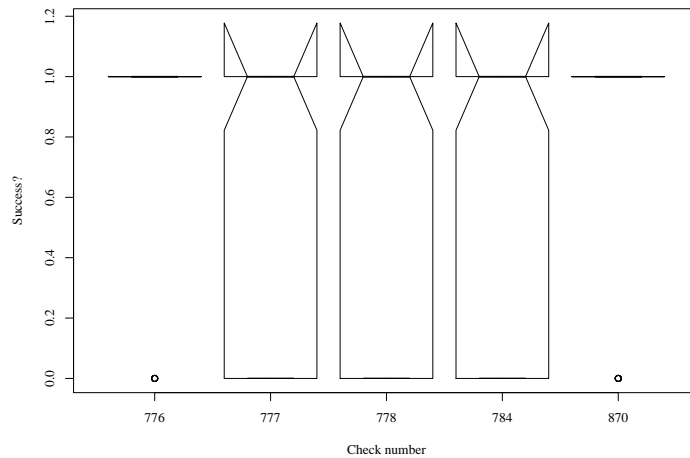


Figure 7.12: Results of check-file-exists

### 7.2.5 Exercise duration

The next question to investigate is if the time reserved for exercises was long enough, or if more time was required to solve them, i.e. if students ended the exercise first, or if it was ended by the timeout. To answer the question, the ending time of exercises was observed in relation to the start time, both of which were available for each booked exercise.

A list of all exercises that were taken in the Virtual Unix Lab is displayed in the results of query 21 in appendix C.3<sup>1</sup>. The “duration” is calculated by the difference between starttime and end time. Duration of NIS and NFS exercises are 90 minutes (1.5 hours, 01:30:00) each. As can be seen from the list, several exercises were not within the normal exercise period between 0 and 90 minutes. Negative durations and those that were significantly over 90 minutes (2 hours and up) indicate that technical problems arose during the exercise, and that manual intervention was needed by an administrator. In the following discussion, these exercises are thus excluded. Another set of exercises is also of interest - there are several exercises that took more than 90 minutes, with ranges between 90:05 and 98:45 minutes. Possible reasons for this delay could be too much system workload on the Virtual Unix Lab machine (a 85MHz Sun SPARCstation 5 that also had to serve other services than the Virtual Unix Lab in summer semester 2004), or that the mechanism to implement the timeout was inaccurate. Assuming either of these reasons, the results of the exercises of up to 10 minutes after the “official” end

<sup>1</sup> See query 21 in appendix C.3 on page 379

## 7.2. ANALYSIS OF DATA GATHERED DURING STUDENT EXERCISES 147

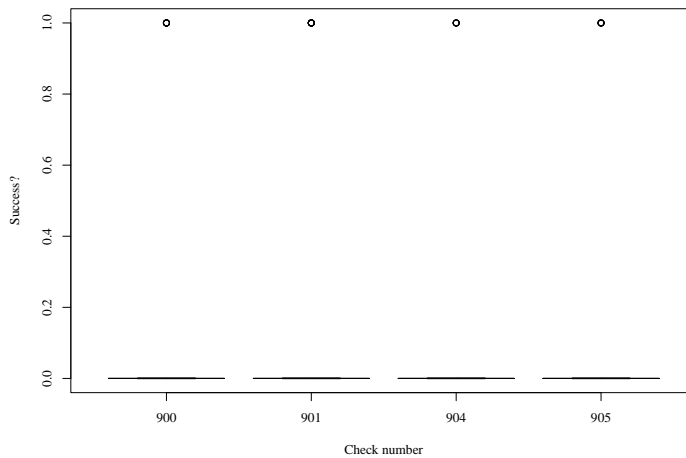


Figure 7.13: Results of netbsd-check-installed-pkg

of an exercise were included in the following analysis. Future versions of the Virtual Unix Lab should be extended to keep record if an exercise was ended by a user or aborted by timeout.

The time that students needed for exercises varied. A histogram of the various times needed by students to perform all NIS and NFS exercises is shown in figure 7.17<sup>1</sup>. Here, the bigger, white boxes in figure 7.17 a) indicate exercises accumulated over 10 minute intervals, while the smaller grey boxes in figures 7.17 a) and b) indicate exercises within a resolution of one minute.

The histogram in figure 7.17 a) shows that most exercises took between 40 and 90 minutes, with a significant number of exercises ending in the final 10 minutes. This could either be that the time reserved for the exercise was exactly right for most students, or that many exercises were aborted by timeout. As there is no record about the exercises terminated by timeout, a closer look at the list of exercises completed around 90 minutes in figure 7.17 b) shows that most exercises were actually ended *before* the timeout of 90 minutes, and that very few exercises ended later - if either by timeout or voluntarily is unknown, but of minor significance, assuming that no exercise was terminated by timeout before 90 minutes.

The overview of all exercises includes a total of 100 NIS and NFS exercises<sup>2</sup>, many of which were finished in between 40 and 90 minutes, and esp. shortly before 90 minutes.

<sup>1</sup> See query 23 in appendix C.3 on page 381

<sup>2</sup> See query 22 in appendix C.3 on page 381

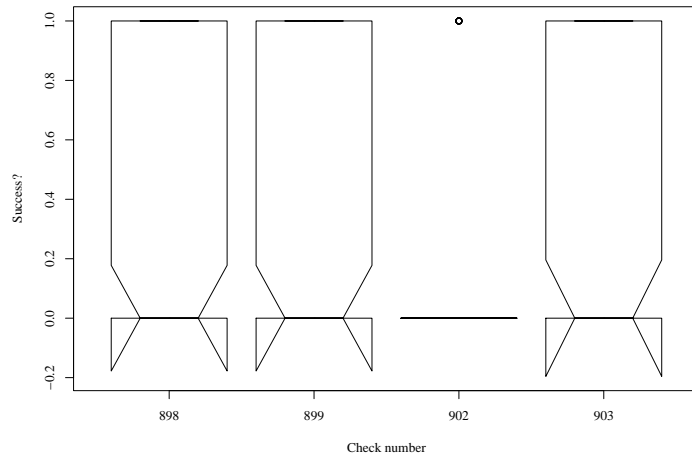


Figure 7.14: Results of solaris-check-installed-pkg

While no timeout ended any of those exercises, a more detailed analysis seems to be in order about the distribution, based on the two separate exercises for NIS and NFS. The 100 exercises observed so far consist of 59 NIS exercises<sup>1</sup> and 41 NFS exercises<sup>2</sup>, histograms for the NIS exercises are displayed in figure 7.18<sup>3</sup>, the same histograms for NFS are shown in figure 7.19<sup>4</sup>. Both figures include the number of exercises ended in 10 minute intervals (white boxes) and 1 minute boxes, and also contain an overview of the exercise duration of 100 minutes in figures 7.18 a) and 7.19 a) as well as zoomed to the 90th minute in figures 7.18 b) and 7.19 b).

Similar observations as for all exercises can be made for NIS and NFS separately – most exercises took between 40 and 90 minutes, with an absolute majority ending in the last few minutes, but before the timeout. As such, there seem to be no difference between the NIS and the NFS exercise. Figure 7.20 compares the distribution of the NIS and NFS end times, and the overlapping of the notches shows that there is no significant difference between the two exercises' durations, i.e. students take equally long for the NIS and the NFS exercises.

These findings answer the question if the time reserved for exercises was sufficient: According to the given data, exercise times for NIS as well as NFS were long enough, but close to the limit. More time for each of the two exercises should be considered, e.g. by changing the exercise time from 90 to 120 minutes, while reducing the post

<sup>1</sup> See query 1 in appendix C.3 on page 371

<sup>2</sup> See query 3 in appendix C.3 on page 372

<sup>3</sup> See query 2 in appendix C.3 on page 371

<sup>4</sup> See query 4 in appendix C.3 on page 372



## 7.2. ANALYSIS OF DATA GATHERED DURING STUDENT EXERCISES 149

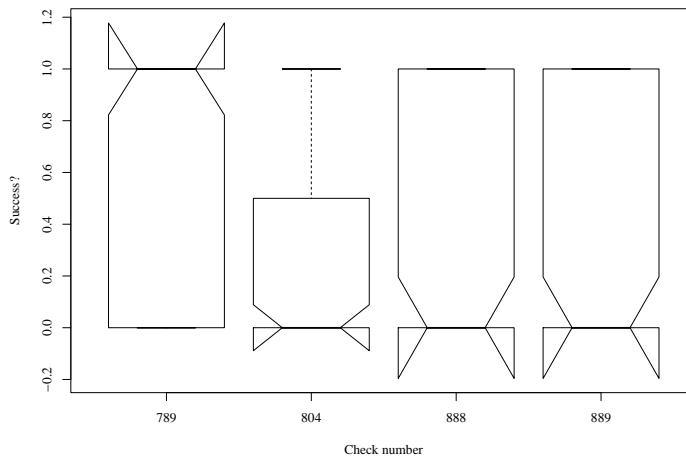


Figure 7.15: Results of unix-check-user-exists

processing time from 45 minutes to 15 minutes, to keep the 3 hour raster.

A change that should be made to the database structure of the Virtual Unix Lab is to record if an exercise was ended by the user or by a timeout, to reduce the need for heuristics to determine between “normal” exercise ends and those terminated by timeout.

### 7.2.6 Exercise time

The last aspect of the Virtual Unix Lab exercise results that is evaluated is the time of day that exercises were taken. The question that is expected to be answered here is, at what times students prefer (not) to exercise. This information could be used to schedule maintenance periods and other downtime.

Exercises in the Virtual Unix Lab can start every three hours, i.e. at 0am, 3am, 6pm, etc. Table 7.3 lists the start times and number of exercises that were started at the corresponding time<sup>1</sup>, figure 7.21 displays the histogram of the same data<sup>2</sup>.

As can be seen from figure 7.21, most of the exercises were performed in the afternoon and evening (12am to 9pm). Late evening and early morning (0am and 9pm) were less popular, and almost no exercises were booked during the later night and early

<sup>1</sup> See query 24 in appendix C.3 on page 382

<sup>2</sup> See query 25 in appendix C.3 on page 382

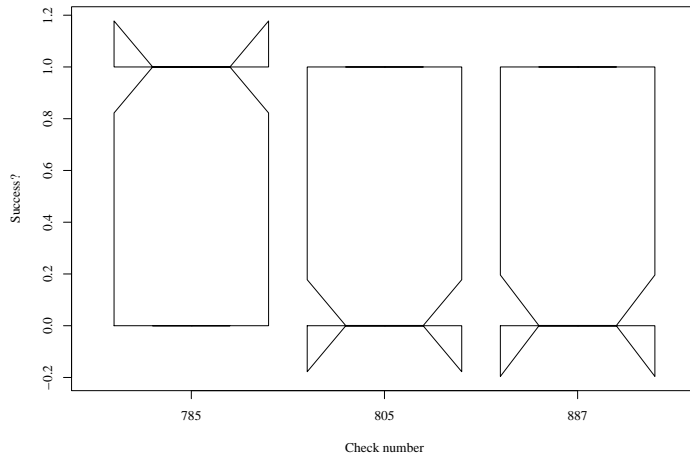


Figure 7.16: Results of check-directory-exists

morning hours. This information can be used to determine times for testing and system maintenance to not disturb students in their “regular” (preferred) practicing hours.

### 7.2.7 Summary

In this section, the data collected during exercises performed in the Virtual Unix Lab in summer 2004 and their results were observed under a number of aspects.

Looking at the frequency and results of the booked exercises showed that many users

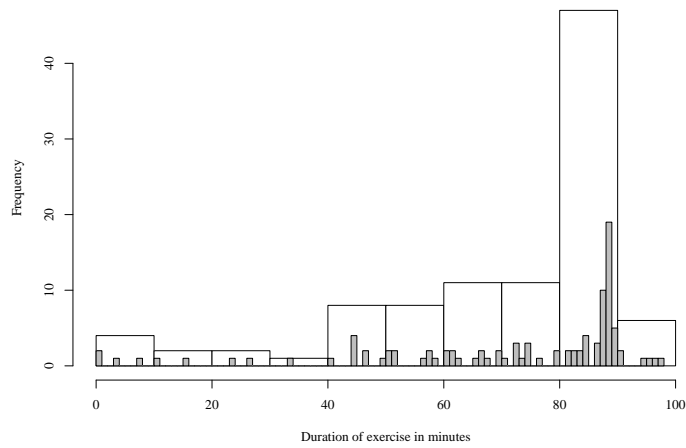
count	startzeit
7	00:00:00
1	06:00:00
11	09:00:00
24	12:00:00
27	15:00:00
32	18:00:00
27	21:00:00

(7 rows)

Table 7.3: Distribution of exercise start times

## 7.2. ANALYSIS OF DATA GATHERED DURING STUDENT EXERCISES 151

a)



b)

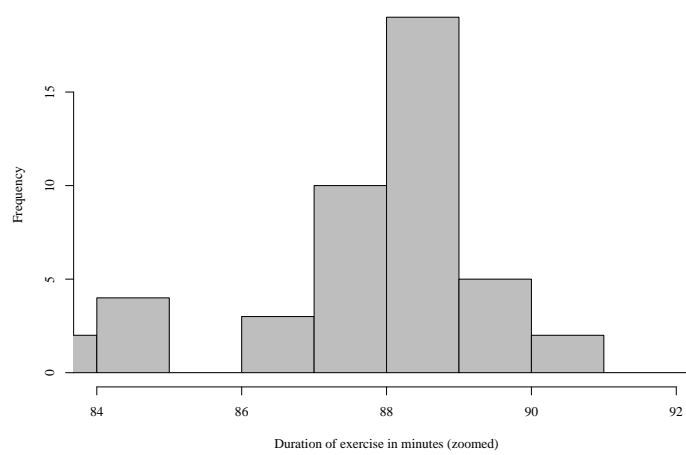
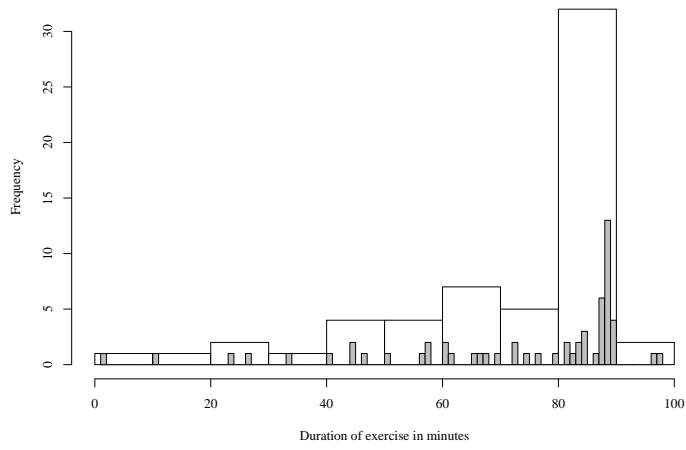


Figure 7.17: Duration of all exercises: a) overview and b) zoomed to the end of exercise

a)



b)

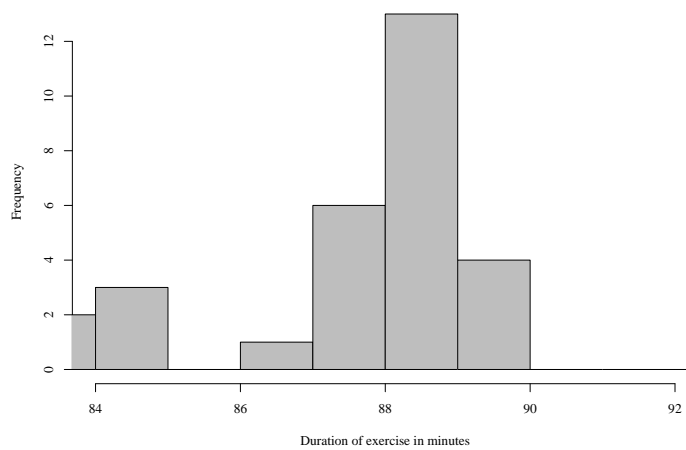
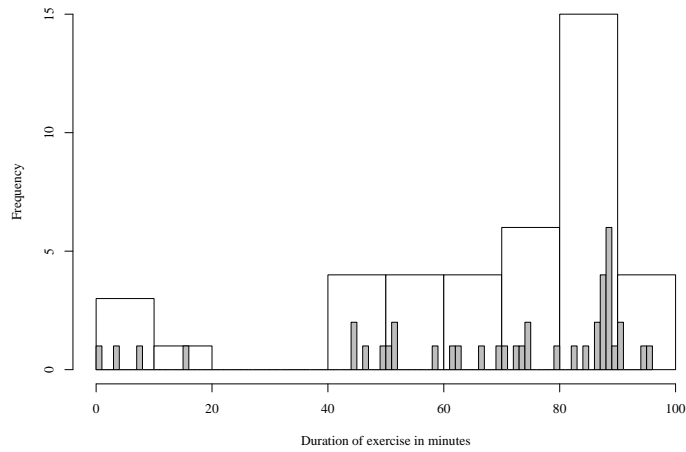


Figure 7.18: Duration of NIS exercises: a) overview b) zoomed to the end of exercises

## 7.2. ANALYSIS OF DATA GATHERED DURING STUDENT EXERCISES 153

a)



b)

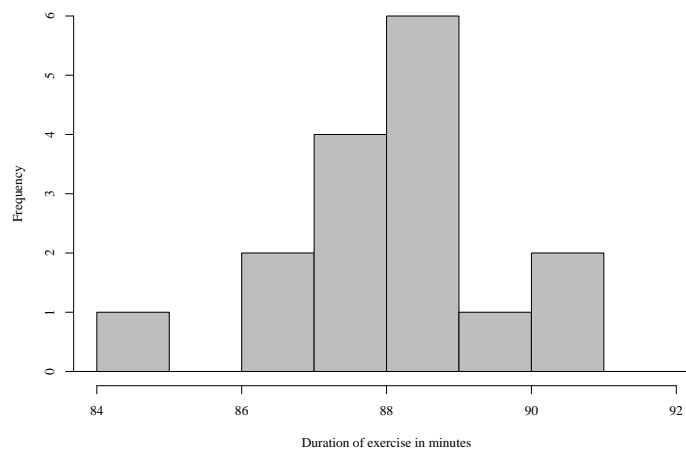


Figure 7.19: Duration of NFS exercises: a) overview b) zoomed to the end of exercises

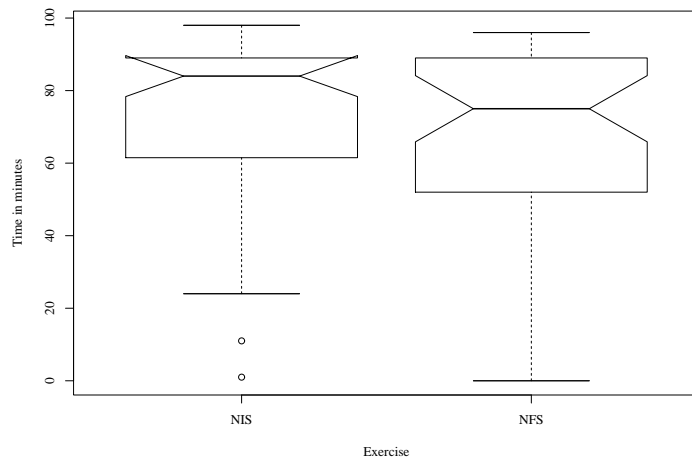


Figure 7.20: Comparison of durations of NIS and NFS exercises

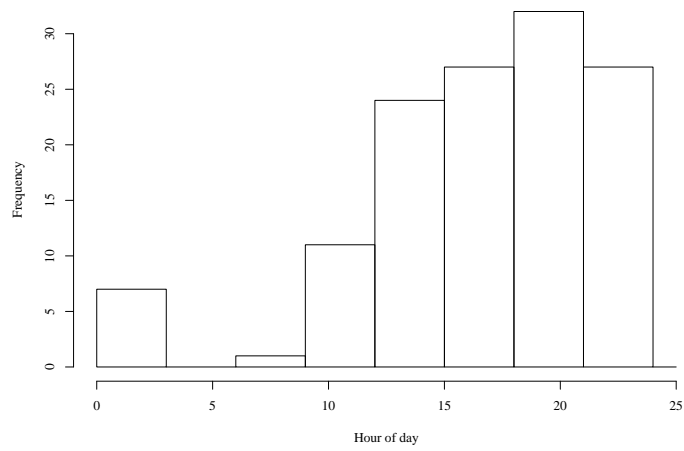


Figure 7.21: Starttime of exercises

booked the requested exercises more than once each. A direct connection between weak/strong performance in the first exercise and improved performance in the last exercise could not be seen. Possible reasons that weak results in the first exercise were not directly connected to (relatively) strong results in the last exercises may be due to problems in handling of the system or in understanding of the tasks requested to perform in the exercises. However, comparisons of the performance in students' first and last exercise showed that there was a significant improvement in overall performance for both NIS and NFS, which confirms that the request for a controlled testing environment from section 7.1 is valid, and that there seems to be a positive impact of the Virtual Unix Lab.

The investigation about solving of similar tasks in various exercises, as defined by the use of the same check script, revealed that a number of tasks were solved properly by most students, but that there are also a number of areas in which students need to get trained better or have better information available during the exercise.

Looking at the time and duration of exercises, the system is used least at 6am, which can be used e.g. as a maintenance window. In contrast, many exercises were performed in the afternoon, evening and night, which – in correspondence with the opening hour of the school – emphasizes the virtual component of the Virtual Unix Lab. The time available for students to take the NIS and NFS exercises is very tight, and offering longer exercises, for example 120 instead of 90 minutes, could make a difference. Another worthwhile change for future investigations would be to record if an exercise was aborted by timeout or by a student finishing the exercise.

### **7.3 Analysis of the user questionnaire**

Focus of the evaluation of the Virtual Unix Lab is to evaluate the system as a whole, and if students accept it as a useful aid in the learning process. To find out about students' acceptance and if they see a benefit in the Virtual Unix Lab, they were asked to fill out a questionnaire.

Performing a questionnaire was chosen due to the relatively low effort needed, because it does not influence students during the exercises. Besides the opinion of students about the Virtual Unix Lab, it shows how student cope with the course of exercises, gets details on students' use and preference of learning material, and learn about their overall motivations and background.

#### **7.3.1 Methodology of the questionnaire analysis**

Before describing the evaluation of the questionnaire's results in the next sections, this section gives an overview of the evaluated aspects, describes design and implemen-

tation of the questionnaire, and describes the methods used in the evaluation of the questionnaire.

### 7.3.1.1 Aspects evaluated by the questionnaire

Insight of the following aspects is expected from a survey taken by students who used the Virtual Unix Lab after the “System Administration” lecture:

**User acceptance:** The first question is, if users find the Virtual Unix Lab a usable addition to the teaching aids used in the “System Administration” class. Questions and answers to find out are discussed in section 7.3.2.

**Course of the exercise:** The question here was how students dealt with the exercises. This went from choosing the time of the exercises over accessing the Virtual Unix Lab and mastering the exercises to evaluation and feedback on the exercise results. The findings are discussed in section 7.3.3.

**Use of learning material:** The Virtual Unix Lab is intended to supplement the “normal” exercises as well as the lecture, but what other learning materials are popular among students? This question is answered in section 7.3.4.

**Target audience:** While it is known what semester the majority of students who used the Virtual Unix Lab were in, there is no direct connection from that to their knowledge and interests which is to be determined to better accustom lecture, lecture notes and practices. The results for these questions are discussed in section 7.3.5

### 7.3.1.2 Design and implementation of the questionnaire

The first step in conducting the questionnaire on users of the Virtual Unix Lab was to design it. Theories about questionnaires provide check lists to help during the design<sup>1</sup>. Decisions made for the questionnaire were to use a Web based approach for conducting. It was not split on multiple pages to prevent users aborting the survey before the final page. To encourage users to provide decisive answers, no options were included to voice “no opinion” in most of the cases. If a rank had to be assigned, odd numbers of options were avoided to prevent undecided users from taking “the middle way.” For easy evaluation, the use of “free form” text was avoided in favor of offering multiple choices. The questionnaire shown in appendix C.1 was used for the survey in the summer term 2004.

To realize the web-based survey, a software was needed to feed in the questionnaire, and get the HTML pages for the forms, database (table) setup as well as methods to

---

<sup>1</sup> [Bortz and Döring, 2002] pp. 244



retrieve the values for evaluation. No money was available for contracting the survey or buying commercial software, and the few available Open Source packages found to fulfill the needs were mostly based on different databases than used for the PostgreSQL database used for the Virtual Unix Lab. As an example, the “PHP Easy Survey Package” (phpESP) was HTML-based, but used MySQL as database<sup>1</sup>. As a solution, a processor “txt2survey” was written to transform the textual description of the questionnaire into the necessary HTML, PHP and SQL files.

Before asking students to perform the survey, a group of 20 “beta-testers” was chosen to perform a test of the survey and ensure that all key items were covered. The beta-testing group was selected to contain people with basic understanding of Unix system administration and the areas covered in the Virtual Unix Lab exercises, NIS and NFS. As the whole Virtual Unix Lab as well as the survey were designed to be in German language, proper understanding of that language was also a requirement for the testers.

After the beta testing period, students who attended the “System Administration” class in summer 2004 were asked to take the NIS and NFS exercise of the Virtual Unix Lab. Handing in printouts of feedback on both exercises and answering the questionnaire was made a mandatory pre-requirement for each student to pass the end-of-term test. This ensured that 28 out of 33 students who took the exercises filled out the questionnaire, even though some questions were left blank. The results are printed in appendix C.2, the various aspects will be discussed throughout this section.

### 7.3.1.3 Evaluation methods

Most data in the questionnaire asked is on an ordinal scale, the rest are on a nominal scale<sup>2</sup>. Due to this, care has to be taken when choosing the statistical methods used to analyze the results. In the following discussion, median and modus will be used. The median is used to determine which value has 50% of the results above and 50% of the results below it, and thus requires a definition of “above” and “below”, which can only be found on an ordinal, but not on a nominal scale. The modus is used with both scales to describe the answer which was chosen most often, on an absolute base<sup>3</sup>.

As common statistical methods for comparison require not ordinal but interval scaled values, they cannot be used directly to compare items on an ordinal scale. Ordinal scales define ordering of items, but not “distance” between them, which prevents applying methods for interval scaled data. By introducing an assumption of a certain “distance” between the values, it is possible to transform values from an ordinal scale to an interval scale, and thus be able to use methods of analytical statistics. In this discussion, the assumption is to assign fixed distances to values used to describe the learning materials. They will be used to calculate mean value and quartiles. Box-plots

---

<sup>1</sup> [phpESP, 2007]

<sup>2</sup> [Fahrmeir, 2003] pp. 17

<sup>3</sup> [Fahrmeir, 2003] pp. 53

are described in section 7.2.1, they are used to visualize the preference of learning materials.

For each aspect, results from various questions from the questionnaire are presented along with references to the exact questions and results in appendix C.2.

## 7.3.2 Evaluation of user acceptance

Evaluation of user acceptance is a major goal of the questionnaire, and as such, the relevant parts are discussed first here.

### 7.3.2.1 Questionnaire results

The first question asked to students was if they found the Virtual Unix Lab a reasonable supplement to the “system administration” lecture. Most students (15 out of 28) found it a very reasonable supplement, the remaining 13 students thought it was a reasonable supplement<sup>1</sup>.

Asking students if the system was easy to use, most (17 out of 28) agreed. From the remaining students, more found it to be cumbersome (7 out of 28) rather than very easy (4 out of 28)<sup>2</sup>.

When asked if the students felt a general benefit from the Virtual Unix Lab, most found the benefit as positive (15 out of 28), the majority of the remaining students (8 out of 28) found it as very positive, 4 students found it as neutral and only one felt a negative benefit<sup>3</sup>.

The last item of the questionnaire was a free-form field where students could write any comments they wanted. From the 13 students that used this opportunity, statements regarding user acceptance show that a two students found the exercise machines to be slow. Other than that, students wished that the Virtual Unix Lab machines were available for practicing various topics covered during the full time of the semester, and that new exercises be added for setup of firewalls, email, installation of software, and performing system updates. In general, several students indicated that they had fun practicing in the Virtual Unix Lab, and that it was a useful supplement to the existing lecture<sup>4</sup>.

---

<sup>1</sup> See question #9 in appendix C.2 on page 356

<sup>2</sup> See question #11 in appendix C.2 on page 356

<sup>3</sup> See question #10 in appendix C.2 on page 356

<sup>4</sup> See question #52 in appendix C.2 on page 369

### 7.3.2.2 Interpretation of the questionnaire results

Investigating user acceptance of the Virtual Unix Lab showed that students regard the system as a very good supplement to the existing lecture. They found it easy to use and that it had a positive benefit on them. This was confirmed by the wishes students expressed for using the Virtual Unix Lab for more than just two exercises, and having it available all the time as well. In general, students indicated having fun using the Virtual Unix Lab.

The only negative point noted here were slow exercise machines, which is no surprise, given that the machines run on 75MHz SPARC CPUs, while current Intel and compatible CPUs run at 3GHz. Possible solutions here would be to use faster machines or emulate the exercise machines, see section 2.9.

### 7.3.3 Evaluation of the course of exercises

To gather data about the course of the exercises performed in the Virtual Unix Lab, a number of questions were reserved in the questionnaire. Other methods, like supervising the exercises by an instructor and/or video, would have been possible in theory, but hard to implement, due to the fact that the students were intended to do the exercises at times and places of their choice. The questions discussed here are in the order of the exercise process.

#### 7.3.3.1 Questionnaire results

Most students (26 of 28) found that there were enough dates available for exercising. Only two students found that there were too few<sup>1</sup> possible dates.

After booking, most students (20 out of 28, 71%) were using their home machines to access the Virtual Unix Lab for practicing, while the remaining 7 students (28%) were using school machines. No student indicated doing the exercises from another place (e.g. from a company they were working at)<sup>2</sup>. Most students absolved the exercises on their own (17 out of 28), 5 of them were in groups of two and 5 in groups of three students – apparently not everyone filled out the questionnaire<sup>3</sup>.

When asked if the setup of the machines for the exercises was adequate, no student found it to be too spartan. Many (11 out of 28) thought of it as slightly spartan, and the majority of 14 out of 28 considered it acceptable. Only three students characterized

<sup>1</sup> See question #12 in appendix C.2 on page 357

<sup>2</sup> See question #13 in appendix C.2 on page 357

<sup>3</sup> See question #16 in appendix C.2 on page 358

the setup as “comfortable”<sup>1</sup>. The instructions which described the exercise to perform that were given to students were found to be too much by only two students out of 28. The information was exactly right for 9 students, and the majority (16 out of 28) of the students have wished for more information<sup>2</sup>.

During the exercise, a majority of students (19 out of 27) wished they had a chance to ask for more help<sup>3</sup>. Even more students (22 out of 27) wished the system would have detected problems automatically, and provided appropriate assistance in that case<sup>4</sup>.

The time reserved for practice – 90 minutes for the NIS as well as the NFS exercise, each – was “too short” for most of the students (15 out of 27), one student found the time “much too short”, and for 11 out of 27 students the time was “just right”<sup>5</sup>.

When asked if the feedback given after exercises was detailed enough to understand mistakes made, about two third of the students (17 out of 26) were able to learn from their mistakes, while the remaining 9 students still were not sure about what they did wrong<sup>6</sup>.

In the field reserved for giving free-form feedback at the end of the questionnaire, several students asked for more time and information to solve the exercises. Also, more information was requested by a few students for the feedback after the exercises, esp. for tasks that were not solved successfully<sup>7</sup>.

### 7.3.3.2 Interpretation of the questionnaire results

The schedule of exercises being available in a three-hour pattern was accepted by most students. Most of them used the “virtual” component of the Virtual Unix Lab to make the exercises from a location of their choice, instead of being physically present at school. A similar number of students solving the exercises alone instead of in groups may lead to the conclusion that students working from their homes did them alone could not be found true when examining a correlation between these results<sup>8</sup>.

The exercise machines’ setup was rather spartan when comparing the NetBSD and Solaris installation to e.g. modern Linux distributions like SuSE, which students were expected to be most familiar with. As a result, it was expected that students would find the installation of the exercise machines rather spartan and inadequate for performing the exercises. The results show that most students found the setup acceptable

<sup>1</sup> See question #15 in appendix C.2 on page 357

<sup>2</sup> See question #18 in appendix C.2 on page 358

<sup>3</sup> See question #19 in appendix C.2 on page 359

<sup>4</sup> See question #20 in appendix C.2 on page 359

<sup>5</sup> See question #17 in appendix C.2 on page 358

<sup>6</sup> See question #41 in appendix C.2 on page 366

<sup>7</sup> See question #52 in appendix C.2 on page 369

<sup>8</sup> The Spearman correlation coefficient between results in question #13 and question #16 was found to be -0.52.

or at most slightly spartan, which indicates that the assumptions made about students' expectations were wrong, in favor of the default installation provided.

An area where work is needed from the teacher's side are the instructions provided on the exercise to be performed, as students wished for more information here. Care must be taken when addressing that point to not give away too much of the solution to the exercises.

The need for more information and help was also expressed by students. They wished to either request more help manually, or have the system automatically detect situations where intervention was needed, and provide help in those situations.

Statements of students that the time for exercises was too short indicate similar problems: The tasks to perform do not take up much time when understood and all the needed procedures and commands needed to run are known. Problems in understanding the objectives and how to reach them – searching for information and applying theories and concepts – cost time, which students seem to lack, as indicated by previous observations. More practical exercises may be appropriate to make students more familiar with methods for practical problem solving.

Finally, while a majority of students were able to learn from the feedback given to them after the exercises, one third of the students needed more help and explanations to understand what they did wrong to not meet the exercise goals. More elaborated feedback than the one-line summaries given could help here. Possible help could point at descriptions of the scenarios, procedures to apply in the lecture notes, and lists of useful commands for setup and troubleshooting for the particular problem.

In summary, it seems students need more information during the exercises to understand what their tasks are and how to solve them. They also want more data given on feedback to learn from errors.

### **7.3.4 Evaluation of the use of learning material**

Students were provided with a wide range of learning materials for the class, as described in section 3.2.4. This section discusses the part of the questionnaire that attempted to find out what material was preferred by students.

#### **7.3.4.1 Impact of learning materials in general**

The first set of questions asked what source of information the student used most to learn about the topic of system administration. Possible answers were

1. The “SA” lecture<sup>1</sup>
2. The lecture notes for the “SA” lecture<sup>2</sup>
3. Practical exercises accompanying the “SA” lecture<sup>3</sup>
4. The Virtual Unix Lab<sup>4</sup>
5. Analyzing school machines<sup>5</sup>
6. Analyzing own machines<sup>6</sup>
7. Books<sup>7</sup>
8. Online information<sup>8</sup>

For each source of information, the students were asked to indicate how much they learned from it. The possible answers and their assumed weight as discussed in section 7.3.1.3 were:

- 4 = A lot (“Sehr viel”)
- 3 = Some (“Einiges”)
- 2 = Average (“Geht so”)
- 1 = Few (“Wenig”)
- 0 = Nothing (“Nichts”)

Figure 7.22 displays the distribution of the learning materials’ popularity to allow a comparison with box-plots as described in section 7.2.1. The columns on the  $x$ -axis show the learning materials, while the  $y$ -axis displays their popularity as described above.

It is obvious that most students did not like to read books (item #7 in figure 7.22), and that analysis of school machines (#5) was not very popular either. The most popular sources of information where students learned most about system administration were visiting the “SA” lecture (#1) and analyzing of the students’ own machines (#6). The remaining items were equally popular, with the Virtual Unix Lab among them.

<sup>1</sup> See question #1 in appendix C.2 on page 353

<sup>2</sup> See question #2 in appendix C.2 on page 354

<sup>3</sup> See question #3 in appendix C.2 on page 354

<sup>4</sup> See question #4 in appendix C.2 on page 354

<sup>5</sup> See question #5 in appendix C.2 on page 355

<sup>6</sup> See question #6 in appendix C.2 on page 355

<sup>7</sup> See question #7 in appendix C.2 on page 355

<sup>8</sup> See question #8 in appendix C.2 on page 355

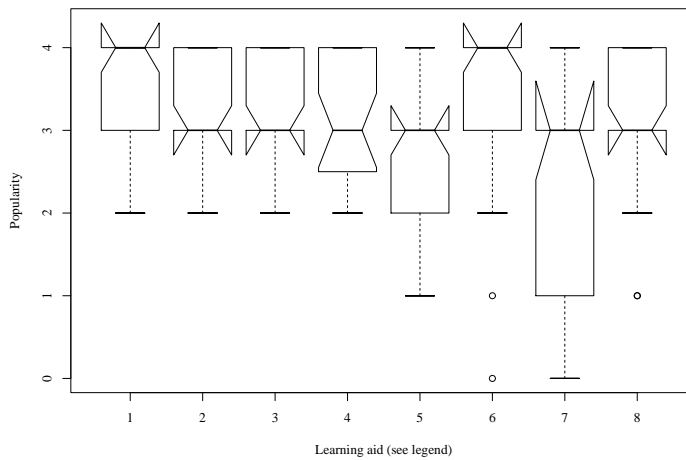


Figure 7.22: Popularity of learning materials among students

#### 7.3.4.2 Impact of learning materials during Virtual Unix Lab exercises

The next set of questions asked was how much any of the following learning materials helped students solve the exercises in the Virtual Unix Lab:

1. The “SA” lecture<sup>1</sup>
2. Lecture notes for the “SA” lecture<sup>2</sup>
3. Practical exercises<sup>3</sup>
4. Analyzing school machines<sup>4</sup>
5. Analyzing own machines<sup>5</sup>
6. Books<sup>6</sup>
7. Online information<sup>7</sup>

<sup>1</sup> See question #21 in appendix C.2 on page 359

<sup>2</sup> See question #22 in appendix C.2 on page 360

<sup>3</sup> See question #23 in appendix C.2 on page 360

<sup>4</sup> See question #24 in appendix C.2 on page 360

<sup>5</sup> See question #25 in appendix C.2 on page 361

<sup>6</sup> See question #26 in appendix C.2 on page 361

<sup>7</sup> See question #27 in appendix C.2 on page 361

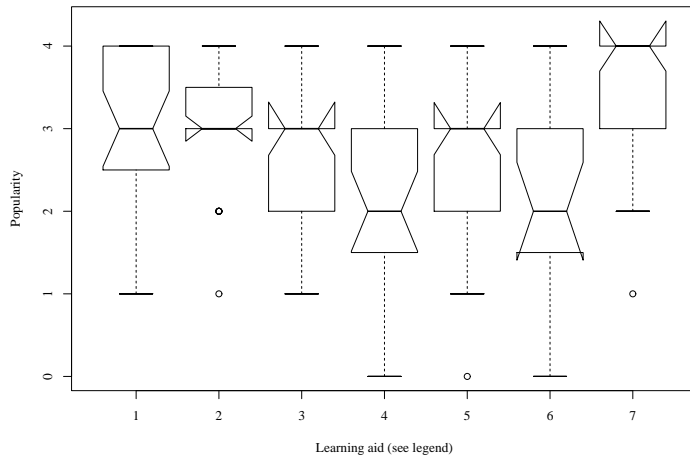


Figure 7.23: Helpful learning material in the Virtual Unix Lab

For each learning material, students were asked to indicate how much the resource helped them. The possible answers and their numbers were:

- 5 = Not used (“Nicht genutzt”) – not included in plot
- 4 = A lot (“Sehr viel”)
- 3 = Some (“Einiges”)
- 2 = Average (“Geht so”)
- 1 = Few (“Wenig”)
- 0 = Nothing (“Nichts”)

Figure 7.23 shows that the most popular medium that was used during exercises in the Virtual Unix Lab were online information (item #7). This is followed by knowledge gained in the lecture (#1) and the lecture notes (#2), practical exercises performed by students outside the Virtual Unix Lab (#3) as well as analysis of own machines (#5) were less popular, and analysis of school machines (#4) and books (#6) were least used.



### 7.3.4.3 Impact of the “SA” lecture for exercises in the Virtual Unix Lab

After observations of the various learning materials, special emphasis was given to effect of the “SA” lecture and the lecture notes. The next set of questions asked how much visiting the lecture helped during exercises in the Virtual Unix Lab, in particular for a number of different tasks:

1. NIS server setup<sup>1</sup>
2. NIS client setup<sup>2</sup>
3. NFS server setup<sup>3</sup>
4. NFS client setup<sup>4</sup>
5. Handling of Solaris in general<sup>5</sup>
6. Handling of NetBSD in general<sup>6</sup>
7. General problem solving<sup>7</sup>

For each topic, students were asked to indicate how much the lecture helped them. Again, numerical values are assigned to allow employing statistical methods for comparison. The possible answers and their numbers were:

- 4 = A lot (“Sehr viel”)
- 3 = Some (“Etwas”)
- 2 = Average (“Geht so”)
- 1 = Few (“Wenig”)
- 0 = Nothing (“Nichts”)

One observation from figure 7.24 is that no student indicated he learned nothing from the lecture for any of the topics, because it is not present on the scale. The median of all results shows that students consider having learned above-average skills for all areas, with the most impact coming from the lecture, i.e. a tendency towards having learned a lot, in the topics of NFS client setup (item #4) and handling of Solaris (#5) and NetBSD (#6) in general.

<sup>1</sup> See question #28 in appendix C.2 on page 362

<sup>2</sup> See question #29 in appendix C.2 on page 362

<sup>3</sup> See question #30 in appendix C.2 on page 362

<sup>4</sup> See question #31 in appendix C.2 on page 363

<sup>5</sup> See question #32 in appendix C.2 on page 363

<sup>6</sup> See question #33 in appendix C.2 on page 363

<sup>7</sup> See question #34 in appendix C.2 on page 364

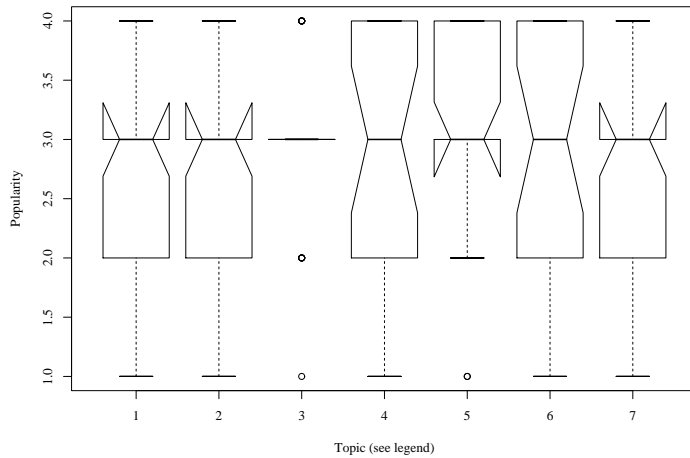


Figure 7.24: Impact of the “SA” lecture on various topics of the Virtual Unix Lab exercises

#### 7.3.4.4 Impact of the “SA” lecture notes for exercises in the Virtual Unix Lab

After asking about the impact of the lecture, the importance of the lecture notes during the Virtual Unix Lab exercises on the same areas were asked:

1. NIS server setup<sup>1</sup>
2. NIS client setup<sup>2</sup>
3. NFS server setup<sup>3</sup>
4. NFS client setup<sup>4</sup>
5. Handling of Solaris in general<sup>5</sup>
6. Handling of NetBSD in general<sup>6</sup>

<sup>1</sup> See question #35 in appendix C.2 on page 364

<sup>2</sup> See question #36 in appendix C.2 on page 364

<sup>3</sup> See question #37 in appendix C.2 on page 365

<sup>4</sup> See question #38 in appendix C.2 on page 365

<sup>5</sup> See question #39 in appendix C.2 on page 365

<sup>6</sup> See question #40 in appendix C.2 on page 366

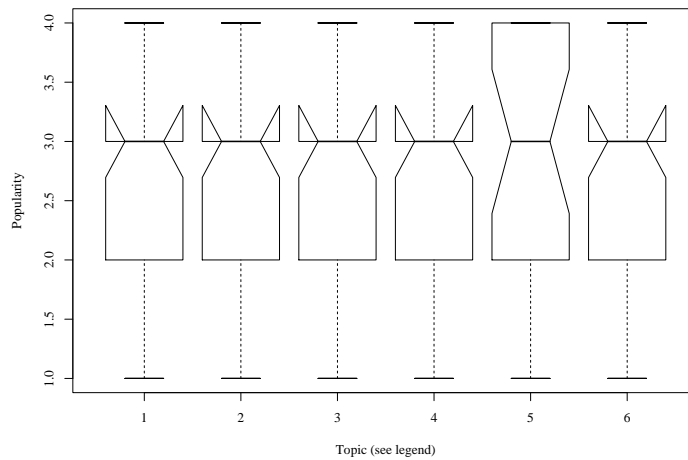


Figure 7.25: Impact of the “SA” lecture notes on various topics of the Virtual Unix Lab exercises

For each topic, students were asked how much the lecture notes helped them. Numerical values are assigned to allow employing statistical methods for comparison. The possible answers and their numbers were:

- 4 = A lot (“Sehr”)
- 3 = Some (“Etwas”)
- 2 = Average (“Geht so”)
- 1 = Few (“Wenig”)
- 0 = Nothing (“Nichts”)

Figure 7.25 shows that the script was regarded as providing above-average help for all areas, with a tendency towards a lot of information for the Solaris operating system (item #5).

#### 7.3.4.5 Interpretation of the questionnaire results

This section made observations about the use and preference of learning materials of students in general, and for the exercises in the Virtual Unix Lab in particular.

In general, students seem to be more interested in reading information online than from books. Instead of analyzing school machines, which can be assumed to be properly configured for the various tasks, students use their own (home) machines to learn. The “System Administration” (SA) lecture seems to be considered a vital source of information for practical exercises, and the lecture notes (which are available online!) as well as other online information are preferred for the practical exercises in the Virtual Unix Lab. Students confirmed this by stating that the Internet was the #1 reference material used during the Virtual Unix Lab exercises<sup>1</sup>. The reason for this may be that information is quicker and easier to search and obtain when the exact location is not known. A similar case is that students prefer analyzing (and probably configuring) their own machines over the school machines. It can be assumed that they knew their own machines better, plus they had the credentials to change the configuration there, in contrast to the rather unknown school machines, where they were not even allowed to tune the configuration. More investigation could be done on this topic, which would go beyond the scope of this investigation.

Visiting the lecture is considered to be important by most students, as it helps somewhat in the various practical tasks that are part of the Virtual Unix Lab. The lecture script is considered a good source of information, too, especially when it comes to the Solaris operating system. Possible improvements that could be made are in the areas of NFS client setup as well as general handling of NetBSD. The question if the demand for NFS client setup was influenced by the demand for general NetBSD documentation in the lecture notes would need separate investigation, and goes beyond the scope of this work.

Another point that may be worth looking into is if the style of online information – usually short, non-prose and keyword-type style – is more appropriate for system administration than the style of books – prose with introduction, more detailed description of problems and solutions, and possibly exercises and references. Different areas to observe are material for acquiring knowledge about concepts and theories in contrast to reference material and their usage during the course of the learning cycle.

### 7.3.5 Evaluation of the target audience

Finding out more information about background knowledge and motivation of the students participating in the “System Administration” lecture was one of the goals of the questionnaire which students were asked after completing the NIS and NFS exercise in the Virtual Unix Lab.

---

<sup>1</sup> See question #52 in appendix C.2 on page 369

### 7.3.5.1 Questionnaire results

An absolute majority (21 of 27) of the students had “very big” interest in their study subject, while the remaining 6 students indicated a “big” interest<sup>1</sup>. Interest in Unix(like) operating systems like Linux, Solaris and NetBSD was indicated as “very big” by 16 of 27 students, and with the exception of one student who was only moderately interested, all other students indicated “big” interest<sup>2</sup>. In contrast, interest in the “System administration” topic was a bit lower. While most students (15 of 27) still indicated “very big” interest, 7 indicated “big” interest and 5 were only moderately interested<sup>3</sup>. Asking about the number of lectures visited, 23 of 27 (85%) students visited 9-10 lectures out of ten, most others (11%) came to 4-8 out of ten lectures, and only one student went to 0-3 out of ten lectures<sup>4</sup>.

The operating system that students used to start the exercise was some Unix-variant for most of the students (21 out of 28) while only 7 used Windows<sup>5</sup>.

To learn more about students’ interest in specific topics that were covered in detail in class as well as in the Virtual Unix Lab, they were queried how they estimated the importance of the “Network File System” (NFS) and “Network Information System” (NIS). Results show that NFS is considered “big” by most (14 out of 27) students with a tendency towards moderate interest by a large part (8 of 27) of the remaining group<sup>6</sup>. For NIS, importance was only considered as “moderate” by most of the students (12 of 27), with an equal number of students considering its importance as “big” (6 of 27) and “less” (also 6 of 27)<sup>7</sup>.

Querying the students if they had prior experience with system administration, e.g. during internships or from home usage, the majority (22 of 27) of the students did have prior experience<sup>8</sup>. To find out what operating system students administrated most, they were asked to name the operating system they used most, with only one answer possible to focus on the system they had the most experience with. Answers showed that most of the students had administrative experience with Linux (16 of 23). 5 Students had experience with Windows, and one student had worked with Solaris and Novell each<sup>9</sup>.

The students who participated in the survey were mostly in their 4th semester (23 out of 27), one was below the 4th semester, and three students were in semesters 8 or above<sup>10</sup>. Gender distribution among students was 96% (26 of 27) male, and 4% (1 of

<sup>1</sup> See question #42 in appendix C.2 on page 366

<sup>2</sup> See question #44 in appendix C.2 on page 367

<sup>3</sup> See question #43 in appendix C.2 on page 367

<sup>4</sup> See question #47 in appendix C.2 on page 368

<sup>5</sup> See question #14 in appendix C.2 on page 357

<sup>6</sup> See question #46 in appendix C.2 on page 368

<sup>7</sup> See question #45 in appendix C.2 on page 367

<sup>8</sup> See question #48 in appendix C.2 on page 368

<sup>9</sup> See question #49 in appendix C.2 on page 368

<sup>10</sup> See question #50 in appendix C.2 on page 369

27) female.

### 7.3.5.2 Interpretation of the questionnaire results

Comparing the interest of students in their studies in general, in Unix(like) operating systems, and in system administration in particular, it seems that they have less interest in the subject of system administration than in the other subjects. This may be due to the fact that the SA lecture is mandatory to all students, in contrast to the SY lecture which used to be available to volunteer students. This may have an impact on motivation of students and test result, as was found in section 7.2.

The lack of interest may be due to the selection of topics covered. The fact that the students consider advanced topics like that of networked filesystems (NFS) and system management in a distributed environment (NIS) as moderately important supports this assumption.

If students were confronted with system administration outside the lecture previously, it was more often with Linux systems than with Windows. Reasons that would need further investigation (but are outside the scope of this work) could be that Linux systems need more “administration” than other (Windows) systems, or that students are more interested in setting up and tuning single-user workstations than being interested in advanced topics like the management of workstation clusters discussed in the “System Administration” (SA) lecture.

Most students participating in the questionnaire were in their 4th semester, where the lecture is mandatory for students of (General) Computer Science (“Allgemeine Informatik”). A few students were from higher semesters. From talking to these students, it can be said that they were not studying (General) Computer Science, but Computer Science with either technical or economical emphasis, and that they took the system administration course voluntarily.

The ration of male to female students was typical for technical study courses.

### 7.3.6 Summary

Evaluation of the online questionnaire confirms many of the approaches taken in the Virtual Unix Lab described so far, and also what future improvements can be made in each of the observed areas:

**User acceptance:** Students found the Virtual Unix Lab easy to use, that it is a reasonable supplement of the existing “System Administration” (SA) lecture, and that use of the Virtual Unix Lab has an overall positive benefit.

Students requested to keep the Virtual Unix Lab running permanently to allow doing investigations and testing configuration when needed, including system privileges. This request could be easily carried out.

Other requests from students to create and offer more exercises in the Virtual Unix Lab as well as making the machines faster would require more effort. Human resources and funding are required for creating new exercises, and upgrading machines needs changes to the Virtual Unix Lab's software, especially for the automatic setup and preparation of exercise machines. In addition, funding would be needed for hardware purchases and associated software changes.

**Course of the exercise:** The number of possible dates for exercises were enough. The duration of exercises could have been longer, as was already observed in section 7.2.5.

Moving from the current 45+90+45 scheme for exercises, which uses 45 minutes for preparing the exercise machines, 90 minutes for the exercise and 45 minutes for postprocessing, i.e. the evaluation of the lab machines, would give chances for either more or longer exercises. For example, going to a 30+90+15 scheme would allow ten exercises per day, 30+90+10 would allow eleven exercises per day, or 45+120+15 would allow eight exercises as right now (which was found OK by students), but allow longer exercises. Preparation and postprocessing would need to be kept in bounds of the limits set by the hardware (for preparation) and the exercise (for postprocessing). The above 45+120+15 scheme would work for the existing setups, assuming exercises do not hang the system for postprocessing.

Most students used the "virtual" component of the Virtual Unix Lab and accessed it from home.

More information should be provided with the instructions on the Virtual Unix Lab exercises, without giving away too much of the solutions. More information should be provided in the feedback given to students after exercises, esp. on tasks that were not completed successfully.

If the student needs more information during the exercise, it should be available on request. As an alternative, instead of having the student ask for help, the system could monitor the progress, and detect that help is needed or if a situation is critical, and offer appropriate help automatically. See chapter 10 for more ideas in that direction.

**Use of learning materials:** Students prefer online information to reading books. Students also prefer their home machines for analysis over school machines, despite the fact that the latter are known to be properly configured for tasks, whereas this is unknown for the former.

Visiting the "System Administration" (SA) lecture is considered important by students, and the lecture notes are used to a great extent for the practical exercises in the Virtual Unix Lab.

Further studies in this area could compare the use of school vs. students' machines as passive vs. active learning materials. Another investigation could be to evaluate the impact of existing online vs. offline (i.e. book) information on topics like learning of concepts and theories, in contrast to reference material for the topic of system administration.

**Target audience:** Students have great interest in their studies and Unix(like) operating systems in general. Interest in system administration is a bit less. An important factor for this may be that the class is mandatory.

The question if students are less interested in system administration and more interested in using systems to perform tasks not related to the machine and operating system configuration could be the work of future research, but is outside the scope of this document.

To conclude, the questionnaire showed overall user acceptance of the Virtual Unix Lab as evaluated, but also that there is a demand for more information in the exercise description, during the exercises, and when giving feedback after the exercises.

## 7.4 Other aspects to evaluate

There are a number of aspects under which the Virtual Unix Lab could be evaluated. While none of these evaluations is carried out as part of this work, they may lead to overall benefits for the users of the Virtual Unix Lab.

**Quality of the VUDSL:** One major component that the original version of the Virtual Unix Lab lacked and that was described in the previous chapters is the DSL for the verification of exercise results. The question about the quality of that language arises. Along with that, the question on how to establish the "quality" of a language arises, though.

Literature on creating languages is scarce, and the situation that judges the quality of languages is very similar, unfortunately. No established methods were found by which to judge how "good" the VUDSL is. Possible metrics to apply could include maintainability, scalability/extendability and tracability of the programming language. Similar metrics may be applied not only to the language (and its manifestation in various programs as exercise texts and their verifications), but also to the processor of the VUDSL itself.

Dijkstra suggests that a major goal of designing a programming language should be that its functions can be verified. For that, a language should offer a "small number of concepts, the more general the better, the more systematic the better."<sup>1</sup> This emphasis on verifiability can also be found in modern approaches

---

<sup>1</sup> [Dijkstra, 1961] p. 4



to software engineering, e.g. the V-model specifies use and test cases for each software feature before looking further at the implementations of the software features and the tests<sup>1,2</sup>.

A practical approach to determine the quality of the VUDSL could be to sit down and write many exercises and the corresponding tests, use the existing stereotypes (check scripts) and possibly refine them. New language features as mentioned in section 6.8 could be added as need arises, and the overall quality of the VUDSL and its processor could be judged by how well they support those extensions. This approach is tedious, time consuming, and (most likely) incomplete, and as such not recommended.

No further evaluation attempt at evaluating the “quality” of the VUDSL is made at this point. The extensions proposed in section 11.6 give hope that the basic system is extensible and scalable within reasonable amounts of maintenance.

**Mobile education:** Given its “virtual” nature, the Virtual Unix Lab can be accessed from anywhere, which was very much accepted by students, as found in the above evaluation. Accessing the Virtual Unix Lab is not only possible from “traditional” access devices for Unix – PCs, workstations or even dialup-terminals– but also from mobile devices like PDAs, cell and smart phones. A number of restrictions still apply for such “mobile” nodes which do not affect the access devices commonly used, and problems with format and formatting of the Virtual Unix Lab’s user interface can be expected.

A more in-depth discussion of the issues and challenges of mobile education can be found in [Nösekabel, 2005].

**Accessibility:** A number of considerations are needed to adjust software and user interfaces to be accessible. Use of color, size of fonts, layout of user interface components, use of language, using keyboard and mouse alternatively are just a few examples given in many guidelines that are intended to make software accessible through various laws<sup>3,4,5</sup>, general accessibility standards<sup>6,7</sup> and a rich choice of software interfaces and style guides<sup>8,9,10,11</sup>.

While the benefit of making software accessible to those that depend on it is recognized as important, no effort in that direction is made in this document with respect to the Virtual Unix Lab. It is left to future works to evaluate, judge and/or improve the existing situation of the Virtual Unix Lab.

---

<sup>1</sup> [iABG, 2007]

<sup>2</sup> [Versteegen, 2001]

<sup>3</sup> [Government of the United Kingdom, 2001]

<sup>4</sup> [BGG, 2002]

<sup>5</sup> [Thomas, 2000]

<sup>6</sup> [ISO 16071, 2003]

<sup>7</sup> [BITV, 2002]

<sup>8</sup> [The KDE Project, 2007]

<sup>9</sup> [Trolltech, 2007]

<sup>10</sup> [The GNOME Project, 2007]

<sup>11</sup> [World Wide Web Consortium, 2007]

**Security:** While the system was designed with security in mind, it was never evaluated under that aspect. Areas that could be observed in such an evaluation are the web based user interface, its implementation<sup>1</sup>, and precautions against users breaking out of the exercise lab's network into the production network to which the Virtual Unix Lab is connected<sup>2</sup>. Methods for evaluation could range from code audits<sup>3,4,5</sup> over network audits and penetration tests<sup>6,7,8,9</sup> to general practices of network and system security<sup>10,11,12</sup>.

**Privacy:** As an aspect of security, privacy of the system, its users, and their data is not evaluated per se, as stated above. But in the context of user modeling, privacy is a concern, and while no full audit is performed, related issues are discussed in section 8.2.

**Usability:** The existing user interface was taken as part of the preliminary work on the Virtual Unix Lab done for the HWP project "Practical Unix cluster setup", and the design goals were modeled to the functional requirements there<sup>13</sup>.

An evaluation of the existing user interface could identify potential improvements for user guidance in general, and how to realize improved user guidance for tutoring in particular. While the latter is further discussed in chapter 10, a full evaluation of the user interface and dialog structures of the Virtual Unix Lab under usability aspects are considered outside of the scope of this work.

Possible methods for further research would be focus groups, expert reviews, personas and usage scenarios, among others<sup>14,15,16</sup>.

Check lists for desirable goals in user interface design can be found in a number of standards like ISO 9241<sup>17</sup> and the VDE 5005 standard for "software-ergonomics in office communication"<sup>18</sup>. They describe information presentation<sup>19</sup>, dialog guidance via on-screen forms<sup>20</sup>, fundamentals of dialog design<sup>21</sup>,

---

<sup>1</sup> [Zimmermann, 2003] pp. 9

<sup>2</sup> [Feyrer, 2004d]

<sup>3</sup> [Heffley and Meunier, 2004] pp. 90278

<sup>4</sup> [Hill, 1988] pp. 291

<sup>5</sup> [Huang et al., 2004] pp. 45

<sup>6</sup> [Lytle et al., 2005] pp. 197

<sup>7</sup> [McNab, 2004] pp. 57

<sup>8</sup> [nmap, 2007]

<sup>9</sup> [Nessus, 2007]

<sup>10</sup> [Herold, 2005] pp. 1

<sup>11</sup> [Schneier, 2005] pp. 1

<sup>12</sup> [Trček, 2005] pp. 43

<sup>13</sup> [Zimmermann, 2003] pp. 9

<sup>14</sup> [Jakob Nielsen, 1997] pp. 94

<sup>15</sup> [Gibbs, 1997]

<sup>16</sup> [Shneiderman, 2004] pp. 139-172

<sup>17</sup> [ISO 9241, 2003] pp.37

<sup>18</sup> [VDI-Gesellschaft Entwicklung Konstruktion Vertrieb, 1990]

<sup>19</sup> [ISO 9241, 2003] ISO 9241-12, pp. 111

<sup>20</sup> [ISO 9241, 2003] ISO 9241-17, pp. 227

<sup>21</sup> [ISO 9241, 2003] ISO 9241-10, pp. 81

and user guidance<sup>1</sup>.

Full evaluation of all the details addressed in these standards is a lot of work, even if these standards do provide their own checklists for easier testing and evaluation. Other approaches to perform these evaluations would be to employ evaluation methods like IsoNorm<sup>2</sup>, IsoMetric<sup>3</sup>, the Questionnaire for User Interface Satisfaction (QUIS) by Shneiderman, Slaughter and Norman<sup>4</sup>, the System Usability Scale (SUS) by Brooke<sup>5</sup>, the Web Usability Index<sup>6</sup> and others<sup>7,8</sup>, in addition to the existing guidelines for usability design and engineering<sup>9,10</sup>.

**User Guidance:** An analysis of the dialog structure of the Virtual Unix Lab could be performed to identify places in the user interface that could be improved for better handling by the users as well as adding components to introduce active user guidance. Such active user guidance could include user modeling, tutoring and user adaption.

An analysis of the Virtual Unix Lab's user interface under these aspects as well as investigations on how to realize a tutoring component and user adaption will be given in chapters 10 and 11.

## 7.5 Conclusion of the evaluation

After observing several aspects of the existing Virtual Unix Lab, this section draws a conclusion on the evaluations performed, i.e. about the Virtual Unix Lab exercise results in section 7.2.7 and the results from the questionnaire in section 7.3.6.

Examining the results of students who repeated an excises in the Virtual Unix Lab more than once, and comparing their first and last results did show some significant improve in performance. No Areas where the learning experience can be improved are partly possible within today's incarnation of the Virtual Unix Lab, and partly need deeper changes to it. Among the items that can easily achieved are making exercises longer (e.g. 120 instead of 90 minutes) and offering more exercises.

Giving more information to students during exercises was requested repeatedly. To supply more information for assistance of the exercise requires a change in the current model of the exercise procedure, and thus a change in the Virtual Unix Lab itself.

<sup>1</sup> [ISO 9241, 2003] ISO 9241-13, pp. 148

<sup>2</sup> [Prümper and Anft, 2006] "Fragebogen ISONORM 9241/10"

<sup>3</sup> [Gediga et al., 1999] pp. 151

<sup>4</sup> [Harper and Norman, 1993] pp. 224

<sup>5</sup> [Brooke, 1996]

<sup>6</sup> [Harms et al., 2002]

<sup>7</sup> [UsabilityNet, 2007] "Questionnaire ressources"

<sup>8</sup> [Baseline, 2007] "Frequently Asked Questions about User Validation: Questionnaires"

<sup>9</sup> [Nielsen, 2001]

<sup>10</sup> [Nielsen, 1994]

Possible areas for providing better information and assistance are a tutorial component and user adaption, which are discussed in the following chapters, 9, 10 and 11.

## **Part III**

### **Tutoring and user adaption**



# Chapter 8

## Introduction of tutoring and user adaption

The previous parts of this work have introduced the Virtual Unix Lab in general, and how verification of exercise results can be realized with the help of a domain specific language. The third part of this work describes how the foundation laid out so far can be used to add tutoring and adaption to the Virtual Unix Lab.

This chapter covers the fundamentals for tutoring and user adaption that are used for defining corresponding components in the Virtual Unix Lab.

### 8.1 Fundamentals of tutoring

Using computers to help in teaching is old, and has grown a number of related acronyms like Computer Aided Instruction (CAI), Intelligent Computer Aided Instruction (ICAI)<sup>1</sup>, and Intelligent Tutoring Systems (ITS)<sup>2</sup>. Related concepts are discussed in this section are “knowledge”, “communication”<sup>3</sup>, “intelligence.”

The named concepts are implemented in learning management system (LMS), learning environments, and tutoring systems, as described in [Darbhamulla and Lawhead, 2004]. A more in-depth discussion of the differences between a tutor, an assistant and a consulted is given in [Wenger, 1987, pp. 232] and [Davies et al., 2001, pp. 54].

While the Virtual Unix Lab offers a wide field of applications, some topics will not be discussed here to narrow the focus; references to literature are given here for further information.

---

<sup>1</sup> [Wenger, 1987] pp. 3

<sup>2</sup> [Freedman et al., 2000] pp. 1

<sup>3</sup> [Wenger, 1987] pp. 6

**Group teaching:** Tutoring of groups of students – in contrast to tutoring of a single student – allows applying many advanced tutoring techniques in the area of communication, e.g. discussion forums, mailing lists, promoting students as tutors of their co-students, and many more - see the methods listed under “Constructivism” in section 3.1.2. Keywords to mention for this area of research are Computer Supported Collaborative Work (CSCW) and Computer Supported Collaborative Learning (CSCL)<sup>1</sup>.

Texts that discuss group teaching are [Suebnuakarn and Haddawy, 2004] for application in medical teaching, and [Yin et al., 2000] for a knowledge-based approach for designing intelligent team training systems. Further texts that cover group teaching include [Yacef, 2004, pp. 343], [Haake et al., 2004] and [Kölle, 2007].

Even though group teaching is not discussed in detail here, tutoring and user adaption can be applied to individual members of a group in ways that would not be possible otherwise. As such, group teaching is considered an extension to the tutoring applied to single students that is discussed here.

**Natural language processing:** There’s a large base of literature for natural language processing that is specifically targeting the Unix operating system, its user interface, and how to apply it to tutoring for users that are new to the system. The area of natural language interfacing is not considered here, as the existing user interfaces of the Unix operating system should be learned, and no additional interfaces be provided to (possibly) make learning and using the system easier.

One of the problem areas with natural language interfaces is to understanding commands, which makes them sub-optimal for the general application area. See [Hegner, 2000, p. 183] for more information on these problems.

Further discussions of using natural language processing with the Unix operating system’s user interface can be found in [Wilensky et al., 1984], [Manaris et al., 1994], [Manaris and Pritchard, 1993], [Chin, 1983], [Wilensky et al., 1988] and [Kevitt, 2000]

The discussion led in the following sections covers learning theories and instruction design as discussed in section 3.1.1, including areas from instruction theory.

### **8.1.1 Approaching tutoring**

The basis for tutoring can be found in communication models<sup>2</sup>. When analyzing tutorial support, communication processes and related models of communicable knowledge have to be observed<sup>3</sup>. In this communication process, the computer acts as representational medium, the domain acts as subject matter and the student as a source of

---

<sup>1</sup> [Haake et al., 2004]

<sup>2</sup> [Wenger, 1987] pp. 6

<sup>3</sup> [Wenger, 1987] pp. 307



variability in the models of expertise<sup>1</sup>. Kobsa also states that user models are “a necessary prerequisite for a dialog system to exhibit cooperative dialog behavior”<sup>2</sup>, and even for non-cooperative dialog systems user models provide an important improvement of communication and flexibility.

Knowledge communication consists of several components and levels applied in a number of communication models<sup>3</sup>. The following discussion focuses on the model of intelligent tutorial systems (ITSs) as described in [Freedman et al., 2000] and [Schulmeister, 2007, p. 171].

The four basic components can be identified<sup>4</sup>:

- The domain model.
- The teaching model, also called pedagogical or didactic model.
- The user model, also referenced as student model.
- The user interface.

The following sections describe a top-down approach towards a tutoring system that employs this approach by first analyzing the didactic realization of the teaching model, providing an analysis of the topics to teach for the domain model, then investigating tutorial and adaptive help for the student in the user model, and finally by observing any changes in the user interface of the Virtual Unix Lab that could assist in that process.

Implementation of the tutoring architecture described here and the tutoring design outlined in chapter 10 is beyond the scope of this work. For an actual implementation, it is expected that an iterative development model similar to the two-step approach chosen to implement verification of exercise results in chapter 6 can be employed.

### 8.1.2 The teaching model

The teaching model in tutoring systems describes how teaching is performed, and what related didactic operations are performed. This section outlines a number of possible approaches that can be used, and gives some guidelines on how to determine which approach to choose. This serves as decision base in chapter 10.

---

<sup>1</sup> [Wenger, 1987] p. 309

<sup>2</sup> [Kobsa, 1990] p. 4

<sup>3</sup> [Wenger, 1987] pp. 417

<sup>4</sup> [Wenger, 1987] pp. 13

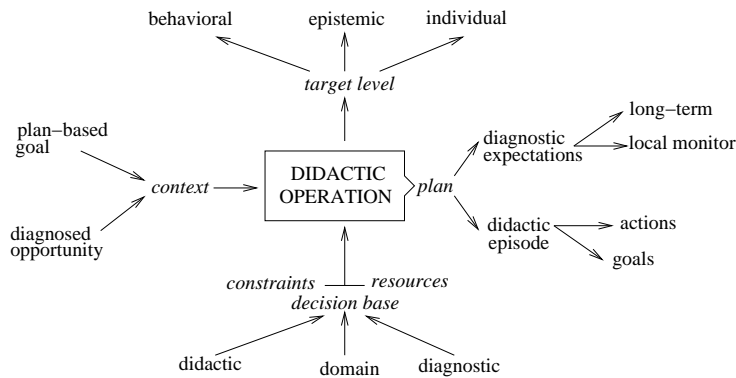


Figure 8.1: Aspects of a didactic operation. Image source: [Wenger, 1987, p. 397]

### 8.1.2.1 Teaching and didactic operations

The teaching model in intelligent tutoring systems is also referred to as didactic or pedagogic model. It focuses on pedagogical activities that are intended for a direct effect on students, not only on diagnostic activities<sup>1</sup>. There are four characteristic aspects of a didactic operation: the plan of action that enacts a didactic operation, the strategic context in which the operation is triggered, the decision base that provides constraints and resources for the construction of the operation, and the target level of the student at which the operation is aimed<sup>2</sup>. Figure 8.1 illustrates the relationship.

Didactic operations influence the plans of actions. Plans exist on all levels, and every action – however small – can be viewed as a plan. Episodes can be used to create diagnostic expectations, and the results of the plan can be monitored. For the monitoring, the bandwidth of the communication channel between the entity that performs the monitoring and the monitored entity is of importance. Likewise, the fact that uncertainty is a fundamental fact of all communication should be remembered<sup>3</sup>.

For the pedagogical context, the assumption is that “instruction has goals.”<sup>4</sup> Activities can happen either as part of a plan, or on impulse because it is considered appropriate in the given context. This may happen if a situation in which an activity happens was not brought up as part of a plan, but if the right final step is still recognized as appropriate. For plan based contexts and strategies, the pedagogic goals dominate. Mixed strategies are still possible to follow alternative goals at the same time<sup>5</sup>.

The decision base for didactic operations consists of resources and constraints. Re-

<sup>1</sup> [Wenger, 1987] p. 395

<sup>2</sup> [Wenger, 1987] p. 396

<sup>3</sup> [Wenger, 1987] p. 397

<sup>4</sup> [Wenger, 1987] p. 398

<sup>5</sup> [Wenger, 1987] pp. 398

sources are required as building material, and constraints ensure didactic effectiveness and often imply the resolution of conflicts between various competing factors that are affecting decisions. There are three potential sources of both resources and constraints: didactic, domain specific, and diagnostic information. The didactic base provides pedagogical principles and sequencing schemes, e.g. simple to complex and focused vs. diversified. Tailored interventions and tailored sequences address specific issues that are found during didactic processes, and they describe intervention strategies for those processes and situations. For this, they can use a lot of diagnostic information if available<sup>1</sup>.

The target level of a didactic operation is defined as the level of the student model at which an operation seeks immediate modification. Possible levels are behavioral, epistemic or individual<sup>2</sup>. At the behavioral level, guidance of a task is performed without addressing any internalized knowledge<sup>3</sup> in any direct or organized fashion. Possible types of behavioristic actions are specific hints, general advice, error correction by direct or indirect indication, or suggestions of better solutions<sup>4</sup>. Guidance at the epistemic target level explicitly seeks to modify the student's knowledge state, either via direct communication or via practice. The latter is intended to exposing the student to specific experiences<sup>5</sup>, see "situated learning" in section 3.1.4. Actions at the individual target level have positive effects on student learning even if they perform no direct form of knowledge communication. Instead, this can be individual motivation (e.g. for despaired students, so further knowledge communication is possible at all), adjusting of speed, abandoning a topic temporarily, or congratulating students on a small success. Using individual actions reflects on the student as a knowing, performing and learning being itself. Such a reflexion may be critical to the whole operation of what teaching strategies to apply. They can only be discovered in an interactive way, it is not possible in other "passive" media like books or films<sup>6</sup>.

Bridging the gap between communicable knowledge and a model of the student is "extremely difficult and computationally costly." As a result, applied didactic knowledge is applied in compiled form. This could happen via established curricula, tested lesson plans, libraries of activities, and presentation techniques<sup>7</sup>.

The organization of teaching can happen in either linear form, where the network of a student's knowledge grows incrementally, or as web of topics where the learning determines the depth rather than the breadth of knowledge. In both organizations, a difference is made between active and passive assistants. Data acquisition can e.g. happen by keystroke analysis and other diagnostic methods, and wrong steps can be determined by applying an existing so-called "theory of bugs". The interplay of top-

---

<sup>1</sup> [Wenger, 1987] pp. 401–407

<sup>2</sup> [Wenger, 1987] pp. 408

<sup>3</sup> See "Cognitivism" in section 3.1.1

<sup>4</sup> [Wenger, 1987] pp. 408

<sup>5</sup> [Wenger, 1987] pp. 410

<sup>6</sup> [Wenger, 1987] pp. 411

<sup>7</sup> [Wenger, 1987] p. 314

down expectations and bottom-up reconstruction can be modeled with diagnostic plan analyzers<sup>1</sup>. More details on factual vs. procedural knowledge and the relationship between various levels (behavioristic, cognitivistic, constructivistic) of didactic analysis can be found in chapter 3 and in [Feyrer, 2005].

### 8.1.2.2 Methods for plan recognition and assistance

After laying out the didactic foundations of teaching and the requirements for plan recognition, assistance, and diagnosis, this section describes how to realize them. It introduces a number of approaches that can be chosen from, starting with classical approaches, going from cognitive and linguistic methods to artificial intelligence. This will be used as base for selecting methods that can be used in the Virtual Unix Lab, which is described in the next section, 8.1.2.3.

#### 8.1.2.2.1 Classical approaches

There are several classical approaches for plan recognition and assistance, which are introduced here.

The overlay model takes a list of domain concepts as input, and forms its user model by noting to what extent each concept is believed to be known by the student<sup>2,3</sup>. Domain knowledge is split into independent components, and the tagging system maps the knowledge of the student on the predefined components. The model defined that way is a subset of the full model, which provides a simple mechanism to determine candidate areas of pedagogical actions<sup>4</sup>. One limitation of the overlay model is that it is restricted to the knowledge of the domain expert. This can be solved by utilizing several experts plus a meta-expert<sup>5</sup>. Another issues is that interdependencies between single concepts are not considered, which are important for procedural knowledge<sup>6</sup>. Improvements of the static overlay model can be made by segmenting tasks and components as well as adjusting them to a genetic graph as described in [Wenger, 1987, pp. 140]. See section 8.1.4 for a further discussion of the student model.

The differential student model offers a procedural network as contrast to a semantic network, where major tasks are split into smaller tasks, and tests for possible errors are included<sup>7</sup>. In contrast to statistical models, procedural networks can also tell what wrong thinking (“bug”) may cause a problem, due to their deterministic deep-structure

---

<sup>1</sup> [Wenger, 1987] pp. 224–226

<sup>2</sup> [Carr and Goldstein, 1977]

<sup>3</sup> [Wenger, 1987] p. 199

<sup>4</sup> [Wenger, 1987] p. 346

<sup>5</sup> [Wenger, 1987] pp. 232

<sup>6</sup> [Wenger, 1987] pp. 137

<sup>7</sup> [Wenger, 1987] pp. 154

model. Limitations of this model are that help is pre-determined by the model, and if a case is not modeled, it can not be trained. Furthermore, no reasoning can be given *why* a wrong decision was made at a certain point<sup>1</sup>. A problem of this model is constructing it, as many cases need to be considered – Wenger mentions “110 observed bugs for subtraction”, “a test capable of distinguishing among 1200 compound bugs with only 12 problems” and a “place-value subtraction with borrowing [...] turns out to involve as many as 58 subskills.” He also adds that an existing theory of bugs for the problem domain will be of valuable help, as it turns the “art of test design” into a formal process<sup>2</sup>.

The Leeds Modeling System (LMS) is related in that it compares rules similar to the before-mentioned sub-goals. But in contrast to only modeling and comparing rules, it also knows about mal-rules. Those may be generated automatically from existing data<sup>3</sup>.

The User Modeling Front End (UMFE) is another extension to the overlay model that assigns a scale to the tagged domain concepts, indicating how well known a concept is, see the “three levels of knowledge” in [Michaud et al., 2000]. It also assumes connections between the concepts, and does not treat them independently. Sources of evidence on which conclusions are drawn include a stereotype of the user’s initial understanding, a user’s statements, and inferences drawn via rules. All this is represented in a framework of the problem space that is split between primitive operators and conditionals<sup>4</sup>.

#### 8.1.2.2.2 Cognitive approach

The difference between classical and cognitive tutoring approaches is that the former try to reflect the learner’s internal line of thinking, while the latter “only” provide a procedure for building up knowledge<sup>5</sup>.

Tutoring and plan recognition with cognitive approaches assumes that knowledge is first acquired declaratively through instruction, and then converted and reorganized into procedures through experience, following Piaget’s concept of assimilation and accommodation described in section 3.1.1. In the first step of knowledge compilation, general pieces of information are “proceduralized” into specific rules that apply to a special class of cases. These are collapsed into few rules that are used in sequence to achieve a goal, which in turn is used to compose a single rule that combines all their efforts<sup>6</sup>. The representation of cognitive functions is assumed to happen as set of

---

<sup>1</sup> [Wenger, 1987] pp. 156

<sup>2</sup> [Wenger, 1987] p. 165

<sup>3</sup> [Wenger, 1987] pp. 194

<sup>4</sup> [Wenger, 1987] p. 221

<sup>5</sup> [Wenger, 1987] pp. 304

<sup>6</sup> [Wenger, 1987] p. 291

production rules, and no limit on the size of the long-term memory is assumed<sup>1</sup>.

Comparison with human tutors shows that they never state the productions that have to be learned in a declarative way explicitly, especially as productions are the representation of the skills to be learned. Instead, they provide a problem solving context and point out factual problems, without telling how to fix them, i.e. not giving away procedural knowledge. Those procedural skills should be compiled by the learner during problem solving by properly understanding, integrating, and later recalling and adapting<sup>2</sup>.

Feedback is considered important in the learning process, and the earlier feedback happens, the better<sup>3</sup>. Immediate feedback is considered useful, but care needs to be taken in debugging and troubleshooting situations. Testing is done through rules and mal-rules<sup>4</sup>.

### **8.1.2.2.3 Linguistic approach**

Using a linguistic approach to plan recognition and tutoring allows to separate syntactic, semantic and pragmatic views on a problem. The challenge is inference of the pragmatic and semantic view from the corresponding semantic and syntactic view, though<sup>5,6</sup>. Augmented Transition Networks (ATNs) and Planning ATNs can help to realize this approach, which is comparable to parsing networking protocols like TCP/IP<sup>7</sup>. In that regard, the same principles like Jon Postel's paradigm of "be liberal in what you accept and conservative in what you emit"<sup>8</sup> can be applied to parsing of the various semiotic layers. Miller, Goldstein and Genesereth also support this idea by viewing plan recognition as instance of a parsing problem<sup>9,10</sup>, where plan recognition is performed in a "bottom-up" fashion, matching expectations that are modeled "top-down." Also, software engineering techniques can be applied that way to describe natural language problems verbally, and make a transition of the problem domain from natural language processing to programming<sup>11</sup>.

### **8.1.2.2.4 Artificial intelligence**

---

<sup>1</sup> [Wenger, 1987] p. 291

<sup>2</sup> [Wenger, 1987] pp. 291

<sup>3</sup> [Heer et al., 2004] p. 468

<sup>4</sup> [Wenger, 1987] pp. 296

<sup>5</sup> [Wenger, 1987] pp. 228

<sup>6</sup> [Morris, 1938]

<sup>7</sup> [Wenger, 1987] p. 229

<sup>8</sup> [Postel, 1981] p. 13

<sup>9</sup> [Wenger, 1987] p. 234

<sup>10</sup> [Genesereth et al., 1982] pp. 124

<sup>11</sup> [Wenger, 1987] p. 235

The idea of using artificial intelligence (AI) and expert systems in computer added instruction (CAI) goes back to Carbonell in 1970<sup>1</sup>. In contrast to the “bottom-up” linguistic approach described in the previous section, this approach assumes a complete domain model, in which navigation and searching can be performed. This approach is related to semantic networks, and addresses the drawbacks that they have in the areas of representation of procedural knowledge by using simulations for feedback and exercise of debugging and troubleshooting<sup>2</sup>.

Assistance is given by recognizing the way that the student is solving a problem, and comparing it with other possible steps. If the student’s approach differs from a possible solution, he can be brought back on track. Solutions for plan recognition can be found with strategies like a width first or depth first search of the domain model, or by using hill-climbing or means-end analysis with the student’s input as starting points<sup>3</sup>.

Historic examples include a framework for integration of intelligent tutoring systems in a gaming simulation<sup>4</sup> and for simulation education<sup>5</sup>. Kerner and Freedman describe a “Content Knowledge Base” that takes a start and end condition as well as a list of known steps and effects, and then determines the proper solution automatically<sup>6</sup>. Couch and Gilfix also apply logic programming to plan recognition and tutoring, and they use Prolog to describe a system that compares current and target state, and that identifies necessary changes<sup>7</sup>. Similar systems that solve problems for the domain of Unix system management are cfEngine<sup>8</sup> and LCFG(ng)<sup>9</sup>. [Narain, 2005] describes a system that designs a network setup for a given problem description, including all configuration parameters.

While those systems start to be of use for solving problems, they do not offer a base for reasoning, and as such are not ripe for use in education and training yet. The time that has passed between Couch and Gilfix’ Prolog-based approach and Narain’s implementation show that there’s some more time needed to grow mature.

Another aspect is complexity of the application domain. [Kautz and Selman, 1992] describes a very simple block-world that needs eleven rules. For a complex system like Unix, the effort will be much higher. Furthermore, to implement rollback of actions like deleting files and terminating processes is considered hard, even if not impossible through means of speculative execution<sup>10</sup>.

In summary, realizing plan recognition and tutoring via AI is considered very difficult

---

<sup>1</sup> [Carbonell, 1970]

<sup>2</sup> [Wenger, 1987] p. 30

<sup>3</sup> [Haberlandt, 1999] pp. 157

<sup>4</sup> [Angelides and Paul, 1993]

<sup>5</sup> [Taylor and Siemer, 1996]

<sup>6</sup> [Kerner and Freedman, 1990]

<sup>7</sup> [Alva L. Couch and Gilfix, 1999]

<sup>8</sup> [Burgess, 1995]

<sup>9</sup> [Anderson and Scobie, 2002]

<sup>10</sup> [Su et al., 2007]

to realize<sup>1</sup>.

#### 8.1.2.2.5 Semantic networks and ontologies

Semantic networks are directed graphs with nodes that are connected by directed relations. The nodes represent properties, and the relations indicate a semantic relationship between the nodes<sup>2,3</sup>. While the properties and relations can be from a wide field, the primary area of application is in establishing linguistic models of natural languages. Associated theories like Schank's Conceptual Dependency Theory assists in the modeling process for semantic networks<sup>4</sup>.

Ontologies describe the relations between nodes and relations<sup>5</sup>. The relation can either be within a specific domain, forming a domain-specific ontology, or on a more general scheme across multiple domains, forming a so-called upper ontology<sup>6</sup>.

Applications of semantic networks can be found in mind maps<sup>7,8</sup> and the extension of the World Wide Web with semantic information into Tim Berners-Lee's "Semantic Web"<sup>9</sup>. Many notations are available for describing elements in semantic networks and ontologies. Example languages include the Web Ontology Language (OWL)<sup>10</sup>, the Resource Description Framework (RDF)<sup>11</sup>, and CycL, the language of the Cyc knowledge base<sup>12</sup>.

Semantic networks can be used as model for knowledge representation. It is only a factual, objective model without personal connotations as would be expected from cognitive learning theories. Furthermore, the semantics are appropriate as a model for knowledge, but not to reflect a learning process<sup>13</sup>.

Semantic networks can be extended from reflecting pure factual knowledge into handling procedural knowledge to some extent. The main application lies within factual knowledge, though<sup>14</sup>. Pedagogical actions that can be based upon the knowledge representation allow reasoning about the student's knowledge, concepts he has already learned and what facts he missed, and a system can give hints on related concepts and

---

<sup>1</sup> [Wenger, 1987] p. 18

<sup>2</sup> [Quillian, 1967]

<sup>3</sup> [Chaffin, 1992]

<sup>4</sup> [Schank, 1972]

<sup>5</sup> [Staab and Studer, 2004]

<sup>6</sup> [Gruber, 2008]

<sup>7</sup> [Buzan and Buzan, 2006]

<sup>8</sup> [Nast, 2006]

<sup>9</sup> [Berners-Lee et al., 2001]

<sup>10</sup> [W3C, 2004a]

<sup>11</sup> [W3C, 2004b]

<sup>12</sup> [Lenat and Guha, 1991]

<sup>13</sup> [Quillian, 1988] p. 80

<sup>14</sup> [Quillian, 1988] p. 81



terms that the student is found not to be fluent with.

Examples of learning environments that use semantic networks alone or in combination with other systems include the friendly intelligent tutoring environment described in [Jerinic and Devedzic, 2000], the WeKnow project described in [Sattari et al., 2007], and the Electronic Learning Assistant (ELA) described in [Kolovski et al., 2004].

#### 8.1.2.2.6 Frames and scripts

Frames organize and store knowledge in units that represent e.g. situations or objects. Frames consist of slots and filters. Slots describe properties, and filters describe values and can link to other frames<sup>1</sup>. If a slot didn't have a specific value, a default value is assumed. This reflects the separation into general and specific knowledge<sup>2</sup>. Scripts describe sequences of events in a particular context, i.e. provide an extension of the factual knowledge represented in frames for procedural knowledge<sup>3,4</sup>.

The application of frames and scripts in education requires a full model of the domain in frames and scripts. Assuming the existence of such a domain model, frames and scripts can be used to identify what solution a user pursues for a given problem, detect deviations from common procedures in the user's actions, and answer general questions about the knowledge domain. The drawback of the model is that the underlying domain model, i.e. the frames and scripts, are hard to model, esp. in complex domains and/or for complex tasks. Frame databases like Lenat's Cyc project<sup>5,6</sup> and the Maryland PARKA project<sup>7,8</sup> show the complexity and effort needed for limited areas of application.

A rare practical example of a project using frames and scripts is Script Applier Mechanism (SAM) project, which allowed to answer questions about fairy tales. Their knowledge representation was based on frames, and scripts were applied to a database that was derived by semantic network techniques in the MARGIE project<sup>9</sup>. Other projects use frames and scripts for educational projects in combination with other techniques, like the frame-based interaction and learning model for ubiquitous learning in [Si et al., 2006].

---

<sup>1</sup> [Minsky, 1975]

<sup>2</sup> [Brewer, 2007]

<sup>3</sup> [Schank and Abelson, 1975]

<sup>4</sup> [Kirsch, 2003] pp. 11

<sup>5</sup> [Lenat and Guha, 1990]

<sup>6</sup> [Cycorp, 2007]

<sup>7</sup> [Evet, 1994]

<sup>8</sup> [PLUS, 2007]

<sup>9</sup> [Cullingford, 1981]

### 8.1.2.2.7 Bayesian networks

Bayesian networks represent a probabilistical model that is based on an directed acyclic graph (DAG). In addition, statistical methods exist to model the inter-dependencies between the nodes. They can be used to predict and classify statements about behaviour and development within the domain model. A major advantage of bayesian networks is that they can create a domain model, or improve an existing domain model without explicit manual effort<sup>1</sup>.

The underlying graph of the network needs to exist or be created, and the probability of moving from one node to another one can be modeled dynamically. Based on this model, statements about certain developments in user behaviour can be made based on previous user interactions that built the domain mode. Verification of the statements is needed, though.

Some applications of bayesian networks in teaching emphasise their supporting nature of other teaching models, e.g. when recognizing what kind of learner a certain student is: [Garcia et al., 2007] shows that detection is possible if a student's learning mode is reflecting or acting, steadily or in fits and starts, and if he learns intuitively or sensitively. As such, bayesian networks find their application within a so-called "decision support system" (DSS), e.g. within intelligent tutoring systems (ITSs). Other examples include the long-term modeling of a user's factual knowledge in the form of english capitalization and punctuation in [Mayo and Mitrovic, 2001], the web-based ITS described in [Butz et al., 2006], and [Conati et al., 2002], which describes how bayesian networks can help to cope with uncertainty in student modeling.

In summary, bayesian network can be considered to be rather a supporting tool for other methods than as a standalone teaching model<sup>2</sup>.

### 8.1.2.3 Choosing a method

The previous section outlined various approaches for realizing assistance in tutoring systems. To determine which approach is best fit, a number of questions need to be answered.

The first decision is if a mode based on a psychological approach should be chosen over a learning theory based on pedagogics<sup>3</sup>. The system should classify students by their success or failure<sup>4</sup>, and it should be considered that constructivistic approaches are

---

<sup>1</sup> [Ben-Gal, 2007]

<sup>2</sup> [Ben-Gal, 2007]

<sup>3</sup> [Wenger, 1987] p. 305

<sup>4</sup> [Wenger, 1987] p. 17, pp. 153

considered difficult to realize<sup>1</sup> and “extremely difficult and computationally costly.”<sup>2</sup> The diagnostic process may require descriptions in domain specific languages, and the diagnostic process is accounting for the required data<sup>3</sup>.

Section 10.1 describes the selection process for tutoring in the Virtual Unix Lab, and its outcome.

### 8.1.3 The domain model

The domain model of tutoring system describes the object that the communication is about. In this regard, a computer acts as the representational medium, the domain is the subject matter, and students are a source of variability in the model of expertise<sup>4</sup>. The domain model contains data about the application domain in both compiled and articulate form<sup>5</sup>. Furthermore, with the help of the domain model it is – to some extent – possible to automatically create exercises for students<sup>6</sup>.

Compiled knowledge is used for several reasons. First, it can be used to indicate specific circumstances, e.g. illustrate a connection, illustrate causal or temporal connections, or to connect specific actions with specific situations. Second, compiled knowledge can make working along a certain model or by a specific approach easier. Third, compiled knowledge can serve a specific purpose e.g. to present a certain setup in a specific light, possibly simplifying or omitting facts that are not of importance to the situation or methods to learn. The advantages of compiled knowledge in this context is that it is highly efficient and simple to apply<sup>7</sup>.

One application of compiled knowledge could be to evaluate the possible steps for the domain concept, model the student, offer direct problem solving with suggestions, select problems to optimize learning according to its student model, solve examples step by step for the student to follow and learn when presented with a better solution. All these points can be used in adaptive interaction, see section 8.2.

Besides compilation of knowledge, articulation can improve the form of communication towards a certain goal, e.g. causality, structure, functionality, teleology, constraints and definitional semantics<sup>8</sup>. As such, processes can be divided into fine-grained steps that not only allow to understand the final result, but also the order of events and decisions that lead to those results. Decomposition can be applied to the subject matter to split it into various views: a curriculum view for learnable units and

---

<sup>1</sup> [Schulmeister, 2007] pp. 218

<sup>2</sup> [Wenger, 1987] pp. 314

<sup>3</sup> [Wenger, 1987] p. 18

<sup>4</sup> [Wenger, 1987] p. 309

<sup>5</sup> [Wenger, 1987] pp. 325–327

<sup>6</sup> [Shah and Kumar, 2002] pp. 170

<sup>7</sup> [Wenger, 1987] p. 329

<sup>8</sup> [Wenger, 1987] pp. 331

a diagnostic view for perceptual units. The various parts and views of the subject matter are then compiled again to adjust it to the corresponding type of teaching<sup>1</sup>. As an example, in the Virtual Unix Lab the subject matter could be decomposed into various check primitives, and an exercise could then be compiled from those checks, see section 8.1.2.2.1.

After analyzing the data of the problem domain, a possible application is to use that data to automatically create exercises for students. While exercises are usually created manually, the task is challenging, and if the volume of required exercises raises in volume, automation is desirable<sup>2</sup>. For simple problems, automatic exercise creation can be done by finding exercises with structural analogy, which is difficult for complex problems as a whole. For them, it would be possible to apply this technique to smaller areas again. An alternative to analogies would be to use domain-specific operators in the form of compiled productions. Again, these are found difficult to communicate and encode without falling back to using examples<sup>3</sup>.

Examples for automated creation of exercises are discussed in [Fischer and Steinmetz, 2000, pp. 49] and [Fischer, 2001]. The taxonomy and ontology used in their examples are not available for the present domain of system administration or any of its parts, though. Furthermore, the approach is good for creating exercises which check declarative knowledge of facts, but not so much for procedural knowledge. A transition from declarative knowledge to procedural knowledge is also of questionable use. [Helic et al., 2004] goes one level higher in abstraction and performs automatic course sequencing based on pre-defined building blocks. This approach can help to integrate exercises into existing learning management systems (LMS) by using interfaces like the ones described in the “Sharable Content Object Reference Model” (SCORM)<sup>4</sup>. As written in section 8.1, this work covers the Virtual Unix Lab as explorative learning system with user adaption, and will not concentrate on learning management systems and how to interface with them. The area of how to provide help – either on demand or automatically – is considered of much importance, though, and thus covered later on.

### 8.1.4 The user model

The model of the user in instructional systems goes back to J. D. Fletcher<sup>5</sup>. It tries to answer the question of what’s going on inside the student. In the model of knowledge communication, the student is the receiver of information<sup>6</sup>, and he has both correct and incorrect knowledge<sup>7</sup>. Representation of knowledge in this model can be done in

---

<sup>1</sup> [Wenger, 1987] pp. 336

<sup>2</sup> [Shah and Kumar, 2002] pp. 170

<sup>3</sup> [Wenger, 1987] p. 303

<sup>4</sup> [ADL Technical Team, 2004]

<sup>5</sup> [Fletcher, 1975] pp. 118

<sup>6</sup> [Wenger, 1987] pp. 307

<sup>7</sup> [Wenger, 1987] pp. 16

several ways. One possible solution is to use primitives of a language for the domain that spans both correct and incorrect knowledge. This can be described e.g. within an expert system or an AI component as described for the teaching model in section 8.1.2.2.4. Another solution is to use a data or model driven approach with a known set of errors and misconceptions<sup>1</sup>. A list of possible attributes to store for a user, and a list of academic and commercial systems that implement user modeling can be found in [Kobsa, 2001a].

When applying a cognitive learning model, there is a difference between final (absolute) expertise and the expertise as possessed by a student involved in learning. The dimensions of variety in knowledge states include scope, incorrect knowledge, and viewpoints<sup>2</sup>. [Wenger, 1987] describes how to build a “genetic graph” that describes such a model, and how to determine a finegrained genetic graph for a given domain using decomposition of the subject matter and an overlay model<sup>3</sup>. As an additional plus, the learning curve can be determined from a genetic graph as described in [Chin, 1986, p. 25]. Taking a genetic graph and overlay model as base for a tutor, the information about which deviations to expect in a domain play an important role in a tutor’s ability to communicate knowledge<sup>4</sup>. An existing theory of bugs can provide this information.

#### 8.1.4.1 Theories of bugs

The term “theory of bugs” describes an enumeration of all mistakes that a student can make when learning to become proficient in a specific domain. There are several enumerative, reconstructive and generative theories of bugs that can be used to determine mistakes by a student<sup>5</sup>.

Enumerative theories of bugs are usually implemented as catalogue or library of items that can go wrong, and can be built empirically by observing student errors. Their representation can exist as a description or rule, which can be detected by machine-executable forms as “mal-rule” or “incorrect plan”, or by matching corresponding patterns as described in section 8.1.2.2.3. “Simple” errors can be grouped into classes or errors, and extended with descriptions of which general errors and wrong assumptions are being made. From these classes of errors, heuristics for error detection can be determined. A problem with enumerative theories of bugs is that they are unstructured and usually extensive<sup>6</sup>.

Reconstructive theories of bugs try to overcome the limitations of enumerative theories of bugs. Instead of having fixed catalogues of errors, classes of errors are recognized

---

<sup>1</sup> [Wenger, 1987] p. 45, pp. 205

<sup>2</sup> [Wenger, 1987] p. 345

<sup>3</sup> [Wenger, 1987] p. 346

<sup>4</sup> [Wenger, 1987] p. 347

<sup>5</sup> [Wenger, 1987] pp. 347, Figure 16.1

<sup>6</sup> [Wenger, 1987] pp. 348

by general descriptions. They are usually data driven and constructed in a bottom-up approach, and use data mining procedures on data from existing exercises. Description of error classes can happen in a (domain) specific language that is specified on a rather finegrained base<sup>1</sup>. Parallels to the decomposition process as for the domain model in section 8.1.3 can be found here.

Generative theories of bugs intend to go one step further and not only recognize classes of errors, but to also lead them back on what was learned wrong in the past, explain to the student what was done wrong, and what he has to do to improve in the future<sup>2</sup>. In practice this could be achieved by giving “better” feedback during the exercises. Wenger found that “incorporation of such dynamic generative theories into a complete tutoring system has never been tried”<sup>3</sup>, which hints at the availability of practical experiences for this area, upon which a real system could be built.

When deciding which approach to use for a theory of bugs, there are several considerations to make. The enumerative approach is easy to realize from empirical observations and it can even cover corner cases that are difficult to catch otherwise. The downside is that the amount of cases that can be handled is limited. Reconstructive theories of bugs can often be derived from enumerative ones. By reducing the focus of reconstructive and generative approaches, more than one assumption can be true for an error, and there will be the problem of choosing which one really is in effect<sup>4</sup>. Models and theories from psychology and artificial intelligence can assist here, see the approach of having several experts plus a meta-expert in classical approaches for plan recognition and assistance, and overlay architecture in section 8.1.2.2.1.

Mislearning and forgetting are two more problems that need to be addressed. In planing nets, steps that were learned wrong or that were forgotten can only be recognized to a limited extent. Replacing planing nets through knowledge in compiled form helps here, and there is some belief that theories based on planing nets and AI are less reliable in practice than simply enumerative approaches<sup>5</sup>.

#### 8.1.4.2 Viewpoints

Errors and “wrong” knowledge can be viewed from two sides. On the one side they are the manifestation of wrong actions and beliefs, on the other side they are points where correct knowledge should be taught and wrongly learned knowledge should be corrected. Basically, viewpoints allow looking at an issue from more than one side, and allow detecting errors and also correct them with “right” knowledge<sup>6</sup>.

---

<sup>1</sup> [Wenger, 1987] p. 349

<sup>2</sup> [Wenger, 1987] pp. 349

<sup>3</sup> [Wenger, 1987] p. 350

<sup>4</sup> [Wenger, 1987] p. 351

<sup>5</sup> [Wenger, 1987] p. 353

<sup>6</sup> [Wenger, 1987] p. 355

Viewpoints are usually situation-specific<sup>1</sup>. To accommodate them to a problem, instructions need to be adapted for multiple approaches and solutions – problems can be viewed and solved in multiple ways, solutions can be constructed in multiple ways, and identical decisions can have different sources.

The background for viewpoints is that each person has its individual view on the world, as learning within a domain is tied to background-knowledge (“culture”). In the context of scientific research, there are also “research-traditions” which build a set of general assumptions about the entities and processes in a domain of study, and about the appropriate methods to be used for investigating the problems and constructing the theories in that domain. Wenger recommends that it is useful to view the student involved in learning as a microcosm of the scientific community<sup>2</sup>.

Several different viewpoints can exist for a problem, and they can overlap or contradict each other. It is thus not right or useful to merge them. For full comprehension, a viewpoint must be assumed that covers all the relevant details, which is also why in practice the sum of all knowledge about a certain topic or domain is called the (singular!) viewpoint of a person<sup>3</sup>. Knowledge communication systems used in diagnosis should ideally maintain one viewpoint to an issue, while also offering different alternative viewpoints at the same time, which is often difficult to realize in reality<sup>4</sup>.

#### 8.1.4.3 Diagnosis

Diagnosis describes the task of providing feedback to the learner for the activities engaged during the learning process. It consists of three tasks: inferences, interpretation, and classification. Inference reconstructs internal processes either by assembling primitives from data in a “bottom up” manner, inferring semantics from syntax<sup>5</sup>, or by testing variations in a model in a “top down” approach. The basic assumption in both cases is that the reconstruction is performed in a deterministic way. Interpretation then places the observations made into context, tries to make sense from actions via viewpoints and goals, and tries to understand the student before helping him, with possible rationalizations to explain observations. Classification then characterizes or evaluates observations and inferences according to expectations<sup>6</sup>.

In general, diagnosis is performed on communication, and the communication happens over a communication channel that has a specific bandwidth. Usually this is a keyboard, sometimes it is a mouse in addition. Quantitative and qualitative analysis of data is still available for diagnostic purposes even with such limited bandwidth. The design of the diagnostic interface can be critical to the success of the diagnos-

<sup>1</sup> See Mandl’s “situated learning” in section 3.1.4

<sup>2</sup> [Wenger, 1987] p. 358

<sup>3</sup> [Wenger, 1987] p. 359

<sup>4</sup> [Wenger, 1987] p. 359

<sup>5</sup> [Morris, 1938]

<sup>6</sup> [Wenger, 1987] pp. 368

tic module. E.g. the impact of granularity of information retrieved for building the student model is considered as an additional difficulty to the pedagogical challenge. Ideally, the level of granularity should match the granularity level of the compiled model knowledge for optimal operation. One problem to also consider is that immediate steps may have different semantics than the final solution. E.g. when configuring a system, it is only natural for it to be in an inconsistent state halfway during the process.

Another problem, esp. for complex domains, is that reasoning itself can go through different phases, which makes intermediate steps much more difficult to understand than the final solution. For the current area of application in the Virtual Unix Lab, this means that while final exercise results are easy to judge purely by the fact that they are final, in contrast to providing assistance during the exercise. While a final solution can be analyzed behaviorally in terms of notions of correctness in the domain and of implemented goals, the precise interpretation of intermediate steps requires a complete model of reasoning within the domain. Without such a model, these steps cannot be interpreted as they reflect a process, not a state<sup>1</sup>.

There are three levels at which information can be relevant for pedagogical purpose and providing diagnosis. The behavioral level is considered as a pure product of behavior without tying to any perceived knowledge state involved in its generation. The epistemic level also evaluates the knowledge of the student, including factual knowledge about the domain and strategic knowledge applied to inference procedures. Everything else is considered at the individual level. This mostly covers all the items that make the deterministic deduction of intents and knowledge non-deterministic, e.g. due to aspects of the teaching and domain architecture, learning model, stereotypes, motivation, circumstances, and reflexive and reciprocal intents<sup>2</sup>.

The following sections describe the levels of diagnosis, gives hints at where and how they can be applied in chapter 10, and discusses acquisition of diagnostic data.

#### 8.1.4.3.1 Behavioral diagnosis

Behavioral diagnosis can be applied to get an overview of the student's knowledge. It can happen via a number of different activities, an overview of which is given in figure 8.2. The activities that are of interest in the context of the Virtual Unix Lab are reconstructive post-hoc and on-line interpretation. They both use inferential reconstruction via reconstructive interpretation to draw inferences about the present situation from events from past events<sup>3</sup>.

In the case of post-hoc reconstruction, only the final resulting situation is analyzed and

---

<sup>1</sup> [Wenger, 1987] p. 367–390

<sup>2</sup> [Wenger, 1987] pp. 368–369

<sup>3</sup> [Wenger, 1987] pp. 371



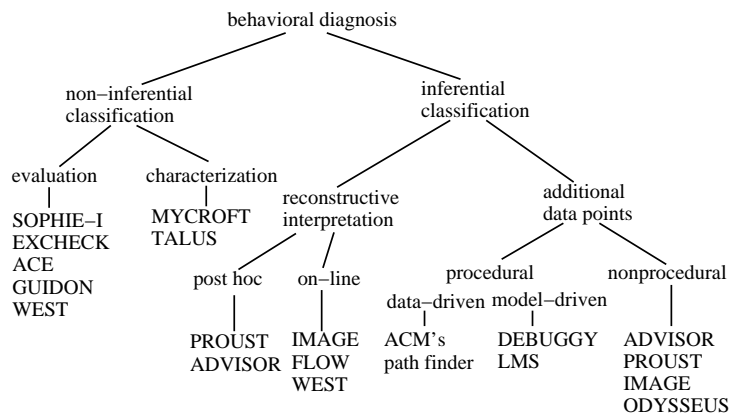


Figure 8.2: Taxonomy of behavioral diagnostic processes. Image source: [Wenger, 1987, p. 372]

inferences are drawn on how they could have happened. In the Virtual Unix Lab this is realized via check scripts as described in chapter 6. On-line diagnosis can collect diagnostic data during the exercise, and draws similar inferences<sup>1</sup>.

#### 8.1.4.3.2 Epistemic diagnosis

Cognitive psychology offers theories of human knowledge as introduced in section 3.1.1. Epistemology describes the “theory of knowledge” that covers the philosophical analysis of human knowledge<sup>2</sup>. A connection between these areas was described<sup>3,4</sup>, and epistemic diagnosis helps to identify areas where tutoring is needed.

Epistemic diagnosis happens in three phases. First, direct assignment of credit and blame determines which elements of knowledge are of interest. Next, structural consistency describes the impact of the affected elements of knowledge on the overall state of knowledge, and last, longitudinal consistency updates the student (user) model due to newly learned knowledge<sup>5</sup>. The following sections provides more details.

##### 8.1.4.3.2.1 Direct assignment of credit and blame

<sup>1</sup> [Wenger, 1987] pp. 373

<sup>2</sup> [Corlett, 1991b] p. 285

<sup>3</sup> [Corlett, 1991b] pp. 285

<sup>4</sup> [Corlett, 1991a] pp. 327

<sup>5</sup> [Wenger, 1987] pp. 376

The purpose of direct assignment of credit and blame's task is twofold. For one, it recognizes both correct and wrong knowledge that was used, e.g. by applying a modeling language. It then compares this against a differential expert model to determine which relevant knowledge was left out and was not applied<sup>1</sup>.

Extraction of epistemic information from the student's behavior can be split into three categories: activities that can be recognized via model tracing, reconstruction of material that can not be observed directly, and applying issues to separate the recognition process from modeling the student behavior. Each of these steps has at least three dimensions: The level of articulation, the degree up to which existing knowledge can be recognized and reproduced, and the amount of information that is required for the diagnosis<sup>2</sup>.

Model tracing takes the knowledge recognized, and compares it against the compiled knowledge available. Compiled knowledge merges many small steps into larger steps without describing in detail which exact steps need to be taken, and is thus very close to the detectable knowledge. As a result, "model tracing" is actually on the border between behavioral and epistemic diagnosis, as it assigns credit and blame for internal pieces of knowledge on the one side, but also verifies behavior and knowledge via wrong or missing rules on the other side<sup>3</sup>. The automated testing via rules and mal-rules in model tracing is similar to the list of possible answers in frame-based systems, where decisions lead to forks in the teaching path. The important difference is that in model tracing, the results are used to update the whole student model, and can thus have an influence on any further teaching activities. Of course, this depends on how much the student model impacts the pedagogic decisions<sup>4</sup>.

While model tracing draws inference from directly measurable activities, the goal of reconstruction is to derive beliefs from data that is not directly measurable. This is considered non-trivial in general, and the analysis of the existing exercises in the Virtual Unix Lab confirm that this is complex. As a result, this approach is not pursued further here. More information is available in [Wenger, 1987, pp. 379].

Issues are curriculum elements whose participation in decisions can be recognized and discussed without being modeled explicitly. They can be anything of pedagogical interest to the system, even a misconception. They are not directly tied to behavior, and any number of them can be independently recognized as having participated or not participated in a decision. As issues are not explicitly modeled, it is difficult to include them into the diagnostic process, though. A possible solution is to have a separate "expert-module" that knows about relevant issues and brings them up at the right time. For example, when a student just reboots the system after changing configuration of a service instead of manually restarting the service, ask the student why he did it that way – maybe he does not know how to restart a service manually. Realizing such an

---

<sup>1</sup> [Wenger, 1987] pp. 376

<sup>2</sup> [Wenger, 1987] pp. 378

<sup>3</sup> [Wenger, 1987] pp. 378

<sup>4</sup> [Wenger, 1987] pp. 379

expert-model is non-trivial and not considered further here, more information on the topic is available in [Wenger, 1987, pp. 381].

A hybrid approach of using model tracing, reconstruction and issues together is possible, as each of them has a different focus, either epistemic or behavioral. With this approach, it is possible to trace the student's actions while reconstructing intermediate steps to infer his plans<sup>1</sup>.

#### 8.1.4.3.2.2 Structural consistency

Additional constraints provided by the structure of knowledge states can increase the influence of direct assignment of credit and blame, e.g. correlation between the likelihoods of various pieces of knowledge being mastered as shown in section 7.2.4. The advantage of those networks is that they can be designed in the absence of epistemological structure and of a model of conceptual interactions, i.e. without an analysis of a domain's compiled knowledge<sup>2</sup>.

That way, it is not only easy to tell what was learned so far and what was not, but also defining what the student is about to learn next. This can be done by looking at the concepts he starts to use, but has not mastered yet. A discussion of this concept can be found in [Michaud et al., 2000].

#### 8.1.4.3.2.3 Longitudinal consistency

Collecting state in an ongoing exercise faces two contradicting requirements: on one side it has to be sensitive enough to adapt the tutor's attitude without delay, but on the other side it also has to be stable enough not to be easily disturbed by local variations in performance.

Longitudinal consistency indents to balance this requirement. It is often derived empirically, and implemented via scalar attributes such as numerical weights. The attributes are associated with individual elements of the knowledge state in the context of an overlay, and they are updated by statistical or pseudo-statistical computations as knowledge manifests itself in behavior<sup>3</sup>. As an alternative, Bayesian networks can be used for the implementation as described in [Mayo and Mitrovic, 2001].

An application of longitudinal consistency is to adjust feedback, see section 8.1.4.4.

---

<sup>1</sup> [Wenger, 1987] p. 381

<sup>2</sup> [Wenger, 1987] pp. 381

<sup>3</sup> [Wenger, 1987] pp. 383

### 8.1.4.3.3 Diagnostic data

Diagnostic data is the source upon which didactic analysis is performed, and thus the source of all evidence. It can be collected either passively, actively or interactively. In contrast to passive actions, active diagnosis allows the system to test its hypotheses. Extending the dialog between the user and the system, an interactive diagnosis invites the student to report on his decisions and his knowledge<sup>1</sup>.

Active diagnosis can be used to request more data that is required for discriminating between the competing models that the system inferred from its present data. Difficulties are confidence about what data to acquire, availability of methods for acquiring the exact data required (i.e. if there's an exact exercise/task that can determine the missing data) and not to speak of how to fit it into the exercise without disturbing continuity, or if it fits at all.

Dynamically created exercises help here. When adding additional exercises, care should be taken that a maximum of information is retrieved by them, to minimize the number of extra exercises needed<sup>2</sup>.

Interactive diagnosis can be used to justify predictions or hypotheses in the context of specific cases ("Why you think that this command ...") or asking questions about students' beliefs. The challenges here are to incorporate the interaction seamlessly into the exercise, and to also "understand" the reasoning done by the student, esp. when using natural language. A domain-specific alternative to using natural language would be to use menus or graphics.

Even in interactive systems, some inference should happen. For obvious or simple cases it is better to use inference than to disturb the student with trivial questions. Furthermore, the student may be unlikely to describe his ideas completely, and thus additional inference would be needed – to ask intelligent questions, there's a need to "understand" what the student is doing anyways<sup>3</sup>.

Still, an interactive approach to confirm diagnosis is attractive for two reasons. First, because it involves the user early in the process, and second, because it keeps the dialog at the level of beliefs, where misconceptions are expected to occur, and hiding the actual diagnostic process from the user. Once a misconception has been detected and confirmed, an alternative most closely in line with the student's own plan can be offered<sup>4</sup>.

In diagnosis, comparisons with human teachers are usually not very illuminating, because classroom teaching is very different from the type of tutoring performed by

---

<sup>1</sup> [Wenger, 1987] p. 390

<sup>2</sup> [Wenger, 1987] p. 390

<sup>3</sup> [Wenger, 1987] p. 392

<sup>4</sup> [Wenger, 1987] p. 236

computer systems, and because human private tutors and their students share extensive conversational capabilities and common backgrounds that are completely inaccessible to current computers<sup>1</sup>.

#### 8.1.4.4 Feedback

In the diagnostic process, feedback serves two purposes: it informs the student about the state and progress of his own knowledge, and it helps the system to verify if its own didactic measures were appropriate<sup>2</sup>.

The foundations of the “feedback loop” to update the student (user) model are laid out in section 8.1.4.3.2, more information is available in [Kerner and Freedman, 1990, p. 895]. See also section 3.1.2 for the meaning of feedback in the various learning theories.

Giving feedback to the student leads to an improved locus of control<sup>3</sup> and is expected to increase his motivation. There are different times at which feedback can be given to the student. Either immediate<sup>4</sup>, late<sup>5</sup> or post-exercise as implemented in chapter 6. Furthermore, feedback can be provided by the system on its own, or on demand by the student, where detailed feedback is only given when the student asks for it<sup>6,7</sup>. Longitudinal adaption to the learner can be done to give ideal, possibly immediate, feedback to the student, while preventing him from doing the work by repeated requests for help, i.e. what is called “gaming the system.”<sup>8,9</sup>

Verifying the impact of didactic operations allows determining what the student has already learned, what he has not learned yet, and what he is about to learn – [Michaud et al., 2000] describes a “zone of proximal development” (ZPD) that highlights the latter. To implement this, some domain specific heuristics are again needed.

To realize a system that pays attention to zones of proximal development, not all information that is available should be fed to the user immediately, to not confuse or overburden him. A possible method of filtering such messages is outlined in [Hylton et al., 2005]. If filtering of information is not sufficient, other techniques like changing the viewpoint of the system may help, as outlined in [Dietrich et al., 1993].

Last, when giving elaborated feedback<sup>10</sup>, hints should be given in great detail, e.g.

---

<sup>1</sup> [Wenger, 1987] p. 394

<sup>2</sup> [Heer et al., 2004] pp. 463

<sup>3</sup> [Corbett and Anderson, 2001] pp. 245

<sup>4</sup> [Corbett and Anderson, 2001] pp. 245

<sup>5</sup> [Nathan, 1990] pp. 407

<sup>6</sup> [Shah and Kumar, 2002] p. 171

<sup>7</sup> [Corbett and Anderson, 2001] pp. 245

<sup>8</sup> [Baker et al., 2004] pp. 383

<sup>9</sup> [Nathan, 1990] pp. 407

<sup>10</sup> See “elaboration” in section 3.1.4

what scenario and approach are appropriate, which commands could help, and which commands would be good for troubleshooting. See section 7.3.3 for more details on what students considered helpful during the course of exercises in the Virtual Unix Lab.

### 8.1.5 The user interface

While the pedagogical, domain and student model provide the base for didactic actions and interactions, the user interface is what separates them from the user. It processes data flowing between the user and the system in both directions, and in the process of doing so translates between the system's internal representation and some language that the user understands. The user interface component works closely with the other components, but as the design for it is still influenced by a number of unique decisions, it is considered as a separate component<sup>1</sup>.

The practical impact of the user interface on the success of the system of knowledge communication has two reasons. First, it is what the user faces and interacts with, and the external presentation of internal material what makes the student learn in the end. Qualities like ease of use, attractiveness, usability, and minimizing the load on working memory can be essential for the acceptance of the system<sup>2</sup>. Second, the constant progress in media technology keeps on providing new tools that may be fit better for the task of knowledge communication than their predecessors, and they can thus have an impact on the design of the entire system<sup>3,4</sup>.

Beyond giving status reports for the user, an important task of the user interface is to provide on-line analysis of the ongoing activities of the user, and communicate feedback. The system can provide more interactivity that way, but the user interface has to support this style of interaction to e.g. prevent or mitigate dangerous situations. A possible implementation for the Virtual Unix Lab would require to extend the check-based testing done on the pragmatial layer realized in chapter 6 to testing on the syntactical or semantical layers. See the discussion of on-line diagnosis in section 8.1.4.3.1, and section 10.3 for further thoughts.

An advanced requirement of the user interface is that it should support splitting a full exercise into several smaller parts which may be presented separately, and where each part is only displayed when the preceding part is solved properly<sup>5</sup>. Care should be taken to not limit the explorative character of the exercise system by halting the flow of the exercise, though. For the Virtual Unix Lab, it would be possible to display the full exercise text, but to also have a separate tutorial component that interacts with

---

<sup>1</sup> [Wenger, 1987] pp. 21

<sup>2</sup> [Shneiderman, 2004] pp. 451

<sup>3</sup> [Wenger, 1987] pp. 21, pp. 298, pp. 314

<sup>4</sup> [Shneiderman, 2004] pp. 173

<sup>5</sup> Compare this to the book suggested by Thorndike and Gates in section 3.1.3

the user on a per-task base, recognizing which task of the exercise the user currently works on, and providing assistance on that task. This assistance can be offered either in cooperative (on-demand) or an automated fashion. See chapter 10 for further discussion.

Finally, communication between the user and the system can happen in many types of interaction<sup>1,2</sup>. Natural language seems obvious, to have the system talk (and understand!) the language of the user, not vice versa. As using natural language in dialog with a machine may impact user acceptance, placing the user interface in the domain of contemporary graphical user interface design seems more appealing, esp. when considering the experiences made by others in that area, as the discussion about using natural language processing in section 8.1 shows.

Guidelines for the implementation of user interfaces can be found in parts 10, 11, and 12 of [ISO 9241, 2003], theoretical foundations can be found in [Nielsen, 2001], [Shneiderman, 2004], and [Norman, 2002]. A number of tools for evaluating the usability of a user interface and to improve it in a latter implementation steps, tools like the ISONORM 9241/10 questionnaire<sup>3</sup> and the IsoMetrics Usability Inventory<sup>4</sup> can be used.

## 8.2 Fundamentals of user adaption

In order to use a technical system, a user needs to have an overview of what information exists in general, which information is available in the system, and how to find it. This requests a lot of effort from the user, and an alternative for the system is to help the user in an active way<sup>5</sup>. [Johansson, 2002, p. 2] cites Fischer for stating that “the challenge in an information-rich world is not only to make information available to people at any time, at any place, and in any form, but specifically to say the right thing at the right time in the right way.” For complex applications, assistance can be provided in several ways: unused functionality should not get in the way of used functionality; unknown existing functionality should be made accessible or delivered at times when it is needed; and commonly used functionality should not be difficult to be learned, used and remembered<sup>6</sup>.

There is an important difference between an assistant and a tutor. While the former helps the user to complete a task by possibly performing actions automatically, a tutor helps in learning how to use the system itself, and will not attempt to do the actual work for the user. A tutor still has to make the right information available at the right

---

<sup>1</sup> [Bruns and Gajewski, 2002] pp. 48

<sup>2</sup> [Shneiderman, 2004] pp. 71

<sup>3</sup> [Prümper and Anft, 2006]

<sup>4</sup> [Gediga et al., 1999]

<sup>5</sup> [Kobsa, 1990] pp. 1

<sup>6</sup> [Johansson, 2002] p. 3

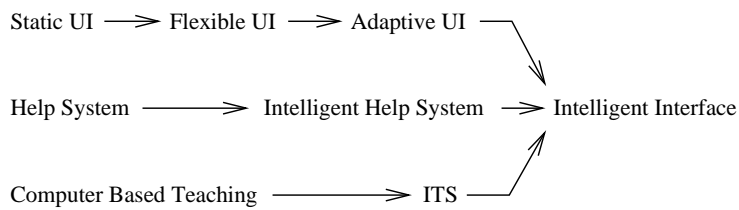


Figure 8.3: Terms: Adaptive User Interfaces and Intelligent Interfaces. Image source: [Dietrich et al., 1993, p. 14, Figure 1]

time. Adaption may be needed in the learning environment and exercises that the user should perform.

The overall benefits that are expected from user adaption in a learning system are increased effectiveness, efficiency and acceptability<sup>1</sup>. In order to offer support in the system, it has to collect data about the user's goals, plans, beliefs, knowledge and assumptions. Those build the user model. By analyzing the information that the user requested, his plans and goals can be learned, and more information can be provided to him. By taking into account what a user knows or doesn't know in a specific situation can prevent the teaching process from becoming boring or asking too much of the user. Furthermore, wrong assumptions can be recognized and communicated<sup>2</sup>. Depending on the system and the specific learning context, specific default assumptions can be made<sup>3</sup>, see the discussion of "stereotypes" in section 8.2.3 .

Figure 8.3 lists the various components that are considered as being part of a "intelligent interface", and their relationship. In order to provide optimal help for the learner, an adaptive user interface, an intelligent help system, and the components of an intelligent tutoring as discussed in section 8.1 are considered worthwhile to achieve. Components that are of interest for adaption also include context-sensitive help, adaptive hypertext, assistance in navigation, as well as a personalized news filter in the user interface. Of these components, the first three are of interest for the Virtual Unix Lab, and also under the aspects of a learning environment and its help system.

A comment should be made about the term "adaption" here. There are a number of related terms in that area, e.g. personalization, adaptable, adaption and adaptable systems. "Personalization" is used here to describe activity that is initiated and controlled by the user<sup>4</sup>. It requires a system that can be changed by the user, which is called "adaptable."<sup>5</sup> "Adaption" means a change in the system that is initiated by the system

<sup>1</sup> [Johansson, 2002] p. 4

<sup>2</sup> [Kobsa, 1990] pp. 243

<sup>3</sup> [Kobsa, 1990] p. 5

<sup>4</sup> [Dietrich et al., 1993] p. 17

<sup>5</sup> [Fischer, 1993] pp. 55



itself to react to activities of the user<sup>1</sup>. Such a system is called “adaptive.”<sup>2</sup> The change may be verified with the user, i.e. be “user-controlled”, but need not be. Giving the user final control e.g. by offering adjustable preferences helps increasing the user’s motivation via increased locus of control<sup>3,4</sup>. [Fischer, 1993, pp. 55] continues this discussion about why design environments should be adaptive and adaptable.

The system discussed here is intended to offer adaption to the user by being adaptive. The following topics will not be discussed here to down narrow the focus; references to literature are given here for further information:

**Shared decision making** means both the system and the user perform adaption. This topic is not considered here in order to prevent the user from distraction from learning and the goal of the exercises. The topic is further covered in [Fischer, 1993, pp. 58].

**Support for a specific application:** The student should learn how to use an existing system. Use of the system itself should not be changed for the user, in general. Instead, adaption should happen in the curriculum, exercises and by adjusting the information given to the student. [Johansson, 2002] further covers this topic.

**Group teaching** in user adaption is similar to group teaching in tutoring as discussed in section 8.1. While learning in groups, including adaption of the exercise to the whole group, is not discussed here, using personalization for individual members of the group is considered on-topic for two reasons. First, it is expected to have a very disproportional ratio between students and teachers in the classroom situation that’s being assisted here. As such, the teacher can offer little or no time to take individual students into account. Having a training system that offers user adaption can assist students where a teacher cannot, esp. if the groups grow larger<sup>5</sup>. Second, large groups of students show a variety in background knowledge and motivation. Offering classroom teaching that includes the full range is not always possible, and again a training system that offers adaption to single users can help in this situation. In sum, an adaptive tutoring system can help students in learning, assuming that its role is clear as assistance.

See section 8.1.4 for a discussion of the student model that is being used to perform adaption upon.

**Elderly people and people with disabilities** may need specially tailored information, contents and possibly modes of display. Areas where this may have an impact

---

<sup>1</sup> [Dietrich et al., 1993] p. 17

<sup>2</sup> [Fischer, 1993] pp. 55

<sup>3</sup> [Corbett and Anderson, 2001] pp. 2001

<sup>4</sup> [Schulmeister, 2007] pp. 146

<sup>5</sup> Personal experience shows that tutoring a group of about 8 students is the maximum where you can take care of every student’s individual needs. This was confirmed in the Fall term 2005 at Stevens Institute of Technology in Hoboken, NJ, USA. Student groups at the University of Applied Sciences, Regensburg, Germany, are usually between thirty and fifty students, making considerations of single students in tutoring next to impossible.

are operation of the computer's user interface hardware, recognition of the navigation elements and contents as well as visual impacts like issues in color recognition. Measures may require the use of large user interface elements or non-visual methods, e.g. reading what is written on the screen to a visually impaired user.

While this goal is considered worthwhile in general, it is not in the focus of this work. At the same time, care is being taken that the work described here will pose no further restrictions on people with the named issues. More information on this topic can be found in [Kobsa, 1999].

**Privacy:** Methods are available to recognize, mitigate and/or prevent activities in systems that may lead to deprivation of privacy. While they are usually found in the area of databases or general applications that handle personal data<sup>1</sup>, they could be considered for exercise systems like the Virtual Unix Lab as well.

Influencing factors are users' concerns of privacy as well as existing privacy legislation. Users' own preferences on their privacy can be tuned on a per application base; getting those preferences into accordance with privacy legislation require customized solutions, as they are too different, and no frameworks exist that supply solutions that cover both areas<sup>2</sup>. The fact that privacy laws differ widely between various countries does not make things easier. When planing a system, including tutoring systems, that will be deployed in several countries, this needs to be taken into account upfront to avoid unpleasant surprises later<sup>3</sup>. The impact of different countries' laws can start with different restrictions on the trans-border flow of personal data, which needs to be considered for Internet-based systems common today<sup>4</sup>.

Privacy considerations should be considered when user data is acquired and stored over long terms. For short term learning goals like in exercise systems, this should not be a problem in general<sup>5</sup>, and it is thus not considered further here.

More information on privacy can be found in [Kobsa, 2002], [Kobsa and Schreck, 2003, pp. 149], [Teltzrow and Kobsa, 2004a], and [Teltzrow and Kobsa, 2004b]. Collaboration in groups was explicitly excluded here, yet privacy is of concern there too, see [Patil and Kobsa, 2005, pp. 329].

**Separation of responsibilities:** Responsibilities in a learning environment may include contents and curriculum, realization, interfaces and integration. In larger systems where credibility, accountability and accreditation are of importance it may be required to implement this, to prevent e.g. exam writers from being able to look at students' grades or keep interface designers from being able to see details of tests that should be kept confidential.

---

<sup>1</sup> [Kobsa, 1990] p. 12

<sup>2</sup> [Kobsa, 2001b] pp. 1

<sup>3</sup> [Kobsa, 2002] pp. 1

<sup>4</sup> [Kobsa, 2002] p. 69

<sup>5</sup> [Kobsa, 1990] pp. 13

The solution to this problem is not only of technical nature with respect to database schemes and authorization, but corresponding legally binding non-disclosure agreements should be signed to help in this process<sup>1</sup>. This topic is not followed further here, more information is available in [Conlan et al., 2003, pp. 210] and in the literature on privacy, see above.

The following sections first talk about the meaning of “context” in exercise systems. As main source of information for the user model, it defines the base of user adaption. Adaptive services and multiple agents are discussed next, followed by techniques for modeling the system which leads to a definition of adaptive axes.

### 8.2.1 The meaning of context

In learning systems, “context” means situative information. Following Conlan and Power, it is “any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.”<sup>2</sup> They describe the following types of contexts<sup>3</sup>:

**Computing context:** network connectivity and bandwidth, hardware resources like I/O devices (mouse, keyboard, display, ...)

**User context:** user’s profile, location, nearby people, and current social situation

**Physical context:** lighting, noise level, traffic conditions, temperature

When focusing on computer based training systems, the simulated environment and the tasks given fit between the computing context (for the system itself) and the training system and its tasks (for the user context). The task of an adaptive tutoring system is to analyze the given context for the user, and offer appropriate tutoring<sup>4</sup>.

To avoid that the user feels helpless with the tutoring system making changes behind his back, the decisions should be communicated and possibly confirmed by personalization of the user<sup>5</sup>, see the discussion about locus of control in section 8.2 above, and in section 8.1.4.4.

---

<sup>1</sup> The approach of signing Non-Disclosure Agreements is currently practiced by the BSD Certification Group to prevent question writers and system developers from disclosing information that lead to unfair advantage of third parties during later certification exams. See <http://www.BSDcertification.org/> for more information.

<sup>2</sup> [Conlan et al., 2003] p. 207

<sup>3</sup> [Conlan et al., 2003] pp. 208

<sup>4</sup> [Conlan et al., 2003] p. 208

<sup>5</sup> [Conlan et al., 2003] p. 208

### 8.2.2 Adaptive services and multiple agents

Adaption can affect communication and functionality of the learning system. In the former case, only communication between the system and the student is affected, and the tasks for the student remain the same. In the latter case, the tasks that the student has to absolve are adjusted to his performance<sup>1</sup>. The adaption of interaction can be realized via adaptive services and multiple agents, by using components that know details about the user model and context of the current situation<sup>2</sup>.

Adaptable services and components should be composed in a way that models the flow of control and information between elemental services, and also reflect underlying business process modeling. Realizing a model of adaptable services requires a flow of information between the various components<sup>3</sup>. In any case, components can either act proactive or reactive; it would be an *active* handling of an adaptive system, though, not a passive one of an adaptable system<sup>4</sup>.

When multiple sources of data are used to establish the user model, separate software agents can be used to acquire specific data, act as tutor or – in the case of collaborating groups which is not covered here – as virtual team members<sup>5</sup>. Agents have to understand the task domain, a possible team structure, decision making processes and information about the user model of participating learner(s). Synthetisizing data from multiple sources may need classification and filtering. More information on the topic of agents in intelligent training systems can be found in [Yin et al., 2000].

### 8.2.3 Modeling techniques

Originally, constructors of learning systems wanted to imitate human interaction with computers to reduce cognitive load on the learner. Experience has shown that this was not successful for a variety of reasons. Since then, it has been established that the differences in interaction between humans and machines can be used to retrieve data that can also be used in building the user's model and for user adaption<sup>6</sup>.

Users with different background knowledge do exist, and a number of modeling techniques can be applied to determine in which group a certain user falls. The techniques introduced here are application of stereotypes, clustering, and plan recognition.

Stereotypes are used to recognize patterns of behavior of a user. Based on the assumption that they were recognized properly, further implications can be made about the

---

<sup>1</sup> [Dietrich et al., 1993] pp. 17

<sup>2</sup> [Conlan et al., 2003] p. 208

<sup>3</sup> [Conlan et al., 2003] pp. 208

<sup>4</sup> [Conlan et al., 2003] p. 209

<sup>5</sup> [Kobsa, 2001a] pp. 57

<sup>6</sup> [Johansson, 2002] p. 7

user, including his knowledge, plans, and behavior<sup>1</sup>. Building and recognizing stereotypes is only possible with some uncertainty. Assumptions are made based on the little data available from users' interaction, and which may not be adequate in general<sup>2,3,4</sup>.

Clustering is another approach. It does not try to categorize the user as a whole, but only build the user-model for specific areas called "clusters." The assumption then is that the user will perform according to those areas. For example, when using a content-based approach for clustering, a user may have mastered one area, but he may still be a beginner in another area. In a "clique based" approach, actions of a user are not observed separately, but in the context of a group of users with a similar goal in mind, as is e.g. the case for users in the Virtual Unix Lab, see the discussion on structural consistency in section 8.1.4.3.2.2. Clustering allows for more fine-grained classification here than use of stereotypes<sup>5</sup>.

When exercising plan recognition based on a user's history of interaction with a system followed by analysis and inference on a user, his knowledge, plans and actions, it allows making statements with more certainty than when using stereotypes. While this is desirable, it also requires a lot more data to be obtained in the first place. There's still some remaining uncertainty, but less than when using stereotypes or clustered approaches, as more data is used, and inferences are drawn on smaller areas based on that data<sup>6,7</sup>.

Using stereotypes and clustering require data from existing users. When those are not available, they can be initialized with values that stem from experience with the expected user base. An established method for characterizing typical user scenarios can be found in Cooper's "personas". In a user-centered design, they describes prototypical users, their expectations, wishes, knowledge and other parameters as determined by interviews<sup>8</sup>. While personas were originally used in software engineering, they can also be used on the smaller scale of initializing user models<sup>9</sup>.

Recognition of goals and plans can be achieved through plan composition, which enumerates all possible user actions and their results, then narrows down the number of possible plans until one or few are identified with sufficient confidence; see the use of artificial intelligence for plan recognition in section 8.1.2.2.4 and [Kobsa, 1993, p. 6]. Tools that can help in the process of plan recognition are libraries of plans and common mistakes. The former provides a list of likely goals with associated actions for recognition, the latter provides domain-specific lists of mistakes that can be expected<sup>1</sup>,

---

<sup>1</sup> [Kobsa, 1995] pp. 2

<sup>2</sup> [Kobsa, 1990] p. 7

<sup>3</sup> [Johansson, 2002] pp. 8

<sup>4</sup> [Rich, 1979]

<sup>5</sup> [Johansson, 2002] p. 9

<sup>6</sup> [Kobsa, 1995] pp. 2

<sup>7</sup> [Johansson, 2002] pp. 7

<sup>8</sup> [Cooper, 2004] pp. 123

<sup>9</sup> [Pruitt and Grudin, 2003]

<sup>1</sup> [Kobsa, 1993] p. 6

see the discussion about theories of bugs in section 8.1.4.1. An alternative to giving the user full freedom of learning as suggested in constructivistic learning theories<sup>2</sup> is to specify the task for the learner and guide him in that task. This is e.g. used in the Virtual Unix Lab, and contrasts the “general” help systems that were built to guess user plans like the Unix Consult<sup>3</sup>, OSCON<sup>4</sup>, and GOETHE<sup>5</sup>.

In sum, stereotypes can be used to determine both user model and system behavior, with clustering as possible refinement step. Application of plan recognition and their associated libraries will need further research. More information on how to apply these theories to user adaption in the Virtual Unix Lab are listed in chapter 11.

### 8.2.4 Adaptive axes

Adaptive axes describe the scales and units upon which adaption can be based. One or more of them can be used to determine how a system should adjust to the learner. The number depends on the exact area of application, the learning environment and the amount of data available in the user mode. The scales can contain values that are either visible or invisible to the student.

Visible attributes could be on a physical level that defines the signals and signs that are used in the combination in communication with the user – an example is the link annotation described in [Specht and Kobsa, 1999].

Invisible attributes cover the learning system’s internal logic and can affect attributes like freshness, progressive assistance, method and modularity shift, level of discourse, backtracking and graceful failure<sup>6</sup>. Further examples of levels for adjusting on the taxonomy of adaptive user interfaces include the learner’s skill level, the specific area of the domain that he’s currently practicing, and interaction style. The latter can include using query and answer, menu selection and command languages as well as reactive error correction versus active help<sup>7,8</sup>.

---

<sup>2</sup> See section 3.1.1

<sup>3</sup> [Wilensky et al., 1988] pp. 35

<sup>4</sup> [Kevitt, 2000] pp. 89

<sup>5</sup> [Heyer et al., 1990] pp. 361

<sup>6</sup> [Heer et al., 2004] pp. 463

<sup>7</sup> [Dietrich et al., 1993] p. 19

<sup>8</sup> [Fowler et al., 1987] pp. 345

## Chapter 9

# Design of tutoring and user adaption

The previous chapters from the first main part of this work have laid out the foundations for verification of exercise results in the Virtual Unix Lab. With the discoveries of the evaluation that more help is desired by students, adding extended help to user by means of tutorial and user adaptive components is approached in this second main part of this work.

This chapter outlines the design of the tutorial and user adaptive components for the Virtual Unix Lab. Topics covered include the goals of those components, the methodology used to create them, an overview of the domain model, and an outline of the software architecture.

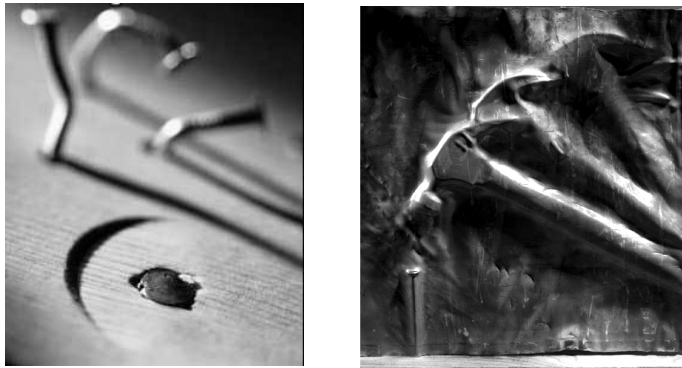
### 9.1 Goals of tutoring and user adaption

After setting up a basic training system, more assistance is desired to support students during their exercises. While a human teacher can look over the shoulder of a student during an in-class exercise, this is not so easy for a computer. The goal here is to imitate a teacher by using a tutorial component, and adapt it to the specific user using an adaptive component<sup>1</sup>.

Currently, the results of an exercise are only verified at the end, establishing success or failure of single parts of the exercise. While this helps to determine what went right and what went wrong, it does not tell *why* things went wrong (if so). A more detailed analysis is needed here. Figure 9.1 illustrates the current and desired situation in a different area of application where not only the final result is of interest, but also the events that lead to it, to possibly improve the final result.

---

<sup>1</sup> [Wilensky et al., 1988] p. 36



a) Verification after the exercise    b) Verification during the exercise

Figure 9.1: System administration is like hitting a nail with a hammer. Sometimes. Image sources: [Bent Nail, 2007], [Morell, 2004]

## 9.2 Methodology of tutoring and user adaption

An architecture for tutorial and user adaptive components for the Virtual Unix Lab is described in chapters 10 and 11. The methodology used is based on the theories and the models for domain, teaching, users and the user interfaces described in section 8.1 and section 8.2. Utilizing the foundations laid for exercise verification via domain specific languages in chapter 6, the architecture of the two components are described. For each component the domain, teaching and user model as well as considerations for the user interface are discussed.

The approach reflects the iterative design that was used to realize verification of exercise results in chapter 6. This includes repeated steps of evaluation and improvements, and is recommended in [Manaris et al., 1994, pp. 34].

## 9.3 The domain model

The domain model describes the topics that the tutoring system is intended to teach. It is established by two steps. First, the items of interest in the specific part of the domain are determined by decomposition of the exercise components. This is followed by a discussion on how to obtain a theory of bugs for this area.



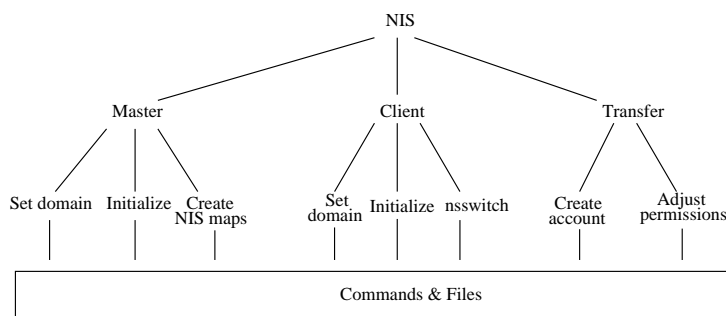


Figure 9.2: Goals and sub-goals of the Network Information System (NIS)

### 9.3.1 Content decomposition

The domain model that is used for tutoring and adaption in the Virtual Unix Lab is not explicitly modeled by rules, but realized by lab exercise machines which reflect the domain in real.

The domain model is thus modeled implicitly. The student is supported in understanding the domain by the exercises about the Network File System (NFS) and the Network Information System (NIS) that are available in the Virtual Unix Lab, the lecture notes, and other material that is available as discussed in section 3.2.4. A more fine-grained analysis of the target domain is discussed in section 8.1.3.

The general goal is to first embed the system as supplement into the existing lecture, and then allowing transition to a purely virtual training system in a second step as described in chapter 1. The basic conditions at which the defined system is still targeted are the same as described section 3, i.e. students of computer science at the University of Applied Sciences Regensburg.

To refine the learning goal of “Unix Cluster Management” with emphasis on the Network File System (NFS) and the Network Information system (NIS), those areas need further analysis for their sub-goals and single tasks. Information on the sub-goals like user and software management, system startup, etc. have to be made available to the student. In general, instructional information should be designed in multiple layers, where one layer adds information to the previous one, giving a hierarchical view on the domain knowledge. This approach is in conformance with Reigeluth’s “elaboration theory” described in [Reigeluth and Stein, 1983]. Figures 9.3 and 9.2 illustrate this analysis and the various layers involved.

Analysis of the exercises into smaller tasks happens according to the classification suggested for the Berkeley Unix Consultant. Classification of the topics involved and ordering to supplement building of a corresponding overlay model also helps with

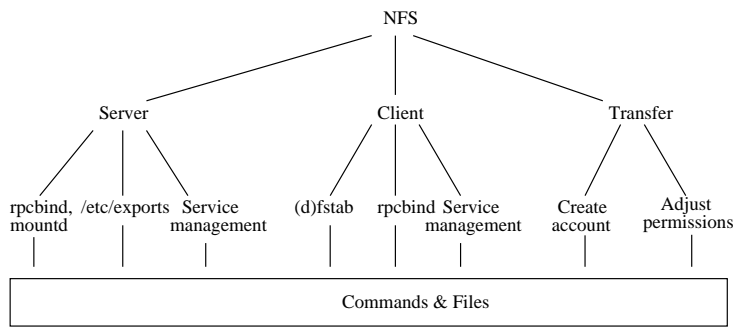


Figure 9.3: Goals and sub-goals of the Network File System (NFS)

defining stereotypes and their attributes<sup>1</sup>. Furthermore, decomposition as suggested in section 8.1.3 may help in defining the learning curve for students.

The methodology chosen here is to discuss a number of questions for each of the topics that are currently covered in the existing exercises in the Virtual Unix Lab, realizing an expert walkthrough. The questions are based on the experiences made in the past from mentoring students for lab exercises and in the Virtual Unix Lab. Alternatives would have been to use classification by assigning single commands from the SINIX manuals to topics and goals, as was done for the “Sinix Consultant”<sup>2</sup>, or cognitive walkthroughs with students. Use of manuals was rejected as there is no single manual that defines all the areas that are covered in the exercises, and cognitive walkthroughs were considered too time-consuming.

The following items were observed for all the topics of the NFS and NIS exercises in the Virtual Unix Lab:

**What does the student have to do?** This identifies on what level the topic is, e.g. either specify commands for items that are more on the behavioristic level, or give an outline of tasks to perform for commands on a higher level that require epistemic diagnosis.

**What problems can occur, how can they be identified?** Issues that may arise either in the learner due to wrong assumptions and beliefs, and that may lead to wrong steps. Also: problems that may arise in system configuration that will lead to future problems. For example when a user destroys parts of the system.

**Help** for the student can be provided either on the behavioristic or epistemic level. Examples would be to just tell a novice user what command to run (assuming he does not know the higher level concepts yet), or just indicate the epistemic

<sup>1</sup> [Chin, 1986] p. 25

<sup>2</sup> [Wahlster et al., 1988] p. 7

level by giving keywords and concepts to the user (assuming he can connect the keywords and concepts as appropriate)

**Believes - what wrong thinking can cause problems?** At times, students have wrong assumptions and/or knowledge about the system, and based on that, they will do the wrong steps. Identifying those steps and the thinking that led to them can prevent mishap.

**What viewpoints may exist:** Some problems may be seen from various positions. When setting up network services, a viewpoint from “inside” the machine (server view) or from “outside” a machine (client view) may be appropriate.

**Issues, Structural and Longitudinal consistency** are important for tutoring. Issues are discussed in the context of a theory of bugs in section 9.3.2, and consistency is covered in sections 11.3 and 11.4.

**Ways of data acquisition:** How can the activity related to the topic be detected? This is done either by analyzing the system state using a check script as described in section 6, or by using “on-line” analysis e.g. via keystroke tracing as described in section 8.1.5.

The above catalogue of questions was applied to the exercises that were previously introduced in chapter 6. Decomposition of the Network Information Service (NIS) exercise can be seen in appendix E, a similar analysis of the Network File System (NFS) exercise was considered but deferred. The results of this analysis and their further application are discussed in section 10.

### 9.3.2 Considerations for a theory of bugs

The term “theory of bugs” describes the specific mistakes that can be made in learning to master a certain domain’s knowledge, as described in section 8.1.4.1. Due to the lack of an existing theory of bugs for the domain of system administration, an attempt was made to create one as outlined below. The intention was to supplement the design of tutorial and user adaptive components in the Virtual Unix Lab as described in chapter 9.

This chapter describes the approach chosen for the creation of a (limited) theory of bugs. It analyzes the data from the Virtual Unix Lab’s database of existing results as well as the results originating from these efforts in a reconstructive approach as described in section 8.1.4.1. The exact database queries and their results that were performed to retrieve the discussed data are listed in appendix D for reference.

### 9.3.2.1 Adjusting the domain model

A full theory of bugs for the complete domain of system administration was considered ways too complex. As a result, a smaller subset of the domain was chosen. As the Virtual Unix Lab only covers a subset of that domain and as data is available for that subset of the domain, the focus of this analysis was set on system administration on the Unix operating system, with special emphasis on administration of both the Network File System (NFS) and the Network Information Service (NIS), including both client and server setup for these areas.

### 9.3.2.2 Analyzing existing exercise data

In order to use a reconstructive approach to determine a (limited) theory of bugs, a number of queries were performed on the database that kept the exercises and their results from past exercises from the Virtual Unix Lab. The following list outlines these queries and their individual results:

1. Determine the overall number of checks performed during exercises on both the client (vulab1) and server (vulab2)<sup>1</sup>.

These numbers are used in subsequent calculations that differentiate between those two machines.

2. Percentual number of checks that students failed to succeed on (numbers calculated manually)<sup>2</sup>.

The results show that the failure rate on the client (vulab2) were higher than on the server (vulab1) for both the NIS and NFS exercises.

3. Determine the number of checks performed for each user and machine<sup>3</sup>.

The results show that the number of checks is balanced between the client and the server. With the previous results, it can be said that more errors are detected on the client than on the server. Reasons for more errors on the client may be that its operating system (NetBSD) is less known, as it is not the primary system used in the lecture accompanying the exercises. It should be noted that the errors may not arise from mistakes (only) in the client configuration, though.

4. The next question is to determine the overall number of how often each check was ran as well as the number of failed checks, and their percentage<sup>4</sup>.

The list of check scripts includes ones “false positives”, i.e. the test is usually successful, and only fail if a user damages a working part of the system. This

---

<sup>1</sup> See results of query 1 in appendix D.

<sup>2</sup> See results of query 2 in appendix D.

<sup>3</sup> See results of query 3 in appendix D.

<sup>4</sup> See results of query 4 in appendix D.

can be seen in figure 9.4. The “false positive” scripts (`check-file-exists` and `unix-check-process-running`) are failed least often, as can be seen from their location near the bottom of the figure.

The remaining scripts find errors with a rate between 53% and 86%, i.e. there is no clear winner that indicates a significant number of errors.

5. A more detailed analysis of the failure distribution is based on both check scripts and also their parameters<sup>1</sup>.

The results confirm the fact that the “false positives” are failed least often. Again, the remaining errors are distributed evenly, no clear class (combination of a certain script with a specific set of parameters) can be found as failed with an extraordinarily high number, compared to the other ones.

6. Taking the percental distribution of each check script with its varying parameters, a box (scatter) plot can be generated to indicate where errors lie in a quantitative fashion. Data is taken from the “perc” column of the results of query 5 in appendix D).

The result is displayed in figure 9.4, it shows several points: The scripts that test “false positives” (scripts #3, #11) do lead to few errors. Scripts that test a wide range of topics and complexity (#2, #4) show a corresponding range, i.e. an equally wide one. Scripts that check for a specific, complex area are more likely to fail, the error rate is more than 50% here. Within the “complex” checks, the error rate is between 60% and 85%, with no clear majority (without further defining what “complex” means in this context). An explanation why testing for installed shells (#17) fails often is because the item is marked as optional in the exercise text.

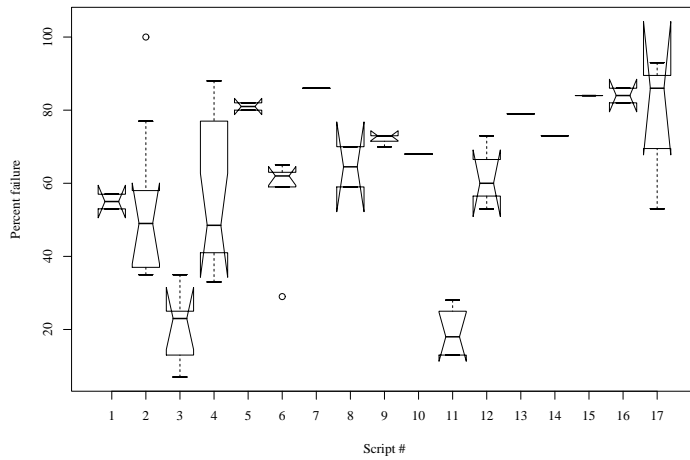
### 9.3.2.3 Results and conclusion

From the above examinations, no usable list of checks that fail frequently and that could be used as foundation for an enumerative theory of bugs can be determined, even for the limited area of system administration under Unix, with a focus on NIS and NFS.

The following sections approach tutoring and adaption without relying on a theory of bugs.

---

<sup>1</sup> See results of query 5 in appendix D.



Legend:

Script #	Script
1	check-directory-exists
2	check-file-contents
3	check-file-exists
4	check-program-output
5	netbsd-check-installed-pkg
6	netbsd-check-rcvar-set
7	netbsd-check-user-shell
8	solaris-check-installed-pkg
9	unix-check-file-owner
10	unix-check-mount
11	unix-check-process-running
12	unix-check-user-exists
13	unix-check-user-fullname
14	unix-check-user-home
15	unix-check-user-ingroup
16	unix-check-user-password
17	unix-check-user-shell

Figure 9.4: Error distribution of check scripts

## 9.4 Software architecture

A lot of software components for learning systems are readily available today, originating from the open source community<sup>1</sup>. Even without looking at the quality of those components, it is difficult to build a learning environment from these components. Difficulties increase with the requirements that are needed for an adaptive learning system<sup>2</sup>. “Adaptive” can have many meanings in this context, e.g. guidance, presentation and collaboration, and one author’s assumption does not necessarily match those of other authors, making it difficult to assemble a system from those components<sup>3</sup>.

Various architectural suggestions exist for constructing an intelligent tutoring system, using e.g. object oriented architectures that organize the tutor around objects that represent the knowledge to be taught, not around the various components of the tutor<sup>4</sup>. While it would be possible to realize tutoring and adaptive components for the Virtual Unix Lab, using this approach would require many changes in the existing system. Integration into the existing Virtual Unix Lab system is considered important for practical realization, and even if the immediate goal is “only” to define an architecture, changing the whole system for the sake of practical realization is beyond the scope of this work. As a consequence, a software architecture is chosen that allows to keep the existing system as a base, and extend it instead. Figure 9.5 shows the components that were added over the first design in figure 4.16 in bold. The following components were added:

**User interface:** Tutorial support and adaption should happen within the existing user interface of the course engine. While attention of the user is currently split between the telnet/ssh interface to the lab systems and the course engine which gives instructions to the user, no third interface should be added.

**User model:** Data about the user and the history of his interactions are already stored in the database. The database scheme can be easily extended to also store information about the ongoing exercises and further data on user interaction that can be used to build the user model.

**Tutor:** The tutoring component monitors the user’s actions during the exercise with the help of the scheduler, and updates the user model based on the available data. It communicates its decisions and any knowledge communication towards the user to the course engine, which acts as user interface for the tutor. See chapter 10 for more information.

**Adaption:** Equally placed as the tutoring component, the adaption component monitors a user’s exercise, compares it against data available in the user model and

---

<sup>1</sup> [Fink et al., 1998] pp. 7

<sup>2</sup> [Nodenot et al., 2004] p. 95

<sup>3</sup> [Nodenot et al., 2004] p. 95

<sup>4</sup> [Bonar et al., 1986] pp. 269

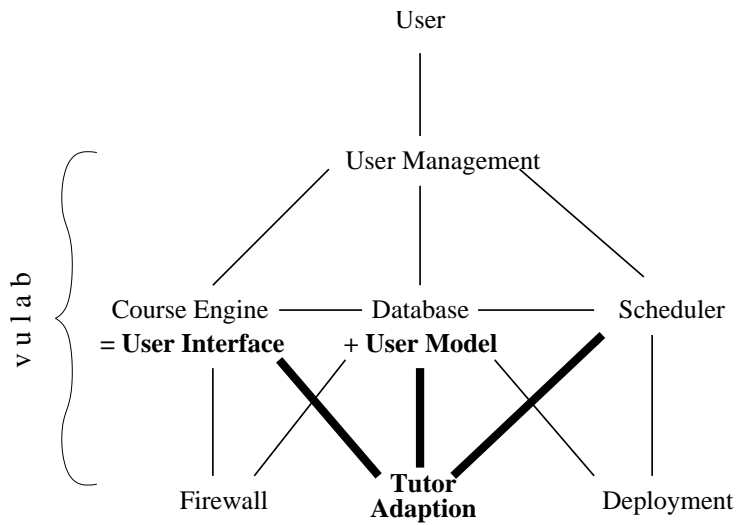


Figure 9.5: The Virtual Unix Lab with tutoring and adaption (new components in bold)

updates the user model so the tutor can adjust its feedback strategy. See chapter 11 for more information.

The original design is organized around the curriculum, and the bite sized architecture allows easy steering and testing<sup>1</sup>. Extensions for tutoring and adaption correspond to the design of the AVANTI system described in [Fink et al., 1998, pp. 5]. The tutoring and adaptive components contain both the domain model and the teaching model to the extent that this can not be placed into the database, preventing hardcoding as much as possible. Communication between the various components is performed by using SQL and the VUDSL described in chapter 6.

More details on tutoring in the Virtual Unix Lab are described in chapter 10, chapter 11 covers user adaption.

<sup>1</sup> [Wenger, 1987] pp. 144



# Chapter 10

## Architecture of tutoring

A tutoring component for the Virtual Unix Lab is expected to guide students in their learning experience, and to support the teacher in giving feedback to the student. The previous chapter outlined the overall design of the Virtual Unix Lab and identified where components for evolving towards a tutoring system would need to be placed. This chapter goes forward in that direction and focusses an architecture that can be used to realize a tutorial component for the Virtual Unix Lab.

Topics discussed here include establishing the teaching model by selecting from a number of possible approaches for tutoring, applying model tracing for diagnosis during exercises, and on-line diagnosis. Giving feedback and assistance is covered with details on the exact goals, assumptions made and challenges encountered, as well as their influences on contents and form of course material. Further topics include considerations for the user model, which is followed by aspects of the user interface.

### 10.1 Establishing the teaching model

A number of models for tutoring in learning environments are available as outlined in section 8.1. For practical realization, the question of which model and methods to use for the intended application arises.

This section describes criteria by which the selection is made for the Virtual Unix Lab, how they apply to a number of choices outlined in section 8.1.2, and decides upon one to implement tutoring. Both terms “pedagogical model” and “teaching model” are used with the same meaning in this section.

### 10.1.1 Selection criteria

This section outlines criteria to be considered for the selection of a pedagogical model for the Virtual Unix Lab. While a general outline of the selection process is given in section 8.1.2.3, emphasis is put on to the following points for the selection process:

**Pedagogical depth:** Does the model offer sufficient pedagogical depth to help the student, e.g. by supporting views, mal-tests and possibly reasoning?

**Procedural knowledge:** Can the method be used to model procedural knowledge, or can it only handle factual knowledge?

**State of the art:** Is the method described in the literature, and/or are reports on their realization available? Is the method known and well tested, or is it rather new and experimental?

**Integration:** How easy is integration of the method in question with the rest of the existing system as described so far? What parts are already there, which ones need to be created? How feasible is the latter?

**Data acquisition:** What effort is needed to acquire data necessary for the method? Is data already available in the system, are extensions to the system needed, or is data only available under certain constraints.

**Exercise maintainability:** Creating a system that can assist students is only one part of the work. Exercises have to be created in the system, and efforts for setup and maintenance of those exercises are considered important.

**Summary:** A short summary of the method.

### 10.1.2 Classical approaches with overlay architecture

The idea behind tutoring with the help of an overlay model is to model the problem domain and trace both correct and incorrect actions by the user. The idea is like putting a transparent slide over a map (the problem domain's model) and tracing the user's steps with a pen. The system then classifies the steps and acts accordingly. Variations exist to target specific student behavior in a differential manner, to improve detection of bad decisions and various extensions to adapt to the user. Classical approaches to tutoring via an overlay architecture are described in detail in section 8.1.2.2.1.

**Pedagogical depth:** The problem domain is modeled by one or more domain experts and/or meta-experts, which will determine relevant concepts and misconceptions for a target area, and define how to react to actions that are identified through the overlay model.

Besides the limitation to the domain experts' knowledge, the system is restricted to pointing out problems, but no reasoning or explanation about the origins that led to the problematic situation is possible. A possible counter-measure for this is the introduction of "mal-rules" to detect when users strive off the "right" path.

**Procedural knowledge:** The overlay method was specifically designed to handle procedural knowledge.

**State of the art:** A large number of papers describe how to employ this technique to real-life scenarios, illustrate how to model problem domains, gather data, and draw conclusions for didactic actions. As such, this approach can be considered well-documented and ripe for practical use.

**Integration:** Considering the application area of the Virtual Unix Lab and the foundations work performed here, the didactic principles for teaching system administration from chapter 3, and the methods for verifying exercise state and results from chapter 6, many of the foundations needed to implement tutoring via overlays are available.

**Data acquisition:** The amount of data available during the exercise is considered sufficient to drive the tutoring process. This applies at least for static analysis of exercise status and progress, and to some extent even for analysis of online interaction of the student.

**Exercise maintainability:** The preparation efforts for exercises – specifying reactions to events etc. – are considered acceptable, with an estimate of linear growth of didactic actions with exercise length/complexity, instead of exponential growth.

**Summary:** The classical approach using an overlay architecture is flexible, and while it remains open for extensions like adaption and more fine grained structures, it is still easy to realize.

### 10.1.3 Cognitive approach

The cognitive approach goes back to cognitive psychology. Therein, no strict rules are outlined for the learner, but (internal) "knowledge" is built up from analyzing and understanding examples that are presented under varying viewpoints. See sections 3.1.1 and 8.1.2.2.2 for more details.

**Pedagogical depth:** The pedagogical offers of this approach are considered the best ones available. Knowledge is gained by the student through learning in different environments, by changing viewpoints, challenges and tasks without predefined learning units. The system ideally gives a maximum of feedback to actions taken by the student, who builds up knowledge about facts and rules on his own.

Variations exist in the form of changing scenarios given to the student, interaction styles, different kinds of interactivity and feedback styles.

**Procedural knowledge:** Cognitive and constructive methods can be used for both procedural and factual knowledge. Methods like decontextualization can be applied in both cases.

**State of the art:** There's a number of documents about the theories behind cognitive approaches in tutoring. Unfortunately the number of experiences from practical realizations of those approaches are rather limited.

**Integration:** Techniques for cognitive tutoring approaches vary widely, ranging from a number of interaction techniques for single students versus groups of students over multiple scenarios and points of view for one scenario to the related methods for data acquisition, which in term vary widely as well.

Within the Virtual Unix Lab, some of these methods and approaches are available or can be implemented with medium amount of efforts. Full support of cognitive tutoring requires a wide number of extra changes to be made to the system, which qualifies this tutoring approach as item for future research, and not an immediate candidate for easy realization.

**Data acquisition:** If availability of an appropriate range of cognitive tutoring tools could be ensured, the existing framework of the Virtual Unix Lab as described so far is expected to deliver the required data to drive this tutoring process.

**Exercise maintainability:** Preparation efforts for exercises with a cognitive tutoring approach are acceptable, again assuming that a set of tools to deliver the contents is available.

**Summary:** The method is considered as very good, but very difficult to realize, if at all. See Schulmeister's judgement in section 3.1.3.

#### 10.1.4 Linguistic approach

The linguistic approach takes events on the syntactical layer, infers actions on the semantical layer, and attempts to determine pragmatical actions from semantical actions. As such, it is appropriate for finding out about a user/learner's plans. Further information is available in section 8.1.2.2.3.

**Pedagogical depth:** The model supports plan recognition through defined relations between syntactical / semantical actions and their corresponding semantical / pragmatical meaning. Input at the appropriate level, with the corresponding amount of details, is required for this. A certain level of fault tolerance against "unimportant" glitches (e.g. mistyped commands) has to be considered, and a comparison between recognized plans, expectations, and exercise goals has to be made to determine if intervention is needed.

Didactic actions that are considered after such comparisons are of general nature and not specific to the linguistic approach of tutoring.

**Procedural knowledge:** Applying linguistic analysis only makes sense to procedural knowledge. Factual knowledge can not be used in this context beyond simple connections of terms, which is better done with semantic networks if needed.

**State of the art:** “Plan recognition” is well covered in tutoring and AI literature, esp. for the domain of Unix operating systems. Most attention is given to recognizing a user’s plan and assisting him, instead of applying tutoring techniques. The overall number of theoretical papers describing how to use plan recognition for tutoring is small, with even fewer practical examples.

**Integration:** To realize tutoring with linguistic methods, a learner’s plan on how to solve a given problem needs to be determined. In the current implementation of the Virtual Unix Lab it is possible to determine the status of the exercise systems. Capturing of user input data (via mouse, keyboard, network, etc.) is not implemented yet, and would be needed for this way of tutoring.

**Data acquisition:** When methods for capturing user input data are available, it is expected that the existing Virtual Unix Lab could deliver enough data from user interaction to properly recognize users’ plans on how they solve given exercises, and comment on them.

**Exercise maintainability:** Preparation of new exercises with support for linguistic tutoring techniques consists of two parts: Defining general rules that apply to each part of the exercises (e.g. how to handle mis-typed command names – when to assume it is a typo, when to assume that a student has problems with the user interface and to intervene, and when to help the student by suggesting what to type), and identifying specific patterns that apply only to one or a small number of steps in an exercise. The overall efforts required here depend on the level of detail that tutoring is intended for the student: Wenger cites 110 possible bugs for subtraction<sup>1</sup>. Considering the complexity of managing an average Unix system with several services gives an impression of the level of complexity that can be reached with this approach.

**Summary:** This approach is good for recognizing unknown plans. For the context of the Virtual Unix Lab this is of less importance, as the general plan is predefined via the exercise text. As such, the method requires too much effort for too little win.

### 10.1.5 Artificial Intelligence based approach

Using Artificial Intelligence (AI) for tutoring is similar to the overlay method, where user input is compared to expected behavior. The main difference here is that the problem domain is not explicitly modeled after an expert’s knowledge, but that start

---

<sup>1</sup> [Wenger, 1987]

and goal are defined, and the 'way' (list of steps to perform) is found by applying techniques usually found in AI. See section 8.1.2.2.4 for more information.

**Pedagogical depth:** Using the AI based approach can lead to finding the possible line or lines of thinking a student has done to reach a given point in an exercise, by e.g. taking preconditions and wrong assumptions, determining the possible lines of steps to reach the current situation to what the student actually did. It is expected to be able to find wrong assumptions and conclusions this way, that can then be reacted on with didactic measures.

Like for the linguistic approach, determining what didactic actions to employ to counter the findings made remains as a separate topic.

**Procedural knowledge:** Applying AI algorithms to search in the problem domain can be done on either factual or procedural knowledge, depending on the model of the domain only. It is expected that a corresponding domain model preferably constructed for procedural knowledge, though. Factual knowledge would better be represented via semantic networks.

**State of the art:** Little literature is available on this approach, esp. not for teaching in the Unix system administration domain. Related papers can be found for setup and verification of computer network setups, but again the number is very low, with no specific discussion about the tutoring techniques stemming from the findings determined by these techniques.

**Integration:** To apply AI methods to tutoring in the Virtual Unix Lab, both a description of possible steps in each situation is needed, as well as an inference mechanism that determines possible steps, compares them with what the user actually did, and then communicates the misconceptions that it has found in the user. While no such inference mechanism is available in the Virtual Unix Lab today, it is considered to be realizable with acceptable efforts, using common AI methods.

**Data acquisition:** Acquiring data for the AI inference engine is similar to the data acquisition and user input capturing technologies mentioned for overlay and linguistic approaches. Equivalent data would be needed here to compare users' steps against various possible lines of thinking.

**Exercise maintainability:** The real challenge in this approach is the definition of exercises: For one, a complete Unix system (in all its variants, like Solaris, Linux, NetBSD, ...) needs to be modeled with the different commands, their options, in what situation they can be run, and with what effects. With several thousand commands, many options per command and an almost arbitrary number of situation that various commands can be used, this is considered daunting – probably to the extent to not go near this approach for a complex real-life scenario, which would explain the low quantity of existing literature.

**Summary:** While this approach sounds promising in theory, it is way too complex to model in practice, when including all possible steps a user can do in Unix: the NetBSD operating system alone has more than 800 commands in the base install, and each of those has a moderate number of possible options. Adding about 100 configuration files in `/etc` alone gives an idea that this is not going anywhere in a lifetime.

### 10.1.6 Semantic networks and ontologies

Semantic networks describe the connections between terms, and ontologies can be used as formal representation of those connections. See section 8.1.2.2.7 for more information.

**Pedagogical depth:** Semantic networks can be used to both verify a student's competence in a specific domain, and also provide reasoning about concepts he has not learned, or learned wrong.

**Procedural knowledge:** Semantic networks mostly work for factual knowledge. Procedural knowledge is difficult to represent at best.

**State of the art:** A number of projects exist that show the use of semantic networks and related ontologies for teaching projects. The semantic networks are mostly used as a supplement to other teaching mechanisms, though, and rarely applied alone.

**Integration:** Integration of semantic networks into the Virtual Unix Lab would require a corresponding domain model. Given that the requirement for the Virtual Unix Lab is mostly checking procedural knowledge, the network could test if a student knows relevant commands, options and files in a given situation. Verification of this could only be used at specific points in an exercise.

**Data acquisition:** Input required from the Virtual Unix Lab to verify against a semantic network would be single commands and keystrokes, similar to what is needed for the on-line analysis described in sections 8.1.4.3.1 and 10.3.

**Exercise maintainability:** Exercises would need to be extended to identify what parts in a (larger) semantic network are required at a specific point in the exercise, and what relevant concepts, commands and files could or could not be expected by the student. Besides the connection between the exercise's single tasks and the semantic network, a semantic network for the domain of system administration and the Unix (or related) operating systems would be required, possibly requiring a corresponding ontology to be defined first. Most of this work would not be directly related to Virtual Unix Lab. No existing projects taking those efforts are known.

**Summary:** Semantic networks are more useful to apply at factual than at procedural knowledge, and related assessment. This, plus the effort of their creation don't make them a primary candidate for a teaching model in the Virtual Unix Lab. They could be considered as addition to a primary model in a later step, though.

### 10.1.7 Frames and scripts

Frames were introduced by Marvin Minsky to describe situations and objects with associated properties, which can be connected either to stereotypical defaults, or to other frames which further specify the property. Scripts are an extension of frames model procedural knowledge. See section 8.1.2.2.7 for more information.

**Pedagogical depth:** A model consisting of frames and scripts can help to verify a learner's knowledge against what he has already learned, and what he has to learn yet. When verifying a learner's knowledge, areas will be detected where specific facts are required, but where stereotypical information will be provided. This can be used as an indicator that the learner is proficient in the general problem domain by display of the stereotypical information, but not yet with the specific situation at hands. Feedback and teaching can be given accordingly.

**Procedural knowledge:** Frames were created to represent factual knowledge, and to model the relations between facts. Scripts are an extension to frames that describe procedures of events, which happen in the context of frames. The order of events is pre-defined, thus allowing to detect if a user follows a certain plan or not.

**State of the art:** A number of learning systems exist for factual knowledge and minor procedural problems, but no recent results show an application in the domain of system administration or any of the related domains.

**Integration:** To integrate frames and scripts in the Virtual Unix Lab, a corresponding domain model and a processing engine for it are required. Based on that, situations could be modeled with frames, and possible ways of interactions could be described in scripts.

**Data acquisition:** On-line analysis could be used to verify if a script is followed or not. The existing diagnosis that is based on verifying the state of the Virtual Unix Lab would not be sufficient to verify the steps of a script.

**Exercise maintainability:** Description of the exercise would happen by modeling key situations like the start and end, and any important intermediate steps as frames. Scripts would be written to recognize steps that lead from one frame to another one. Those scripts could include correct steps as well as wrong ones, to detect if a learner makes mistakes or follows a wrong line of thinking.



**Summary:** Frames and scripts require on-line diagnosis and a complex domain model. Both require major issue to realize, and thus make this approach unlikely as an extension of the existing Virtual Unix Lab.

The general approach is worth to re-consider when on-line analysis is available in the Virtual Unix Lab, though. This is further discussed in section 10.3.

### 10.1.8 Bayesian networks

Bayesian networks are directed acyclic graphs, which describe situations and the probability of any succeeding situations. The models can be generated automatically as described in section 8.1.2.2.7.

**Pedagogical depth:** Assuming a properly modeled and trained network, bayesian networks can be used to predict a user's behaviour, detect any deviations and provide feedback at what point the deviation happened, and what the proper actions would have been.

**Procedural knowledge:** Bayesian networks can be applied to factual and procedural knowledge equally, and detect changes over time.

**State of the art:** Bayesian networks are used in several projects. On a number of them, they are used as supplement for other methods, though, and aren't used as the primary teaching model.

**Integration:** Integration of bayesian networks into the Virtual Unix Lab would require several steps: first to setup a basic model of the domain, which would then be trained in a second step, to indicate what steps are expected and "good", and also what mistakes can be made. For productive exercises, this model could then be used as described. Data input for the Virtual Unix Lab would preferably be gained though on-line analysis as discussed in section 10.3, although the existing model of analyzing the current situation of the lab machine's state may be used as well, with less confidence for statements.

**Data acquisition:** Depending on the underlying domain model, the existing analysis of the lab machine state can be extended with on-line analysis.

**Exercise maintainability:** Assuming that the Virtual Unix Lab system is extended to train a bayesian network with data from existing exercises, and to provide feedback, it is expected that the overall overhead of exercise maintenance is rather low, assuming that the network is trained by existing exercises. Individual exercises could extend the graph that reflects the underlying domain model to some extent.

**Summary:** Bayesian networks could be used as supplement to an existing teaching method, to ensure predictions and statements. The low overhead of when maintaining exercises is considered good, the necessary infrastructure needs to be implemented in the existing system, though. In summary, bayesian networks could be considered as worthwhile addition for future versions of the Virtual Unix Lab, but not as primary teaching model.

### 10.1.9 Comparison

After the previous sections have given an overview of the possible methods for realizing tutoring, this section compares them and draws a summary on which method is most likely to lead to success.

**Pedagogical depth:** All the named methods have the required pedagogical depths. Overlays are easiest to realize, cognitive methods would be preferable from a strictly pedagogical point of view. The linguistic and AI-based approaches as well as frames & scripts and bayesian networks would be very good to find problems, offer reasoning on how and why things went wrong, and how to improve the situation. Semantic networks and ontologies could be added to verify conceptual knowledge.

**Procedural knowledge:** The Virtual Unix Lab trains mostly procedural knowledge. For this, overlay methods, cognitive, linguistic or AI methods are better fit than semantic networks. Frames & scripts and bayesian networks could be used to some extent, but are not ideal as primary teaching model. They could be used for later improvements.

**State of the art:** The overlay and cognitive methods are well documented, while publications on linguistic and AI-based approaches are rather sparse. This goes for both theories as well as practical projects that use them. Similar observations can be made for semantic networks and ontologies, frames and scripts, as well as bayesian networks.

**Integration:** Overlay methods can be easily integrated, while cognitive methods require a lot more effort, as also observed by Schulmeister. Linguistic and AI-based approaches could be integrated into the existing system with moderate efforts. Similar efforts are needed for semantic networks and ontologies, frames and scripts, and bayesian networks.

**Data acquisition:** This is easy to realize for overlays in the existing system. Cognitive methods are doable if appropriate tutoring tools are added to the existing system. Linguistic methods require inputs from on-line diagnosis, which is challenging to realize. AI-based methods should be easier, but will require data acquisition from on-line sampling instruments, too.

Approach	Pedagogical depth	Procedural knowledge	State of the art	Integration	Data acquisition	Exercise maintenance	Summary
Classical w/ overlay	+	++	+	+	++	+	++
Cognitive	++	++	0	-	0	+	+
Linguistic	+	+	-	0	0	--	0
AI	+	+	-	0	0	--	0
Semantic network	+	-	0	0	-	-	-
Frames & scripts	+	+	-	0	-	0	0
Bayesian networks	0	+	+	0	-	0	0

Figure 10.1: Comparison of tutoring approaches, from best (++) to worst (--)

Semantic networks and ontologies as well as frames and scripts would require on-line diagnosis to exist, which is not available in the Virtual Unix Lab as described so far. Bayesian networks could benefit from on-line diagnosis as well, but they could be based on the existing system as well, with a moderate amount of work.

**Exercise maintainability:** This is considered easy for overlay methods, with a linear connection between the amount of maintenance needed and exercise volume. Bayesian networks are expected to impose a low overhead on exercise maintenance as well. For cognitive methods, semantic networks and frames & scripts the efforts increase rapidly and can quickly explode into an unmanageable state. The effort for both linguistic and AI-based methods is considered very high, too.

**Summary:** The overlay approach is small and manageable. The cognitive approach needs a lot of work on the course engine. Linguistic and AI-based methods are too complicated with respect to exercise preparations. Semantic networks and frames & scripts require both work on the existing Virtual Unix Lab as well as esp. on exercises, whereas bayesian networks would require changes to the existing Virtual Unix Lab system as well.

Table 10.1 summarizes the situation. Following it, the further proceeding is to focus on using an overlay architecture with checks for exercise results, as they are easy to implement and maintain. Furthermore, the paradigm of direct assignment of credit and blame is realized for feedback for the same reasons. Due to the lack of a useful theory of bugs, detection of errors is done in an ad-hoc way with “false positives”, based on experiences made in prior exercises in the Virtual Unix Lab.

## 10.2 Using model tracing for diagnosis during the exercise

The goal of model tracing for diagnosis during the exercise in the Virtual Unix Lab is to determine what skills and topics the student is proficient in, and then improve those skills and topics that need further improvement. That way, cognitive adaption is intended to happen in the learner by giving appropriate feedback to him. See also section 10.4.5.

The approach outlined here is to apply the direct assignment of credit and blame paradigm as described in section 8.1.4.3.2.1. In contrast to the system used for diagnosis so far, no pure behavioristic post-hoc diagnosis is used. Instead, model tracing with the help of an overlay model as described in sections 8.1.4.3.2.1 and 8.1.2.2.1 is used. In contrast to a semantical network that can be used for rather simple behavioristic tasks, those represented in the Virtual Unix Lab are modeled as procedural network, see section 8.1.2.2.1.

Realization in the Virtual Unix Lab includes a full analysis and decomposition of the subject matter as outlined in section 9.3.1, and to define exercises that work from smaller tasks towards specific learning goals. Verification of the tasks and goals can be done by using check scripts, and noting their results to realize model tracing. The overlay model is implicit in the exercise in this case, defining what should be verified, and how – which checks to expect to fail, and which to expect to succeed.

Description of errors and error classes can happen though the VUDSL and its primitives as defined in chapter 6 and section 8.1.4.1. Checks for a common class of errors can be realized by passing parameters to check scripts testing for the class by using the VUDSL. As an example, checks can be made to see if specific settings in the `/etc/rc.conf` file were disrupted, important packages were deinstalled, files deleted or processes terminated. Also, deadlock situations can be detected, like when a NIS/NFS client is started without a corresponding server, which leads to nasty hangs for these services<sup>1</sup>.

## 10.3 Investigating on-line diagnosis

The check scripts described in the previous section test the system state after user actions. While this is largely sufficient, looking closer at the user input while changes are still being made would be useful as supplement, to realize behavioristic on-line diagnostics as described in section 8.1.4.3.1

---

<sup>1</sup> Clients using the Network File System (NFS) wait until their server is (back) up when it is gone. This waiting can cause the whole system to hang and wait, which led to the expansion of NFS to “Nightmare File System” as described in [Weise et al., 1994, pp. 283].

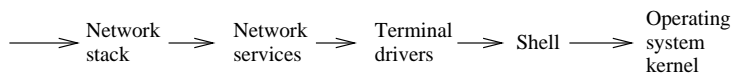


Figure 10.2: The path of incoming information

Theoretically, diagnosis can be done on a number of semiotic layers:

**Syntactical layer:** Analyze network traffic, keystrokes and mouse activity and determine activities performed. This is difficult as no context will be available, e.g. for things like tab completions, history recall or wildcard expansion on a command line interpreter, or for position of a mouse pointer and contents of a screen to tell the effects of a mouse click.

**Semantical layer:** This could analyze the history files written by a shell, or – using a modified shell – commands typed by users directly as e.g. described in [Matthews et al., 2000]

**Pragmatical layer:** This would require knowing/understanding the plan that the student expresses towards the system by activities on the semantic and syntactical layers. Inference of the student’s plans is needed for this, which in turn requires methods that analyze the lower layers again, as the user’s brain cannot be scanned (yet).

While “context” will still be needed to make use of data from the semantical layer, choosing an approach on the syntactical layer even more of data. As such, the focus here will be on the semantical layer.

The general path by which commands enter a Unix based lab system are illustrated in figure 10.2: The lab machine’s operating system receives input over the network, using its network stack. The network stack then passes that data on to one of the network daemons that are used for interaction, e.g. the secure shell (sshd), telnet (telnetd) or remote shell (rshd) daemon. These pass on the characters extracted from the network packets to a command shells using the Unix terminal (tty) or pseudo terminal (pty) interface. The “shell” that gets the characters passed is one of several command line interpreters available in Unix, e.g. standard systems have “sh” and “csh”, but there are others that differ slightly in functionality. In turn, they assemble commands from the single keystrokes passed in via the network, and then either perform internal action or run other commands from the system via the `exec(2)` system call<sup>1,2,3</sup>.

For the practical realization, there are several places in this setup where data can be acquired, defined by the components outlined above. Here is an analysis of what will be needed to get data at that part, and how feasible that is to realize:

<sup>1</sup> [Mayer, 2001] pp. 24

<sup>2</sup> [Stevens, 1992] pp. 325

<sup>3</sup> [Stevens, 1994] pp. 162

**Network stack:** Mining data at this level would mean to sniff and analyze network traffic, or hook into the operating system's network stack. Leaving the technical difficulties, plus the multitude of possible operating systems that this has to be done for aside, analysis on this level would require more context than is available here. I.e. data on the harddisk for wildcard expansion, history substitution, tab completion, or interpretation of mouse clicks in graphical user interfaces. Including the technical difficulties, this can be regarded as not feasible.

**Network services:** Various daemon processes realize the network services that handle input and output for command line interfaces, e.g. rshd, sshd and telnetd. They receive single characters over the network, and again no context is available for reliable inference of the semantic level. Also, technical realization again is problematic due to (non)availability of source code and the number of network services and operating systems that would need patching, which also makes this method not very likely for successful deployment.

**Terminal drivers:** Intercepting user input at this point means modification to every operating system's terminal (tty) and pseudo terminal (pty) drivers, and will yield only syntactic information again. This fact, the number of systems to change, their changeability (i.e. non-availability of source code), and the technical expertise needed for the required modifications speaks against this approach, too.

**Shell:** "The shell" is actually available in several incarnations, some available as open source, but the ones often shipped with commercial Unix systems are of closed source nature, and thus cannot be changed easily. As the shell would be in a good position to determine information on the semantic level this would be a good place to start, as e.g. documented for the USCSH<sup>1</sup>. Problems arise again from the number of shells that would need changing, and the partial (non)availability of source code.

**Operating system kernel:** In the family of Unix(like) operating systems, commands are started by exec(2) and a number of related system calls. As every command is executed that way, they would be an ideal place to harvest diagnostic data. Problems are (un)availability of source code once more, the number of different operating systems that would need changing, and the technical expertise to perform those changes.

An approach of an assistant that performs plan detection and assistance in the Unix environment by analyzing system calls can be found in [Su et al., 2007].

**Others:** Another point where diagnostic data for on-line diagnosis and corresponding analysis could be gained is the firewall that all interaction with the lab machines has to pass through. While sources for the IPfilter firewalling software, which is used in the Virtual Unix Lab, is available, data would be only on the syntactic level again.

---

<sup>1</sup> [Matthews et al., 2000] pp. 121

When available, system accounting can be used to log all commands executed by the system. It requires no modification to the system, and in combination with a separate program that collects and analyzes data, this would be the most promising approach. Feasibility of the approach would require evaluation of system accounting in further depth, esp. on what exact data is available, and with what latency. No system so far is known to use this source of diagnostic data that could be used as reference.

In summary, the idea of collecting on-line data for diagnostic purpose seems easy from the outside, but the implementation details make the effort questionable for the Virtual Unix Lab. For the system level that data is needed for, too many system components would require changes that are either non-trivial from the technical side, or not possible as no source code for modifications is available. As not much literature is available for collecting on-line data for system-level diagnosis either, the following chapters will focus on a system that does not rely on on-line diagnosis, while keeping the option to add this at a later time.

## 10.4 Giving feedback and assistance

This section defines the goals for giving feedback and assistance during an exercise, then describes some of the challenges to consider, and how to master them in the current system. The changes will have an impact on the organization of exercises and course material, which is also discussed.

### 10.4.1 Goal

The goal of giving feedback is to assist the student during the exercise, and show existing problems in the problem-solving context without giving any hints on their solutions (at first), as discussed by Wenger<sup>1</sup> and in section 8.1.2.2.2. During the exercise, feedback is shown for those parts of the exercise that were already worked on, either if the work was successful or not. Additional help is given so that the user understands what the system is verifying. No feedback is displayed for the parts of the exercise that haven't been worked on yet. This scheme scales between the current practice of not displaying any feedback during the exercise at all to showing full feedback for all items after the exercise.

To find the part of the exercise that the student is currently working on, checks on the current situation are performed – see also the Zone of Proximal Development (ZPD) in [Michaud et al., 2000] and the course of an exercise a student may take in figure 10.3.

---

<sup>1</sup> [Wenger, 1987] p. 292

```

1. Step: -----
2. Step: +-----
3. Step: +++++-----
4. Step: ++++++++---+++-----

```

Figure 10.3: Possible course of an exercise (+=done, -=todo)

The didactic model underlying this approach is based on a teacher wandering around among students doing exercises in a classroom or lab, looking over their shoulder, analyzing the current situation and providing help based on the student's activities.

### 10.4.2 Assumptions

Questions to answer at this point are how to recognize what was already worked on, what is currently being worked on, and what is still open to do for the student. Some assumptions based on the existing Virtual Unix Lab and its purpose are being made here:

- Exercise parts should be worked on in a linear fashion
- Later parts build up on earlier parts; as such, it is better to learn (only) the basics in early parts, than to learn advanced skills in later parts, after passing the early and "easy" parts.
- Even if the student has skipped a part of the exercise and works on a later one, he will have a reason for this. To be less invasive, it is suggested to support the student in his move, instead of forcing him to go back to the skipped part.

With these assumptions, the procedure to recognize what part of the exercise the student currently works on is as follows: Check from the 'last' (figure 10.3: rightmost) part of the exercise to the 'first' (figure 10.3: leftmost) part to find what part was solved successfully last. Assuming that the student works in a linear fashion, he will work on the next unsolved part. Figure 10.4 illustrates this process.

### 10.4.3 Challenges

Two of the challenges that will be encountered are if a student skips parts of the exercise, and/or if he does not perform them in a linear sequence.



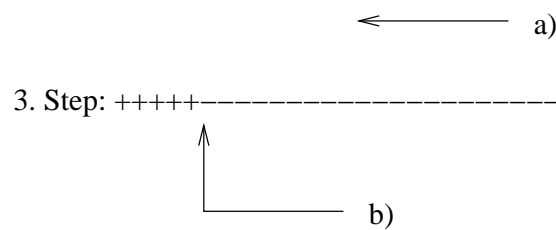


Figure 10.4: Going backward to find the latest (a) and next part being worked on (b)

A skipped part is displayed in the 4th step of figure 10.3. Assuming that the student skipped the missing parts for a reason, no efforts are made to bring him back “on track” to finish the missing parts if there are other parts still missing in the sequence of exercises. If the last (rightmost) part of the exercise is found to be completed, guidance and direction can be offered to help the student solve the missing parts.

Handling exercise parts that are not performed in sequential order is harder. This situation can occur in two cases. First, when a student performs his first pass through the exercise parts, and second, after he has finished the majority of the exercises and gets to pick the parts that are still missing. The assumption that exercise parts are approached in a sequential order from the start may be less true than for the general flow of the exercise.

Besides the assumption that the student always starts with the first (leftmost) unsolved exercise part, some heuristics have to be employed to verify what parts he *really* works on. Applying the linguistic approach for tutoring based on on-line analysis could offer the needed data. For the realization, the data model described in figure 6.15 and section 6.6 would need to be extended. The extension would define how to recognize a student working on the corresponding part of an exercise not only via (post-mortem, so to speak) checks as right now, but also by what patterns of interaction and commands he uses.

#### 10.4.4 Realization

To realize feedback, its content and form have to be considered. This is discussed in the following sections.

##### 10.4.4.1 Contents

Feedback is considered to be part of the user model as described in section 8.1.4.4. To provide elaborated feedback, the contents have to contain sufficient details. Feedback

provided by users of the Virtual Unix Lab (see section 7.3.3) and analysis of the existing exercises (see appendix E) show that the following points should be considered for detailed feedback:

- What scenario is appropriate?<sup>1</sup>
- What approach is appropriate?<sup>2</sup>
- What commands may be of help to solve the given exercise part?<sup>3</sup>
- What commands may be of help to troubleshoot the given exercise part?<sup>4</sup>

When providing help, two orders are possible: either provide general cognitive / epistemic help first (i.e. hinting at relevant topics, trouble shooting strategies, and so on), and then move on to behavioristic help (i.e. telling what commands to use, possibly including the necessary arguments for the situation at hands). If this does't help the student, he requests more help. The alternative is move into the opposite direction, giving behaviouristic help first, and if the student needs more help, give the necessary background later via epistemic help. The former approach is considered appropriate here, based on the assumption that the student will learn what topics are appropriate to consider to solve the problem, instead of blindly typing in the commands that the help system may provide when giving behavioristic feedback.

The effect that Baker describes as students "gaming the system"<sup>5</sup> can happen in the first incarnation of the system. Updating the student model each time help is given to the student and applying adaption based on structural and longitudinal consistency can be used to detect if a student abuses the feedback system as discussed in sections 10.5 and 11.5. Also, the help previously offered to the student in addition to the other data in his user model helps to determine what help is offered next, if more help is needed.

Discussion of the technical realization of displaying feedback during the exercise is discussed in section 10.6.

#### 10.4.4.2 Form of feedback

The form that feedback is given in can be either in a cooperative (on-demand) way, or automatically. The underlying pedagogical model for cooperative feedback corresponds to a student asking the teacher for help during a lab exercise. The teacher has to gain an overview over the situation that the student is in, the attempts that the student has made to solve the exercise part at hands, plus the real state of the system to

<sup>1</sup> See "viewpoints" in appendix E

<sup>2</sup> See "what does the student have to do" in appendix E

<sup>3</sup> See "help" in appendix E

<sup>4</sup> See "help" in appendix E

<sup>5</sup> [Baker et al., 2004]

give appropriate help. For automatic feedback, the pedagogical model corresponds to a teacher roaming around among students in the lab, looking over their shoulder and commenting their work if needed. Feedback can be given either immediately when a noteworthy situation is found, or after some delay, giving the student time to correct mistakes on his own, as is discussed in section 8.1.4.4, section 3.1.2 and [Wenger, 1987, pp. 296].

#### 10.4.5 Impact on organization of exercises and learning material

Providing feedback can have an impact on organization of exercises and learning material for students. Depending on the pedagogical model to apply, exercises and learning material can be split into tiny pieces, in order to adjust the flow of exercises more dynamically to the student's performance. Organization of content in such a way is common in constructivistic learning environments and their assorted learning management systems. The requirements of such systems and their relation to the Virtual Unix Lab are discussed in chapter 1.

Possible approaches for sequencing exercises and learning material are outlined in [Darbhamulla and Lawhead, 2004] and [Helic et al., 2004], a possible architecture that employs a cycle of “direction → capture → analysis → feedback” that would be compatible with the Virtual Unix Lab can be found in [Heer et al., 2004]. The mapping of specific checks to didactic topics can be done via topic-specific parameters and weights, see the discussion on “weighted polynomials” in [Brusilovsky and Cooper, 2002, pp. 28].

No focus is set on the topic of splitting and sequencing of course material for the Virtual Unix Lab at this point. Many theoretical foundations for that area are available, e.g. in [Kobsa et al., 2001], [Helic et al., 2004], [Fischer, 2001] and in the whole corpus of hypertext and hypermedia literature. If need arises, finer grained course sequencing can be performed at a later step to improve and fine-tune tutoring.

When discovering that more training is needed for a prerequisite, interrupting an ongoing exercise to learn those prerequisites in a separate exercise is challenging not only to the learning environment and the learning material, but also to the cognitive load on the student, involving a switch of focus and context back and forth. An alternative is to not interrupt the ongoing exercise, give the student the option to either learn the skill on the current exercise, or abort it, then to offer the second training and after that allow the student to take the first exercise again. Past experience shows that giving students the opportunity to repeat exercises without penalties is accepted by students, as described in section 7.2.2.

Figure 10.5 illustrates these two approaches to arrange a “main” exercise that the student practices (1) and a second exercise that teaches basics for the main exercise (2). Image 10.5 a) shows interruption of the main exercise (1) for the prerequisite exer-

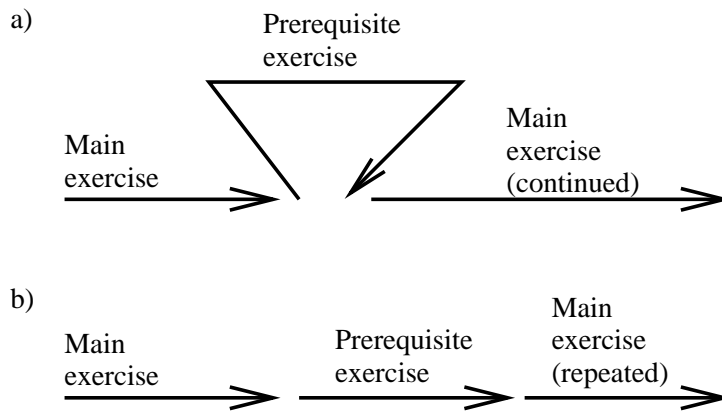


Figure 10.5: Learning prerequisites by a) interrupting and b) repeating

cise (2) and the following continuation of the main exercise (1). Image 10.5 b) shows that the main exercise (1) is fully completed first, and after non-success has been established, the necessary basics are practiced (2) before repeating the main exercise (1).

Advantages of the second approach are:

- Only a recommendation is given, the final control is left to the student, both *if* he takes the prerequisite exercise, and *when*.
- No distraction is made for the student while he is focused on the main exercise.
- Implementation is a lot easier, both for the course system itself, and for organization and granularity of course material.
- Giving recommendations on what exercise the student should take next can be considered as a “light” version of a learning management system.

The feedback given to the student can be used to judge if more basics need to be learned, in addition to get an evaluation of his performance in the exercise. Individual feedback can be given here by utilizing user adaptation, see section 11.5.

## 10.5 Considerations for the user model

The approach to giving feedback and assistance to the student in the previous section also requires considerations for the user model. Data collected during the exercise includes:

- What specific exercise is currently on, identified by the booking ID. This helps to find the type of exercise, the associated checks, and what user is working on the exercise, for updating his user model.
- The number of the verification run within the given exercise. Currently there is only one such run through all the check scripts at the end of the exercise, so no bookkeeping is needed. With several runs, this has to change.
- Each exercise's check has a unique check ID. For each combination of booking ID, check run, and check ID, the result of the test has to be recorded for evaluation, updating the student model and giving feedback.
- If assistance is given by printing help text, the time and details on the text are recorded in the student model. If further help is needed at the same point, this information will help to get more specific and not repeat help previously given.
- The reason why help was provided. If help was provided by the system in an automatic way, or if the user made a request for help. This can help to identify when users abuse the help system.

Based on Nathan's statement that "ITSs do too much thinking"<sup>1</sup>, the system is implemented as unintelligent tutor that provides late or very late feedback, with possible user adaption. In the given context, "late" feedback would be after a part of an exercise was worked on as discussed in the previous section, "very late" feedback would be after the whole exercise, as is currently done in the Virtual Unix Lab. In contrast, immediate feedback on the exercise part that the student is still working on is considered inappropriate as it may lead to confusion of the student, if his line of thinking (and configuration works) is interrupted by panic messages that the system is in an inconsistent state (which is a natural thing during system administration configuration work). Updates to the user model happen accordingly by the data outlined above.

While Nathan suggests to not give any feedback at all, the Virtual Unix Lab is intended to support the student by giving feedback, so that the burden of assessment of his work is not placed on him alone<sup>2</sup>. This approach can later be extended for individual feedback and user adaption, see chapter 11.

## 10.6 Impact on the user interface

In intelligent tutoring systems, the user interface is used for communication between the student and the tutoring system. To extend the Virtual Unix Lab with capabilities to provide feedback and assistance, extensions to the current user interface are needed.

---

<sup>1</sup> [Nathan, 1990] pp. 407

<sup>2</sup> [Nathan, 1990] p. 413

This needs considering of the relevant communication channels, the current user interface, and how to blend information into the existing web-based user interface.

### 10.6.1 Communication channels

The user interface provides communication in two ways, taking users' input, and presenting learning material and feedback to the student. User input comes as interaction with the website, keystrokes from on-line diagnosis (keylogging) and the checks performed by the system to verify the current state of the exercise. Interaction of the users happens via a web interface for presentation of the exercise text and some feedback like remaining time, the lab machines are accessed via separate applications for FTP/SSH/telnet.

The major challenge to the user interface when adding tutoring to the Virtual Unix Lab is to integrate the feedback provided by the system to the user in a seamless, non-intrusive way.

### 10.6.2 Analysis of the current user interface

An analysis of the current interaction in the Virtual Unix Lab can help identify the modules that need to be extended for providing further communication with the student, for both acquiring data and providing feedback. An overview of the current user interactions can be seen in figures 10.6 and 10.7. The general menu structure with its five main menu items is displayed in figure 10.6. The exercise itself happens between the "Start exercise" and "End exercise" steps. A zoomed version of that flow of the exercise itself, with the countdown to end the exercise, possible help for accessing the lab machines, and the main interaction can be found in figure 10.7. This main interaction is happening via a command line interface (CLI) in a separate terminal application besides the web application, where the user logs into the two lab machines, types commands and interacts with the systems to fulfill the requirements of the exercise.

The two modules that have been identified for possibly providing help to the system are marked with "Help #1" and "Help #2" in figure 10.7, where the former would be placed within the web interface, and the latter within the systems' command line interface that the student is accessing with the terminal application. As such, help can be web-based and/or shell-based. Web-based feedback would happen within the existing user interface, shell-based feedback would require deeper modifications of the lab machines. An impression of the work needed can be obtained by observations made in the Berkeley Unix Consultant project described in [Wilensky et al., 1988] and in the Unix assistant introduced in [Manaris and Pritchard, 1993].

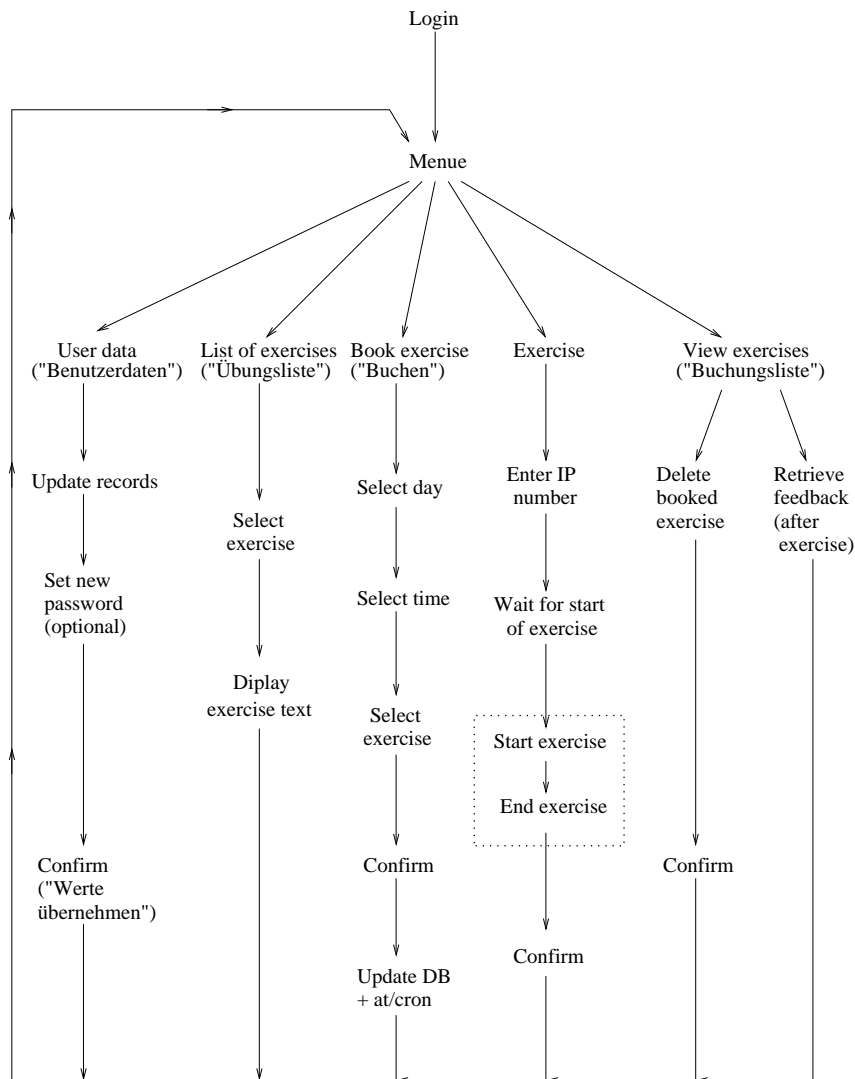


Figure 10.6: The current user interface: Menue structure

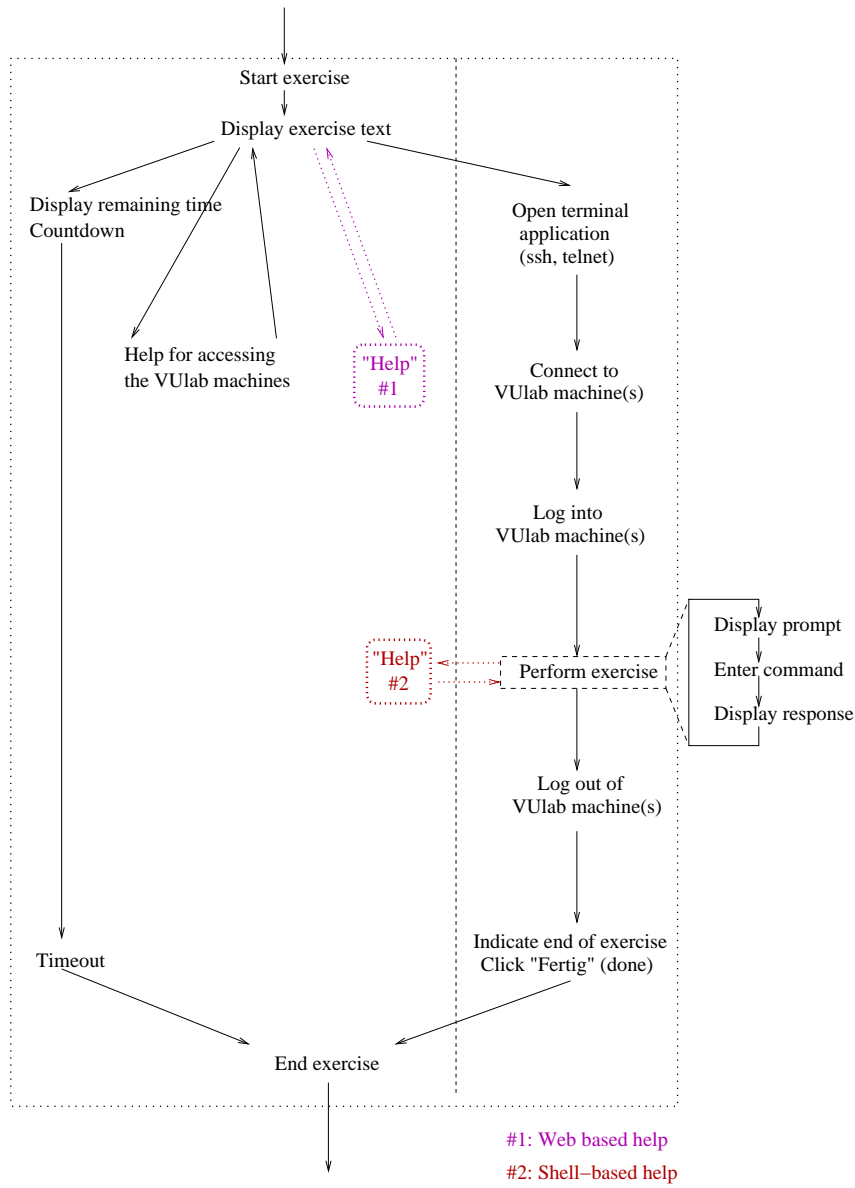


Figure 10.7: The current user interface: During the exercise



### 10.6.3 Blending information into the web-based user-interface

When providing feedback in the existing web interface, this can be done either synchronous or asynchronous. Synchronous feedback would be in connection to some user event in the web interface. As there is currently only one button in the web interface to indicate the exercise has finished before time is up, this is no help for feedback during the exercise. A new “help”-button could be introduced to realize the pedagogical model of a student raising his hand to call the teacher in a lab exercise, see “cooperative feedback” section 10.4.4.2.

For asynchronous help, the course engine would have to display feedback without the user asking for it explicitly. The course engine would consult the user model for its decisions, and then communicate with the web interface to display the information. As there may be several items to display to the user, some selection would have to be made on what feedback to really give to the user. Information to consider would be the current exercise situation, the student’s previous history and other data in the user model. Selection and presentation could be displayed as suggested in the RSS-based scheme in [Hylton et al., 2005].

For both synchronous and asynchronous delivery of feedback, actually displaying the feedback should not disturb the information displayed by the student at that point, i.e. exercise text and remaining exercise time should continue to be visible. A part of the screen could be reserved to display feedback, possibly with a chance to scroll through previous messages from the “teacher.” Updates of the text would have to happen in an asynchronous way based on either actions from the user and the system. Within the existing web framework, this could happen by running a JavaScript-based engine that constantly communicates with the course engine, and which displays information to the user when needed, without any action from him. A possible implementation of such a JavaScript-based engine called “Asynchronous JavaScript and XML” or in short “Ajax” was introduced in [Garrett, 2005].

## 10.7 Summary

The major components that need work for extending the Virtual Unix Lab to add tutoring during the exercise have been discussed in this chapter. While time constraints prevent realizing them, it is expected that the foundations laid here are sufficient for further work on tutoring in the Virtual Unix Lab.



# Chapter 11

## Architecture of user adaption

For advanced learning topics, adapting of the learning system to users is considered beneficial. This chapter describes an architecture for user adaption in the Virtual Unix Lab.

The adaptive component introduced in this chapter is built upon result verification with Domain Specific Languages and the “simple” tutorial component as described in the previous chapters, and extends them. The user model of the tutorial component is used as base for this extension, as it already contains data on progress of students’ performance during a particular exercise. The data available in the user model reflects the situative context described in section 8.2.1, which is determined by check scripts.

The data can be used for several applications: An overview of the progress of exercises and students’ learning in general can be gained, and the topics learned can be verified to be consistent. Better support can be given during the exercise with respect to giving help and feedback, while at the same time preventing students from abusing the help system. Further data can be collected to analyze usage of online help contents (Unix manual pages, online lecture notes) and optimize them. The latter approach is useful for moving towards a full learning management system, which is discussed briefly in chapter 1. The other points are discussed in the following sections.

### 11.1 Establishing and maintaining the user model

The user model contains data about the user. In the Virtual Unix Lab described so far, this means data about the students’ performance during and after an exercise. To adjust the system to the user, it has to draw certain inferences about the user upon which to act.

This section outlines how the user model’s view of the user is initialized, what data is

considered relevant, how the initial user model is updated to approximate the real user behavior, and what inferences to draw from the updated model.

### 11.1.1 Initialization

Initialization of the user model is based on data available on the user and his exercises with the intent to classify the user into one of several roles like “beginner”, “expert” or a number of intermediate states. Average values for all students as determined in the evaluation in section 7.2.4 could be used for an initial classification, to which the student is then compared. In the easiest scenario, below average would mean a beginner, above average an expert. Of course this can be extended to introduce more roles if need arises.

To compare a student against the average, data about the student has to be known. As this is not available for a new student taking his first exercise, several solutions are possible. It can be assumed that the student performs on average, and assume just the average values of all previous students’ values. This can be updated later when more and more data becomes available, see below.

Another possibility to initialize the user’s initial state is by a questionnaire, asking the student questions. This can be done in addition to the use of the existing exercise data. Some effort is needed to determine the list of questions, and evaluate answers. User acceptance has to be taken into account, too, when considering using questionnaires. It is suggested that this approach should be kept for a later stage, if the first iteration of going with average values turns out not to be sufficient.

Classification into roles can be done on a general scale for all tasks that the student can be asked to perform, or more fine grained, allowing a student to be a beginner in one topic, but an expert in others. This is recommended, as it allows to determine what topics the student has already learned, and what he still has to learn. Topics can be differentiated either “only” by the check script used to evaluate performance in the corresponding area, or it can also take the specific parameters given to the script into account.

An example for the former would be to determine a student’s “editing skills” via the `check-file-contents` check scripts after something needs to be changed in an exercise. Another example would be to see if he has the required “install software skills” by looking if the `netbsd-check-installed-pkg` and `solaris-check-installed-pkg` check scripts find a requested package installed.

An example that also takes the parameters given to a check script into account would be for scripts that cover broader areas, like verifying the output of a program via `check-program-output`.

### 11.1.2 Clustering

Clustering can be applied if the number of check scripts, possibly in combination with parameters, turns out to be too much data. This way, checks that verify “similar” areas can be grouped together, like the “software install skill” example above that takes both package management on Solaris and NetBSD into account. More information on clustering is available in section 8.2.3. During the first implementation of user adaption, clustering will not be considered, saving it as possible future optimization.

### 11.1.3 Observed data

The data received, stored, and analyzed for tutoring and adaption can be split into several groups, some of which are again optional for the first implementation.

Data on the progress of exercises, both during the exercises and after them is already discussed in the previous chapter, see section 10.3. The data about skills and concepts learned can be determined by analyzing the check script results as discussed above and in sections 10.2 and 10.3. On-line diagnostics can help to determine what commands the user is typing and what information he is looking up in manual and web pages. This information tells if the user is investigating the right solution, possibly even before he’s issuing the corresponding commands. A problem with documentation is that the user may get the required information “out of band” in a way which can not be measured, e.g. by looking up manual pages on a different system, browsing an offline copy of the lecture notes or consulting a printout.

### 11.1.4 Updating the user model

The initial data that is stored in the user model is updated during the exercises by the new findings made. More information can be inferred by comparing the initial state and later updates<sup>1</sup>. The data updated during the exercise is the same that is covered during initialization of the user model. Specific knowledge-areas and skills are updated by analyzing the results from check-scripts and possibly their parameters as described above, and multiple areas and skills can be combined by applying clustering.

When updating the user model, keeping old values for reference and analysis can be useful. For one, this can help to determine if the initial values chosen for the user model were chosen appropriately. If many updates happen at the start of an exercise, the initial values were probably chosen suboptimally. Also, long term observations can be made about the change in students’ behavior, e.g. if someone was classified as “beginner” initially, and gradually changes into an “expert.” Such observations can be made along one or several exercises.

---

<sup>1</sup> [Chin, 1986] pp. 26

### 11.1.5 Accommodating plan recognition

The data from the user model is mainly used to give feedback in the tutorial component, see chapter 10. Adapting the data to the user allows to give personalized feedback that is tailored towards each user individually. There are a number of other possible applications, though.

Inferences can be drawn about the plans, intents and knowledge of the user, as outlined in chapter 10 and by [Fink et al., 1998]. In contrast to universal help systems like COMFOHELP<sup>1</sup> or the Berkeley Unix Consultant, the task for the student to complete in the Virtual Unix Lab is specified, and as a result the student's plan is assumed to be known<sup>2</sup>. So far, "plan recognition" has not been covered within the Virtual Unix Lab due to this predetermination.

When a task allows to be solved in more than one way, the result verification performed in the current version of the Virtual Unix Lab can be supplemented by a more detailed analysis of what the student is actually doing. For this, on-line diagnosis as described in section 10.4.3 will be required. If it is available, the exact task that the student is currently working on can be determined, and then his exact steps can be observed. The steps can be evaluated if they lead towards the given goal, or away from it. A separation between "good" (appropriate, right, leading towards the goal) and "bad" (wrong, counter productive, leading away from the given goal) can be made, and needs to be reflected in the exercise definition. The exercise specification described in section 6.6 will need to be extended for this, also possibly indicating *how* good or bad a particular step is.

A possible application would be to allow the system to interrupt in critical situations before or right when the student performs an action that would render the system unusable. Examples would be if he removes a file that is critical to the system's operation, or if he has setup a NIS or NFS client without a server, and reboots – the client system would hang infinitely upon reboot, waiting for the server to come up.

Another extension would be to apply "fuzzy" matching to the on-line diagnosis with the data stored in the exercise, and allow deviations of a certain degree from them to be more fault tolerant. This can be applied to recognize typing errors in commands that would be "on track" otherwise, or when output is not 100% as expected, but is slightly different. The level to which fuzziness is acceptable or not would have to be defined for each individual case, and would also require reflection in the exercise definition.

Those components could greatly help assisting students in the Virtual Unix Lab. Before realizing them, on-line diagnosis is required as basic building block, though.

---

<sup>1</sup> [Krause et al., 1993]

<sup>2</sup> [Chin, 1986] p. 24

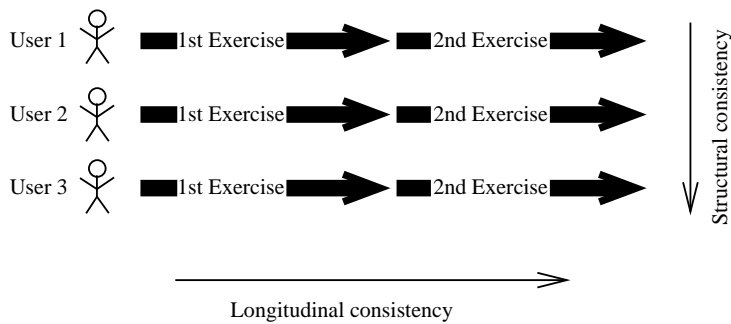


Figure 11.1: Structural and longitudinal consistency in the Virtual Unix Lab

## 11.2 Adaptive axes

Adaptive axes describe the scale on which a specific attribute of an adaptive system can be set. They describe the characteristics for achieving the given state, and the results stemming from a particular state to which the system has been adapted as described in section 8.2.4. This section introduces the adaptive axes identified for the Virtual Unix Lab.

Section 8.1.4.3.2 identified a number of approaches for epistemic diagnosis. In addition to the direct assignment of credit and blame described in section 10.2, structural and longitudinal consistency are considered relevant. Consistencies are established with and compared against data from the user model. As a result, there is an interaction between the tutorial component and the adaptive component of the Virtual Unix Lab. In order to keep the architectural design easy, they were split, and the adaptive component is defined here, instead of overloading the tutorial component:

- **Tutorial component:** collects data, stores it in user model, and gives feedback based on the data in the user model
- **Adaptive component:** evaluates data in the user model and updates the decision base for the tutorial component

When observing consistency in the Virtual Unix Lab, both structural and longitudinal consistency are taken into account. Sections 8.1.4.3.2.2 and 8.1.4.3.2.3 introduced the theoretical foundations, and figure 11.1 illustrates the difference within the Virtual Unix Lab:

- **Structural consistency** compares results of various users against each other, to compare a single student against a group.

- **Longitudinal consistency** compares data of a single user against other incarnations of the data, e.g. about profiles built from an earlier phase of the same exercise, or from other exercises done by the student.

The adaptive axes of structural and longitudinal consistency are discussed further in sections 11.3 and 11.4. The third axis is personalized feedback. As indicated in the previous chapter, feedback is important for tutoring, and with the data collected in the user model, feedback can (and should!) be personalized for optimal feedback and learning effect in the student. Personalized feedback is discussed in section 11.5.

## 11.3 Structural consistency

The focus of structural consistency is to compare one student's performance to that of other students. The student's performance is the base for updating the user model, which in turn is used to alter the system's way of handling feedback. This section discusses exercise velocity, mastered skills, help requests as source of data, outlines how to reflect this on the user model, and raises the issue on proper metrics for evaluations.

### 11.3.1 Observing exercise velocity

The first observation made is the speed at which a student makes progress in a given exercise. The results are compared against similar values determined from other students' exercises, with special attention to the zone of proximal development (ZPD) described in section 10.4.1.

When comparing a student's performance at a given point in time against that of other students at the same time in the same exercise, it is easy to determine if the student is faster or slower than the other students, with a certain confidence. For example, if a student has absolved more checks than the average of other students at a certain point in time, inferences can be drawn about his performance. Of course repeated evaluation will help to determine if the inferences were drawn correctly.

Taking the average of all other students from a group as the standard against which the student is compared has drawbacks: A group that is in general weak or unmotivated can give unfair advantage to a student with average strength and motivation due to their low performance. Also, if a new term or exercise starts, no data is available for comparison. For such cases, the exercises can be extended to include data on what performance is expected from students during an exercise: What milestone should be passed after what time, which concepts and skills should be mastered safely, etc. When noting students' expectations in the exercise, more time would be reserved for "difficult" tasks than for "trivial" ones.



When extending exercise texts with information about what expectations to put into students' performance, this should be done in a second step, after measuring and evaluating students' performance first.

Proper metrics are needed to determine what is and can be expected from students, see section 11.3.5.

### 11.3.2 Observing mastered skills

Going plainly by number of exercises correctly solved over time is one approach to establish how a student compares to others. A more fine-grained approach is to look at his performance for each skill, as identified by a certain check script and possibly any related parameters, and then compare that among students.

For example the "edit file" skill can be allowed to fail for 1-2 times, but should be expected to be mastered after that number of attempts. If a student fails more often, then there is probably a general problem that needs attention. Also, if the failure pattern is not at random but turns from repeated failures into repeated success, then a point at which the student did learn the skill in question can be established.

Grouping of "easy" skills may be needed to determine if a "difficult" skill was learned, e.g. "user management" may require mastering of skills like creating directories, changing owners on filesystem objects and modifying system databases. On the other hand, mastering a skill may well require more knowledge than just that of managing all prerequisites. A detailed analysis of empirical data would be needed to be able to tell details.

In general, it can be expected that "simple" skills are learned faster/sooner than "difficult" skills. After an initial round of tests to determine how student's perform on average, this can again be reflected in the exercise setup, by noting expected values. Examples could be in the form of "the 'modify system database' skill must never fail" and "programs printing proper output may fail a few times at the easy/starting or voluntary/difficult parts."

### 11.3.3 Observing help requests

Another source of data for judging a student's performance in relation to a group of students are the offers and/or requests for help he makes in one way or other. In that context, those requests can be:

- Active help requests by the student, e.g. by pressing a "Help!" button.

- Offers for help by the system, based on analysis of the current exercise as discussed in section 10.4.
- Possibly requests for online manual pages and webpages with information on how to solve the exercise. Recognizing this can be challenging as the requests can happen outside of the Virtual Unix Lab as described in section 11.1.3.

Analyzing help requests can tell if a student asks for help more often than is expected. On one hand side, there may be valid reasons for the student to request more information, e.g. plain curiosity on what the system has to offer, or maybe he did not fully understand the description of the exercise, and hopes to clarify this by asking for help. On the other side, analyzing help requests can prevent abuse of the help system by students who are “gaming the system.” The number of help requests that are considered “normal” or “too many” in this context can be determined by looking at data from all students again. Each request for help updates the student’s user model with data on what part of the exercise that help was requested, and what help was given in case there’s more than one hint available – see the separation into “behavioral” and “epistemic” help in section 10.4.4.1.

Observing help requests of all students also helps to identify tasks where not only a single student has problems, but where the majority of the student group requests more help than was expected. This can indicate general problems of understanding that should be addressed in an appropriate way, e.g. with intensified training in the classroom or by providing extra exercises for the topic that students find difficult.

Data about the expected volume of help requests can be noted in the exercise text again. This can e.g. happen in the form of noting tasks where increased help requests should be expected, or it can also be in combination with the level of skills learned - e.g. that increased help requests are acceptable as long as a specific skill is not noted as learned.

#### 11.3.4 Adjusting the user model

The observations and collection of data discussed in the previous sections can be used to update the user model of each individual student:

- If the student is faster or slower than the majority of students, in general.
- If he has mastered various skills better or worse than the average.
- What help was given to the student, and at what help level to give the next hint for the same task, if repeated help is requested.

This in turn can lead to reactions such as personalized feedback as discussed in sections 11.1.4 and 11.5.

Furthermore, the data can also be used to determine average patterns of behavior of whole groups of students against which an individual student can be compared. Metrics for these comparisons are discussed in the following section.

### 11.3.5 A metric for evaluation

To classify what “faster / slower”, “learned” and “too many help-requests” means that metrics are needed, which can be used to judge what the average learning speed is, after how many (and which) repetitions a topic can be considered as “learned”, and asking how many questions and using how much aid is acceptable before considering it as being “too much.”

Another question is, against what data set the current student is compared, exactly. Possible options are the average of all students in the same group, median of the group, and a possible statistical distribution with a certain confidence. When using descriptive methods instead of indicative methods, a graphical tool like the box/whisker-plots used during the evaluation in section 7.2.1 will be useful.

For a first pass, using an arithmetic average for non-boolean values like general results and for help-requests, with some percental margin, e.g. a confidence of 95%. For boolean values like skills mastered, the median can be used for orientation.

More precise statements can only be made after evaluating data from a first round of exercises. For that, a first round of tests and data gathering is required. Of course this is preceded by implementing a system that acts accordingly, and delivers the required data.

After evaluating data from life exercises, hints can be put into the exercises to start the next round with better stereotypes. The format of the Virtual Unix Lab’s Verification Unit Domain Specific Language (VUDSL) as described in section 6.6 would need corresponding extensions. Possible facts stated can include how values are measured (averages, median) and what possible deviation (absolute or in percent) would be acceptable. A few examples in human-readable notation could be “This part of the exercise should be completed after 30 minutes  $\pm 5$  minutes”, “Skill *S* should be mastered from here  $\pm 2$  tasks” and “The student should not be slower than 10% of average of the group.” Of course the exercise would require those specifications in a machine readable representation available via the VUDSL. Section 11.6 outlines some of these extensions.

## 11.4 Longitudinal consistency

In contrast to the structural consistency observed in the previous section, the focus here is to analyze a single exercise, and try to understand the pace at which a student is making progress in that exercise. The analysis is based on measuring the time between solving various parts of an exercise and acquiring certain skills as outlined in the previous section, with the eventual goal of identifying problems in the progress of the exercise.

The following sections cover the assumptions made on progress of the exercise during the analysis, performing calculations for establishing longitudinal consistency, and how to perform descriptive and indicative analysis.

### 11.4.1 Assumptions and methodology

Longitudinal consistency looks at one specific exercise. Data acquisition happens during the periodical status verification by running check scripts for the tutorial component as described in section 10. This can be compared to the “flashlight” view shown in figure 9.1 b).

Results of each check-run, i.e. the single results of each check script, will be saved by using the ID of the booked exercise (“buchungs\_id”), and an increasing number or a timestamp to identify the check-run within the booked exercise.

A number of assumptions are made on how progress happens within an exercise:

- There’s linear progress of the exercise, as shown in figure 10.3
- No parts of the exercise are skipped, ideally
- Parts of the exercise are numbered strictly increasing:  $u_0, u_1, \dots (u_x)$
- Check scripts that implement verification of each part of an exercise do not “block” but succeed immediately. Blocked scripts can skew the timing observations made.
- The “state” of an exercise is defined by the last part of an exercise solved successfully, i.e. the one located rightmost in figure 10.3

Those assumptions result in a strictly increasing value that can be used for calculations. For the following analysis,  $t_x$  denotes the time at which the exercise is at state  $u_x$ .

The calculations described here aim at specific attributes and properties of a student’s learning process. Other techniques can be used to establish the student model, e.g. via Bayesian networks as described in [Mayo and Mitrovic, 2001].

### 11.4.2 Descriptive analysis

There are a number of exercise parts between the two states  $u_0$  and  $u_1$ , with their associated times  $t_0$  and  $t_1$ . The average velocity that each part of the exercise between those two states was taken with can be calculated as  $(u_1 - u_0)/(t_1 - t_0) = \Delta u/\Delta t$ .

#### 11.4.2.1 Interpolation vs. more data

By increasing the frequency of the state checks, i.e. by decreasing  $\Delta t$ , more precise information can be learned about every single part of the exercise as verified by each individual check script. An alternative to decreasing  $\Delta t$  is to define the ratio between the various parts by adding scalars that reflect the ratio. For example, in a 30 minute period with three exercise parts, this could indicate that the first part takes twenty minutes, and the second and third part take 5 minutes each.

#### 11.4.2.2 Detecting speed changes

By getting more data – either through interpolation or by more frequent data collection – the average time needed for each exercise part by the particular student can be determined. If there's a constant value, the student works with constant speed and is making progress. Whether he's slower or faster than other students can be found by looking at other students' speed values. If progress of the students is not constant, this indicates behavior that needs attention. Information about average durations for each exercise part and at what time a specific check can be expected to be positive due to the underlying skill being learned can also be determined by methods described in section 11.3.

#### 11.4.2.3 Observations for repeated exercises

If a student repeats an exercise, he may expose different speed behavior than in previous runs of the same exercise by him. While changes can be expected from the results in section 7.2.3, the point where a student's speed changes from fast to slow may change, and it is this point that calls for attention: The student may need more time because he is not fluent in the required skills or needs more information. This point the equivalent of Michaud et al's "Zone of Proximal Development" (ZPD), see section 10.4. An comparison between the part of the exercise at which the ZPD shows up between various repetitions of an exercise may reveal more information.

#### 11.4.2.4 Speed and acceleration of progress

By observing the derivation of the change in the exercise progress  $\Delta u$  by time  $\Delta t$  with  $\Delta t \rightarrow 0 = u'$ , the speed of progress of the exercise at the point at which the student is currently working can be determined. Movement of the point can be used to make a statement if the exercise is progressing fast or slow. By observing the change of speed over time  $-\Delta u'$  derived by  $\Delta t = u''$  – statements on the change of the progress' speed, i.e. its acceleration, can be made. E.g. it can be told if the student is speeding up or slowing down<sup>1</sup>.

#### 11.4.2.5 Data model and storage

The data model for storing the check results within an exercise should be modeled to identify each individual check's result. Currently, each check result is stored in addition with the booking ID, the check ID and the result of the check script in the `ergebnis_checks` table as described in appendix B. For multiple check runs during an exercise, a counter or timestamp needs to be added.

If data from a previous check run is available, information about the exercise's speed of progress can be calculated and stored in the database. Likewise, if the data from the previous run does include information about the speed of progress at that time, information about the change of exercise speed can be calculated and stored.

The question of what data to store exactly should be addressed last. For each check-run during the exercise, speed and acceleration at that point in time can be calculated as outlined above, and it would be sufficient to draw conclusions about the progress of the exercise.

Data about progress of the exercise – indicated by the movement of the part that the student is currently working on, and reflected by the speed and acceleration of the exercise – can be stored in a separate database table. The table would have the booking ID of the exercise and a counter or timestamp to identify the individual check run as keys, and would include data on speed and acceleration for the current snapshot.

#### 11.4.2.6 Drawing conclusions from speed and acceleration

After determining speed and acceleration of an exercise at a certain points, an evaluation may result in adjustments of the system to the student. While it remains to be determined what a “slow” or a “fast” student exactly is, the system can recognize such students. For “slow” students, it can provide additional help to solve the exercise by immediately giving hints that would be given to other students only on demand,

---

<sup>1</sup> [Serway and Jewett, 2004] pp. 1

or after some time. The system could even adjust itself to those students, and save them doing parts of the exercise. The question of fairness to other students should be considered when adjusting the exercise system, though.

“Fast” students probably do not need any special attention, likewise no reactions to speedups (acceleration) is needed during the exercise.

When negative acceleration, i.e. slowdown, is detected during an exercise, this would be of more interest. Depending on the exercise this may be expected at some points e.g. when new knowledge is required. For example in the NIS setup, operations like adding a user to the NIS system is expected to be a challenge to students, and thus more time should be planned as things may go slow at that point. At other points, a slowdown may indicate that the student has a problem at that point, and this could – possibly with some threshold to prevent distraction from the internal learning process – be interpreted as need for help.

An exercise could store those points and indicate what speed is “too slow” at what points in an exercise a slowdown is expected as normal, when it should not happen, and how to react if it still does. See section 11.6.2 below for further discussion on VUDSL extensions.

### 11.4.3 Indicative analysis

Instead of computing values to compare students’ performance in exercises, the same raw data that is used for those calculations can be used as base for indicative analysis. Using graphical methods like the box/whisker plots introduced in section 7.2.1, it is possible to compare a student to a group of students, and also look at different exercises or snapshots of the same exercise from a single student.

When using box/whisker plots for visualization, statements can be made with a certain confidence. Examples that compare results of later exercises with those of early exercises can be seen in figures 7.1, 7.4 a), and 7.4 b).

Other methods that do not offer statements on confidence may still be useful to indicate trends when comparing earlier and later exercises of the same user. Investigations on general performance and details on every individual part of an exercises can be made. Examples for the former can be seen in figures 7.5 a) and b), examples for the latter can be seen in figures 7.6 a) and b). In any case, statements can be made if there is a positive (increasing) or negative (decreasing) trend in performance and exercise results.

Data that can be observed and compared this way includes:

- The number of checks solved properly, exercise speed and acceleration for a

single student's exercise(s) as the basic data to observe.

- A student's overall progress, compared to a group. This can be used to tell if the student is working faster or slower than the reference group.
- A student's overall progress, compared with his earlier exercises. If no progress can be seen here, demand for increased help and information can be inferred.
- A student's progress on a particular exercise over time, observing several snapshots of results. More detailed statements on progress, speed of progress and also change in progress, i.e. speedup/slowdown, can be made.
- At what points in an exercise do slowdowns (or even speedups) occur? If those points show up on all students' exercises, information and teaching effort seems appropriate to cover the challenge in the related exercise parts.

While indicative methods that use visualization are good for human interpretation, they are less useful as decision base for computer programs. Still, they may be very valuable for verifying operation of the learning system and for status reports for both the student as well as the teacher.

## 11.5 Personalizing feedback

In tutoring systems, help can either be cooperative (on demand) or provided automatically, see section 10.4. In both cases, interaction with the user model should happen to determine what help should be given. Data from the user model that would be required is:

- What help was already offered at this place
- A general classification of the student: beginners could get behavioral / comprehensive help e.g. by giving exact commands to type, or hinting them at useful commands; experts could get epistemic / high level / conceptual help.

The ultimate goal is to personalize the feedback given to the student to match his speed and level of knowledge. At the same time, abuse of the help system through the student ("gaming the system") should be prevented.

### 11.5.1 Adjusting of help contents

If a student is found to be either very far behind the expected knowledge, or far in advance of it, some feedback can be retained from him to not overburden or bore him.



To realize this, items can be classified by where they usually appear on the learning curve, and the student can then be compared against that curve to see where he is related to that<sup>1</sup>. See also the discussion on the Genetic Graph in section 8.1.4. Possible implementation options are to not display information at all, or adjust the information presented in an “incremental linking” style. Another set of possibilities arises from the choice whether to use restrictive or non-restrictive adaptive methods, e.g. if link annotation or link hiding should be used<sup>2</sup>. Also, when on-line diagnosis recognizes wrong or even dangerous commands being issued, it could warn or plain refuse to issue these commands.

### 11.5.2 Handling non-standard exercise progress

Section 11.4 shows how to detect if a student exposes non-standard behavior in solving exercises, e.g. if he is too fast or too slow, how that can be determined either in general, or if there is a speedup or slowdown at a specific point in the exercise. One effect that will happen is that slow students may not complete the exercise and run into timeouts, but that is not a problem per se - the students can repeat the exercise, and no special action is needed. If a student makes progress at a certain pace and then slows down at a certain point, that is of more interest: What happened at that point? Does the student need help or assistance, or did he just leave for a smoke? On-line analysis can help to determine the exact circumstances in more detail.

### 11.5.3 Adjusting the system

If a real slowdown is detected, the question on how to react arises. Possible reactions include<sup>3</sup>:

- **No action:** Having the system offer immediate advice may be too fast and confuse the student who may just be thinking. The didactic model of this would be a teacher spotting a problem while looking over a student’s shoulder in a classroom exercise, but not speaking up when it is obvious that the student is working on a solution to the problem.
- **Give information on current situation:** The system may or may not fully understand the current problem at hands. Based on the level of the student as stored in the user model, help can be given at various levels, ranging from behavioristic hints to epistemic help.

---

<sup>1</sup> [Chin, 1986] p. 25

<sup>2</sup> [Specht and Kobsa, 1999] pp. 1

<sup>3</sup> [Schulmeister, 2007] pp. 181

- **Offer an alternative viewpoint:** If a student is recognized to approach a certain problem in a specific manner (as e.g. detected by using on-line diagnosis), he may have forgotten or not be aware of the fact that there are other approaches to solve the problem. Information can be given on how to proceed in the current line of thinking that the student is currently in, or other viewpoints can be outlined, possibly with details on how to approach them (depending on the student's experience and level).
- **Hint at other exercises:** If a student did previously slow down at similar exercise parts of the same category, he may need more practice and deeper understanding of the matter at hands. Besides giving more information, suggesting a specific exercise to the student to improve his skills in that area may help. Of course this assumes that an appropriate exercise is available. The data for this hint may be stored in the user model and suggested to the student after the exercise, to not distract him more than necessary from learning.
- **Perform the step requested automatically:** In theory, the system could perform the steps required to proceed at the point that the student is currently stuck at. This should be done with quite some consideration though: Users may eventually abuse the help system, and the question of fairness to other students that have successfully mastered the point at hands has to be taken into account.

Besides exposing these actions when a slowdown in exercise is detected, most of them can also be applied when the user asks for help actively.

#### 11.5.4 Preventing abuse of the help system

When the above points are realized, detecting when a user is trying to abuse the help system is possible as described in [Baker et al., 2004]: if a user works slowly but at a steady pace and he starts requesting help repeatedly at one point, care should be taken. A comparison with the user's history of help requests can determine if he needs some help in general, or if he just tries to trick the system into giving the right information without making efforts on his own.

### 11.6 Extending the VUDSL for user adaption

The previous sections have discussed user adaption in the Virtual Unix Lab in the context of structural and longitudinal consistency as well as personalizing feedback. Each of these areas reflects on the definition of the exercise, and this section discusses possible extensions of the VUDSL for them.

To extend the existing VUDSL as described in section 6.6, considerations need to be made about how to extend it. An easily realizable way that is still readable would be to add an extra keyword to each line that indicates what exactly the line is for, followed by data specific to that purpose. E.g. currently data in the `auswertung_teiluebung()` PHP calls look like this:

```
<?php auswertung_teiluebungen(
    ??? // vulabl: check-program-output PROGRAM=ypwhich OUTPUT_SHOULD='vulabl'
        // Does ypwhich(1) return 'vulabl'?
); ?>
```

In the PHP comments (“//”), the first line describes what check to run on a specific system, and the second one determines the feedback given to the student after the exercise. Adding keywords “On” and “Feedback”, this could e.g. result in the following:

```
<?php auswertung_teiluebungen(
    ??? // On vulabl: check-program-output PROGRAM=ypwhich OUTPUT_SHOULD='vulabl'
        // Feedback: Does ypwhich(1) return 'vulabl'?
); ?>
```

The general approach in the next sections is to first define what needs to be added to the exercise, before showing how it can be added.

### 11.6.1 VUDSL extensions for structural consistency

The Verification Unit Domain Specific Language (VUDSL) describes exercises in the Virtual Unix Lab as introduced in section 6.6. This section covers extensions of the VUDSL to accommodate data for establishing changes for structural consistency as discussed in section 11.3.

1. **Speed of progress:** The timeframe in which an exercise should be solved is given by the overall time available for the exercise. Within the exercises, regions and milestones can be identified which should be completed at specific times, though, see 11.3.1. E.g. if an exercise consists of two parts, the first part may require 30% of the time, and the second part may take the remaining 70%.

This can be noted in the exercise e.g. by noting at what time a milestone should be completed. Here is an example telling that the point in question is expected to be solved after 30 minutes:

```
Exercise: Minor task

<?php auswertung_teiluebungen(
    ??? // On vulabl: check-minor-task
```

```

        // Feedback: Was the minor task performed successfully
        // Expected after: 30min
    ); ?>

    Exercise: Major task
    ...

```

Giving time like this is probably easiest for a start. An alternative would be to give time relative to the exercise's overall duration, e.g. "Expected after: 25%" to note that after one quarter of the time the exercise is due.

2. **Data about expected mastering of skills:** If a specific skill is required at a certain point, the system could observe the user's past history, and act according to it. For example, it could give more help from the start, or adapt to the situation. In order to realize this, the skill required is implicitly encoded by the check script that verifies the user's results. What's lacking is a metric to determine if previous failures indicate critical misunderstanding or not. To solve this, checks could contain data on their importance, so that only changes tagged as "Important skill" are taken into account.

Here is an example:

```

Exercise 1: Perform important task!

<?php auswertung_teiluebungen(
    ??? // On vulabl: check-task
        // Feedback: Was the important task done?
        // Important skill: yes
    ); ?>

Exercise 2: Perform unimportant task!

<?php auswertung_teiluebungen(
    ??? // On vulabl: check-task
        // Feedback: Was the unimportant task done?
        // Important skill: no
    ); ?>

Exercise 3: Perform another important task!

<?php auswertung_teiluebungen(
    ??? // On vulabl: check-task
        // Feedback: Was the other important task done?
        // Important skill: yes
    ); ?>

```

After exercises 1 and 2, their results can be observed to see what to do when the student reaches exercise 3: If the student failed the "important" first task, this will have a different impact than when he failed the "unimportant" second task. For this observation, it is important that only skills with the same check script (and possibly parameters, though not used here), are observed.

3. **Data on overall number of acceptable help requests:** Section 11.3.3 describes that the exercise may know about the number of help requests that is acceptable as "normal" either for the whole exercise, or parts of it. This would serve as lower bound above which actions will be taken by the system as noted by the

“Globally acceptable help requests” and “Acceptable help requests” tags in the following example:

```
// Globally acceptable help requests: 1

<li> Do something!

<?php auswertung_teiluebungen(
    ??? // On vulabl: check-something
    // Feedback: Was something done properly?
    // Acceptable help requests: 2
    // Hint 1: Further explain task to do
    // Hint 2: Given even more details, and first hints on how to solve
    // Hint 3: Hint at how to solve the problem

); ?>
```

If help requests are accepted as method to further explain the task that the student has to do, then this should be reflected by appropriate hints that just explain things in more detail, without giving away help on how to solve the problem at hands immediately.

4. **Data on acceptable help requests considering (un)learned skills:** Section 11.3.3 also describes a schema where the number of acceptable help-requests could be given more fine-grained, based on the finding that the student possibly has not mastered the required skill for that exercise part (as identified by the check script). Numbers for “beginners” that did not master the skill and “experts” could be given:

```
<li> Do something!

<?php auswertung_teiluebungen(
    ??? // On vulabl: check-something
    // Feedback: Was something done properly?
    // Acceptable help requests for beginners: 2
    // Acceptable help requests for experts: 1
    // Hint 1: Further explain task to do
    // Hint 2: Given even more details, and first hints on how to solve
    // Hint 3: Hint at how to solve the problem

); ?>
```

5. **What metric to use for measuring and comparing:** In the comparisons described so far, absolute numbers were used either to describe a user’s expected behavior, or of any deviations. Section 11.3.5 outlines the alternatives, esp. when giving the bounds in which deviations from given standard values are describe.

Instead of a fixed limit, tolerances can be given, e.g. for a given time an absolute value – given in minutes or as count – can be used to indicate that e.g. for novice users, more time would be acceptable than for advanced users:

```
// Expected after: 30min +/- 5min
...
// Acceptable help requests: 2 +/- 1
); ?>
```

Instead of giving the acceptable deviation as absolute number, a relative number can be given as well:

```
// Expected after: 30min +/- 10%
...
// Acceptable help requests: 2 +/- 50%
); ?>
```

When establishing the average speed of progress, fixed values can be used as outlined for the speed of progress above. Instead of taking fixed values, values from the exercises of other users could be used instead. The values could indicate what time the majority of users took to solve a specific part of an exercise, or how much time was needed to master a given skill to a certain degree. These numbers can be determined from the existing exercise results in the Virtual Unix Lab.

A question is how to exactly calculate them, though. Possible ways would include average, median and modus values for the given data. E.g. for time, an average may make more sense while for a yes/no item like a skill learned, the modus may make more sense.

This information can be put into the exercise text as well:

```
// Expected after: median
...
// Important skill: yes (modus)
); ?>
```

Besides the exact method on *how* to perform the calculation, the question of *what* to compare against is important as well. Possible items could be the time after which an item was solved, or a skill that was mastered.

As in the previous examples, adding knobs to tune – methods to test, and scales to apply – is only the first part of tuning the exercise. Future research with practical examinations will reveal what values to apply for a given exercise.

## 11.6.2 VUDSL extensions for longitudinal consistency

This section describes extensions of the Verification Unit Domain Specific Language to accommodate data for establishing changes for longitudinal consistency as discussed in section 11.4.

### 1. Adjustments for descriptive analysis:

- (a) **What is “too slow”?** When observing progress of a single exercise, the speed of progress can be determined as outlined in section 11.3.1. Possible ways to determine valid values here include absolute values from empirical

methods, and compare them against other students' exercises as outlined in the previous section. The extensions to the VUDSL would be the same, so no extra changes are needed.

- (b) **How to handle slowdown** can be defined by marking exercise parts where slowdown is allowed and acceptable without immediate action, while lack of progress in other parts may need different behavior from the exercise system. Hints on where a slowdown is acceptable, where it should or must not happen can be encoded into the exercise as shown in the following examples:

```
// Slowdown: ok
// Slowdown: acceptable
// Slowdown: should-not
// Slowdown: must-not
); ?>
```

These hints will help the Virtual Unix Lab to react to a possible user behavior. The question on what to do exactly in that case is discussed in section 11.6.3.

## 2. Adjustments for indicative analysis:

In contrast to the methods described for adjustments of descriptive analysis, results from indicative analysis are not evaluated by the exercise system. Instead, the statistics and graphs produced are intended to be used by students and teachers, and should thus be comparable and not changed individually.

### 11.6.3 VUDSL extensions for personalized feedback

Personalizing feedback requires adjustment of the presentation of help contents, and a decision base on what help should be given. For a first implementation, keeping those purely in the Virtual Unix Lab's Course Engine (see chapter 9) should be sufficient. If customizations turn out to be needed for an exercise or part of an exercise, the VUDSL can be extended based on these findings at a later step.

### 11.6.4 Other VUDSL extensions

Besides the extensions discussed in the previous sections, other extensions may be useful for a number of aspects where the VUDSL is used. Here is a list of areas for possible future extensions:

1. **Giving multiple hints:** If the system knows more than one hint at a given exercise, an order needs to be defined for that. This can e.g. be done by numbering the exercises, and then giving them one after the other. The following example tags the hints with "Hint" and a number to distinct them.

Exercise: Determine the currently used NIS server.

```
<?php auswertung_teiluebungen(
??? // On vulabl: check-program-output PROGRAM=yppwhich OUTPUT_SHOULD='vulabl'
    // Feedback: Does yppwhich(1) return 'vulabl'?
    // Hint 1:   What command prints the current NIS server?
    // Hint 2:   Try running yppwhich(1)
    // Hint 3:   Does yppwhich(1) print 'vulabl'?
); ?>
```

The number may not be needed technically, but it can be used to offer some ranking or ordering of the hints, so the system knows what hint to give first.

2. **Marking hints that can be given to a user during the exercise as behavioristic or epistemic.** Existing hints can be annotated that way. Let's observe the ones in this exercise part:

Exercise: Determine the currently used NIS server.

```
<?php auswertung_teiluebungen(
??? // On vulabl: check-program-output PROGRAM=yppwhich OUTPUT_SHOULD='vulabl'
    // Feedback: Does yppwhich(1) return 'vulabl'?
    // Hint 1:   What command prints the current NIS server?
    // Hint 2:   Try running yppwhich(1)
    // Hint 3:   Does yppwhich(1) print 'vulabl'?
); ?>
```

The hints could be given in order, assuming that given epistemic, high level hints will lead to the proper associations in the student, which of course assumes that he has learned them already. If that's not the case, the second hint would give the command to run. If that's still not enough, another hint could also give the expected results.

Different approaches to tutoring can be given, e.g. either first give epistemic hints and then behavioristic ones if the first hints do not lead to proper results. Or the exercise text could be supplemented with the behavioristic hints immediately if needed, without any further action by the user.

In either case, the system would need to know what of what kind a specific hint is. This could be told explicitly, e.g. by tagging the hints as "Behavioral" and "Epistemic":

Exercise: Determine the currently used NIS server.

```
<?php auswertung_teiluebungen(
??? // On vulabl: check-program-output PROGRAM=yppwhich OUTPUT_SHOULD='vulabl'
    // Feedback: Does yppwhich(1) return 'vulabl'?
    // Epistemic Hint 1:   What command prints the current NIS server?
    // Behavioral Hint 1:   Try running yppwhich(1)
    // Behavioral Hint 2:   Does yppwhich(1) print 'vulabl'?
); ?>
```

3. **Handling transient states:** The Virtual Unix Lab only observes the current state of the exercise systems. If a transient event – e.g. a button being pushed and released – needs to be recorded, this is challenging, as the button may no longer be pressed when the system's state is observed. If the button does not have a permanent effect that can be determined later, or if the effect is possibly



reversed or changed at some point, drawing clear inferences is challenging. To solve this problem, exercises should be setup in a way to not rely on transient events.

If this is not an option, increasing the intervals at which the system examines the exercise systems is an option to increase the likelihood to catch the transient event. Of course this still depends on the event itself – if it is very short-lived it may be challenging to do busy-polling on the systems. Giving hints here at which time intervals such behavior would be required to catch such events can be noted in the exercise.

Here is an outline how to realize an exercise of pressing a button for (say) 10 seconds. Besides pressing the button, the exercise would require preparation and finalizing steps:

```
Part 1: Make sure you know where The Button is.

Part 2: Press The Button and hold it down for 10 seconds, then
        release it.

Part 3: Continue with the exercise.
```

After part 3, observing the system will not show if the button was pressed or not. Taking the time between part 1 and 3 as “critical section” can be used to notify the system that increased awareness is required, i.e. that the scanning interval in which check scripts are ran should be set to something as low as 5 seconds. Later on, a check would be needed to see if the test for the button was ever true, and then act appropriately.

Here is an example that increments and decrements scanning, and then tests and reacts whether the button was pushed after some time. The three changes to the VUDSL are adding a “Scan” tag to increase check scan intervals, reset them to the default for the exercise, and add a new PHP function `auswertung_teiluebung_ever()` that does not print if the status of the named check was true at the end of the exercise, but if it *ever* was true, and give feedback accordingly:

```
Part 1: Make sure you know where The Button is.

<?php auswertung_teiluebungen(
    ??? // On vulabl: unix-check-process-running PROCESS=buttonprog
        // Feedback: Is the button-program running?
        // Hint:      Start the ``button``-program
        // Scan:      5s
    ); ?>

Part 2: Press The Button and hold it down for 10 seconds, then
        release it.

<?php auswertung_teiluebungen_ever(
    ??? // On vulabl: check-button STATUS=presed
        // Feedback: Was the button pressed for 10 seconds?
        // Hint:      Press the button for 10 seconds.
    ); ?>

Part 3: Continue with the exercise.

<?php auswertung_teiluebungen(
```

```
??? // On vulabl: check-other
    // Feedback: The exercise was continued successfully!
    // Hint: Relax!
    // Scan: default
); ?>
```

## 11.7 Summary

This chapter discussed user adaption in the Virtual Unix Lab, with special attention to establishing and maintenance of the user model that stores information about students, an overview of the adaptive axes used and special attention to structural and longitudinal consistency. Applying these adaptations was discussed for personalizing feedback to the student, and a number of hints were given on how to extend the existing VUDSL to give the exercise system more hints on how to handle user behavior.

Many of the items discussed can be implemented with the VUDSL that is available in the Virtual Unix Lab. Based on this implementation, fine tuning of the precise values to use in exercises, and what exact metrics to use cannot be told at this time. Furthermore, on-line diagnosis would be of benefit for fine-grained analysis. These areas are expected to provide material for future research.

# Chapter 12

## Conclusion

The focus throughout this work was on defining a learning system for system administration. Emphasis was put on the architecture for result verification and on feedback to the learner. After laying out the didactic foundations of system administration, the system was described, realized, and evaluated. Advanced topics for tutoring and adaption were discussed, building up on the basic Virtual Unix Lab system.

The result contains more work on the foundations of tutoring and didactics of system administration as was originally expected. Still, a system was realized that is usable in practice, and that can be used as foundation for future works.

Further areas of work have been identified, most notably teaching of system administration and realizing and tuning of tutoring and user adaption. Additional work should be put into translations of the system from German to English language, investigations of virtualization techniques and their integration in the deployment of the Virtual Unix Lab, and creation of a theory of bugs for system administration as a whole, or in parts.

As the system with its basic functionality has proven useful for the education of system administration, another possible step for the future would be to market the system, e.g. to supplement existing training situations as offered by various companies. Companies that could be interested include specific Unix vendors as well as independent training institutes. The necessary funding for future research in the areas named above could be achieved that way.

Beyond that, it can be said that IT systems keep on growing in complexity, and that demand on system administrators increases accordingly. And with it, the contents that need to be taught to them to cope with their workload. This increase in information and requirements can only be solved by more and better education in system administration, and related tools like the Virtual Unix Lab.



## List of figures

1.1	Instructions for the command line and a graphical user interface. Image source: [Emzy Bilder Galerie, 2007] . . . . .	5
1.2	Topics related to system administration . . . . .	6
1.3	Topics of information science . . . . .	7
3.1	Behavioristic approach of teaching. Image Source: [Kerres, 1998, p. 46]	25
3.2	The TOTE model. Image source: [Miller et al., 1960, p. 26] . . . . .	28
3.3	Structure of the “System Administration” lecture . . . . .	41
3.4	Thematic groups in the “System Administration” lecture . . . . .	42
3.5	Levels of difficulty in the “System Administration” lecture . . . . .	43
3.6	Examples help learning without a computer . . . . .	45
3.7	Change in learning paradigm with advancing level . . . . .	47
4.1	Logging into the Virtual Unix Lab . . . . .	54
4.2	Entering data for a new login . . . . .	54
4.3	Welcome to the Virtual Unix Lab . . . . .	55
4.4	Booking an exercise: selecting date and time . . . . .	56
4.5	Booking an exercise: selecting the exercise . . . . .	57
4.6	Booking an exercise: confirmation . . . . .	58
4.7	An exercise is prepared and waiting . . . . .	58

---

4.8	Configuring access to the lab machines . . . . .	59
4.9	Waiting for start of exercise time . . . . .	60
4.10	Display of the exercise text . . . . .	61
4.11	Logging into lab machines for the exercise . . . . .	62
4.12	End of exercise . . . . .	62
4.13	Feedback on an exercise taken . . . . .	63
4.14	The initial implementation of the Virtual Unix Lab . . . . .	64
4.15	Accessing the lab clients . . . . .	65
4.16	Software components of the Virtual Unix Lab . . . . .	66
6.1	Verifying on the semantic and pragmatic layer . . . . .	85
6.2	Step 0: Separate exercise text and verification check script . . . . .	86
6.3	Exercise text with no associated checks, in plain ASCII . . . . .	88
6.4	Exercise text with no associated checks, rendered in web browser . . . . .	89
6.5	Step I: Preparation . . . . .	92
6.6	Step I: Exercise . . . . .	92
6.7	Step I: Verification . . . . .	93
6.8	Defining an exercise, step 1: general properties . . . . .	94
6.9	Defining an exercise, step 2: which image to deploy on which lab machine . . . . .	94
6.10	Defining an exercise, step 3: what checks to run on which machine . . . . .	95
6.11	Extended web interface to enter parameters for check script . . . . .	99
6.12	Listing existing checks . . . . .	100
6.13	Possible parameters of a check script, and their description . . . . .	100
6.14	Exercise text and checks: a) uncoupled in step I, b) coupled in step II . . . . .	103
6.15	Example exercise text with check data . . . . .	106

6.16	Giving feedback on an exercise for a single user . . . . .	108
6.17	Giving teacher/admin feedback for all users which took an exercise . .	109
6.18	Defining an admin-only exercise to update the Solaris image, step 1: only “admin” may book . . . . .	112
6.19	Defining an admin-only exercise to update the Solaris image, step 2: Solaris will be preinstalled . . . . .	112
6.20	Defining an admin-only exercise to update the Solaris image, step 3: the disk will be cleaned and put into an image file after the exercise . .	113
6.21	Step II: Preparation . . . . .	115
6.22	Step II: Exercise . . . . .	115
6.23	Step II: Verification . . . . .	116
6.24	Step II: Feedback . . . . .	117
6.25	Preparing an exercise, part 1: Writing exercise text and hints . . . . .	118
6.26	Preparing an exercise, part 2: Extracting hints into database and writ- ing new text with check-numbers for feedback hints . . . . .	118
6.27	Preparing an exercise, part 3: Comparing original and updated exercise text . . . . .	119
6.28	Preparing an exercise, part 4: Moving the updated exercise into place and saving to the CMS . . . . .	119
6.29	The list of booked exercises contains both completed exercises for which feedback can be requested (“freigegeben: nicht-mehr”) as well as uncompleted exercises that have not yet started (“freigegeben: nein”) 121	121
6.30	Buttons for a) retrieving feedback on completed exercises, and b) delet- ing uncompleted exercise that have not yet started . . . . .	121
6.31	VUDSL example for verifying one aspect of the exercise . . . . .	123
6.32	VUDSL example for verifying multiple aspects of the exercise in one go 124	124
6.33	Various forms of non-linear exercises . . . . .	126
7.1	Comparison of all scores between students’ first and last exercise . . .	135
7.2	Score of all first and last exercises ordered ascending . . . . .	135

---

7.3	Score of all first and last exercises ordered by first exercise . . . . .	136
7.4	Comparison of a) NIS and b) NFS scores between students' first and last exercise . . . . .	137
7.5	Score of first and last exercise ordered ascending for a) NIS and b) NFS exercise . . . . .	138
7.6	Score of first and last exercise ordered by first exercise for a) NIS and b) NFS exercise . . . . .	139
7.7	Results of check-program-output . . . . .	141
7.8	Results of check-file-contents . . . . .	142
7.9	Results of unix-check-process-running . . . . .	143
7.10	Results of netbsd-check-rvar-set . . . . .	144
7.11	Results of unix-check-file-owner . . . . .	145
7.12	Results of check-file-exists . . . . .	146
7.13	Results of netbsd-check-installed-pkg . . . . .	147
7.14	Results of solaris-check-installed-pkg . . . . .	148
7.15	Results of unix-check-user-exists . . . . .	149
7.16	Results of check-directory-exists . . . . .	150
7.17	Duration of all exercises: a) overview and b) zoomed to the end of exercise . . . . .	151
7.18	Duration of NIS exercises: a) overview b) zoomed to the end of exercises	152
7.19	Duration of NFS exercises: a) overview b) zoomed to the end of exercises	153
7.20	Comparison of durations of NIS and NFS exercises . . . . .	154
7.21	Starttime of exercises . . . . .	154
7.22	Popularity of learning materials among students . . . . .	163
7.23	Helpful learning material in the Virtual Unix Lab . . . . .	164
7.24	Impact of the "SA" lecture on various topics of the Virtual Unix Lab exercises . . . . .	166



---

7.25	Impact of the “SA” lecture notes on various topics of the Virtual Unix Lab exercises . . . . .	167
8.1	Aspects of a didactic operation. Image source: [Wenger, 1987, p. 397]	182
8.2	Taxonomy of behavioral diagnostic processes. Image source: [Wenger, 1987, p. 372] . . . . .	197
8.3	Terms: Adaptive User Interfaces and Intelligent Interfaces. Image source: [Dietrich et al., 1993, p. 14, Figure 1] . . . . .	204
9.1	System administration is like hitting a nail with a hammer. Sometimes. Image sources: [Bent Nail, 2007], [Morell, 2004] . . . . .	212
9.2	Goals and sub-goals of the Network Information System (NIS) . . . . .	213
9.3	Goals and sub-goals of the Network File System (NFS) . . . . .	214
9.4	Error distribution of check scripts . . . . .	218
9.5	The Virtual Unix Lab with tutoring and adaption (new components in bold) . . . . .	220
10.1	Comparison of tutoring approaches, from best (++) to worst (--) . . . . .	231
10.2	The path of incoming information . . . . .	233
10.3	Possible course of an exercise (+=done, -=todo) . . . . .	236
10.4	Going backward to find the latest (a) and next part being worked on (b)	237
10.5	Learning prerequisites by a) interrupting and b) repeating . . . . .	240
10.6	The current user interface: Menue structure . . . . .	243
10.7	The current user interface: During the exercise . . . . .	244
11.1	Structural and longitudinal consistency in the Virtual Unix Lab . . . . .	251



## List of tables

2.1	Education of computer science at European universities [cited 2007-08-16] . . . . .	12
2.2	Education of system administration at universities [cited 2007-08-16]	13
2.3	Harddisk image cloning software [cited 2007-08-18] . . . . .	15
2.4	Virtualization and emulation software [cited 2007-08-16] . . . . .	22
7.1	Exercise popularity . . . . .	132
7.2	Check scripts and their usage in various checks . . . . .	140
7.3	Distribution of exercise start times . . . . .	150



# Bibliography

- [Abelson et al., 1985] Abelson, H., Sussman, G. J., and Sussman, J. (1985). *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, MA, USA. Available from: <http://mitpress.mit.edu/sicp/full-text/book/book.html> [cited 2007-10-05].
- [Adams and Erickson, 2001] Adams, D. R. and Erickson, C. (2001). Teaching networking and operating systems to information systems majors. In *SIGCSE '01: Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*, pages 85–89, New York, NY, USA. ACM Press.
- [Adams and Laverell, 2005] Adams, J. C. and Laverell, W. D. (2005). Configuring a multi-course lab for system-level projects. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education*, pages 525–529, New York, NY, USA. ACM Press.
- [ADL Technical Team, 2004] ADL Technical Team. Sharable Content Object Reference Model (SCORM) Documentation Suite [online]. (2004) [cited 2007-10-15]. Available from: <http://www.adlnet.gov/downloads/DownloadPage.aspx?ID=237>.
- [Aho et al., 1988] Aho, A. V., Kernighan, B. W., and Weinberger, P. J. (1988). *The AWK Programming Language*. Addison Wesley, Boston, MA, USA.
- [Aho et al., 2003] Aho, A. V., Sethi, R., and Ullman, J. D. (2003). *Compilers. Principles, Techniques and Tools*. Addison Wesley, Boston, MA, USA.
- [Alexander, 1995] Alexander, C. (1995). *Eine Muster-Sprache*. Löcker Verlag, Vienna, Austria.
- [Alva L. Couch and Gilfix, 1999] Alva L. Couch, D. and Gilfix, M. (1999). It's Elementary, Dear Watson: Applying Logic Programming To Convergent System Management Processes. In *LISA '99: Proceedings of the 13th USENIX conference on System administration*, pages 123–138, Boston, MA, USA. USENIX Association.
- [Alvisi et al., 2002] Alvisi, L., Bhatia, K., and Marzullo, K. (2002). Causality tracking in causal message-logging protocols. *Distributed Computing*, 15(1):1–15.

- [Anderson et al., 2006] Anderson, D. S., Hibler, M., Stoller, L., Stack, T., and Lepreau, J. (2006). Automatic online validation of network configuration in the emulab network testbed. In *Proceedings of the Third IEEE International Conference on Autonomic Computing (ICAC 2006)*, Los Alamitos, CA, USA. IEEE Computer Society Press. Available from: <http://www.cs.utah.edu/flux/papers/linktest-icac06.pdf> [cited 2007-10-05].
- [Anderson and Scobie, 2002] Anderson, P. and Scobie, A. (2002). LCFG: The next generation. In *Proceedings of the UKUUG Winter Conference 2002*, Buntingford, UK. United Kingdom Unix User Group. Available from: <http://www.lcfg.org/doc/ukuug2002.pdf> [cited 2007-10-05].
- [Angelides and Paul, 1993] Angelides, M. C. and Paul, R. J. (1993). Towards a framework for integrating intelligent tutoring systems and gaming-simulation. In *WSC '93: Proceedings of the 25th conference on Winter simulation*, pages 1281–1289, New York, NY, USA. ACM Press.
- [Baker et al., 2004] Baker, R. S., Corbett, A. T., Koedinger, K. R., and Wagner, A. Z. (2004). Off-task behavior in the cognitive tutor classroom: when students "game the system". In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 383–390, New York, NY, USA. ACM Press.
- [Ball, 1999] Ball, T., editor (1999). *Proceedings of the 2nd Conference on Domain-Specific Languages*. USENIX Association, Boston, MA, USA.
- [Baseline, 2007] BASELINE, editor. Frequently Asked Questions about User Validation: Questionnaires [online]. (2007) [cited 2007-10-05]. Available from: <http://www.ucc.ie/hfrg/baseline/questionnaires.html>.
- [Beale and Rogers, 2007] Beale, J. and Rogers, R. (2007). *Nessus Network Auditing*. Syngress Publishing, Amsterdam, Netherlands.
- [Beck, 1999] Beck, K. (1999). *Extreme Programming Explained*. Addison Wesley, Boston, MA, USA.
- [Beck, 2002] Beck, K. (2002). *Test Driven Development*. Addison Wesley, Boston, MA, USA.
- [Ben-Gal, 2007] Ben-Gal, I. (2007). Bayesian Networks. In Ruggeri, F., Kenett, R., and Faltin, F., editors, *Encyclopedia of Statistics in Quality and Reliability*. John Wiley & Sons, Indianapolis, IN, USA.
- [Bent Nail, 2007] Bent Nail. New & Used Building Supplies - Deconstruction, Demolition & Salvage [online]. (2007) [cited 2007-10-05]. Available from: <http://www.bentnail.org/bennai/Profile.html>.
- [Bentley, 1986] Bentley, J. (1986). Programming pearls: Little languages. *Communications of the ACM*, 29(8):711–721.

- [Berners-Lee et al., 1999] Berners-Lee, T., Fischetti, M., and Dertouzos, M. L. (1999). *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. Harper San Francisco, San Francisco, CA, USA.
- [Berners-Lee et al., 2001] Berners-Lee, T., Lassila, O., and Hendler, J. (2001). The semantic web. *Scientific American*, 284(5):34–43.
- [BGG, 2002] Bundesministerium für Gesundheit und Soziale Sicherung, editor. *Gesetz zur Gleichstellung behinderter Menschen (BGG)* [online]. (2002) [cited 2007-10-05]. Available from: <http://bundesrecht.juris.de/bundesrecht/bgg/>.
- [BITV, 2002] Bundesministerium des Inneren, editor. *Verordnung zur Schaffung barrierefreier Informationstechnik nach dem Behindertengleichstellungsgesetz (BITV)* [online]. (2002) [cited 2007-10-05]. Available from: <http://bundesrecht.juris.de/bundesrecht/bitv/>.
- [Boctor, 1999] Boctor, D. (1999). *Microsoft Office 2000: Visual Basic for Applications Fundamentals*. Microsoft Press, Redmond, WA, USA.
- [Bonar et al., 1986] Bonar, J., Cunningham, R., and Schultz, J. (1986). An object-oriented architecture for intelligent tutoring systems. In *OOPSLA '86: Conference proceedings on Object-oriented programming systems, languages and applications*, pages 269–276, New York, NY, USA. ACM Press.
- [Border, 2007] Border, C. (2007). The development and deployment of a multi-user, remote access virtualization system for networking, security, and system administration classes. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 576–580, New York, NY, USA. ACM Press.
- [Bortz and Döring, 2002] Bortz, J. and Döring, N. (2002). *Forschungsmethoden und Evaluation für Human- und Sozialwissenschaftler*. Springer Verlag, Heidelberg, Germany.
- [Brewer, 2007] Brewer, W. F. Learning Theory – Schema Theory [online]. (2007) [cited 2007-12-16]. Available from: <http://education.stateuniversity.com/pages/2175/Learning-Theory-SCHEMA-THEORY.html>.
- [Brooke, 1996] Brooke, J. (1996). A quick and dirty usability scale. In Jordan, P. W., Thomas, B., Weerdmeester, B. A., and McClelland, I. L., editors, *Usability Evaluation in Industry*. Taylor & Francis, London, UK.
- [Bruner, 1961] Bruner, J. S. (1961). The act of discovery. *Harvard Educational Review*, 31(1):21–32.
- [Bruns and Gajewski, 2002] Bruns, B. and Gajewski, P. (2002). *Multimediales Lernen im Netz – Leitfaden für Entscheider und Planer*. Springer Verlag, Heidelberg, Germany.

- [Brusilovsky and Cooper, 2002] Brusilovsky, P. and Cooper, D. W. (2002). Domain, task, and user models for an adaptive hypermedia performance support system. In *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces*, pages 23–30, New York, NY, USA. ACM Press.
- [Bundesministerium für Bildung und Forschung, 2004] Bundesministerium für Bildung und Forschung, editor (2004). *Kursbuch eLearning 2004: Produkte aus dem Förderprogramm Neue Medien in der Bildung - Hochschule*. Bundesministerium für Bildung und Forschung, Bonn, Germany. Available from: [http://www.bmbf.de/pub/nmb\\_kursbuch.pdf](http://www.bmbf.de/pub/nmb_kursbuch.pdf) [cited 2007-10-05].
- [Burgess, 1995] Burgess, M. (1995). A site configuration engine. *Computing Systems*, 8(2):309–337. Available from: <http://www.iu.hio.no/~mark/papers/paper1.pdf> [cited 2007-10-05].
- [Burgess, 2000] Burgess, M. (2000). Theoretical system administration. In *LISA '00: Proceedings of the 14th USENIX conference on System administration*, pages 1–14, Boston, MA, USA. USENIX Association.
- [Burgess and Frisch, 2007] Burgess, M. and Frisch, A. (2007). *A System Engineer's Guide to Host Configuration and Maintenance Using Cfengine*. USENIX Association, Boston, MA, USA. Available from: [http://www.sage.org/pubs/16\\_cfengine/](http://www.sage.org/pubs/16_cfengine/) [cited 2007-10-05].
- [Butz et al., 2006] Butz, C. J., Hua, S., and Maguire, R. B. (2006). A web-based bayesian intelligent tutoring system for computer programming. *Web Intelligence and Agent System*, 4(1):77–97.
- [Buzan and Buzan, 2006] Buzan, T. and Buzan, B. (2006). *The Mind Map Book*. Random House, New York, NY, USA.
- [Campbell and Cohen, 2005] Campbell, W. and Cohen, R. (2005). Using system administrator education in developing an IT degree in a computer science department. In *SIGITE '05: Proceedings of the 6th conference on Information technology education*, pages 319–321, New York, NY, USA. ACM Press.
- [Carbonell, 1970] Carbonell, J. R. (1970). Mixed-initiative man-computer instructional dialogues. Technical Report 1971, Bolt, Beranek and Newman, Cambridge.
- [Carr and Goldstein, 1977] Carr, B. and Goldstein, I. P. (1977). Overlays: a Theory of Modelling for Computer Aided Instruction. Technical Report AI Memo 406 (Logo Memo 40), Massachusetts Institute of Technology, Cambridge, MA, USA.
- [Chaffin, 1992] Chaffin, R. (1992). The concept of a semantic relation. In Lehrer, A., editor, *Frames, Fields and contrasts*, pages 253–288. Lawrence Erlbaum Associates, Publishers, Hillsdale, NJ, USA.
- [Chambers, 1983] Chambers, J. M. (1983). *Graphical methods for data analysis*. Wadsworth International Group, Belmont, CA, USA.



- [Chassell, 2004] Chassell, R. J. (2004). *An Introduction to Programming in Emacs Lisp*. Free Software Foundation, Boston, MA, USA.
- [Chauvin, 1991] Chauvin, Y. (1991). MENIX: A Unix user adaptable Interface. *SIGCHI Bulletin*, 23(4):64–65.
- [Chin, 1983] Chin, D. N. (1983). Knowledge structures in uc, the unix consultant. In *Proceedings of the 21st annual meeting on Association for Computational Linguistics*, pages 159–163, Morristown, NJ, USA. Association for Computational Linguistics.
- [Chin, 1986] Chin, D. N. (1986). User modeling in uc, the unix consultant. In *CHI '86: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 24–28, New York, NY, USA. ACM Press.
- [Clark, 2000] Clark, D. Developing Instruction or Instructional Design [online]. (2000) [cited 2007-10-05]. Available from: <http://www.nwlink.com/~donclark/hrd/learning/development.html>.
- [Cocke and Schwartz, 1970] Cocke, J. and Schwartz, J. T. (1970). *Programming Languages and their Compilers*. Courant Institute of Mathematical Sciences, New York Universities, New York, NY, USA.
- [Cole, 2005] Cole, J. (2005). *Using Moodle*. O'Reilly, Sebastopol, CA, USA.
- [Conati et al., 2002] Conati, C., Gertner, A., and VanLehn, K. (2002). Using Bayesian Networks to Manage Uncertainty in Student Modeling. *User Modeling and User-Adapted Interaction*, 12(4):371–417.
- [Conlan et al., 2003] Conlan, O., Power, R., Higel, S., O'Sullivan, D., and Barrett, K. (2003). Next generation context aware adaptive services. In *ISICT '03: Proceedings of the 1st international symposium on Information and communication technologies*, pages 205–212. Trinity College Dublin.
- [Cooper, 2004] Cooper, A. (2004). *The Inmates Are Running the Asylum*. Sams Publishing, Indianapolis, IN, USA.
- [Corbesero, 2003] Corbesero, S. G. (2003). Teaching system and network administration in a small college environment. *Journal of Computing Sciences in Colleges (JCSC)*, 19(2):155–163.
- [Corbett and Anderson, 2001] Corbett, A. T. and Anderson, J. R. (2001). Locus of feedback control in computer-based tutoring: impact on learning rate, achievement and attitudes. In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 245–252, New York, NY, USA. ACM Press.
- [Corlett, 1991a] Corlett, J. A. (1991a). Epistemology and experimental cognitive psychology: A reply to fuller, schmitt, and greenwood. *New ideas in Psychology*, 9:327–334.

- [Corlett, 1991b] Corlett, J. A. (1991b). Some connections between epistemology and cognitive psychology. *New ideas in Psychology*, 9:285–306.
- [Cornell University, 2007] Cornell University. Program Overview: Information Science [online]. (2007) [cited 2007-10-05]. Available from: <http://www.infosci.cornell.edu/about/>.
- [CPAN, 2007] Comprehensive Perl Archive Network [online]. (2007) [cited 2007-10-05]. Available from: <http://www.cpan.org/>.
- [CSTA, 2007] Homepage of the Computer Science Teachers Association [online]. (2007) [cited 2007-10-05]. Available from: <http://csta.acm.org/>.
- [Cullingford, 1981] Cullingford, R. (1981). SAM. In Schank, R. and Reisbeck, C., editors, *Inside Computer Understanding*. Lawrence Erlbaum Associates, Publishers, Hillsdale, NJ, USA.
- [Cunningham, 2001] Cunningham, W. Manifesto for Agile Software Development [online]. (2001) [cited 2007-12-22]. Available from: <http://www.agilemanifesto.org/>.
- [Cycorp, 2007] Cycorp, editor. Cycorp inc. homepage [online]. (2007) [cited 2007-12-12]. Available from: <http://www.cyc.com/>.
- [Dagdilelis and Satratzemi, 1999] Dagdilelis, V. and Satratzemi, M. (1999). Didactics too, not only technology. In *ITiCSE '99: Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education*, page 183, New York, NY, USA. ACM Press.
- [Darbhhamulla and Lawhead, 2004] Darbhhamulla, R. and Lawhead, P. (2004). Paving the way towards an efficient learning management system. In *ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference*, pages 428–433, New York, NY, USA. ACM Press.
- [Davies et al., 2001] Davies, J. R., Gertner, A. S., Lesh, N., Rich, C., Sidner, C. L., and Rickel, J. (2001). Incorporating tutorial strategies into an intelligent assistant. In *IUI '01: Proceedings of the 6th international conference on Intelligent user interfaces*, pages 53–56, New York, NY, USA. ACM Press. Available from: <http://www.merl.com/reports/docs/TR2000-30.pdf> [cited 2007-10-05].
- [Derose, 1997] Derose, S. J. (1997). *The SGML FAQ Book: Understanding the Foundation of HTML and XML*. Klower Academic, Dordrecht, Netherlands.
- [Deutsches Institut für Normung, 2003] Deutsches Institut für Normung, editor (2003). *DIN-Taschenbuch 354: Software-Ergonomie*. Beuth Verlag, Berlin, Germany.
- [Dhanjani, 2004] Dhanjani, N. Writing nessus plugins [online]. (2004) [cited 2007-10-05]. Available from: [http://www.oreillynet.com/pub/a/security/2004/06/03/nessus\\_plugins.html](http://www.oreillynet.com/pub/a/security/2004/06/03/nessus_plugins.html).

- [di Forino, 1969] di Forino, A. C. (1969). Programming languages. In *Advances in Information Systems Science*, volume I. Plenum Press, New York, NY, USA.
- [Dietrich et al., 1993] Dietrich, H., Malinowski, U., Kühme, T., and Schneider-Hufschmidt, M. (1993). State of the art in adaptive user interfaces. In Schneider-Hofschmidt, M., Kühme, T., and Malinowski, U., editors, *Adaptive User Interfaces*. Elsevier Science Publishers, Amsterdam, Netherlands.
- [Dijkstra, 1961] Dijkstra, E. W. (1961). On the Design of Machine independent Programming Languages. Technical Report MR 34, Stichting Mathematical Centrum, Amsterdam. Available from: <http://www.cs.utexas.edu/users/EWD/MCReps/MR34.PDF> [cited 2007-10-20].
- [Dike, 2006] Dike, J. (2006). *User Mode Linux*. Pearson Studium Verlag, München, Germany.
- [Dougherty and Robbins, 1997] Dougherty, D. and Robbins, A. (1997). *Sed & Awk*. O'Reilly, Sebastopol, CA, USA.
- [Edwards et al., 1997] Edwards, P., Rivett, R., and McCall, G. (1997). Towards an automotive safer subset of C. In Daniel, P., editor, *Proceedings of the 16th International Conference on Computer Safety, Reliability and Security*, pages 185–195, Heidelberg, Germany. Springer Verlag.
- [Eide et al., 2006] Eide, E., Stoller, L., Stack, T., Freire, J., and Lepreau, J. (2006). Integrated scientific workflow management for the emulab network testbed. In *Proceedings of the 2006 USENIX Annual Technical Conference*, pages 363–368, Boston, MA, USA. Available from: <http://www.cs.utah.edu/flux/papers/workflow-usenix06-base.html> [cited 2007-10-05].
- [Eikenbusch and Leuders, 2004] Eikenbusch, G. and Leuders, T., editors (2004). *Lehrer-Kursbuch Statistik*. Cornelsen Verlag Scriptor, Berlin, Germany.
- [Elliston et al., 2000] Elliston, B., Gkioulas, Taylor, Trome, and Vaughan (2000). *Autoconf, Automake and Libtool*. Que Publishing, Indianapolis, IN, USA.
- [Emulab, 2007a] Emulab - Network Emulation Testbed Home [online]. (2007) [cited 2007-10-05]. Available from: <http://www.emulab.net/>.
- [Emulab, 2007b] Other Emulab Testbeds [online]. (2007) [cited 2007-10-05]. Available from: <http://www.emulab.net/docwrapper.php3?docname=otheremulabs.html>.
- [Emzy Bilder Galerie, 2007] Emzy Bilder Galerie. NT versus Unix [online]. (2007) [cited 2007-10-05]. Available from: <http://www.emzy.de/gallery/fun/NTvsUnix>.
- [Ernst, 2004] Ernst, T. (2004). Mögliche Szenarien für das Virtuelle Unix Labor. Technical report, Fachhochschule Regensburg, Computer Science Department.

- [Evelt, 1994] Evelt, M. P. (1994). *PARKA: A System for Massively Parallel Knowledge Representation*. PhD thesis, University of Maryland.
- [Fachhochschule Regensburg, 2007] Fachhochschule Regensburg. Laboratory of Communication Technologies [online]. (2007) [cited 2007-10-05]. Available from: <http://comserver.fh-regensburg.de/>.
- [Fahrmeir, 2003] Fahrmeir, L. (2003). *Statistik*. Springer Verlag, Heidelberg, Germany.
- [FernUniversität Hagen, 2007] FernUniversität Hagen. Virtuelles Informatik-Labor [online]. (2007) [cited 2007-10-05]. Available from: <http://pi7.fernuni-hagen.de/vilab/>.
- [Feyrer, 2001] Feyrer, H. Mit dem Regensburger Marathon-Cluster durch's Ziel [online]. (2001) [cited 2007-10-05]. Available from: <http://www.feyrer.de/marathon-cluster/>.
- [Feyrer, 2004a] Feyrer, H. (2004a). An Introduction to Sysadmin Training in the Virtual Unix Lab. In *EuroBSDCon 2004 Proceedings*, Karlsruhe, Germany. Available from: <http://www.feyrer.de/Texts/Own/eurobsdcon2004-vulab-paper.pdf>.
- [Feyrer, 2004b] Feyrer, H. Virtuelles Unix Labor - Deployment der Übungsrechner [online]. (2004) [cited 2007-10-05]. Available from: <http://vulab.fh-regensburg.de/~feyrer/vulab/hubertf/deployment>.
- [Feyrer, 2004c] Feyrer, H. Virtuelles Unix Labor - Design [online]. (2004) [cited 2007-10-05]. Available from: <http://vulab.fh-regensburg.de/~feyrer/vulab/hubertf/design>.
- [Feyrer, 2004d] Feyrer, H. Virtuelles Unix Labor - Firewall [online]. (2004) [cited 2007-10-05]. Available from: <http://vulab.fh-regensburg.de/~feyrer/vulab/hubertf/firewall>.
- [Feyrer, 2004e] Feyrer, H. Virtuelles Unix Labor - Kursengine [online]. (2004) [cited 2007-10-05]. Available from: <http://vulab.fh-regensburg.de/~feyrer/vulab/hubertf/kursengine>.
- [Feyrer, 2004f] Feyrer, H. Virtuelles Unix Labor - Netboot Setup [online]. (2004) [cited 2007-10-05]. Available from: <http://vulab.fh-regensburg.de/~feyrer/vulab/hubertf/netboot-doku.txt>.
- [Feyrer, 2005] Feyrer, H. (2005). Didaktik der Systemadministration. In *GUUG Frühjahrsfachgespräch 2005 Proceedings*, Munich, Germany. Available from: <http://vulab.fh-regensburg.de/~feyrer/vulab/hubertf/guug-sa-did.pdf> [cited 2007-10-05].

- [Feyrer, 2007a] Feyrer, H. (2007a). Education of System Administration. Technical report, Computer Science Department of the University of Applied Sciences Regensburg and Information Science Department of the University of Regensburg. Available from: <http://www.feyrer.de/Texts/Own/article-vulab-didactics.pdf> [cited 2007-12-04].
- [Feyrer, 2007b] Feyrer, H. g4u - Harddisk Image Cloning for PCs [online]. (2007) [cited 2007-10-05]. Available from: <http://www.feyrer.de/g4u/>.
- [Feyrer, 2007c] Feyrer, H. (2007c). Impact of the Virtual Unix Lab: Evaluation of end-of-semester papers tests. Technical report, Computer Science Department of the University of Applied Sciences Regensburg and Information Science Department of the University of Regensburg. Available from: <http://www.feyrer.de/Texts/Own/article-vulab-eval-papers.pdf> [cited 2007-12-04].
- [Feyrer, 2007d] Feyrer, H. (2007d). Implementing exercise result verification for the Virtual Unix Lab. Technical report, Computer Science Department of the University of Applied Sciences Regensburg and Information Science Department of the University of Regensburg. Available from: <http://www.feyrer.de/Texts/Own/article-vulab-resver-implementation.pdf> [cited 2007-12-04].
- [Feyrer, 2007e] Feyrer, H. Systemadministration unter Unix [online]. (2007) [cited 2007-10-05]. Available from: <http://www.feyrer.de/SA/>.
- [Fink et al., 1998] Fink, J., Kobsa, A., and Nill, A. (1998). Towards a user-adapted information environment on the web. In *Proceedings of Multimedia and Standardization 98*, Paris, France. Available from: <http://www.isr.uci.edu/~kobsa/papers/1998-must-kobsa.pdf> [cited 2007-10-05].
- [Finkel et al., 1995] Finkel, R., Ortega, L., and Shanklin, C. (1995). *Advanced Programming Languages*. Addison Wesley, Boston, MA, USA. Available from: <ftp://aw.com/cseng/authors/finkel/apld> [cited 2007-10-05].
- [Fischer, 1993] Fischer, G. (1993). Shared knowledge in cooperative problem-solving systems – integrating adaptive and adaptable components. In Schneider-Hofschmidt, M., Kühme, T., and Malinowski, U., editors, *Adaptive User Interfaces*. Elsevier Science Publishers, Amsterdam, Netherlands.
- [Fischer, 2001] Fischer, S. (2001). Course and exercise sequencing using metadata in adaptive hypermedia learning systems. *Journal on Educational Resources in Computing (JERIC)*, 1(1es):5.
- [Fischer and Steinmetz, 2000] Fischer, S. and Steinmetz, R. (2000). Automatic creation of exercises in adaptive hypermedia learning systems. In *HYPertext '00: Proceedings of the eleventh ACM on Hypertext and hypermedia*, pages 49–55, New York, NY, USA. ACM Press.

- [Fletcher, 1975] Fletcher, J. D. (1975). Modeling of learner in computer-based instruction. *Journal of Computer-Based Instruction*, 1:118–126.
- [Floyd, 1979] Floyd, R. W. (1979). The paradigms of programming. *Communications of the ACM*, 22(8):455–460.
- [Fowler et al., 1987] Fowler, C. J. H., Macaulay, L. A., and Siripoksup, S. (1987). An evaluation of the effectiveness of the adaptive interface module (aim) in matching dialogues to users. In *Proceedings of Third Conference of the British Computer Society Human-Interaction on People and computers III*, pages 345–359, Cambridge, MA, USA. Cambridge University Press.
- [Frank, 1969] Frank, H. (1969). *Kybernetische Grundlagen des Lernens und Lehrens*. AGIS Verlag, Baden-Baden, Germany.
- [Freedman et al., 2000] Freedman, R., Ali, S. S., and McRoy, S. W. (2000). What is an intelligent tutoring system? *Intelligence*, 11(3):15–16. Available from: <http://www.cs.niu.edu/~freedman/papers/link2000.pdf> [cited 2006-05-16].
- [Friedl, 1997] Friedl, J. E. F. (1997). *Mastering Regular Expressions*. O'Reilly, Sebastopol, CA, USA.
- [Gagné, 1967] Gagné, R. (1967). *The conditions of learning*. Holt, Rinehart and Winston, New York, NY, USA.
- [Gagné and Briggs, 1974] Gagné, R. M. and Briggs, L. J., editors (1974). *Principles of instructional design*. Holt, Rinehart and Winston, New York, NY, USA.
- [Garcia et al., 2007] Garcia, P., Amandi, A., Schiaffino, S., and Campo, M. (2007). Evaluating Bayesian networks' precision for detecting students' learning styles. *Computers & Education*, 49(3):794–808. Available from: <http://dx.doi.org/10.1016/j.compedu.2005.11.017> [cited 2007-12-17].
- [Garrett, 2005] Garrett, J. J. Ajax: A new approach to web applications [online]. (2005) [cited 2007-10-05]. Available from: <http://www.adaptivepath.com/publications/essays/archives/000385.php>.
- [Garrett and Nash, 2001] Garrett, L. and Nash, J. C. (2001). Issues in Teaching the Comparison of Variability to Non-Statistics Students. *Journal of Statistics Education*, 9(2):12–16. Available from: <http://www.amstat.org/publications/jse/v9n2/garrett.html> [cited 2007-10-05].
- [Gediga et al., 1999] Gediga, G., Hamborg, K.-C., and Düntsch, I. (1999). The isometrics usability inventory: An operationalisation of iso 9241-10. *Behaviour and Information Technology*, 18:151–164.
- [Genesereth et al., 1982] Genesereth, M., Kehler, T., Barr, A., Finin, T., Friedland, P., Miller, J., Miller, M., Soloway, E., and Tennant, H. (1982). Intelligent assistance

- for complex systems. In *ACM 82: Proceedings of the ACM '82 conference*, page 124, New York, NY, USA. ACM Press.
- [GI, 2007] Gesellschaft für Informatik, editor. Informatik und Ausbildung / Didaktik der Informatik (IAD) [online]. (2007) [cited 2007-10-05]. Available from: <http://www.gi-ev.de/gliederungen/fachbereiche/informatik-und-ausbildung-didaktik-der-informatik-iad/>.
- [Gibbs, 1997] Gibbs, A. (1997). Focus Groups. *Social Research Update*, 19.
- [Glickstein, 2004] Glickstein, B. (2004). *Writing GNU Emacs Extensions: Editor Customizations and Creations with Lisp*. O'Reilly, Sebastopol, CA, USA.
- [Gosling and McGilton, 1996] Gosling, J. and McGilton, H. The Java Language Environment White Paper [online]. (1996) [cited 2007-10-05]. Available from: <ftp://ftp.javasoft.com/docs/papers/langenviron-pdf.zip>.
- [Government of the United Kingdom, 2001] Government of the United Kingdom, editor (2001). *Special Educational Needs and Disability Act 2001 (SENDA)*. The Stationery Office, London, UK.
- [Gruber, 2008] Gruber, T. (2008). Ontology. In *Encyclopedia of Database Systems*. Springer Verlag, Heidelberg, Germany.
- [Guruprasad et al., 2005] Guruprasad, S., Ricci, R., and Lepreau, J. (2005). Integrated network experimentation using simulation and emulation. In *Proceedings of the first International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (Tridentcom 2005)*, Trento, Italy. Available from: <http://www.cs.utah.edu/flux/papers/simem-tridentcom05a.pdf> [cited 2007-10-05].
- [Haake et al., 2004] Haake, J., Schwabe, G., and Wessner, M., editors (2004). *CSSL-Kompendium. Lehr- und Handbuch zum computerunterstützten kooperativen Lernen*. Oldenbourg Verlag, München, Germany.
- [Haberlandt, 1999] Haberlandt, K. (1999). *Human Memory*. Allyn & Bacon, Boston, MA, USA.
- [Hall, 2007] Hall, J. M. (2007). Beachhead: Beneath the surface. *Linux Journal*, 2007(154):16.
- [Hamilton, 2007] Hamilton, B. Rosetta Stone for Unix [online]. (2007) [cited 2007-10-05]. Available from: <http://bhami.com/rosetta.html>.
- [Hammer and Elby, 2000] Hammer, D. and Elby, A. (2000). Epistemological resources. In Fishman, B. J. and O'Connor-Divelbiss, S. F., editors, *International Conference of the Learning Sciences – Facing the Challenges of Complex Real-World Settings*, pages 4–5, University of Michigan, Ann Arbor, USA.

- [Harms et al., 2002] Harms, I., Schweibenz, W., and Strobel, J. (2002). Usability Evaluation von Web-Angeboten mit dem Usability-Index. In *Proceedings der 24. DGI-Online-Tagung 2002 - Content in Context*, Frankfurt am Main, Germany.
- [Harper and Norman, 1993] Harper, B. D. and Norman, K. L. (1993). Improving user satisfaction: The questionnaire for user interaction satisfaction. In *Proceedings of the 1st Annual Mid-Atlantic Human Factors Conference*, pages 224–228, Virginia Beach, VA, USA.
- [Heer et al., 2004] Heer, J., Good, N. S., Ramirez, A., Davis, M., and Mankoff, J. (2004). Presiding over accidents: system direction of human action. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 463–470, New York, NY, USA. ACM Press. Available from: <http://jheer.org/publications/2004-Direction-CHI.pdf> [cited 2007-10-05].
- [Heffley and Meunier, 2004] Heffley, J. and Meunier, P. (2004). Can Source Code Auditing Software Identify Common Vulnerabilities and Be Used to Evaluate Software Security? In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04)*, page 90277. Available from: <http://doi.ieeecomputersociety.org/10.1109/HICSS.2004.1265654>.
- [Hegner, 2000] Hegner, S. J. (2000). Plan realization for complex command interaction in the unix help domain. *Artificial Intelligence Review*, 14(3):181–228.
- [Heinichen et al., 2007] Heinichen, J., Raue, S., and Assman, A. Dokumentation rootlab [online]. (2007) [cited 2007-10-05]. Available from: <http://vsr.informatik.tu-chemnitz.de/backup3/Rootlab.pdf>.
- [Helic et al., 2004] Helic, D., Maurer, H., and Scerbakov, N. (2004). Combining individual tutoring with automatic course sequencing in wbt systems. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 456–457, New York, NY, USA. ACM Press.
- [Herold, 2005] Herold, R. (2005). *Managing an Information Security and Privacy Awareness and Training Program*. Auerbach Publications, New York, NY, USA.
- [Hewlett Packard, 2007] Hewlett Packard. HP TestDrive [online]. (2007) [cited 2007-10-05]. Available from: <http://www.testdrive.hp.com/>.
- [Heyer et al., 1990] Heyer, G., Kesse, R., Oemig, F., and Dudda, F. (1990). Knowledge representation and semantics in a complex domain: the unix natural language help system goethe. In *Proceedings of the 13th conference on Computational linguistics*, pages 361–363, Morristown, NJ, USA. Association for Computational Linguistics.
- [Hibler et al., 2004] Hibler, M., Ricci, R., Stoller, L., Duerig, J., Guruprasad, S., tim Stack, Webb, K., and Lepreau, J. (2004). Feedback-directed virtualization techniques for scalable network experimentation. Technical Note FTN-2004-02, Flux Group, University of Utah.



- [Hilfinger, 1981] Hilfinger, P. N. (1981). *Abstraction mechanisms and language design*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA.
- [Hill, 1988] Hill, G. (1988). A rule-based software engineering tool for code analysis. In *Proceedings of the Seventh Annual International Phoenix Conference on Computers and Communications*, pages 291–295.
- [Hoare, 1973] Hoare, C. A. R. (1973). Hints on programming language design. Technical Report AIM-224, STAN-CS-73-403, Stanford University, Artificial Intelligence Laboratory, Computer Science Department.
- [Holland and Skinner, 1961] Holland, J. G. and Skinner, B. F. (1961). *The analysis of behaviour*. McGraw Hill, New York, NY, USA.
- [Hu et al., 2004] Hu, J., Meinel, C., and Schmitt, M. (2004). Tele-lab it security: an architecture for interactive lessons for security education. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 412–416, New York, NY, USA. ACM Press.
- [Huang et al., 2004] Huang, Y.-W., Yu, F., Hang, C., Tsai, C.-H., Lee, D.-T., and Kuo, S.-Y. (2004). Security and privacy: Securing web application code by static analysis and runtime protection. In *Proceedings of the 13th International Conference on World Wide Web*, pages 40–52.
- [Hubwieser, 2000] Hubwieser, P. (2000). *Didaktik der Informatik*. Springer Verlag, Heidelberg, Germany.
- [Humbert, 2006] Humbert, L. (2006). *Didaktik der Informatik - mit praxiserprobtem Unterrichtsmaterial*. Teubner Verlag, Wiesbaden, Germany.
- [Hylton et al., 2005] Hylton, K., Rosson, M. B., Carroll, J. M., and Ganoe, C. (2005). When news is more than what makes headlines. *ACM Crossroads: Human-Computer Interaction*, 12(2):13–17. Available from: <http://www.acm.org/crossroads/xrds12-2/rss.html> [cited 2007-10-05].
- [iABG, 2007] iABG, editor. Das V-Modell® - Homepage [online]. (2007) [cited 2007-11-05]. Available from: <http://www.v-modell.iabg.de/>.
- [ISO 16071, 2003] ISO 16071 (2003). ISO 16071:2003(E) – Ergonomics of human-system interaction – Guidance on accessibility for human-computer interfaces. In [Deutsches Institut für Normung, 2003].
- [ISO 23270, 2006] ISO 23270 (2006). ISO 23270:2006(E) – C# Language Specification. Available from: [http://standards.iso.org/ittf/PubliclyAvailableStandards/c042926\\_ISO\\_IEC\\_23270\\_2006\(E\).zip](http://standards.iso.org/ittf/PubliclyAvailableStandards/c042926_ISO_IEC_23270_2006(E).zip) [cited 2007-11-19].
- [ISO 8879, 1986] ISO 8879 (1986). *ISO 8879:1986(E) – Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML)*. International Organization for Standardization, Geneva, Switzerland.

- [ISO 9241, 2003] ISO 9241 (2003). ISO 9241:2003(D) – Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten. In [Deutsches Institut für Normung, 2003].
- [Jakob Nielsen, 1997] Jakob Nielsen (1997). The Use and Misuse of Focus Groups. *IEEE Software*, 14(1):94–95.
- [Jerinic and Devedzic, 2000] Jerinic, L. and Devedzic, V. (2000). The friendly intelligent tutoring environment. *SIGCHI Bull.*, 32(1):83–94.
- [Johansson, 2002] Johansson, P. (2002). User modeling in dialog systems. Technical Report SAR 02-2, St. Anna. Available from: [http://www.ida.liu.se/~ponjo/downloads/papers/johansson\\_sar2002.pdf](http://www.ida.liu.se/~ponjo/downloads/papers/johansson_sar2002.pdf) [cited 2007-10-05].
- [Johnson, 1975] Johnson, S. C. (1975). Yacc: Yet Another Compiler Compiler. Technical Report Computing Science Technical Report No. 32, Bell Laboratories, Murray Hill, NJ, USA.
- [Kamp and Watson, 2007] Kamp, P.-H. and Watson, R. N. M. Jails: Confining the omnipotent root [online]. (2007) [cited 2007-10-05]. Available from: <http://docs.freebsd.org/44doc/papers/jail/jail.html>.
- [Kautz and Selman, 1992] Kautz, H. and Selman, B. (1992). Planning as satisfiability. In *ECAI '92: Proceedings of the 10th European conference on Artificial intelligence*, pages 359–363, Indianapolis, IN, USA. John Wiley & Sons.
- [Kay, 1972] Kay, A. (1972). A personal computer for children of all ages. In *Proceedings of the ACM National Conference*, Boston, MA, USA. Available from: <http://www.mprove.de/diplom/gui/Kay72a.pdf> [cited 2007-11-19].
- [Keller and Krüger, 2001] Keller, H. and Krüger, S. (2001). *ABAP Objects: Einführung in die SAP-Programmierung*. SAP Press, Bonn, Germany.
- [Kerner and Freedman, 1990] Kerner, J. T. and Freedman, R. S. (1990). Developing intelligent tutoring systems with a Hypermedia Object-Based Intelligent Educator (HOBIE). In *IEA/AIE '90: Proceedings of the 3rd international conference on Industrial and engineering applications of artificial intelligence and expert systems*, pages 890–897, New York, NY, USA. ACM Press.
- [Kernighan and Ritchie, 1988] Kernighan, B. W. and Ritchie, D. M. (1988). *The C Programming Language*. Prentice-Hall, Upper Saddle River, NJ, USA.
- [Kernighan and Ritchie, 1994] Kernighan, B. W. and Ritchie, D. M. (1994). The M4 Macro Processor. In *4.BSD Programmer's Supplementary Documents*. O'Reilly, Sebastopol, CA, USA.
- [Kernighan, 1975] Kernighan, K. W. (1975). RATFOR – A Preprocessor for a Rational Fortran. *Software Practice and Experience*, 5:395–406.

- [Kerres, 1998] Kerres, M., editor (1998). *Multimediale und telemediale Lernumgebungen: Konzeption und Entwicklung*. Oldenbourg Verlag, München, Germany.
- [Kevitt, 2000] Kevitt, P. M. (2000). The oscon operating system consultant. *Artificial Intelligence Review*, 14(1-2):89–119.
- [Kirsch, 2003] Kirsch, S. M. (2003). Frames, Scripts and Plans. Technical report, Institut für Kommunikationsforschung und Phonetik, Rheinische Friedrich-Wilhelms-Universität Bonn. Available from: <http://sites.inka.de/moebius/docs/framescripts-ho.pdf> [cited 2007-12-16].
- [Kobsa, 1990] Kobsa, A. (1990). User modeling in dialog systems: potentials and hazards. *AI & Society*, 4(3):214–231. Available from: <http://www.isr.uci.edu/~kobsa/papers/1990-AISoc-kobsa.pdf> [cited 2007-10-05].
- [Kobsa, 1993] Kobsa, A. (1993). Adaptivität und Benutzermodellierung in interaktiven Softwaresystemen. In *Proceedings der 17. Fachtagung für Künstliche Intelligenz*, Informatik Aktuell series, pages 152–166, Heidelberg, Germany. Springer Verlag. Available from: <http://www.isr.uci.edu/~kobsa/papers/1993-DKIT93-kobsa.pdf> [cited 2007-10-05].
- [Kobsa, 1995] Kobsa, A. (1995). Supporting user interfaces for all through user modeling. In *Proceedings of the Sixth International Conference on Human-Computer Interaction*, volume I, pages 155–157. Available from: <http://www.ics.uci.edu/~kobsa/papers/1995-HCI95-kobsa.pdf> [cited 2007-10-05].
- [Kobsa, 1999] Kobsa, A. (1999). Adapting web information to disabled and elderly users. In *Proceedings of WebNet 99- World Conference on the WWW and Internet*, volume 2, pages 32–37, Charlottesville, VA, USA. Association for the Advancement of Computing in Education (AACE). Available from: <http://www.ics.uci.edu/~kobsa/papers/1998-NRHM-kobsa.pdf> [cited 2007-10-05].
- [Kobsa, 2001a] Kobsa, A. (2001a). Generic user modeling systems. *User Modeling and User-Adapted Interaction*, 11(1-2):49–63.
- [Kobsa, 2001b] Kobsa, A. (2001b). Tailoring privacy to users' needs. In *UM '01: Proceedings of the 8th International Conference on User Modeling 2001*, pages 303–313, Heidelberg, Germany. Springer Verlag. Available from: <http://www.ics.uci.edu/~kobsa/papers/2001-UM01-kobsa.pdf> [cited 2007-10-05].
- [Kobsa, 2002] Kobsa, A. (2002). Personalized hypermedia and international privacy. *Communications of the ACM*, 45(8):64–67.
- [Kobsa et al., 2001] Kobsa, A., Koenemann, J., and Pohl, W. (2001). Personalized Hypermedia Presentation Techniques for Improving Online Customer Relationships. *The Knowledge Engineering Review*, 16:111–155.

- [Kobsa and Schreck, 2003] Kobsa, A. and Schreck, J. (2003). Privacy through pseudonymity in user-adaptive systems. *ACM Transactions on Internet Technology (TOIT)*, 3(2):149–183. Available from: <http://www.ics.uci.edu/~kobsa/papers/2003-TOIT-kobsa.pdf> [cited 2007-10-05].
- [Kolovski et al., 2004] Kolovski, V., Jordanov, S., and Galletly, J. (2004). An electronic learning assistant. In *CompSysTech '04: Proceedings of the 5th international conference on Computer systems and technologies*, pages 1–6, New York, NY, USA. ACM.
- [Kolter and Maloof, 2006] Kolter, J. Z. and Maloof, M. A. (2006). Learning to detect and classify malicious executables in the wild. *The Journal of Machine Learning Research*, 7:2721–2744.
- [Kopp and Michl, 2000] Kopp, H. and Michl, W. (2000). *MeiLe - Neue Medien in der Lehre*. Luchterhand Literaturverlag, Köln, Germany.
- [Krause et al., 1993] Krause, J., Mittermaier, E., and Hirschmann, A. (1993). The Intelligent Help System COMFOHELP. *User Modeling and User-Adapted Interaction*, 3(3):249–282.
- [Kuhlen and Laisiepen, 2004] Kuhlen, R. and Laisiepen, K. (2004). *Grundlagen der praktischen Information und Dokumentation*. Saur Verlag, München, Germany, 5th edition.
- [Kuncicky and Wynn, 1998] Kuncicky, D. and Wynn, B. A. (1998). *Short Topics in System Administration #4: Educating and Training System Administrators: A Survey*. USENIX Association, Boston, MA, USA.
- [Kuyper, 1998] Kuyper, M. (1998). *Knowledge Engineering for Usability*. PhD thesis, University of Amsterdam, Roetersstraat 15, 1018 WB Amsterdam, Netherlands.
- [Kölle, 2007] Kölle, R. (2007). *Java lernen in virtuellen Teams*. Verlag Werner Hülsbusch, Boizenburg, Germany.
- [Lave and Wenger, 1991] Lave, J. and Wenger, E. (1991). *Situated learning*. Cambridge University Press, Cambridge, MA, USA.
- [Ledgard, 1971] Ledgard, H. F. (1971). Ten mini-languages: A study of topical issues in programming languages. *ACM Computing Surveys (CSUR)*, 3(3):115–146.
- [Lenat and Guha, 1990] Lenat, D. B. and Guha, R. V. (1990). *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project*. Addison Wesley, Boston, MA, USA.
- [Lenat and Guha, 1991] Lenat, D. B. and Guha, R. V. (1991). The evolution of cycl, the cyc representation language. *SIGART Bull.*, 2(3):84–87.

- [Lepreau, 2006] Lepreau, J. (2006). Emulab: Recent Work, Ongoing Work. In *Proceedings of the DETER Community Meeting*, USC/ISI. Available from: <http://www.cs.utah.edu/flux/testbed-docs/emulab-dev-jan06.pdf> [cited 2007-10-05].
- [Lesk and Schmidt, 1975] Lesk, M. E. and Schmidt, E. (1975). Lex - A Lexical Analyzer Generator. Technical Report Computing Science Technical Report No. 39, Bell Laboratories, Murray Hill, NJ, USA.
- [Lévénéz, 2007] Lévénéz, E. Unix History [online]. (2007) [cited 2007-10-05]. Available from: <http://www.levenez.com/unix/>.
- [LireFire Labs, 2007a] Description Internet Lab [online]. (2007) [cited 2007-10-05]. Available from: [http://www.livefirelabs.com/info/internet\\_lab.htm](http://www.livefirelabs.com/info/internet_lab.htm).
- [LireFire Labs, 2007b] Homepage of LifeFire Labs [online]. (2007) [cited 2007-10-05]. Available from: <http://www.livefirelabs.com/>.
- [LireFire Labs, 2007c] UNIX System Administration course information [online]. (2007) [cited 2007-10-05]. Available from: [http://www.livefirelabs.com/course\\_info/UNIX\\_System\\_Administration.htm](http://www.livefirelabs.com/course_info/UNIX_System_Administration.htm).
- [Lytle et al., 2005] Lytle, D. P., Resendez, V., and August, R. (2005). Security in the residential network. In *Proceedings of the 33rd annual ACM SIGUCCS conference on User services SIGUCCS '05*, pages 197–201.
- [Lütticke and Helbig, 2004] Lütticke, R. and Helbig, H. (2004). Practical courses in distance education supported by an interactive tutoring component. In Benrath, U. and Szücs, A., editors, *3rd EDEN Research Workshop, Bibliotheks- und Informationssystem der Universität Oldenburg (BIS): Supporting the Learner in Distance Education and E-Learning*, pages 441–447. Available from: <http://pi7.fernuni-hagen.de/papers/luett/luett-2004-ed-en-bis.pdf> [cited 2007-10-05].
- [Ma and Nickerson, 2006] Ma, J. and Nickerson, J. V. (2006). Hands-on, simulated, and remote laboratories: A comparative literature review. *ACM Computing Surveys (CSUR)*, 38(3).
- [Madhavapeddy et al., 2007] Madhavapeddy, A., Ho, A., Deegan, T., Scott, D., and Sohan, R. (2007). Melange: creating a "functional" internet. *ACM SIGOPS Operating Systems Review*, 41(3):101–114.
- [Manaris and Pritchard, 1993] Manaris, B. Z. and Pritchard, J. W. (1993). Constructing natural language interface applications to operating systems. In *CSC '93: Proceedings of the 1993 ACM conference on Computer science*, pages 425–432, New York, NY, USA. ACM Press.

- [Manaris et al., 1994] Manaris, B. Z., Pritchard, J. W., and Dominick, W. D. (1994). Developing a natural language interface for the unix operating system. *SIGCHI Bulletin*, 26(2):34–40.
- [Mandl et al., 1994] Mandl, H., Gruber, H., and Renkl, A. (1994). Situiertes Lernen in multimedialen Lernumgebungen. In Issing, L. J. and Klimsa, P., editors, *Information und Lernen mit Multimedia*, pages 167–178. Psychologie Verlags Union, Weinheim, Germany.
- [Mata-Toledo and Reyes-Garcia, 2002] Mata-Toledo, R. A. and Reyes-Garcia, C. A. (2002). A model course for teaching database administration with personal oracle 8i. *Journal of Computing Sciences in Colleges (JCSC)*, 17(3):125–130.
- [Matsumoto, 2001] Matsumoto, Y. (2001). *Ruby In A Nutshell*. O'Reilly, Sebastopol, CA, USA.
- [Matthews et al., 2000] Matthews, M., Pharr, W., Biswas, G., and Neelakandan, H. (2000). Uscsh: An active intelligent assistance system. *Artificial Intelligence Review*, 14(1-2):121–141.
- [Mayer, 2001] Mayer, A. (2001). *Shell-Programmierung in Unix*. Computer und Literatur Verlag, Böblingen, Germany.
- [Mayo and Mitrovic, 2001] Mayo, M. and Mitrovic, A. (2001). Optimising ITS Behaviour with Bayesian Networks and Decision Theory. *International Journal of Artificial Intelligence in Education*, 12:124–153. Available from: [http://ai.ed.inf.ed.ac.uk/abstracts/Vol\\_12/mayo.html](http://ai.ed.inf.ed.ac.uk/abstracts/Vol_12/mayo.html) [cited 2007-12-17].
- [McGill et al., 1978] McGill, R., Tukey, J. W., and Larsen, W. A. (1978). Variations of boxplots. *The American Statistician*, 32(1):12–16.
- [McNab, 2004] McNab, C. (2004). *Network Security Assessment*. O'Reilly, Sebastopol, CA, USA.
- [Medvidovic and Rosenblum, 1997] Medvidovic, N. and Rosenblum, D. S. (1997). Domains of concern in software architectures and architecture description languages. In *Proceedings of the Conference on Domain-Specific Languages*, Santa Barbara, California, USA. Available from: <http://www.usenix.org/publications/library/proceedings/dsl97/medvidovic.html>.
- [Mernik et al., 2005] Mernik, M., Heering, J., and Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Computing Surveys (CSUR)*, 37(4):316–344.
- [Merrill, 1983] Merrill, D. (1983). Component display theory. In [Reigeluth, 1983], pages 279–333.
- [Merrill et al., 1991] Merrill, D., Li, Z., and Jones, M. (1991). Second Generation Instructional Design (ID<sub>2</sub>). *Educational Technology*, 30(1):7–11. Available from: <http://id2.usu.edu/Papers/ID1&ID2.PDF> [cited 2007-10-05].

- [Meunier, 1995] Meunier, R. (1995). The pipes and filters architecture. In *Pattern Languages of Program Design*, pages 427–440. Addison Wesley, Boston, MA, USA.
- [Michaud et al., 2000] Michaud, L. N., McCoy, K. F., and Pennington, C. A. (2000). An intelligent tutoring system for deaf learners of written english. In *Assets '00: Proceedings of the fourth international ACM conference on Assistive technologies*, pages 92–100, New York, NY, USA. ACM Press.
- [Miller et al., 1960] Miller, G. A., Galanger, E., and Pribram, K. H. (1960). *Plans and the structure of behavior*. Holt, Rinehart and Winston, New York, NY, USA.
- [Minsky, 1975] Minsky, M. A. (1975). Framework for representing knowledge. In Winston, P. H., editor, *The Psychology of Computer Vision*, pages 211–277. McGraw Hill, New York, NY, USA.
- [Moodle, 2007] Moodle. A Free, Open Source Course Management System for Online Learning [online]. (2007) [cited 2007-10-05]. Available from: <http://www.moodle.org/>.
- [Morell, 2004] Morell, A. Motion Study of Hammer on Lead [online]. (2004) [cited 2007-10-05]. Available from: [http://www.abelardomorell.net/recentwork/Motion\\_Study\\_Hammer\\_full.jpg](http://www.abelardomorell.net/recentwork/Motion_Study_Hammer_full.jpg).
- [Morris, 1938] Morris, C. W. (1938). Foundations of the theory of signs. In Neurath, O., editor, *International Encyclopedia of Unified Science*. University of Chicago Press, Chicago, MI, USA.
- [Nakatani et al., 1986] Nakatani, L. H., Egan, D. E., Ruedisueli, L. W., Hawley, P. M., and Lewart, D. K. (1986). TNT: A talking tutor 'n' trainer for teaching use of interactive computer systems. In *CHI '86: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 29–34, New York, NY, USA. ACM Press.
- [Narain, 2005] Narain, S. (2005). Network Configuration Management via Model Finding. In *Proceedings of the 19th Large Installation System Administration (LISA) Conference*, San Diego, California, USA.
- [Nast, 2006] Nast, J. (2006). *Idea Mapping*. John Wiley & Sons, Indianapolis, IN, USA.
- [Nathan, 1990] Nathan, M. J. (1990). Empowering the student: prospects for an un-intelligent tutoring system. In *CHI '90: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 407–414, New York, NY, USA. ACM Press.
- [Nessus, 2007] Nessus – the network vulnerability scanner [online]. (2007). Available from: <http://www.nessus.org/>.

- [Nielsen, 1994] Nielsen, J. (1994). *Usability Engineering*. Morgan Kaufman Publishers, San Francisco, CA, USA.
- [Nielsen, 2001] Nielsen, J. (2001). *Designing Web Usability*. Markt+Technik Verlag, München, Germany.
- [Nipkow and von Oheimb, 1998] Nipkow, T. and von Oheimb, D. (1998). JavaJight is type-safe – definitely. In *Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, San Diego, California, USA.
- [Nixsys, 2007] Nixsys. Public Access UNIX System (PAUS) [online]. (2007) [cited 2007-10-05]. Available from: <https://nixsyspaus.org/>.
- [nmap, 2007] Nmap – free security scanner for network exploration & security audits [online]. (2007) [cited 2007-10-05]. Available from: <http://www.insecure.org/nmap/index.html>.
- [Nodenot et al., 2004] Nodenot, T., Marquesuzaá, C., Laforcade, P., and Sallaberry, C. (2004). Model based engineering of learning situations for adaptive web based educational systems. In *WWWAlt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 94–103, New York, NY, USA. ACM Press.
- [Norman, 2007] Norman, D. (2007). The next ui breakthrough: command lines. *interactions*, 14(3):44–45.
- [Norman, 2002] Norman, D. A. (2002). *The Design of Everyday Things*. Basic Books, New York, NY, USA.
- [Nösekabel, 2005] Nösekabel, H. (2005). *Mobile Education*. GITO Verlag, Berlin, Germany.
- [openQRM, 2007] openQRM – The open source systems management platform [online]. (2007) [cited 2007-10-05]. Available from: <http://www.openqrm.org/>.
- [Ossanna and Kernighan, 1976] Ossanna, J. F. and Kernighan, B. W. (1976). Nroff/troff user's manual. Technical Report CSTR #54, AT&T Bell Laboratories, Murray Hill. Available from: <http://cm.bell-labs.com/cm/cs/cstr/54.ps.gz> [cited 2007-10-05].
- [Papert, 1982] Papert, S. (1982). *Mindstorms: Kinder, Computer und neues Lernen*. Birkhäuser Verlag, Basel, Switzerland.
- [Patcha and Park, 2007] Patcha, A. and Park, J.-M. (2007). An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, 51(12):3448–3470.



- [Patil and Kobsa, 2005] Patil, S. and Kobsa, A. (2005). Privacy in collaboration: Managing impression. In *Proceedings of the First International Conference on Online Communities and Social Computing*, Las Vegas, NV, USA. Available from: <http://www.ics.uci.edu/~kobsa/papers/2005-ICOCSC-kobsa.pdf> [cited 2007-10-05].
- [Pawlow, 1972] Pawlow, I. P., editor (1972). *Die bedingten Reflexe*. Kindler Verlag, München, Germany.
- [phpESP, 2007] PHP Easy Survey Package [online]. (2007) [cited 2007-10-05]. Available from: <http://phpesp.sourceforge.net/>.
- [Piaget, 1967] Piaget, J. (1967). *Six psychological studies*. Random House, New York, NY, USA.
- [PLDI, 2007] PLDI, A. S. Programming language design and implementation (pldi) [online]. (2007) [cited 2007-10-05]. Available from: <http://www.acm.org/sigplan/pldi.htm>.
- [PLUS, 2007] PLUS, editor. Large-Scale Knowledge Representation: The PARKA Project [online]. (2007) [cited 2007-12-12]. Available from: <http://www.cs.umd.edu/projects/plus/Parka/>.
- [Poskanzer, 2007] Poskanzer, J. The extended portable bitmap toolkit (pbmplus) [online]. (2007) [cited 2007-10-05]. Available from: <http://www.acme.com/software/pbmplus/>.
- [Postel, 1981] Postel, J. (1981). RFC 793: Transmission Control Protocol. Available from: <ftp://ftp.internic.net/rfc/rfc793.txt> [cited 2007-10-05].
- [Pratt and Zelkowitz, 2001] Pratt, T. W. and Zelkowitz, M. V. (2001). *Programming Languages: Design and Implementation*. Prentice-Hall, Upper Saddle River, NJ, USA.
- [Pressey, 1926] Pressey, S. L. (1926). A simple apparatus which gives tests and scores - and teaches. *School and Society*, 23(586):373–376.
- [Pressey, 1927] Pressey, S. L. (1927). A machine for automatic teaching of drill material. *School and Society*, 25(645):549–552.
- [Pruitt and Grudin, 2003] Pruitt, J. and Grudin, J. (2003). Personas: practice and theory. In *DUX '03: Proceedings of the 2003 conference on Designing for user experiences*, pages 1–15, New York, NY, USA. ACM Press.
- [Prümper and Anft, 2006] Prümper, J. and Anft, M. Fragebogen ISONORM 9241/10 [online]. (2006) [cited 2007-10-05]. Available from: [http://www.ergo-online.de/site.aspx?url=html/software/verfahren\\_zur\\_beurteilung\\_der/fragebogen\\_isonorm\\_online.htm](http://www.ergo-online.de/site.aspx?url=html/software/verfahren_zur_beurteilung_der/fragebogen_isonorm_online.htm).

- [Public Access Networks Corporation, 2007] Public Access Networks Corporation. Panix Shell Services [online]. (2007) [cited 2007-10-05]. Available from: <http://www.panix.com/shell.html>.
- [py, 2007] py – Write parser programs in perl [online]. (2007) [cited 2007-10-05]. Available from: <http://perl.plover.com/py/>.
- [Quilici et al., 1986] Quilici, Dyer, and Flowers (1986). Aqua: An intelligent unix advisor. In *Proceedings of the 7th European Conference on Artificial Intelligence (ECAI), Volume II*, pages 33–38, Brighton, England.
- [Quilici, 2000] Quilici, A. (2000). Using justification patterns to advise novice unix users. *Artificial Intelligence Review*, 14(4-5):403–420.
- [Quillian, 1967] Quillian, M. R. (1967). Word concepts. a theory and simulation of some basic semantic capabilities. *Behavioral Science*, 12:410–430.
- [Quillian, 1988] Quillian, M. R. (1988). Semantic memory. In Minsky, M., editor, *Semantic information processing*, pages 216–270. MIT Press, Cambridge, MA, USA.
- [Ramming, 1997] Ramming, C., editor (1997). *Proceedings of the Conference on Domain-Specific Languages October 15-17*. USENIX Association, Boston, MA, USA.
- [Rawls and Hagen, 1998] Rawls, R. R. and Hagen, M. A. (1998). *Autolisp Programming: Principles and Techniques*. O’Reilly, Sebastopol, CA, USA.
- [Raymond, 2003] Raymond, E. S. (2003). *The Art of UNIX Programming*. Addison Wesley, Boston, MA, USA. Available from: <http://www.faqs.org/docs/artu/> [cited 2007-10-05].
- [Reductive Labs, 2007a] Reductive Labs. Cfengine vs. Puppet [online]. (2007) [cited 2007-10-05]. Available from: <http://reductivelabs.com/trac/puppet/wiki/CfengineVsPuppet>.
- [Reductive Labs, 2007b] Reductive Labs. Puppet [online]. (2007) [cited 2007-10-05]. Available from: <http://puppet.reductivelabs.com/>.
- [Reed, 2007] Reed, D. IP Filter [online]. (2007) [cited 2007-10-05]. Available from: <http://coombs.anu.edu.au/~avalon/>.
- [Reigeluth, 1983] Reigeluth, C., editor (1983). *Instructional-Design Theories and Models*. Lawrence Erlbaum Associates, Publishers, Hillsdale, NJ, USA.
- [Reigeluth and Stein, 1983] Reigeluth, C. and Stein, F. (1983). The elaboration theory of instruction. In [Reigeluth, 1983], pages 335–382.
- [Rice, 2006] Rice, W. (2006). *Moodle E-Learning Course Development*. Packt Publishing Limited, Birmingham, UK.

- [Rich, 1979] Rich, E. (1979). User Modeling via Stereotypes. *Cognitive Psychology*, 3:329–354.
- [Richey, 1986] Richey, R., editor (1986). *The theoretical and conceptual bases of instructional design*. Kogan Page, London, UK.
- [Robberecht, 2007] Robberecht, R. (2007). Interactive nonlinear learning environments. *The Electronic Journal of e-Learning (EJEL)*, 5(1):59–68. Available from: <http://www.ejel.org/Volume-5/v5-i1/Robberecht.pdf> [cited 2008-11-24].
- [Root-Lab, 2007a] Hardwareausstattung Root-Labor [online]. (2007) [cited 2007-10-05]. Available from: <http://www.tu-chemnitz.de/informatik/friz/pool/374/hardware.php>.
- [Root-Lab, 2007b] Nutzungshinweise Root-Labor [online]. (2007) [cited 2007-10-05]. Available from: <http://www.tu-chemnitz.de/informatik/friz/pool/374/nutzung.php>.
- [Rossum and Drake, 2003] Rossum, G. V. and Drake, F. L. (2003). *An Introduction to Python*. Network Theory, Bristol, UK. Available from: <http://www.network-theory.co.uk/python/manual/> [cited 2007-10-05].
- [Rossum and Fred L. Drake, 2003] Rossum, G. V. and Fred L. Drake, J. (2003). *The Python Language Reference*. Network Theory, Bristol, UK. Available from: <http://www.network-theory.co.uk/python/language/> [cited 2007-10-05].
- [Ryan, 1991] Ryan, B. (1991). Dynabook revisited with alan kay. *BYTE Magazine*, 16(2):203–ff.
- [Salus, 1994] Salus, P. H. (1994). *A Quarter Century of UNIX*. Addison Wesley, Boston, MA, USA.
- [Sattari et al., 2007] Sattari, S., Backhaus, W., and Henning, K. (2007). The web-based knowledge map: the combination of practise-oriented and scientific knowledge. In *WBED'07: Proceedings of the sixth conference on IASTED International Conference Web-Based Education*, pages 475–480, Anaheim, CA, USA. ACTA Press.
- [Schank, 1972] Schank, R. C. (1972). Conceptual Dependency: A Theory of Natural Language Understanding. *Cognitive Psychology*, 3(4):pages 532–631.
- [Schank and Abelson, 1975] Schank, R. C. and Abelson, R. P. (1975). Scripts, plans, and knowledge. In *Proceedings of the 4th International Joint Conference on Artificial Intelligence*.

- [Schaumann, 2004] Schaumann, J. (2004). Netbsd/desktop: Scalable workstation solutions. In *EuroBSDCon 2004 Proceedings*, pages 141–159, Karlsruhe, Germany. Available from: <http://www.netbsd.org/~jschauma/netbsd-desktop.pdf> [cited 2007-10-05].
- [Schneier, 2005] Schneier, B. (2005). *Applied Cryptography*. John Wiley & Sons, Indianapolis, IN, USA.
- [Schubert and Schwill, 2004] Schubert, S. and Schwill, A. (2004). *Didaktik der Informatik*. Spektrum Akademischer Verlag, Heidelberg, Germany.
- [Schulmeister, 2002] Schulmeister, R. (2002). *Lernplattformen für das virtuelle Lernen*. Oldenbourg Verlag, München, Germany.
- [Schulmeister, 2007] Schulmeister, R. (2007). *Grundlagen hypermedialer Lernsysteme*. Oldenbourg Verlag, München, Germany, 4. edition.
- [Schuman, 2007] Schuman, L. Perspectives on Instruction: What are the problems and strengths of these theories? [online]. (2007) [cited 2007-10-05]. Available from: <http://edweb.sdsu.edu/courses/edtec540/Perspectives/Perspectives.html>.
- [Seidel and Lipsmeier, 1989] Seidel, C. and Lipsmeier, A. (1989). *Computerunterstütztes Lernen – Entwicklungen, Möglichkeiten, Perspektiven*. Verlag für Angewandte Psychologie, Stuttgart, Germany.
- [Serway and Jewett, 2004] Serway, R. A. and Jewett, J. W. (2004). *Physics for Scientists and Engineers*. Thomson Brooks/Cole, Belmont, CA, USA, 6th edition.
- [Shah and Kumar, 2002] Shah, H. and Kumar, A. N. (2002). A tutoring system for parameter passing in programming languages. In *ITiCSE '02: Proceedings of the 7th annual conference on Innovation and technology in computer science education*, pages 170–174, New York, NY, USA. ACM Press.
- [Shannon and Weaver, 1949] Shannon, C. E. and Weaver, W. (1949). *The mathematical theory of communication*. University of Illinois Press, Urbana, IL, USA.
- [Shneiderman, 2004] Shneiderman, B. (2004). *Designing the User Interface*. Addison Wesley, Boston, MA, USA, 4th edition.
- [Si et al., 2006] Si, N.-K., Weng, J.-F., and Tseng, S.-S. (2006). Building a Frame-Based Interaction and Learning Model for U-Learning. *Lecture Notes in Computer Science*, 4159:796–805.
- [SIGCSE, 2007] Homepage of the ACM Special Interest Group on Computer Science Education [online]. (2007) [cited 2007-10-05]. Available from: <http://www.sigcse.org/>.

- [Skinner, 1947] Skinner, B. F. (1947). 'Superstition' in the pigeon. *Journal of Experimental Psychology*, 38:168–172. Available from: <http://psychclassics.yorku.ca/Skinner/Pigeon/> [cited 2007-10-05].
- [Skinner, 1968] Skinner, B. F. (1968). *The technology of teaching*. B. F. Skinner Foundation, Cambridge, MA, USA.
- [sol.net Network Services, 2007] sol.net Network Services. Solaria Public Access UNIX [online]. (2007) [cited 2007-10-05]. Available from: <http://www.sol.net/~jgreco/solaria/>.
- [SourceForge, 2007] SourceForge. Document E07-04: Project Shell Service [online]. (2007) [cited 2007-10-05]. Available from: [http://sourceforge.net/docman/display\\_doc.php?docid=4297&group\\_id=1#shell](http://sourceforge.net/docman/display_doc.php?docid=4297&group_id=1#shell).
- [Specht and Kobsa, 1999] Specht, M. and Kobsa, A. (1999). Interaction of domain expertise and interface design in adaptive educational hypermedia. In *TUE Computing Science Report 99-07: Proceedings of the Second Workshop on Adaptive Systems and User Modeling on the World Wide Web*, pages 89–93, Eindhoven, Netherlands. Eindhoven University of Technology. Available from: <http://www.wis.win.tue.nl/asum99/specht/specht.html> [cited 2007-10-05].
- [Spinellis, 2001] Spinellis, D. (2001). Notable design patterns for domain-specific languages. *The Journal of Systems and Software*, 56(1):91–99.
- [Spinellis, 2003] Spinellis, D. (2003). *Code Reading*. Addison Wesley, Boston, MA, USA.
- [Spinellis, 2007] Spinellis, D. How to embed citations in diagrams [online]. (2007) [cited 2007-10-05]. Available from: <http://www.spinellis.gr/blog/20070204/index.html>.
- [Spinellis and Gritzalis, 2000] Spinellis, D. and Gritzalis, D. (2000). A domain-specific language of intrusion detection. In *Proceedings of the 1st ACM Workshop on Intrusion Detection Systems*. ACM. Available from: <http://www.spinellis.gr/pubs/conf/2000-CCS-DSLID/html/paper.html>.
- [Spinellis and Guruprasad, 1997] Spinellis, D. and Guruprasad, V. (1997). Lightweight languages as software engineering tools. In *Proceedings of the Conference on Domain-Specific Languages*, Santa Barbara, California, USA. Available from: <http://www.usenix.org/publications/library/proceedings/dsl97/spinellis.html>.
- [Staab and Studer, 2004] Staab, S. and Studer, R. (2004). *Handbook on ontologies*. Springer Verlag, Heidelberg, Germany.
- [Stevens, 1992] Stevens, R. W. (1992). *Advanced Programming in the Unix Environment*. Addison Wesley, Boston, MA, USA.

- [Stevens, 1994] Stevens, R. W. (1994). *TCP/IP Illustrated, Volume 1: The Protocols*. Addison Wesley, Boston, MA, USA.
- [Stroustrup, 1994] Stroustrup, B. (1994). *The Design and Evolution of C++*. Addison Wesley, Boston, MA, USA.
- [Su et al., 2007] Su, Y.-Y., Attariyan, M., and Flinn, J. (2007). Autobash: improving configuration management with operating system causality analysis. In *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 237–250, New York, NY, USA. ACM.
- [Suebnuakarn and Haddawy, 2004] Suebnuakarn, S. and Haddawy, P. (2004). A collaborative intelligent tutoring system for medical problem-based learning. In *IUI '04: Proceedings of the 9th international conference on Intelligent user interface*, pages 14–21, New York, NY, USA. ACM Press.
- [Sun Microsystems, 2007] Sun Microsystems. BigAdmin: Solaris Containers (Zones) [online]. (2007) [cited 2007-10-05]. Available from: <http://www.sun.com/bigadmin/content/zones/>.
- [Super Dimension Fortress, 2007] Super Dimension Fortress. SDF Public Access UNIX System - Free Shell Account and Shell Access [online]. (2007) [cited 2007-10-05]. Available from: <http://sdf.lonestar.org/>.
- [Taylor and Siemer, 1996] Taylor, S. J. E. and Siemer, J. (1996). Enhancing simulation education with intelligent tutoring systems. In *WSC '96: Proceedings of the 28th conference on Winter simulation*, pages 675–680, New York, NY, USA. ACM Press.
- [Teltzrow and Kobsa, 2004a] Teltzrow, M. and Kobsa, A. (2004a). Communication of privacy and personalization in e-business. In *Proceedings of the 1st Workshop WHOLES: A Multiple View of Individual Privacy in a Networked World*, Stockholm, Sweden. Available from: [http://www.sics.se/privacy/wholes2004/papers/teltzrow\\_kobsa.pdf](http://www.sics.se/privacy/wholes2004/papers/teltzrow_kobsa.pdf) [cited 2007-10-05].
- [Teltzrow and Kobsa, 2004b] Teltzrow, M. and Kobsa, A. (2004b). Impacts of user privacy preferences on personalized systems: a comparative study. In *Designing personalized user experiences in eCommerce*, pages 315–332. Kluwer Academic Publishers, Norwell, MA, USA. Available from: <http://www.ics.uci.edu/~kobsa/papers/2004-PersUXinECom-kobsa.pdf> [cited 2007-10-05].
- [The FreeBSD Documentation Project, 2007] The FreeBSD Documentation Project. FreeBSD Handbook [online]. (2007) [cited 2007-10-05]. Available from: [http://www.freebsd.org/doc/en\\_US.ISO8859-1/books/handbook/](http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/).
- [The GNOME Project, 2007] The GNOME Project, editor. ATK - Accessibility Toolkit [online]. (2007) [cited 2007-10-05]. Available from: <http://developer.gnome.org/doc/API/2.0/atk/index.html>.

- [The KDE Project, 2007] The KDE Project, editor. KDE Accessibility Project [online]. (2007) [cited 2007-10-05]. Available from: <http://accessibility.kde.org/>.
- [The Linux Foundation, 2007] The Linux Foundation. Lab Activities [online]. (2007) [cited 2007-10-05]. Available from: [http://old.linux-foundation.org/lab\\_activities/](http://old.linux-foundation.org/lab_activities/).
- [The NetBSD Foundation, 2007] The NetBSD Foundation. Diskless netbsd how-to [online]. (2007) [cited 2007-10-05]. Available from: <http://www.NetBSD.org/Documentation/network/netboot/>.
- [The Open Group, 2004] The Open Group (2004). *Single Unix Specification*. American National Standards Institute, 1430 Broadway, New York, NY 10018, USA. IEEE Std 1003.1, 2004 Edition. Available from: <http://www.opengroup.org/onlinepubs/007904975/toc.htm> [cited 2007-10-05].
- [The PHP Project, 2007] Hojtsy, G., editor. PHP Manual [online]. (2007) [cited 2007-10-05]. Available from: <http://www.PHP.net/>.
- [Thomas, 2000] Thomas, S. B. (2000). College Students and Disability Law. *The Journal of Special Education*, 33(4):248–257. Available from: [http://www.ldonline.org/ld\\_indepth/legal\\_legislative/college\\_students\\_and\\_dis\\_law.html](http://www.ldonline.org/ld_indepth/legal_legislative/college_students_and_dis_law.html) [cited 2007-10-05].
- [Thorndike, 1911] Thorndike, E. L. (1911). *Animal Intelligence*. MacMillan Publishing, New York, NY, USA. Available from: <http://psychclassics.yorku.ca/Thorndike/Animal/> [cited 2007-10-05].
- [Trolltech, 2007] Trolltech. Cross-Platform Accessibility Support in Qt 4 [online]. (2007) [cited 2007-10-05]. Available from: <http://doc.trolltech.com/4.0/qt4-accessibility.html>.
- [Trček, 2005] Trček, D. (2005). *Managing Informatino Systems Security and Privacy*. Springer Verlag, Heidelberg, Germany.
- [Tucek et al., 2007] Tucek, J., Lu, S., Huang, C., Xanthos, S., Zhou, Y., Newsome, J., Brumley, D., and Song, D. (2007). Sweeper: a lightweight end-to-end system for defending against fast worms. *ACM SIGOPS Operating Systems Review*, 41(3):115–128.
- [Tukey, 1977] Tukey, J. W. (1977). *Exploratory data anlysis*. Addison Wesley, Boston, MA, USA.
- [Tulodziecki, 2000] Tulodziecki, G. (2000). Computerunterstütztes Lernen aus mediendidaktischer Sicht. In Kammerl, R., editor, *Computerunterstütztes Lernen*, pages 53–72. Oldenbourg Verlag, München, Germany.

- [Tyler and Treu, 1989] Tyler, S. W. and Treu, S. (1989). An interface architecture to provide adaptive task-specific context for the user. *International Journal of Man-Machine Studies*, 30(3):303–327.
- [University of Cypria, Department of Computer Science, 2007a] University of Cypria, Department of Computer Science. *Εργαστήριο UNIX με SUN Solaris* (Unix and Solaris lab) [online]. (2007) [cited 2007-10-05]. Available from: <http://www5.cs.ucy.ac.cy/Computing/en/Labs/solaris.html>.
- [University of Cypria, Department of Computer Science, 2007b] University of Cypria, Department of Computer Science. New Users Guide to Computing Systems [online]. (2007) [cited 2007-10-05]. Available from: [http://www5.cs.ucy.ac.cy/Computing/en/User\\_Guides/newuserguide.html](http://www5.cs.ucy.ac.cy/Computing/en/User_Guides/newuserguide.html).
- [UsabilityNet, 2007] UsabilityNet, editor. Questionnaire resources [online]. (2007) [cited 2007-10-05]. Available from: [http://www.usabilitynet.org/tools/r\\_questionnaire.htm](http://www.usabilitynet.org/tools/r_questionnaire.htm).
- [User Mode Linux, 2007] User Mode Linux. Home Page [online]. (2007) [cited 2007-10-05]. Available from: <http://user-mode-linux.sourceforge.net/>.
- [VDI-Gesellschaft Entwicklung Konstruktion Vertrieb, 1990] VDI-Gesellschaft Entwicklung Konstruktion Vertrieb, editor (1990). *Software-Ergonomie in der Bürokommunikation*. Beuth Verlag, Berlin, Germany.
- [Versteegen, 2001] Versteegen, G. (2001). *Das V-Modell in der Praxis*. dPunkt Verlag, Heidelberg, Germany.
- [Virtuelle Hochschule Baden-Württemberg, 2007] Virtuelle Hochschule Baden-Württemberg. Verbund Virtuelles Labor [online]. (2007) [cited 2007-10-05]. Available from: <http://www.vvl.de/VVL/index.html>.
- [Virtuelle Hochschule Bayern, 2001] Virtuelle Hochschule Bayern (2001). Verordnung über die Virtuelle Hochschule Bayern. *Hochschulrecht in Bayern*, 1180.
- [Vollrath and Jenkins, 2004] Vollrath, A. and Jenkins, S. (2004). Using virtual machines for teaching system administration. *Journal of Computing Sciences in Colleges (JCSC)*, 20(2):287–292.
- [W3C, 2004a] W3C, editor. OWL Web Ontology Language – Guide [online]. (2004) [cited 2007-12-11]. Available from: <http://www.w3.org/TR/owl-guide/>.
- [W3C, 2004b] W3C, editor. Resource Description Framework (RDF) [online]. (2004) [cited 2007-12-11]. Available from: <http://www.w3.org/RDF/>.
- [Wahlster et al., 1988] Wahlster, W., Hecking, M., and Kemke, C. (1988). SC: Ein intelligentes Hilfesystem für SINIX. In Gollan, B., Paul,



- W. J., and Schmitt, A., editors, *Innovative Informationsinfrastrukturen. Informatik-Fachberichte 184*. Springer Verlag, Heidelberg, Germany. Available from: [http://www.dfki.de/~wahlster/Publications/SC\\_Ein\\_intelligentes\\_Hilfesystem\\_fuer\\_SINIX.pdf](http://www.dfki.de/~wahlster/Publications/SC_Ein_intelligentes_Hilfesystem_fuer_SINIX.pdf) [cited 2007-10-05].
- [Wall et al., 2000] Wall, L., Christiansen, T., and Orwant, J. (2000). *Programming Perl*. O'Reilly, Sebastopol, CA, USA.
- [Wall et al., 1996] Wall, L., Christiansen, T., and Schwartz, R. L. (1996). *Programming Perl*. O'Reilly, Sebastopol, CA, USA.
- [Watson, 1913] Watson, J. B. (1913). Psychology as the Behaviourist Views it. *Psychological Review*, 20:158–177. Available from: <http://psychclassics.yorku.ca/Watson/views.htm> [cited 2007-10-05].
- [Weidenmann, 1993] Weidenmann, B. (1993). *Pädagogische Psychologie*. Psychologie Verlags Union, Weinheim, Germany.
- [Weise et al., 1994] Weise, D., Garfinkel, S., and Strassmann, S., editors (1994). *The UNIX Hater's Handbook*. IDG Books, Boston, MA, USA. Available from: <http://research.microsoft.com/~daniel/unix-haters.html> [cited 2007-10-27].
- [Wenger, 1987] Wenger, E. (1987). *Artificial Intelligence and Tutoring systems – Computational and Cognitive Approaches to the Communication of Knowledge*. Morgan Kaufman Publishers, San Francisco, CA, USA.
- [Wexelblat, 1976] Wexelblat, R. L. (1976). Maxims for malfeasant designers, or how to design languages to make programming as difficult as possible. In *Proceedings of the 2nd International Conference on Software Engineering*, pages 331–336, San Francisco, CA, USA. IEEE Computer Society Press.
- [Wiener, 1948] Wiener, N. (1948). *Cybernetics, or control and communication in the animal and machine*. MIT Press, Cambridge, MA, USA.
- [Wiggins, 1989] Wiggins, G. (1989). A True Test: Toward More Authentic and Equitable Assessment. *Delta Phi Kappan*, 70(9):7–11.
- [Wikipedia, 2007] Wikipedia - the free encyclopedia [online]. (2007) [cited 2007-10-05]. Available from: <http://www.wikipedia.org/>.
- [Wilensky et al., 1984] Wilensky, R., Arens, Y., and Chin, D. (1984). Talking to unix in english: an overview of uc. *Communications of the ACM*, 27(6):574–593.
- [Wilensky et al., 1988] Wilensky, R., Chin, D. N., Luria, M., Martin, J., Mayfield, J., and Wu, D. (1988). The berkeley unix consultant project. *Computer Linguistics*, 14(4):35–84.

- [Wirth, 1974] Wirth, N. (1974). On the design of programming languages. In *Proceedings of IFIP Congress 74*, pages 23–30, Stockholm, Sweden.
- [Witschital, 1990] Witschital, P. (1990). *Intelligente Tutorielle Systeme in der Programmierausbildung*. PhD thesis, Technische Universität Braunschweig.
- [World Wide Web Consortium, 2007] World Wide Web Consortium, editor. Web Accessibility Initiative (WAI) [online]. (2007) [cited 2007-10-05]. Available from: <http://www.w3.org/WAI/>.
- [Yacef, 2004] Yacef, K. (2004). Making large class teaching more adaptive with the logic-ita. In *CRPIT '04: Proceedings of the sixth conference on Australian computing education*, pages 343–347, Darlinghurst, Australia, Australia. Australian Computer Society.
- [Yang, 2001] Yang, T. A. (2001). Computer security and impact on computer science education. In *CCSC '01: Proceedings of the sixth annual CCSC northeastern conference on The journal of computing in small colleges*, pages 233–246, Shelbyville, IN, USA. Consortium for Computing Sciences in Colleges.
- [Yin et al., 2000] Yin, J., Miller, M. S., Ioerger, T. R., Yen, J., and Volz, R. A. (2000). A knowledge-based approach for designing intelligent team training systems. In *AGENTS '00: Proceedings of the fourth international conference on Autonomous agents*, pages 427–434, New York, NY, USA. ACM Press.
- [Zhang et al., 2007] Zhang, Q., Reeves, D. S., Ning, P., and Iyer, S. P. (2007). Analyzing network traffic to detect self-decrypting exploit code. In *ASIACCS '07: Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 4–12, New York, NY, USA. ACM Press.
- [Zheng et al., 2007] Zheng, W., Bianchini, R., and Nguyen, T. D. (2007). Automatic configuration of internet services. *ACM SIGOPS Operating Systems Review*, 41(3):219–229.
- [Zimmermann, 2003] Zimmermann, S. (2003). Webbasiertes User-Management des Virtuellen Unix Labors. Technical report, Fachhochschule Regensburg, Computer Science Department.
- [Zoulas, 2007] Zoulas, C. (2007). Private email communication as of 2007-08-16 (Message ID: 20070816081952.F162156407@rebar.astron.com).

# Appendix A

## Example exercise components

### A.1 Exercise texts for users

The exercise texts displayed in this section are the plain text given to the user for practicing. They were the same for step I and II of the Virtual Unix Lab, and were rendered from HTML into plain text using “lynx -dump”.

#### A.1.1 Network Information System (NIS) exercise

The following text displays the NIS exercise's text:

```
Übung: NIS Master und Client Setup

In dieser Übung soll auf den beiden vulab-Rechner der Network
Information Service (NIS) installiert werden. Dabei wird auf dem
Rechner "vulab1" der NIS-Master, auf dem Rechner "vulab2" der
NIS-Client installiert.

1. Master (Solaris): vulab1

* Stellen Sie sicher dass die nötigen Pakete (SUNWypr, SUNWypu,
SUNWsprot, ...) installiert sind.
* Setzen Sie den NIS-Domänenname auf "vulab" (/etc/defaultdomain &
domainname(1))
* Setzen Sie den Rechner mit "ypinit -m" als NIS Master auf
* Sorgen Sie dafür dass die nötigen Serverprozesse (ypbind, ypserv,
...) beim booten gestartet werden.
* Starten Sie die Serverdienste!
* Welcher NIS-Server wird verwendet?
* Welche Datei wird für die Gruppen-Daten verwendet?
* Welche Datei wird für die Passwort-Daten verwendet?
* Überprüfen Sie ob Gruppen- und Passwort-Informationen über NIS
abgefragt werden können.
* Vergleichen Sie den Passwort-Eintrag des Benutzers "vulab" im NIS
und in den /etc-Dateien. Was stellen Sie fest?
* Sorgen Sie dafür, dass die Passwort-Informationen künftig in der
Datei /var/yp/passwd gehalten werden. Die existierenden Logins
```

- sollen dabei nicht übernommen werden.
- \* Legen Sie im NIS eine Kennung "ypuser" mit eindeutiger UID, Home-Verzeichnis "/usr/homes/ypuser", Korn-Shell als Login-Shell, und Passwort "ypuser" an.
  - \* Stellen Sie sicher dass der User "ypuser" via finger(1) sichtbar ist
  - \* Stellen Sie sicher dass sich der User "ypuser" via telnet, ssh und ftp einloggen kann!
  - \* Stellen Sie sicher, dass der User "ypuser" sein Passwort mit yppasswd(1) ändern kann.

## 2. Client (NetBSD): vulab2

- \* Setzen Sie den Domainnamen auf den selben Namen wie beim NIS-Master oben.
- \* Ist das aufsetzen des Clients mit "ypinit -c" nötig? Ist es sinnvoll? Warum (nicht)?
- \* Stellen Sie sicher dass die nötigen Dienste (ypbind, ...) beim booten gestartet werden.
- \* Starten Sie die Dienste!
- \* Welcher NIS-Server wird verwendet?
- \* Stellen Sie sicher dass die NIS Maps (group, hosts, ...) abgerufen werden können
- \* Stellen Sie sicher dass die NIS-Benutzer mit finger(1) abgefragt werden können
- \* Stellen Sie sicher dass sich der oben angelegte Benutzer "ypuser" auf dem Client einloggen kann. Erstellen Sie das Home-Verzeichnis dazu vorerst manuell.
- \* Betrachten Sie das Passwort-Feld der Passwort-Datei des Users "ypuser" auf dem NIS Master!.
- \* Ändern Sie das Passwort von "ypuser" vom Client aus im NIS auf ``mynlspw``.
- \* Betrachten Sie das Passwort-Feld der Passwort-Datei des Users "ypuser" auf dem NIS Master erneut. Was stellen Sie fest?

## 3. Diverses

- \* Setzen Sie den "Full Name" des Benutzers "ypuser" auf "NIS Testbenutzer". Verifizieren Sie das Ergebnis mit finger(1). Welche Methoden zum setzen existieren auf dem NIS Master? Welche auf dem NIS Client?
- \* Legen Sie eine NIS-Gruppe "benutzer" an, und machen Sie diese zur (primären) Gruppe des Benutzers "ypuser". Welche Group-ID wählen Sie? Warum?
- \* Legen Sie im Home-Verzeichnis des Benutzers "ypuser" auf dem Master und dem Client eine Datei an, und überprüfen Sie, welcher Gruppe sie gehört.
- \* Sorgen Sie dafür dass der Benutzer "ypuser" auf dem NetBSD-System mittels su(1) root-Rechte erhalten kann. Er muss dazu (unter NetBSD) zusätzlich Mitglied der Gruppe "wheel" sein.
- \* Wie bewerten Sie die Tatsache dass das root-Passwort alleine nicht reicht, sondern auch die richtige Gruppenzugehörigkeit Voraussetzung für einen su(1) auf root ist? Vergleichen Sie zwischen NetBSD, Solaris und Linux!
- \* Der Rechner "tab" (IP-Nummer: 194.95.108.32) soll via NIS bekannt gemacht werden. Tragen Sie den Rechner auf dem Server in die entsprechende Hosts-Datei ein, aktualisieren Sie die NIS-Map und verifizieren Sie das Ergebnis mittels ypcat(1) und ping(1) sowohl auf dem NIS-Master als auch auf dem NIS-Client.

## Hinweise:

- \* Solaris-Pakete für bash und tcsh liegen in /cdrom, Installation mit pkgadd(1).
- \* NetBSD-Pakete für bash und tcsh (und weitere) liegen auf <ftp://ftp.de.netbsd.org/pub/NetBSD/packages/1.6.1/sparc/All>, Installation mit pkg\_add(1).

## A.1.2 Network File System (NFS) exercise

The following text displays the NFS exercises's text:

### Übung: NFS Server und Client Setup

In dieser Übung soll auf den beiden vulab-Rechner das Network File System (NFS) installiert werden. Dabei wird auf dem Rechner "vulab1" der NFS-Server, auf dem Rechner "vulab2" der NFS-Client installiert.

#### 1. Server (Solaris): vulab1

Das Dateisystem /usr/homes soll für den zweiten Rechner 'vulab2' per NFS exportiert werden:

- \* Sichern Sie die Datei, in der bisher die NFS-Exports notiert sind
- \* Das Verzeichnis /usr/homes soll für den Rechner "vulab2" freigegeben werden. Tragen Sie dies in die richtige Datei ein.
- \* Laufen die nötigen Serverprozesse? Starten Sie sie ggf. mit Hilfe der passenden Start-Scripten aus /etc/\*.d.
- \* Sorgen Sie dafür dass die Datei (neu) eingelesen wird
- \* Überprüfen Sie mit 'showmount -e' ob die Freigabe besteht!

#### 2. Client (NetBSD): vulab2

Das Verzeichnis /usr/homes soll vom NFS-Server (vulab1) auf /usr/homes gemountet werden:

- \* Existiert der Mountpoint /usr/homes auf dem Client?
- \* Sind Daten im Mountpoint enthalten?
- \* Überprüfen Sie mit 'showmount -e' die NFS-Freigaben des NFS-Servers 'vulab1' (10.0.0.1)
- \* Untersuchen Sie die System-Defaults in /etc/defaults/rc.conf und tragen Sie für NFS nötige Abweichungen in die Datei /etc/rc.conf ein. Achten Sie auf rpc.lockd(8) und rpc.statd(8)!
- \* Starten Sie alle nötigen Hintergrundprozesse.
- \* Überprüfen Sie, ob das Verzeichnis /usr/homes von vulab1 testweise auf /mnt gemountet werden kann. Unmounten Sie es anschliessend wieder!
- \* Sorgen Sie dafür daß das Verzeichnis /usr/homes vom NFS-Server "vulab1" beim Systemstart auf /usr/homes gemountet wird, tragen Sie dies in die passenden Konfigurationsdatei ein
- \* Mounten Sie alle noch nicht gemounteten NFS-Verzeichnisse!
- \* Überprüfen Sie mit df(1) und mount(8) daß das Verzeichnis gemountet ist!

#### 3. Zugriffsrechte

##### 3.1 Rechnerbasiert

- \* Legen Sie als root auf dem NIS-Client ein Verzeichnis /usr/homes/nfsuser an! Wie reagiert das System, und warum?
- \* Lesen Sie auf dem NFS-Server die Manpage zu dfstab(4) und den darin unter "SEE ALSO" verwiesenen Befehlen (etc.), und sorgen Sie dafür, daß Sie als root auf dem NFS-Client vollen Zugriff habe
- \* Machen Sie die nötige Änderung in /etc/dfs/dfstab.
- \* Lesen Sie die Datei neu ein!
- \* Welche Sicherheitsimplikationen hat der eben vorgenommene Konfigurationsschritt? Macht er in der Praxis Sinn? Wie kann man ihn umgehen?
- \* Legen Sie das Verzeichnis /usr/homes/nfsuser an!

##### 3.2 Benutzerbasiert

Es soll ein Benutzer "nfsuser" auf beiden Systemen angelegt werden, der auf jedem System lokal vermerkt ist (Login, Passwort etc. in

```

/etc/...), das Home-Verzeichnis /usr/homes/nfsuser soll aber auf
beiden Rechnern mittels NFS verfügbar sein!
* Legen Sie auf vulab1 den User an: ``useradd -d /usr/homes/nfsuser
nfsuser``
* Legen Sie auf vulab2 denselben User an: ``useradd -d
/usr/homes/nfsuser nfsuser``
* Geben Sie dem Benutzer auf beiden Systemen (getrennt) mittels
passwd(1) ein Passwort
* Geben Sie das Verzeichnis /usr/homes/nfsuser mittels chown(1) dem
Benutzer "nfsuser".
* Loggen Sie sich auf beiden Rechner als User "nfsuser" ein und
legen Sie eine Datei "hallo-von-vulab1" bzw. "hallo-von-vulab2"
an.
* Welches Problem besteht?
* Geben Sie auf beiden Rechnern dem Benutzer "nfsuser" die User-ID
2000, stellen Sie sicher dass das Home-Verzeichnis (inkl. Inhalt)
auch dem User gehört, und legen Sie die beiden Dateien erneut an.

```

Hinweise:

- \* Solaris-Pakete für bash und tcsh liegen in /cdrom, Installation mit pkgadd(1).
- \* NetBSD-Pakete für bash und tcsh (und weitere) liegen auf ftp://ftp.de.netbsd.org/pub/NetBSD/packages/1.6.1/sparc/All, Installation mit pkg\_add(1).

## A.2 Exercises including text and check data

The exercise texts displayed in this section are from step II of the Virtual Unix Lab. They contain the exercise text as well as data for the checks to be run.

### A.2.1 Network Information System (NIS) exercise

```

<!-- DB updated by feyrer on Sun Feb 22 23:53:01 MET 2004 from nis.php -->
<!-- id: nis.php,v 1.23 2004/06/03 10:27:12 feyrer Exp -->
<?php auswertung_ueberschrift(); ?>
<!-- ----->
<h1> NIS Master und Client Setup</h1>
In dieser Übung soll auf den beiden vulab-Rechner der Network
Information Service (NIS) installiert werden. Dabei wird auf dem
Rechner "vulab1" der NIS-Master, auf dem Rechner "vulab2" der
NIS-Client installiert.
<p>
<h2>1. Master (Solaris): vulab1</h2>
<ul>
<li> Stellen Sie sicher dass die nötigen Pakete (SUNWypr, SUNWypu,
SUNNSprot, ...) installiert sind.
<li> Setzen Sie den NIS-Domänenname auf "vulab" (/etc/defaultdomain &
domainname(1))
<?php auswertung_teiluebungen(
    774, // vulab1: check-file-contents FILE=/etc/defaultdomain CONTENT_SHOULD='vulab'
    // Domäne in /etc/defaultdomain gesetzt?
    775 // vulab1: check-program-output PROGRAM=domainname OUTPUT_SHOULD='vulab'
    // Domäne im laufenden System (domainname(1)) gesetzt?
); ?>
<li> Setzen Sie den Rechner mit "ypinit -m" als NIS Master auf
<?php auswertung_teiluebungen(
    776, // vulab1: check-file-exists FILE=/var/yp/Makefile
    // Existiert /var/yp/Makefile?

```

```

777, // vulabl: check-file-exists FILE=/var/yp/binding/vulab/ypservers
//      Existiert /var/yp/binding/vulab/ypservers?

778 // vulabl: check-file-exists FILE=/var/yp/passwd.time
//      Existiert /var/yp/passwd.time?

); ?>

<li> Sorgen Sie dafür dass die nötigen Serverprozesse (ypbind, ypserv,
... ) beim Booten gestartet werden.
<li> Starten Sie die Serverdienste!
<li> Welcher NIS-Server wird verwendet?

<?php auswertung_teilleubungen(
779 // vulabl: check-program-output PROGRAM=ypwhich OUTPUT_SHOULD='vulabl'
//      Gibt ypwhich(1) 'vulabl' zurück?

); ?>

<li> Welche Datei wird für die Gruppen-Daten verwendet?
<li> Welche Datei wird für die Passwort-Daten verwendet?
<li> Überprüfen Sie ob Gruppen- und Passwort-Informationen über NIS
abgefragt werden können.

<?php auswertung_teilleubungen(
780, // vulabl: check-program-output PROGRAM='ypcat passwd | wc -l' OUTPUT_SHOULD='[0]*'
//      Daten in passwd-Map vorhanden?

781, // vulabl: check-program-output PROGRAM='ypcat hosts | wc -l' OUTPUT_SHOULD='[0]*'
//      Daten in host-Map vorhanden?

782 // vulabl: check-program-output PROGRAM='ypcat group | wc -l' OUTPUT_SHOULD='[0]*'
//      Daten in group-Map vorhanden?

); ?>

<li> Vergleichen Sie den Passwort-Eintrag des Benutzers "vulab" im NIS
und in den /etc-Dateien. Was stellen Sie fest?
<li> Sorgen Sie dafür, dass die Passwort-Informationen künftig in der
Datei /var/yp/passwd gehalten werden. Die existierenden Logins
sollen dabei nicht übernommen werden.

<?php auswertung_teilleubungen(
783, // vulabl: check-file-contents FILE=/var/yp/Makefile CONTENT_SHOULD='`PWDIR.*=/var/yp/'
//      PWDIR in /var/yp/Makefile auf /var/yp gesetzt?

784 // vulabl: check-file-exists FILE=/var/yp/passwd
//      Existiert /var/yp/passwd?

); ?>

<li> Legen Sie im NIS eine Kennung "ypuser" mit eindeutiger UID,
Home-Verzeichnis "/usr/homes/ypuser", Korn-Shell als Login-Shell,
und Passwort "ypuser" an.

<?php auswertung_teilleubungen(
785, // vulabl: check-directory-exists DIR=/usr/homes/ypuser
//      Verzeichnis /usr/homes/ypuser existiert?

786, // vulabl: unix-check-user-shell LOGIN=ypuser SHELL_SHOULD="/.*/ksh"
//      Shell von ypuser auf ksh gesetzt?

787, // vulabl: check-program-output PROGRAM='cat /var/yp/passwd | grep ypuser: | wc -l' OUTPUT_SHOULD=1
//      User ypuser in /var/yp/passwd eingetragen?

788 // vulabl: check-program-output PROGRAM='ypcat passwd | grep ypuser: | wc -l' OUTPUT_SHOULD=1
//      User ypuser in passwd NIS Map vorhanden?

); ?>

<li> Stellen Sie sicher dass der User "ypuser" via finger(1) sichtbar
ist

<?php auswertung_teilleubungen(
789, // vulabl: unix-check-user-exists LOGIN=ypuser
//      User existiert (getpwnam(3))?

790, // vulabl: check-file-contents FILE=/etc/nsswitch.conf CONTENT_SHOULD='passwd:.nis'
//      passwd-Information wird in NIS gesucht (/etc/nsswitch.conf)?

791, // vulabl: check-file-contents FILE=/etc/nsswitch.conf CONTENT_SHOULD='group:.nis'
//      group-Information wird in NIS gesucht (/etc/nsswitch.conf)?

792 // vulabl: check-file-contents FILE=/etc/nsswitch.conf CONTENT_SHOULD='hosts:.nis'
//      hosts-Information wird in NIS gesucht (/etc/nsswitch.conf)?

); ?>

<li> Stellen Sie sicher dass sich der User "ypuser" via telnet, ssh und
ftp einloggen kann!
<li> Stellen Sie sicher, dass der User "ypuser" sein Passwort mit

```

```

        yppasswd(1) ändern kann.
</ul>

<h2>2. Client (NetBSD): vulab2</h2>

<ul>
<li> Setzen Sie den Domainnamen auf den selben Namen wie beim NIS-Master
oben.

    <?php auswertung_teiluebungen(
        793, // vulab2: check-file-contents FILE=/etc/defaultdomain CONTENT_SHOULD='vulab'
            // Domainname in /etc/defaultdomain gesetzt?

        794 // vulab2: check-program-output PROGRAM=domainname OUTPUT_SHOULD='vulab'
            // Domainname im laufenden System gesetzt? (domainname(1))

    ); ?>

<li> Ist das aufsetzen des Clients mit "ypinit -c" nötig? Ist es
sinnvoll? Warum (nicht)?
<li> Stellen Sie sicher dass die nötigen Dienste (ypbind, ...) beim
booten gestartet werden.

    <?php auswertung_teiluebungen(
        795, // vulab2: netbsd-check-rcvar-set RCVAR=rc_configured
            // /etc/rc.conf: rc_configured gesetzt?

        796, // vulab2: netbsd-check-rcvar-set RCVAR=rpcbind
            // /etc/rc.conf: rpcbind gesetzt?

        797 // vulab2: netbsd-check-rcvar-set RCVAR=ypbind
            // /etc/rc.conf: ypbind gesetzt?

    ); ?>

<li> Starten Sie die Dienste!

    <?php auswertung_teiluebungen(
        798, // vulab2: unix-check-process-running PROCESS=rpcbind
            // rpcbind läuft?

        799 // vulab2: unix-check-process-running PROCESS=ypbind
            // ypbind läuft?

    ); ?>

<li> Welcher NIS-Server wird verwendet?

    <?php auswertung_teiluebungen(
        800 // vulab2: check-program-output PROGRAM=ypwhich OUTPUT_SHOULD='vulab1'
            // Wird vulab1 als NIS-Server verwendet? (ypwhich(1))

    ); ?>

<li> Stellen Sie sicher dass die NIS Maps (group, hosts, ...) abgerufen
werden können

    <?php auswertung_teiluebungen(
        801, // vulab2: check-program-output PROGRAM='ypcat passwd | wc -l' OUTPUT_SHOULD='[0]*'
            // Daten in passwd-Map vorhanden?

        802, // vulab2: check-program-output PROGRAM='ypcat hosts | wc -l' OUTPUT_SHOULD='[0]*'
            // Daten in hosts-Map vorhanden?

        803 // vulab2: check-program-output PROGRAM='ypcat group | wc -l' OUTPUT_SHOULD='[0]*'
            // Daten in group-Map vorhanden?

    ); ?>

<li> Stellen Sie sicher dass die NIS-Benutzer mit finger(1) abgefragt
werden können

    <?php auswertung_teiluebungen(
        804 // vulab2: unix-check-user-exists LOGIN=ypuser
            // Existiert Benutzer ypuser?

    ); ?>

<li> Stellen Sie sicher dass sich der oben angelegte Benutzer "ypuser"
auf dem Client einloggen kann. Erstellen Sie das Home-Verzeichnis
dazu vorerst manuell.

    <?php auswertung_teiluebungen(
        805 // vulab2: check-directory-exists DIR=/usr/homes/ypuser
            // Existiert Home-Verzeichnis?

    ); ?>

<li> Betrachten Sie das Passwort-Feld der Passwort-Datei des Users
"ypuser" auf dem NIS Master!.
<li> Ändern Sie das Passwort von "ypuser" vom Client aus im NIS auf
'mynispw'.

```



```

<?php auswertung_teiluebungen(
    806 // vulab2: unix-check-user-password LOGIN=ypuser PASSWD_SHOULD=mynlspw
        // PaSwort richtig gesetzt?

); ?>

<li> Betrachten Sie das Passwort-Feld der Passwort-Datei des Users
"ypuser" auf dem NIS Master erneut. Was stellen Sie fest?
</ul>

<h2>3. Diverses</h2>

<ul>
<li> Setzen Sie den "Full Name" des Benutzers "ypuser" auf "NIS
Testbenutzer". Verifizieren Sie das Ergebnis mit finger(1).
Welche Methoden zum setzen existieren auf dem NIS Master? Welche
auf dem NIS Client?

<?php auswertung_teiluebungen(
    807 // vulab2: unix-check-user-fullname LOGIN=ypuser FULLNAME_SHOULD='NIS Testbenutzer'
        // Fullname richtig gesetzt?

); ?>

<li> Legen Sie eine NIS-Gruppe "benutzer" an, und machen Sie diese zur
(primären) Gruppe des Benutzers "ypuser". Welche Group-ID wählen
Sie? Warum?

<?php auswertung_teiluebungen(
    808, // vulab2: unix-check-user-ingroup LOGIN=ypuser GROUP_SHOULD=benutzer
        // Benutzer 'ypuser' Mitglied der Gruppe 'benutzer'?

    809 // vulab2: check-program-output PROGRAM='ypcat group' OUTPUT_SHOULD='benutzer:'
        // Gruppe 'benutzer' existiert in der group NIS-Map?

); ?>

<li> Legen Sie im Home-Verzeichnis des Benutzers "ypuser" auf dem
Master und dem Client eine Datei an, und überprüfen Sie, welcher
Gruppe sie gehört.
<li> Sorgen Sie dafür dass der Benutzer "ypuser" auf dem NetBSD-System
mittels su(1) root-Rechte erhalten kann. Er muss dazu (unter
NetBSD) zusätzlich Mitglied der Gruppe "wheel" sein.

<?php auswertung_teiluebungen(
    810 // vulab2: check-file-contents FILE=/etc/group CONTENT_SHOULD=''^wheel:.*ypuser''
        // ypuser in wheel-Gruppe in /etc/group?

); ?>

<li> Wie bewerten Sie die Tatsache dass das root-Passwort alleine nicht
reicht, sondern auch die richtige Gruppenzugehörigkeit
Voraussetzung für einen su(1) auf root ist? Vergleichen Sie
zwischen NetBSD, Solaris und Linux!
<li> Der Rechner "tab" (IP-Nummer: 194.95.108.32) soll via NIS bekannt
gemacht werden. Tragen Sie den Rechner auf dem Server in die
entsprechende Hosts-Datei ein, aktualisieren Sie die
NIS-Map und verifizieren Sie das Ergebnis mittels ypcat(1)
und ping(1) sowohl auf dem NIS-Master als auch auf dem NIS-Client.

<?php auswertung_teiluebungen(
    811, // vulab2: check-program-output PROGRAM='ypcat hosts' OUTPUT_SHOULD='194.95.108.65.*tab'
        // Eintrag mit IP-Nummer und Rechnername in hosts NIS-Map?

    812 // vulab2: check-program-output PROGRAM='/sbin/ping -c 1 tab 2>&l ; echo result:$?' OUTPUT_SHOULD='result:0$'
        // 'tab' pingbar?

); ?>

</ul>

<h2>Hinweise:</h2>

<ul>
<li> Solaris-Pakete für bash und tcsh liegen in /cdrom, Installation
mit pkgadd(1).

<?php auswertung_teiluebungen(
    898, // vulab1: solaris-check-installed-pkg PKG=SUNWtcsh
        // tcsh auf Solaris installiert? (pkginfo SUNWtcsh)

    899 // vulab1: solaris-check-installed-pkg PKG=SUNWbash
        // bash auf Solaris installiert? (pkginfo SUNWbash)

); ?>

<li> NetBSD-Pakete für bash und tcsh (und weitere) liegen auf
ftp://ftp.de.netbsd.org/pub/NetBSD/packages/1.6.1/sparc/All,
Installation mit pkg_add(1).

<?php auswertung_teiluebungen(

```

```

900, // vulab2: netbsd-check-installed-pkg PKG=tcsh
//      tcsh auf NetBSD installiert? (pkg_info -e tcsh)

901 // vulab2: netbsd-check-installed-pkg PKG=bash
//      bash auf NetBSD installiert? (pkg_info -e bash)

); ?>

</ul>

<!-- ----->
<?php auswertung_zusammenfassung(); ?>

```

## A.2.2 Network File System (NFS) exercise

```

<!-- DB updated by feyrer on Sun Feb 22 23:54:29 MET 2004 from nfs.php -->
<!-- Id: nfs.php.v 1.15 2004/06/03 10:27:12 feyrer Exp -->
<?php auswertung_ueberschrift(); ?>
<!-- ----->

<h1> NFS Server und Client Setup</h1>

In dieser Übung soll auf den beiden vulab-Rechner das Network File
System (NFS) installiert werden. Dabei wird auf dem Rechner "vulab1"
der NFS-Server, auf dem Rechner "vulab2" der NFS-Client installiert.
<p>

<h2>1. Server (Solaris): vulab1</h2>

Das Dateisystem /usr/homes soll für den zweiten Rechner 'vulab2' per
NFS exportiert werden:
<p>

<ul>
<li> Sichern Sie die Datei, in der bisher die NFS-Exports notiert sind
<li> Das Verzeichnis /usr/homes soll für den Rechner "vulab2"
freigegeben werden. Tragen Sie dies in die richtige Datei ein.

<?php auswertung_teiluebungen(
864 // vulab1: check-file-contents FILE=/etc/dfs/dfstab CONTENT_SHOULD='share.*nfs.*usr/homes'
//      'share nfs /usr/homes' in /etc/dfs/dfstab?

); ?>

<li> Laufen die nötigen Serverprozesse? Starten Sie sie ggf. mit Hilfe
der passenden Start-Scripten aus /etc/*.d.

<?php auswertung_teiluebungen(
865, // vulab1: unix-check-process-running PROCESS=rpcbind
//      Läuft rpcbind?

866, // vulab1: unix-check-process-running PROCESS=mountd
//      Läuft mountd?

867, // vulab1: unix-check-process-running PROCESS=nfsd
//      Läuft nfsd?

868, // vulab1: unix-check-process-running PROCESS=statd
//      Läuft statd?

869, // vulab1: unix-check-process-running PROCESS=lockd
//      Läuft lockd?

870 // vulab1: check-file-exists FILE='/etc/rc3.d/S15nfs.server'
//      NFS-Server wird im Runlevel 3 gestartet?

); ?>

<li> Sorgen Sie dafür dass die Datei (neu) eingelesen wird

<?php auswertung_teiluebungen(
871 // vulab1: check-program-output PROGRAM='share' OUTPUT_SHOULD='/usr/homes'
//      share(1M) listet /usr/homes? (unportabel!)

); ?>

<li> Überprüfen Sie mit 'showmount -e' ob die Freigabe besteht!

<?php auswertung_teiluebungen(
872 // vulab1: check-program-output PROGRAM='showmount -e localhost' OUTPUT_SHOULD='/usr/homes'
//      showmount(1) zeigt /usr/homes?

); ?>

</ul>

<h2>2. Client (NetBSD): vulab2</h2>

```

Das Verzeichnis /usr/homes soll vom NFS-Server (vulabl) auf /usr/homes gemountet werden:

```
<p>
```

```
<ul>
```

- <li> Existiert der Mountpoint /usr/homes auf dem Client?
- <li> Sind Daten im Mountpoint enthalten?
- <li> Überprüfen Sie mit 'showmount -e' die NFS-Freigaben des NFS-Servers 'vulabl' (10.0.0.1)

```
<?php auswertung_teiluebungen(
    873 // vulab2: check-program-output PROGRAM='showmount -e vulabl' OUTPUT_SHOULD='/usr/homes'
        //          showmount(1) zeigt /usr/homes?
); ?>
```

- <li> Untersuchen Sie die System-Defaults in /etc/defaults/rc.conf und tragen Sie für NFS nötige Abweichungen in die Datei /etc/rc.conf ein. Achten Sie auf rpc.lockd(8) und rpc.statd(8)!

```
<?php auswertung_teiluebungen(
    874, // vulab2: netbsd-check-rcvar-set RCVAR=rc_configured
        //          /etc/rc.conf: rc_configured gesetzt?

    875, // vulab2: netbsd-check-rcvar-set RCVAR=lockd
        //          /etc/rc.conf: lockd gesetzt?

    876, // vulab2: netbsd-check-rcvar-set RCVAR=statd
        //          /etc/rc.conf: statd gesetzt?

    877 // vulab2: netbsd-check-rcvar-set RCVAR=nfs_client
        //          /etc/rc.conf: nfs_client gesetzt?
); ?>
```

- <li> Starten Sie alle nötigen Hintergrundprozesse.

```
<?php auswertung_teiluebungen(
    878, // vulab2: unix-check-process-running PROCESS=rpcbind
        //          Läuft rpcbind?

    879, // vulab2: unix-check-process-running PROCESS=rpc.lockd
        //          Läuft rpc.lockd?

    880 // vulab2: unix-check-process-running PROCESS=rpc.statd
        //          Läuft rpc.statd?
); ?>
```

- <li> Überprüfen Sie, ob das Verzeichnis /usr/homes von vulabl testweise auf /mnt gemountet werden kann. Unmounten Sie es anschließend wieder!

```
<?php auswertung_teiluebungen(
    881 // vulab2: unix-check-mount MOUNT_FROM=vulabl:/usr/homes MOUNT_ON=/mnt
        //          Manueller mount erfolgreich?
); ?>
```

- <li> Sorgen Sie dafür daß das Verzeichnis /usr/homes vom NFS-Server "vulabl" beim Systemstart auf /usr/homes gemountet wird, tragen Sie dies in die passenden Konfigurationsdatei ein

```
<?php auswertung_teiluebungen(
    882 // vulab2: check-file-contents FILE=/etc/fstab CONTENT_SHOULD='vulabl:/usr/homes.*usr/homes.*nfs.*rw'
        //          Passender Eintrag in /etc/fstab?
); ?>
```

- <li> Mounten Sie alle noch nicht gemounteten NFS-Verzeichnisse!

- <li> Überprüfen Sie mit df(1) und mount(8) daß das Verzeichnis gemountet ist!

```
<?php auswertung_teiluebungen(
    883, // vulab2: check-program-output PROGRAM='df -k | grep : ' OUTPUT_SHOULD='^vulabl:/usr/homes.*usr/homes$'
        //          Mount ist im df(1) Output sichtbar?

    884 // vulab2: check-program-output PROGRAM='mount | grep nfs' OUTPUT_SHOULD='^vulabl:/usr/homes on /usr/homes'
        //          Mount ist im mount(8) Output sichtbar?
); ?>
```

```
</ul>
```

```
<h2>3. Zugriffsrechte</h2>
```

```
<h3>3.1 Rechnerbasiert</h3>
```

```
<ul>
```

- <li> Legen Sie als root auf dem NFS-Client ein Verzeichnis /usr/homes/nfsuser an! Wie reagiert das System, und warum?

```

<li> Lesen Sie auf dem NFS-Server die Manpage zu dfstab(4) und den darin
unter "SEE ALSO" verwiesenen Befehlen (etc.), und sorgen Sie dafür,
daß Sie als root auf dem NFS-Client vollen Zugriff habe
<li> Machen Sie die nötige Änderung in /etc/dfs/dfstab.

<?php auswertung_teiluebungen(
    885 // vulab1: check-file-contents FILE=/etc/dfs/dfstab CONTENT_SHOULD='root='
        // 'root=' Eintrag in dfstab?

); ?>

<li> Lesen Sie die Datei neu ein!

<?php auswertung_teiluebungen(
    886 // vulab1: check-program-output PROGRAM='share' OUTPUT_SHOULD='/usr/homes.*root='
        // share(1M) exportiert /usr/homes für root zugreifbar?

); ?>

<li> Welche Sicherheitsimplikationen hat der eben vorgenommene
Konfigurationsschritt? Macht er in der Praxis Sinn? Wie kann man
ihn umgehen?
<li> Legen Sie das Verzeichnis /usr/homes/nfsuser an!

<?php auswertung_teiluebungen(
    887 // vulab1: check-directory-exists DIR=/usr/homes/nfsuser
        // Existiert Verzeichnis /usr/homes/nfsuser?

); ?>

</ul>

<h3>3.2 Benutzerbasiert</h3>

Es soll ein Benutzer "nfsuser" auf beiden Systemen angelegt werden,
der auf jedem System lokal vermerkt ist (Login, Passwort etc. in
/etc/...), das Home-Verzeichnis /usr/homes/nfsuser soll aber auf
beiden Rechnern mittels NFS verfügbar sein!

<ul>
<li> Legen Sie auf vulab1 den User an: `useradd -d /usr/homes/nfsuser
nfsuser`

<?php auswertung_teiluebungen(
    888 // vulab1: unix-check-user-exists LOGIN=nfsuser
        // Benutzer 'nfsuser' existiert auf vulab1?

); ?>

<li> Legen Sie auf vulab2 denselben User an: `useradd -d
/usr/homes/nfsuser nfsuser`

<?php auswertung_teiluebungen(
    889 // vulab2: unix-check-user-exists LOGIN=nfsuser
        // Benutzer 'nfsuser' existiert auf vulab2?

); ?>

<li> Geben Sie dem Benutzer auf beiden Systemen (getrennt) mittels
passwd(1) ein Passwort
<li> Geben Sie das Verzeichnis /usr/homes/nfsuser mittels chown(1) dem
Benutzer "nfsuser".

<?php auswertung_teiluebungen(
    890 // vulab1: unix-check-file-owner FILE=/usr/homes/nfsuser OWNER_SHOULD=nfsuser
        // Gehört /usr/homes/nfsuser dem Benutzer 'nfsuser' auf vulab1?

    891 // vulab2: unix-check-file-owner FILE=/usr/homes/nfsuser OWNER_SHOULD=nfsuser
        // Gehört /usr/homes/nfsuser dem Benutzer 'nfsuser' auf vulab2?

); ?>

<li> Loggen Sie sich auf beiden Rechner als User "nfsuser" ein und legen Sie
eine Datei "hallo-von-vulab1" bzw. "hallo-von-vulab2" an.
<li> Welches Problem besteht?
<li> Geben Sie auf beiden Rechnern dem Benutzer "nfsuser" die User-ID
2000, stellen Sie sicher dass das Home-Verzeichnis (inkl. Inhalt)
auch dem User gehört, und legen Sie die beiden Dateien erneut an.

<?php auswertung_teiluebungen(
    892 // vulab1: unix-check-file-owner FILE=/usr/homes/nfsuser/hallo-von-vulab1 OWNER_SHOULD=nfsuser
        // hallo-von-vulab1 gehört nfsuser auf vulab1?

    893 // vulab2: unix-check-file-owner FILE=/usr/homes/nfsuser/hallo-von-vulab1 OWNER_SHOULD=nfsuser
        // hallo-von-vulab1 gehört nfsuser auf vulab2?

    894 // vulab1: unix-check-file-owner FILE=/usr/homes/nfsuser/hallo-von-vulab2 OWNER_SHOULD=nfsuser
        // hallo-von-vulab2 gehört nfsuser auf vulab1?

    895 // vulab2: unix-check-file-owner FILE=/usr/homes/nfsuser/hallo-von-vulab2 OWNER_SHOULD=nfsuser
        // hallo-von-vulab2 gehört nfsuser auf vulab2?

); ?>

```

```

</ul>

<h2>Hinweise:</h2>

<ul>
<li> Solaris-Pakete für bash und tcsh liegen in /cdrom, Installation
mit pkgadd(1).

    <?php auswertung_teiluebungen(
        902, // vulab1: solaris-check-installed-pkg PKG=SUNWtcsh
            //          tcsh auf Solaris installiert? (pkginfo SUNWtcsh)

        903 // vulab1: solaris-check-installed-pkg PKG=SUNWnbash
            //          bash auf Solaris installiert? (pkginfo SUNWnbash)
    ); ?>

<li> NetBSD-Pakete für bash und tcsh (und weitere) liegen auf
ftp://ftp.de.netbsd.org/pub/NetBSD/packages/1.6.1/sparc/All,
Installation mit pkg_add(1).

    <?php auswertung_teiluebungen(
        904, // vulab2: netbsd-check-installed-pkg PKG=tcsh
            //          tcsh auf NetBSD installiert? (pkg_info -e tcsh)

        905 // vulab2: netbsd-check-installed-pkg PKG=bash
            //          bash auf NetBSD installiert? (pkg_info -e bash)
    ); ?>

</ul>

<!-- ----->
<?php auswertung_zusammenfassung(); ?>

```

## A.3 The VUDSL processor: uebung2db

```

#!/usr/pkg/bin/perl

use DBI;
use Getopt::Std;

$checkscript_path="/vulab";          # check-script
##F#$checkscript_path="/home/feyrer/work/vulab/docs/hubertf/code";

getopts('dv');

$debug=1
  if $opt_d;
$verbose=1
  if $opt_v or $opt_d;

$uebung_id = $ARGV[0];
$template = $ARGV[1];
$output = $ARGV[2];

die "Usage: $0 [-dv] uebung_id uebung.php-template neue_uebung.php\n"
  if $uebung_id eq ""
  or $template eq ""
  or $output eq ""
  or $template eq $output;

open(OUTPUT, ">$output")
  or die "Can't write $output: ${!}\n";

$dbh = DBI->connect("dbi:Pg:", "vulab", "", { AutoCommit => 0 })
#   or die "cannot connect to DB";
$dbh = DBI->connect("dbi:Pg:dbname=vulab:host=smaug", "vulab", "vulab", { AutoCommit => 0 })
  or die "cannot connect to DB";

$checks_done = ();
$warnings = 0;

header();
main();
delete_old();

close(OUTPUT);

if ($warnings > 0 and !$debug) {
  print "Transaction rolled back, output file removed due to warnings.\n";
  print "Please fix!\n";
  unlink($output);

  $dbh->rollback;
}

```

```

} else {
    $dbh->commit;
}

$dbh->disconnect();

exit(0);

#####
sub warning {
    print "WARNING: @_\n";
    $warnings++;
}

#####
sub header() {
    local($now);
    chomp($now = `date`);
    print OUTPUT "<!-- DB updated by $ENV{'USER'} on $now from $template -->\n";
}

#####
sub main() {
    open(T, $template) or die "can't read $template: $!\n";
    while(<T>) {
        chomp;
        ($check_id, $komma, $rechner, $script, $parameter) =
            m@`s*{[0-9X?]+}([, ])?\s*/\s*{[a-zA-Z0-9_]*}:\s*{[^ ]*check-[ ]*\s*(.*)}@;

        if ($rechner eq "") {
            print OUTPUT "$_\n";
            if !/Generated by.* on .* from;/ # skip header
                next;
        }

        ### 1. Syntax-Check etc.

        ## Check if script present
        if ( ! -f "$checkscript_path/$script" ) {
            warning("missing check-script '$script'");
            next;
        }

        $interpreter = get_interpreter("$checkscript_path/$script");
        print "$check_id: cat $script | ssh $rechner env $parameter '$interpreter'\n";
        if $debug;

        chomp($bezeichnung = <T>);
        if ($bezeichnung !~ m@`s*/\s*\s*\s*@) {
            warning("no comment for $check_id ($rechner: $script $parameter)");
        }
        $bezeichnung =~ `s,\s*/\s*,,;`
        $bezeichnung =~ `s,\s$,,;`
        print "    bezeichnung=\"$bezeichnung\"\n";
        if $debug;
        print "\n";
        if $debug;

        ## Rechner bekannt?
        $sth = $dbh->prepare("SELECT * .
                            FROM rechner .
                            WHERE bezeichnung='$rechner'");

        $sth->execute();
        while(@row = $sth->fetchrow_array) {
            if ($row[0] eq $rechner) {
                print "    rechner OK: $rechner\n";
                if $debug;
            } else {
                warning("rechnercheck unknown host: $rechner");
            }
        }

        ## Check parameters
        # Get possible parms
        open(P, "$interpreter $checkscript_path/$script listparms |")
        or die "Can't listparms for $script: $!\n";
        while(<P>) {
            @p = split(/\|/);
            $par[$p[0]] = $p[1];
            #print " $p[0]";
        }
        close(P);
        #print "\n";

        # Parse into variables using sh & env
        open(P, "env -i $parameter env |")
        or die "Can't env(1) $parameter";
        while(<P>) {
            chomp();
            ($var, $val) = /{[a-zA-Z0-9_]*}=(.*)/;
            #print "    $var -> $val\n";
        }
    }
}

```



```

        if $debug;
        $sth = $dbh->prepare($sql);
        $sth->execute();
        print "old checks removed from database\n";
        if $verbose;
    }
}

#####
sub update_db() {
    local($check_id, $uebung_id, $script, $bezeichnung, $rechner,
          $parameter) = @_;
    local($sth);

    $parameter = " s\\\/\\\/\\\/g;
    $parameter = " s'\/\\\/\\\/g;
    $bezeichnung = " s\\\/\\\/\\\/g;
    $bezeichnung = " s'\/\\\/\\\/g;

    $sql = "UPDATE uebungs_checks ".
        "SET ".
        "    uebung_id='$uebung_id', ".
        "    script='$script', ".
        "    bezeichnung='$bezeichnung', ".
        "    rechner='$rechner', ".
        "    parameter='$parameter' ".
        "WHERE ".
        "    check_id='$check_id'";
    print "    SQL: $sql\n";
    if $debug;
    $sth = $dbh->prepare($sql);
    $sth->execute();
}

#####
sub insert_into_db() {
    local($uebung_id, $script, $bezeichnung, $rechner,
          $parameter) = @_;
    local($sth);

    $parameter = " s\\\/\\\/\\\/g;
    $parameter = " s'\/\\\/\\\/g;
    $bezeichnung = " s\\\/\\\/\\\/g;
    $bezeichnung = " s'\/\\\/\\\/g;

    # 1. insert new
    $sql = "INSERT INTO uebungs_checks ".
        "    ( uebung_id, script, bezeichnung, ".
        "    rechner, parameter ) ".
        "VALUES ".
        "    ( '$uebung_id', '$script', '$bezeichnung', ".
        "    '$rechner', '$parameter' )";
    print "    SQL: $sql\n";
    if $debug;
    $sth = $dbh->prepare($sql);
    $sth->execute();

    # 2. find $new_check_id
    $sql = "SELECT check_id ".
        "FROM uebungs_checks ".
        "WHERE uebung_id='$uebung_id' ".
        "    AND script='$script' ".
        "    AND bezeichnung='$bezeichnung' ".
        "    AND rechner='$rechner' ".
        "    AND parameter='$parameter'";
    print "    SQL: $sql\n";
    if $debug;
    $sth = $dbh->prepare($sql);
    $sth->execute();

    while(@row = $sth->fetchrow_array) {
        $new_check_id = $row[0];
    }
    print "    new check_id=$new_check_id\n";
    if $debug;

    return $new_check_id;
}

#####
sub get_interpreter() {
    local($file) = @_;
    local($i, $rc);

    die "No such file: $file\n";
    if ( ! -f $file );

    open(F, "$file") or die "can't open $file: $!\n";
    $i = <F>;
    close(F);

    if ($i =~ /perl/) {
        #src="perl || /root/vulab/perl";
        $rc="perl";
    }
}

```



```

} elsif ($i =~ /^[^]sh/) {
    $rc = "sh";
}
return $rc;
}

```

## A.4 Complete lists of checks used in exercises

This section provides complete lists of checks that are performed for both the Network Information System (NIS) and the Network File System (NFS) exercises. The data is retrieved from the Virtual Unix Lab's database, and the SQL queries and their results are shown.

### A.4.1 Network Information System (NIS) exercise

This section lists all the checks that are performed by the NIS exercise as stated in the Virtual Unix Lab's database.

```

vulab=> select check_id,bezeichnung from uebungs_checks where uebung_id='nis';
check_id | bezeichnung
-----|-----
775 | Domäne im laufenden System (domainname(1)) gesetzt?
776 | Existiert /var/yp/Makefile?
777 | Existiert /var/yp/binding/vulab/ypservers?
778 | Existiert /var/yp/passwd.time?
779 | Gibt ypwhich(1) 'vulabl' zurück?
780 | Daten in passwd-Map vorhanden?
781 | Daten in host-Map vorhanden?
782 | Daten in group-Map vorhanden?
784 | Existiert /var/yp/passwd?
785 | Verzeichnis /usr/homes/ypuser existiert?
786 | Shell von ypuser auf ksh gesetzt?
787 | User ypuser in /var/yp/passwd eingetragen?
788 | User ypuser in passwd NIS Map vorhanden?
789 | User existiert (getpwnam(3))?
790 | passwd-Information wird in NIS gesucht (/etc/nsswitch.conf)?
791 | group-Information wird in NIS gesucht (/etc/nsswitch.conf)?
792 | hosts-Information wird in NIS gesucht (/etc/nsswitch.conf)?
793 | Domainname in /etc/defaultdomain gesetzt?
795 | /etc/rc.conf: rc_configured gesetzt?
796 | /etc/rc.conf: rpcbind gesetzt?
797 | /etc/rc.conf: ypbind gesetzt?
798 | rpcbind läuft?
799 | ypbind läuft?
800 | Wird vulabl als NIS-Server verwendet? (ypwhich(1))
801 | Daten in passwd-Map vorhanden?
802 | Daten in hosts-Map vorhanden?
803 | Daten in group-Map vorhanden?
804 | Existiert Benutzer ypuser?
805 | Existiert Home-Verzeichnis?
806 | Paßwort richtig gesetzt?
807 | Fullname richtig gesetzt?
808 | Benutzer 'ypuser' Mitglied der Gruppe 'benutzer'?
809 | Gruppe 'benutzer' existiert in der group NIS-Map?
810 | ypuser in wheel-Gruppe in /etc/group?

```

```

811 | Eintrag mit IP-Nummer und Rechnername in hosts NIS-Map?
774 | Domäne in /etc/defaultdomain gesetzt?
794 | Domainname im laufenden System gesetzt? (domainname(1))
783 | PWDIR in /var/yp/Makefile auf /var/yp gesetzt?
812 | 'tab' pingbar?
898 | tcsh auf Solaris installiert? (pkginfo SUNWtcsh)
899 | bash auf Solaris installiert? (pkginfo SUNWbash)
900 | tcsh auf NetBSD installiert? (pkg_info -e tcsh)
901 | bash auf NetBSD installiert? (pkg_info -e bash)
(43 rows)

```

## A.4.2 Network File System (NFS) exercise

This section describes the checks that are performed for the NFS exercise.

```

vulab=> select check_id,bezeichnung from uebungs_checks where uebung_id='nfs';
check_id | bezeichnung
-----|-----
864 | 'share nfs /usr/homes' in /etc/dfs/dfstab?
865 | Läuft rpcbind?
866 | Läuft mountd?
867 | Läuft nfsd?
868 | Läuft statd?
869 | Läuft lockd?
870 | NFS-Server wird im Runlevel 3 gestartet?
874 | /etc/rc.conf: rc_configured gesetzt?
875 | /etc/rc.conf: lockd gesetzt?
876 | /etc/rc.conf: statd gesetzt?
877 | /etc/rc.conf: nfs_client gesetzt?
878 | Läuft rpcbind?
879 | Läuft rpc.lockd?
880 | Läuft rpc.statd?
881 | Manueller mount erfolgreich?
882 | Passender Eintrag in /etc/fstab?
883 | Mount ist im df(1) Output sichtbar?
884 | Mount ist im mount(8) Output sichtbar?
885 | 'root=' Eintrag in dfstab?
887 | Existiert Verzeichnis /usr/homes/nfsuser?
888 | Benutzer 'nfsuser' existiert auf vulab1?
889 | Benutzer 'nfsuser' existiert auf vulab2?
890 | Gehört /usr/homes/nfsuser dem Benutzer 'nfsuser' auf vulab1?
891 | Gehört /usr/homes/nfsuser dem Benutzer 'nfsuser' auf vulab2?
892 | hallo-von-vulab1 gehört nfsuser auf vulab1?
893 | hallo-von-vulab1 gehört nfsuser auf vulab2?
894 | hallo-von-vulab2 gehört nfsuser auf vulab1?
895 | hallo-von-vulab2 gehört nfsuser auf vulab2?
871 | share(1M) listet /usr/homes? (unportabel!)
872 | showmount(1) zeigt /usr/homes?
873 | showmount(1) zeigt /usr/homes?
886 | share(1M) exportiert /usr/homes für root zugreifbar?
902 | tcsh auf Solaris installiert? (pkginfo SUNWtcsh)
903 | bash auf Solaris installiert? (pkginfo SUNWbash)
904 | tcsh auf NetBSD installiert? (pkg_info -e tcsh)
905 | bash auf NetBSD installiert? (pkg_info -e bash)
(36 rows)

```

## A.5 List of check scripts and parameters

This section lists check scripts available in step II of the Virtual Unix Lab, a textual description of what they do as printed by the `what.is` parameter and a list of parameters as printed by the `listparms` parameter. As step II was in German language, so are the descriptions given here. A future implementation of the Virtual Unix Lab may pay attention to internationalization.

**admin-check-clearharddisk:** Festplatte zum Komprimieren optimieren (mit 0-Bits beschreiben)

Parameters:

- (none)

**admin-check-makeimage:** Muss auf localhost laufen! Erzeugt Plattenimage von \$DISK von \$RECHNER in Datei \$IMGFILE.img; Zeit ca. 30min

Parameters:

- RECHNER (Default: 'unset'): Rechner dessen Platte in IMGFILE verpackt werden soll (vulab1, ...)
- IMGFILE (Default: 'unset'): Imagefile, relativ zu /vulab
- DISK (Default: 'sd0'): Platte, von der das Image gemacht werden soll

**netbsd-check-installed-pkg:** Testet ob Paket \$PKG unter NetBSD installiert ist

Parameters:

- PKG (Default: 'tssh'): Package-Pattern fuer pkg\_info -e

**netbsd-check-rcvar-set:** Testet ob Variable RCVAR in /etc/rc.conf gesetzt ist (NetBSD)

Parameters:

- RCVAR (Default: 'rc\_configured'): Variable, die überprüft werden soll

**netbsd-check-user-shell:** Testet ob Shell von User \$LOGIN gleich \$SHELL\_SHOULD in /etc/master.passwd

Parameters:

- LOGIN (Default: 'test'): Benutzer, dessen Shell ueberprueft werden soll
- SHELL\_SHOULD (Default: '\*/tssh'): Regulaerer Ausdruck, gegen den verglichen werden soll.

**solaris-check-installed-pkg:** Testet ob Paket \$PKG unter Solaris installiert ist

Parameters:

- PKG (Default: 'tssh'): Package-Pattern fuer pkginfo

**unix-check-file-owner:** Prüft ob FILE dem Benutzer OWNER\_SHOULD (Login oder UID) gehoert.

Parameters:

- FILE (Default: '/etc/passwd'): Datei oder Verzeichnis, absolut.
- OWNER\_SHOULD (Default: 'root'): Login-Name oder numerische User-ID

**unix-check-mount:** Versucht MOUNT\_FROM auf MOUNT\_ON zu mounten

Parameters:

- MOUNT\_FROM (Default: 'foo'): Erstes Argument fuer mount(8)
- MOUNT\_ON (Default: '/mnt'): Mountpoint, muss existieren
- MOUNT\_ARGS (Default: 'none'): Parameter fuer mount(8)

**unix-check-process-running:** Testet ob PROCESS läuft (Regulärer Ausdruck gegen ps(1)-Output)

Parameters:

- PROCESS (Default: 'init'): Regulärer Ausdruck, gegen den der Output von ps -elf/aux verglichen wird.

**unix-check-user-exists:** Testet ob der Benutzer \$LOGIN existiert (via getpwnam())

Parameters:

- LOGIN (Default: 'test'): Benutzer, dessen Home-Dir ueberprueft werden soll

**unix-check-user-fullname:** Testet ob der volle Name von LOGIN gleich FULLNAME\_SHOULD ist (via getpwnam())

Parameters:

- LOGIN (Default: 'root'): Benutzer, dessen Fullname ueberprueft werden soll
- FULLNAME\_SHOULD (Default: 'Charlie Root'): String auf den der Fullname gesetzt sein sollte

**unix-check-user-home:** Testet ob das Home-Verzeichnis von User \$LOGIN gleich \$HOME\_SHOULD ist (via getpwnam())

Parameters:

- LOGIN (Default: 'test'): Benutzer, dessen Home-Dir ueberprueft werden soll
- HOME\_SHOULD (Default: '\*'): Pfad auf den das Home-Verzeichnis gesetzt sein sollte

**unix-check-user-ingroup:** Testet ob User \$LOGIN in Gruppe GROUP\_SHOULD ist (primary oder supplementary)

Parameters:

- LOGIN (Default: 'test'): Login-Name
- GROUP\_SHOULD (Default: 'wheel'): Primäre oder Supplementäre Gruppe, in der der Benutzer sein sollte

**unix-check-user-password:** Testet ob Passwort von User \$LOGIN gleich \$PASSWORD\_SHOULD (plain)

Parameters:

- LOGIN (Default: 'test'): Benutzer, dessen Passwort ueberprueft werden soll
- PASSWORD\_SHOULD (Default: '\*'): Plaintext-Passwort (unverschlüsselt), gegen das geprueft wird

**unix-check-user-shell:** Testet ob die Login-Shell von User \$LOGIN gleich \$SHELL\_SHOULD ist (via getpwnam())

Parameters:

- LOGIN (Default: 'test'): Benutzer, dessen Login-Shell ueberprueft werden soll
- SHELL\_SHOULD (Default: '/bin/sh'): Pfad auf den die Shell gesetzt sein sollte

## A.6 Selected check scripts

This section shows the full source code of selected check scripts from step I, i.e. before optimizing, and from step II, i.e. after optimizing.

### A.6.1 Step I

#### A.6.1.1 netbsd-check-finger.sh

```
#!/bin/sh
#
# Checks if $user exists, NetBSD-specific
user=test
if [ `finger $user | grep Shell: | wc -l` = 1 ]
then
    rc=ok
```

```

else
    rc=failed
fi
echo $rc

```

### A.6.1.2 netbsd-check-masterpw.sh

```

#!/bin/sh
#
# Checks if $user exists, NetBSD-specific
user=test

grep -l $user /etc/master.passwd 2>&1 >/dev/null
rc=$?

echo $rc

```

### A.6.1.3 netbsd-check-pkginstalled.sh

```

#!/bin/sh

pkg_info -qe tcsh
tcsh_installed=$?

pkg_info -qe bash
bash_installed=$?

echo tcsh_installed=$tcsh_installed
echo bash_installed=$bash_installed

if [ $tcsh_installed = 0 -a $bash_installed = 0 ]
then
    rc=ok
else
    rc=failed
fi

echo $rc

```

### A.6.1.4 netbsd-check-pw.pl

```

#!/usr/local/bin/perl

$user="test";
$should_pwu="vutest";

$is_pwe=(getpwnam($user))[1];
($salt) = ($is_pwe =~ /^(..)/);
$should_pwe=crypt($should_pwu, $salt);

print "is_pwe=$is_pwe\n";
print "salt='$salt'\n";
print "should_pwe=$should_pwe\n";

if ( $is_pwe eq $should_pwe ) {
    $rc = "ok";
}

```

```

} else {
    $rc = "failed";
}

print "$rc\n";

```

### A.6.1.5 netbsd-check-usershell2.sh

```

#!/bin/sh
#
# Tests if shell of user $user is set to $should_shell

user=vulab
should_shell='././bash'

### NO CHANGES FROM HERE

is_shell=`finger $user | grep Shell | awk '{print $4}'`

echo is_shell=$is_shell
echo should_shell=$should_shell

if expr "$is_shell" : "$should_shell" >/dev/null
then
    rc=ok
else
    rc=wrong
fi

echo $rc

```

### A.6.1.6 check-program-output

```

#!/usr/bin/perl
#####
# Kurzbeschreibung: "Dieses Script ($0) $WHATIS\n." (1 Zeile)
$WHATIS='
***
*** Testet ob Ausgabe von PROGRAM den regulären Ausdruck OUTPUT_SHOULD enthält
***
';

# Based on work by Thomas Ernst <herr.ernst@gmx.de>

#####
# Parameter:
# [ "Variable", "Default", "Beschreibung der Variable" ]
#
@vars = (
    [ "PROGRAM", "true",
      "Programm für sh -c '$PROGRAM'" ],
    [ "OUTPUT_SHOULD", "Hallo Welt!",
      "zu suchender Regulärer Ausdruck" ],
    [ "VERBOSE", "",
      "Ausgabe ausgeben" ]
);

#####
# Check-Spezifisch:
sub check()
{

```

```

print "PROGRAM='$PROGRAM'\n";
print "OUTPUT_SHOULD='$OUTPUT_SHOULD'\n";
print "VERBOSE='$VERBOSE'\n";
print "\n";

$rc = "wrong";
if (open(F, "$PROGRAM 2>&1 |")) {
    while(<F>){
        if(/$OUTPUT_SHOULD/){
            print "Match: $_\n";

            $rc = "ok";
            last;
        } elsif ($VERBOSE) {
            print "$_";
        }
    }
    close(F);
}

print "$rc\n";
}

#####
### Common code:

# Variablen übernehmen
sub init()
{
    for($i=0; $i<=$#vars; $i++) {
        if($ENV{$vars[$i][0]}) {
            ${vars[$i][0]} = $ENV{$vars[$i][0]};
        } else {
            ${vars[$i][0]} = $vars[$i][1];
        }
    }
}

# "Hauptprogramm"
if($ARGV[0] eq "listparms") {
    for($i=0;$i<=$#vars;$i++) {
        print "$vars[$i][0]|$vars[$i][1]|$vars[$i][2]\n";
    }
} elsif($ARGV[0] eq "whatis") {
    $WHATIS=~s/^\n*//g;
    $WHATIS=~s/\n\*\*\?//g;
    $WHATIS=~s/^\*\*\?//g;
    $WHATIS=~s/\n*\?//g;
    print "$WHATIS\n";
} elsif($ARGV[0] eq "-h") {
    print "whatis    Kurzbeschreibung des Scripts\n";
    print "listparms   Listet Variablen mit Default und Beschreibung\n";
    print "-h         Alle Parameter\n";
    print "sonst      Check-Script wird ausgefuehrt\n";
} else {
    init();
    check();
}

```



## A.6.2 Step II

### A.6.2.1 admin-check-clearharddisk

```
#!/bin/sh
#####
# Kurzbeschreibung: "Dieses Script ($0) $WHATIS\n." (1 Zeile)
WHATIS='
***
*** Festplatte zum Komprimieren optimieren (mit 0-Bits beschreiben)
***
'

#####
### Parameter:

vars=""

#####
### Check-Spezifisch:
check()
{
    cd /

    echo Cleaning empty blocks...
    dd if=/dev/zero of=0 bs=1048576

    sleep 1
    echo ""
    echo Cleaning up...
    rm -f 0

    echo Done.
    echo ok
}

#####
### AB HIER NICHTS MEHR VERAENDERN !!!
#####
### Common code:
# Variablen uebernehmen
for var in $vars ; do
    eval "$var=\${var:=\${${var}_def}}\"
done

# "Hauptprogramm"
if [ "$1" = 'listparms' ]; then
    for var in $vars ; do
        eval "echo \"\$var|\${${var}_def}|\${${var}_bez}\""
    done
elif [ "$1" = "whatis" ]; then
    echo "$WHATIS" | sed -e 's/^\*\*\*/g' | grep -v '^[ ]*$'
elif [ "$1" = "-h" ]; then
    echo "Usage: $0 [whatis|listargs|-h]"
else
    check
fi
```

### A.6.2.2 admin-check-makeimage

```
#!/bin/sh
#
```

```

# Basiert in guten Teilen auf deploy1
#
# Sollte nur fuer einmalige Uebungen zur Imageerzeugung benutzt werden
# (anschliessend Uebung im VUlab loeschen!)
#

#####
# Kurzbeschreibung: "Dieses Script ($0) $WHATIS\n." (1 Zeile)
WHATIS='
***
*** Muss auf localhost laufen! Erzeugt Plattenimage von $DISK von $RECHNER
*** in Datei $IMGFILE.img; Zeit ca. 30min
***
'

#####
### Parameter:

vars="RECHNER IMGFILE DISK"

RECHNER_def=unset
RECHNER_bez="Rechner dessen Platte in IMGFILE verpackt werden soll (vulab1, ...)"

IMGFILE_def=unset
IMGFILE_bez="Imagefile, relativ zu /vulab"

DISK_def='sd0'
DISK_bez="Platte, von der das Image gemacht werden soll"

#####
### Check-Spezifisch:
check()
{
    imagePath="/vulab"
    imageHost=smaug

    client_log=logs/$RECHNER.log
    deployment_done_cookie="VULab Deployment Done"
    deployment_poll_interval=60          # seconds
    ssh="./rsh-wrapper -p 9999"

    if [ $RECHNER = unset ]; then
        echo RECHNER unset
        echo failed
        exit 0
    fi

    if [ $IMGFILE = unset ]; then
        echo IMGFILE unset
        echo failed
        exit 0
    fi

    if [ "`uname -n`" != $imageHost ]; then
        echo muss auf `localhost` laufen
        echo failed
        exit 0
    fi

    cd $imagePath

    echo "RECHNER=$RECHNER"
    echo "DISK=$DISK"
    echo "IMGFILE=$IMGFILE"
    echo "imagePath=$imagePath"
    echo ""

    echo Starting deployment: `date`

```

```

# Define which image to create
echo ${DISK} ${IMGFILE} >mkimg-${RECHNER}

if [ "$ssh $RECHNER echo READY" != READY ]
then
    echo "Machine $RECHNER didn't respond properly via '$ssh'"
    echo failed
    exit 0
fi

# Setup client logfile
rm -f $client_log
install -m 777 /dev/null $client_log

# Kick client into netboot
echo "Starting netboot on $RECHNER in background..."
# Pfad fuer Solaris ist /usr/sbin/reboot, redirection Shell-abhaengig !!!
ssh $RECHNER "env PATH=/usr/sbin:/sbin /bin/sh -c 'reboot -- net' \
    </dev/null 2>/dev/null >/dev/null" \
    </dev/null 2>/dev/null >/dev/null &
echo "done. (rc=$?)"

# Wait for client to startup on netboot properly
echo "Waiting a bit to get to /etc/rc..."
sleep 120 # takes about 70 seconds, plus some extra
if ! grep ^Starting $client_log >/dev/null 2>/dev/null
then
    echo "Client $RECHNER didn't do netboot properly, aborting."
    exit 1
else
    echo "$RECHNER properly netbooted."
fi

# Client's running, now wait for it to be done
while ! grep -q "$deployment_done_cookie" $client_log
do
    echo 'date': waiting for $RECHNER to finish: `tail -1 $client_log`
    sleep $deployment_poll_interval
done
echo done.

# Clean up
echo Cleaning up ...
rm -f mkimg-${RECHNER}
echo done.

echo Checking if $RECHNER was installed properly
sleep 120 # time for reboot
if [ "$ssh $RECHNER echo READY" != READY ]
then
    echo "$0: machine $RECHNER didn't respond properly via '$ssh'
        after installing"
    exit 1
fi
echo OK.

# Image in Tabelle 'images' eintragen:
echo -n Remember image $IMGFILE in database:
echo "INSERT INTO images (bezeichnung) VALUES ('$IMGFILE');" \
| psql -U vulab

echo Deployment done: `date`

echo ok
}

```

```

#####
### AB HIER NICHTS MEHR VERAENDERN !!!
#####

```

```

### Common code:
# Variablen uebernehmen
for var in $vars ; do
    eval "$var=\${var:=\${${var}_def}}\"
done

# "Hauptprogramm"
if [ "$1" = 'listparms' ]; then
    for var in $vars ; do
        eval "echo \"\$var|\${${var}_def}|\${${var}_bez}\""
    done
elif [ "$1" = "whatis" ]; then
    echo "$WHATIS" | sed -e 's/^\*\*\*/g' | grep -v '^[ ]*$'
elif [ "$1" = "-h" ]; then
    echo "Usage: $0 [whatis|listargs|-h]"
else
    check
fi

```

### A.6.2.3 check-file-contents

```

#!/usr/bin/perl
#####
# Kurzbeschreibung: "Dieses Script ($0) $WHATIS\n." (1 Zeile)
$WHATIS='
***
*** Testet ob FILE den regulären Ausdruck CONTENT_SHOULD enthält
***
';

# Based on work by Thomas Ernst <herr.ernst@gmx.de>

#####
# Parameter:
# [ "Variable", "Default", "Beschreibung der Variable" ]
#
@vars = (
    [ "FILE", "/etc/motd",
      "zu durchsuchende Datei, absoluter Pfad" ],
    [ "CONTENT_SHOULD", "Hallo Welt!",
      "zu suchender Regulärer Ausdruck" ]
);

#####
# Check-Spezifisch:
sub check()
{
    print "FILE=$FILE\n";
    print "CONTENT_SHOULD=$CONTENT_SHOULD\n";
    print "";

    $rc = "wrong";
    if (open(F, "$FILE")) {
        while(<F>){
            chomp();
            #print "$_\n";
            if(/$CONTENT_SHOULD/){
                $rc = "ok";
                last;
            }
        }
        close(F);
    }

    print "$rc\n";
}

```

```

}

#####
### Common code:

# Variablen übernehmen
sub init()
{
    for($i=0; $i<=#vars; $i++) {
        if(exists($ENV{$vars[$i][0]}) {
            ${vars[$i][0]} = $ENV{$vars[$i][0]};
        } else {
            ${vars[$i][0]} = $vars[$i][1];
        }
    }
}

# "Hauptprogramm"
if($ARGV[0] eq "listparms") {
    for($i=0; $i<=#vars; $i++) {
        print "${vars[$i][0]}|${vars[$i][1]}|${vars[$i][2]}\n";
    }
} elseif($ARGV[0] eq "whatis") {
    $WHATIS=~s/^\n*//g;
    $WHATIS=~s/\n\+\*\+ ?//g;
    $WHATIS=~s/^\*\*\+ ?//g;
    $WHATIS=~s/\n*\+//g;
    print "$WHATIS\n";
} elseif($ARGV[0] eq "-h") {
    print "whatis      Kurzbeschreibung des Scripts\n";
    print "listparms    Listet Variablen mit Default und Beschreibung\n";
    print "-h          Alle Parameter\n";
    print "sonst       Check-Script wird ausgefuehrt\n";
} else {
    init();
    check();
}

```

#### A.6.2.4 unix-check-user-exists

```

#!/usr/bin/perl
#####
# Kurzbeschreibung: "Dieses Script ($0) $WHATIS\n." (1 Zeile)
$WHATIS='
***
*** Testet ob der Benutzer $LOGIN existiert (via getpwnam())
***
';

#####
# Parameter:
# [ "Variable", "Default, "Beschreibung der Variable" ]
#
@vars = (
    [ "LOGIN", "test",
      "Benutzer, dessen Home-Dir ueberprueft werden soll" ],
);

#####
# Check-Spezifisch:
sub check()
{
    $login_is=(getpwnam($LOGIN))[0];

    print "login_is=$login_is\n";
}

```

```

print "LOGIN=$LOGIN\n";
print "\n";

if ( $login_is eq $LOGIN ) {
    $src="ok";
} else {
    $src="wrong";
}

print "$src\n";
}

#####
### AB HIER NICHTS MEHR VERAENDERN !!!
#####
### Common code:

# Variablen übernehmen
sub init()
{
    for($i=0; $i<=#vars; $i++) {
        if($ENV{$vars[$i][0]}) {
            ${vars[$i][0]} = $ENV{$vars[$i][0]};
        } else {
            ${vars[$i][0]} = $vars[$i][1];
        }
    }
}

# "Hauptprogramm"
if($ARGV[0] eq "listparms") {
    for($i=0;$i<=#vars;$i++) {
        print "$vars[$i][0]|$vars[$i][1]|$vars[$i][2]\n";
    }
} elsif($ARGV[0] eq "whatis") {
    $WHATIS=~s/^\n*//g;
    $WHATIS=~s/\n\*\*\* ?//g;
    $WHATIS=~s/^\*\*\* ?//g;
    $WHATIS=~s/\n*\$//g;
    print "$WHATIS\n";
} elsif($ARGV[0] eq "-h") {
    print "whatis      Kurzbeschreibung des Scripts\n";
    print "listparms     Listet Variablen mit Default und Beschreibung\n";
    print "-h            Alle Parameter\n";
    print "sonst        Check-Script wird ausgeführt\n";
} else {
    init();
    check();
}

```

### A.6.2.5 unix-check-user-shell

```

#!/usr/bin/perl
#####
# Kurzbeschreibung: "Dieses Script ($0) $WHATIS\n." (1 Zeile)
$WHATIS='
***
*** Testet ob die Login-Shell von User $LOGIN gleich $SHELL_SHOULD ist (via getpwnam())
***
';

#####
# Parameter:
# [ "Variable", "Default", "Beschreibung der Variable" ]
#

```

```

@vars = (
    [ "LOGIN", "test",
      "Benutzer, dessen Login-Shell ueberprueft werden soll" ],
    [ "SHELL_SHOULD", "/bin/sh",
      "Pfad auf den die Shell gesetzt sein sollte" ],
    );

#####
# Check-Spezifisch:
sub check()
{
    $shell_is=(getpwnam($LOGIN))[8];

    print "LOGIN=$LOGIN\n";
    print "shell_is=$shell_is\n";
    print "SHELL_SHOULD=$SHELL_SHOULD\n";
    print "\n";

    if ( $shell_is =~ $SHELL_SHOULD ) {
        $src="ok";
    } else {
        $src="wrong";
    }

    print "$src\n";
}

#####
### AB HIER NICHTS MEHR VERAENDERN !!!
#####
### Common code:

# Variablen uebernehmen
sub init()
{
    for($i=0; $i<=#vars; $i++) {
        if($ENV{$vars[$i][0]}) {
            ${vars[$i][0]} = $ENV{$vars[$i][0]};
        } else {
            ${vars[$i][0]} = $vars[$i][1];
        }
    }
}

# "Hauptprogramm"
if($ARGV[0] eq "listparms") {
    for($i=0;$i<=#vars;$i++) {
        print "$vars[$i][0]|$vars[$i][1]|$vars[$i][2]\n";
    }
}
elseif($ARGV[0] eq "whatis") {
    $WHATIS=~s/^\n*//g;
    $WHATIS=~s/\n\*\*\* ?//g;
    $WHATIS=~s/^\*\*\* ?//g;
    $WHATIS=~s/\n*\*//g;
    print "$WHATIS\n";
}
elseif($ARGV[0] eq "-h") {
    print "whatis      Kurzbeschreibung des Scripts\n";
    print "listparms    Listet Variablen mit Default und Beschreibung\n";
    print "-h          Alle Parameter\n";
    print "sonst       Check-Script wird ausgefuehrt\n";
}
else {
    init();
    check();
}

```

## A.6.2.6 unix-check-user-password

```
#!/usr/bin/perl
#####
# Kurzbeschreibung: "Dieses Script ($0) $WHATIS\n." (1 Zeile)
$WHATIS='
***
*** Testet ob Passwort von User $LOGIN gleich $PASSWD_SHOULD (plain)
***
';

#####
# Parameter:
# [ "Variable", "Default", "Beschreibung der Variable" ]
#
@vars = (
    [ "LOGIN", "test",
      "Benutzer, dessen Passwort ueberprueft werden soll" ],
    [ "PASSWD_SHOULD", "*",
      "Plaintext-Passwort (unverschlusselt), gegen das geprueft wird" ],
);

#####
# Check-Spezifisch:
sub check()
{
    $passwd_is_e=(getpwnam($LOGIN))[1];
    ($salt) = ($passwd_is_e =~ /^(.+)/);
    $PASSWD_SHOULD_u= $PASSWD_SHOULD;
    $PASSWD_SHOULD_e=crypt($PASSWD_SHOULD_u, $salt);

    print "passwd_is_e=$passwd_is_e\n";
    print "salt='$salt'\n";
    print "PASSWD_SHOULD_u=$PASSWD_SHOULD_u\n";
    print "PASSWD_SHOULD_e=$PASSWD_SHOULD_e\n";
    print "\n";

    if ( $PASSWD_SHOULD_e ne "" and $passwd_is_e eq $PASSWD_SHOULD_e ) {
        $src="ok";
    } else {
        $src="wrong";
    }

    print "$src\n";
}

#####
### AB HIER NICHTS MEHR VERAENDERN !!!
#####
### Common code:

# Variablen uebernehmen
sub init()
{
    for($i=0; $i<=$#vars; $i++) {
        if($ENV{$vars[$i][0]}) {
            ${vars[$i][0]} = $ENV{$vars[$i][0]};
        } else {
            ${vars[$i][0]} = $vars[$i][1];
        }
    }
}

# "Hauptprogramm"
if($ARGV[0] eq "listparms") {
    for($i=0;$i<=$#vars;$i++) {
        print "$vars[$i][0]|$vars[$i][1]|$vars[$i][2]\n";
    }
}
```



```

} elsif($ARGV[0] eq "whatis") {
    $WHATIS=~s/^\n*//g;
    $WHATIS=~s/\n\*\*\ ?//g;
    $WHATIS=~s/^\*\*\ ?//g;
    $WHATIS=~s/\n*//g;
    print "$WHATIS\n";
} elsif($ARGV[0] eq "-h") {
    print "whatis      Kurzbeschreibung des Scripts\n";
    print "listparms   Listet Variablen mit Default und Beschreibung\n";
    print "-h          Alle Parameter\n";
    print "sonst       Check-Script wird ausgefuehrt\n";
} else {
    init();
    check();
}

```

### A.6.2.7 unix-check-process-running

```

#!/usr/bin/perl
#####
# Kurzbeschreibung: "Dieses Script ($0) $WHATIS\n." (1 Zeile)
$WHATIS='
***
*** Testet ob PROCESS läuft (Regulärer Ausdruck gegen ps(1)-Output)
***
';

#####
# Parameter:
# [ "Variable", "Default, "Beschreibung der Variable" ]
#
@vars = (
    [ "PROCESS", "init",
      "Regulärer Ausdruck, gegen den der Output von ps -elf/aux verglichen wird." ],
);

#####
# Check-Spezifisch:
sub check()
{
    print "PROCESS=$PROCESS\n";

    $src = "wrong";
    if (open(P, "ps -elf 2>&l || ps -auxww 2>&l |")) {
        while(<P>) {
            if (/ $PROCESS/) {
                print "Match: $_\n";
                $src = "ok";
                last;
            }
        }
        close(P);
    }

    print "$src\n";
}

#####
### AB HIER NICHTS MEHR VERAENDERN !!!
#####
### Common code:

# Variablen übernehmen
sub init()
{

```

```

    for($i=0; $i<=$#vars; $i++) {
        if($ENV{$vars[$i][0]}) {
            ${vars[$i][0]} = $ENV{$vars[$i][0]};
        } else {
            ${vars[$i][0]} = $vars[$i][1];
        }
    }
}

# "Hauptprogramm"
if($ARGV[0] eq "listparms") {
    for($i=0; $i<=$#vars; $i++) {
        print "$vars[$i][0]|$vars[$i][1]|$vars[$i][2]\n";
    }
}
elseif($ARGV[0] eq "whatis") {
    $WHATIS=~s/^\n*//g;
    $WHATIS=~s/\n\*\*\* ?//g;
    $WHATIS=~s/^\*\*\* ?//g;
    $WHATIS=~s/\n*$/g;
    print "$WHATIS\n";
}
elseif($ARGV[0] eq "-h") {
    print "whatis      Kurzbeschreibung des Scripts\n";
    print "listparms     Listet Variablen mit Default und Beschreibung\n";
    print "-h            Alle Parameter\n";
    print "sonst         Check-Script wird ausgefuehrt\n";
}
else {
    init();
    check();
}

```

### A.6.2.8 netbsd-check-rcvar-set

```

#!/usr/bin/perl
#####
# Kurzbeschreibung: "Dieses Script ($0) $WHATIS\n." (1 Zeile)
$WHATIS='
***
*** Testet ob Variable RCVAR in /etc/rc.conf gesetzt ist (NetBSD)
***
';

#####
# Parameter:
# [ "Variable", "Default, "Beschreibung der Variable" ]
#
@vars = (
    [ "RCVAR", "rc_configured",
      "Variable, die überprüft werden soll" ],
);

#####
# Check-Spezifisch:
sub check()
{
    $rc=system(" /etc/rc.subr; ".
              ". /etc/rc.conf; ".
              "checkyesno $RCVAR; ".
              "exit \${?}");

    print "RCVAR=$RCVAR\n";
    print "rc=$rc\n";
    print "\n";

    if ( $rc == 0 ) {
        $rc="ok";
    }
}

```

```

    } else {
        $src="wrong";
    }

    print "$src\n";
}

#####
### AB HIER NICHTS MEHR VERAENDERN !!!
#####
### Common code:

# Variablen übernehmen
sub init()
{
    for($i=0; $i<=#vars; $i++) {
        if($ENV{$vars[$i][0]}) {
            ${vars[$i][0]} = $ENV{$vars[$i][0]};
        } else {
            ${vars[$i][0]} = $vars[$i][1];
        }
    }
}

# "Hauptprogramm"
if($ARGV[0] eq "listparms") {
    for($i=0;$i<=#vars;$i++) {
        print "$vars[$i][0]|$vars[$i][1]|$vars[$i][2]\n";
    }
}
elseif($ARGV[0] eq "whatis") {
    $WHATIS=~s/^\n*//g;
    $WHATIS=~s/\n\*\*\* ?//g;
    $WHATIS=~s/^\*\*\* ?//g;
    $WHATIS=~s/\n*$/g;
    print "$WHATIS\n";
}
elseif($ARGV[0] eq "-h") {
    print "whatis      Kurzbeschreibung des Scripts\n";
    print "listparms     Listet Variablen mit Default und Beschreibung\n";
    print "-h           Alle Parameter\n";
    print "sonst        Check-Script wird ausgeführt\n";
}
else {
    init();
    check();
}

```



# Appendix B

## Database structure

This section describes the database tables used in the Virtual Unix Lab, the SQL-statements that were used to create the tables in the PostgreSQL database, and example database records in a few selected cases.

### B.1 Table: benutzer

This table describes a user in the Virtual Unix Lab.

```
CREATE TABLE benutzer (  
    user_id serial NOT NULL,           -- unique user id  
    vorname varchar(50) NOT NULL,     -- first name  
    nachname varchar(50) NOT NULL,    -- last name  
    matrikel_nr numeric(10) NOT NULL, -- student id  
    email varchar(80) NOT NULL,       -- contact email  
    login varchar(80) NOT NULL,       -- vulab login  
    passwort varchar(15) NOT NULL,    -- password  
    freischalt_secret varchar(30) NOT NULL, -- initial secret  
    anmeldedatum date NOT NULL,      -- sign-on date  
    typ varchar(80) DEFAULT 'user' NOT NULL,  
    PRIMARY KEY (user_id),  
    UNIQUE (user_id),  
    UNIQUE (login),  
    UNIQUE (matrikel_nr)  
);
```

### B.2 Table: rechner

This table contains a list of all the lab machines.

```
CREATE TABLE rechner (  

```

```

    bezeichnung varchar(30) NOT NULL,          -- hostname
    PRIMARY KEY (bezeichnung),
    UNIQUE (bezeichnung)
);

```

### B.3 Table: images

This table contains a list of all possible images that can be installed on the lab machines.

```

CREATE TABLE images (
    bezeichnung varchar(150) NOT NULL,        -- filename
    PRIMARY KEY (bezeichnung),
    UNIQUE (bezeichnung)
);

```

### B.4 Table: uebungen

This table lists all possible exercises with their basic properties.

#### B.4.1 Definition

```

CREATE TABLE uebungen (
    uebung_id varchar(40) NOT NULL,          -- exercise id
    bezeichnung varchar(150) NOT NULL,      -- description
    nur_fuer varchar(40),                   -- user-restriction
    vorlauf time NOT NULL,                  -- preparation time
    dauer time NOT NULL,                    -- exercise duration
    nachlauf time NOT NULL,                 -- time for checks
    wiederholbar boolean NOT NULL,         -- repeatable?
    text varchar(150) NOT NULL,             -- exercise text filename
    mehr_info varchar(150),                 -- more information (unused)
    PRIMARY KEY (uebung_id),
    UNIQUE (uebung_id)
);

```

#### B.4.2 Example records

```

vulab-> select uebung_id, bezeichnung, vorlauf, dauer, text from uebungen;

```

uebung_id	bezeichnung	vorlauf	dauer	text
pruefung	Verwalten von Benutzern mit Hilfe von NIS	00:45:00	01:00:00	pruefung.html
pruefung2	Verwalten von Benutzern mit Hilfe von NFS	00:45:00	01:00:00	pruefung2.html
nfs	Aufsetzen von NFS Client und Server	00:45:00	01:30:00	nfs.php
solaris	Solaris konfigurieren	00:45:00	01:30:00	solaris.php
nis	Aufsetzen von NIS Client und Server	00:45:00	01:30:00	nis.php
netbsd	NetBSD konfigurieren	00:45:00	01:30:00	netbsd.php
update-solaris	Solaris-Image updaten	00:45:00	01:00:00	solaris.php

## B.5 Table: uebung\_setup

This table lists the machines and their images associated with a certain exercise.

```
CREATE TABLE uebung_setup (
  uebung_id varchar(40) NOT NULL,           -- exercise id
  rechner varchar(30) NOT NULL,           -- hostname
  image varchar(150) NOT NULL,           -- image filename
  CONSTRAINT pk_uebung_setup
    PRIMARY KEY (uebung_id, rechner),
  CONSTRAINT fk_uebung
    FOREIGN KEY (uebung_id)
    REFERENCES uebungen (uebung_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT fk_rechner
    FOREIGN KEY (rechner)
    REFERENCES rechner (bezeichnung)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT fk_image
    FOREIGN KEY (image)
    REFERENCES images (bezeichnung)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

## B.6 Table: uebungs\_checks

This table contains a list of checks to make at the end of a certain exercise. The “parameter” field is only available in implementation step II.

### B.6.1 Definition

```
CREATE TABLE uebungs_checks (
  check_id serial NOT NULL,               -- check id
  uebung_id varchar(80) NOT NULL,         -- associated exercise
  script varchar(150) NOT NULL,          -- which script to run
  parameter varchar(300),                -- parameters for script (Step II only!)
  rechner varchar(30) NOT NULL,          -- where to run script
  bezeichnung varchar(150) NOT NULL,     -- text for feedback
  PRIMARY KEY (check_id),
  UNIQUE (check_id),
  CONSTRAINT fk_uebung
    FOREIGN KEY (uebung_id)
    REFERENCES uebungen (uebung_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT fk_rechner
    FOREIGN KEY (rechner)
    REFERENCES rechner (bezeichnung)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

## B.6.2 Example records

```
vulab> select * from uebungs_checks where uebung_id='netbsd';
```

check_id	uebung_id	script	Bezeichnung	rechner
909	netbsd	netbsd-check-installed-pkg	bash installiert? (pkg_info -e bash)	vulabl
910	netbsd	unix-check-user-exists	Benutzer angelegt? (getpwnam(3))	vulabl
911	netbsd	unix-check-user-home	Home-Directory richtig gesetzt?	vulabl
912	netbsd	unix-check-user-shell	Shell auf tcsh gesetzt? (getpwnam(3))	vulabl
913	netbsd	netbsd-check-user-shell	Shell auch in /etc/master.passwd gesetzt?	vulabl
914	netbsd	unix-check-user-password	Passwort richtig gesetzt? (getpwnam(3))	vulabl
915	netbsd	unix-check-user-shell	Shell des Users vulab auf bash gesetzt?	vulabl
908	netbsd	admin-check-clearharddisk	tcsh installiert? (pkg_info -e tcsh)	vulabl

(8 rows)

## B.7 Table: buchungen

This table contains entries for exercises actually booked by users, including time and date of the exercise and which exercise to practice.

### B.7.1 Definition

```
CREATE TABLE buchungen (
  buchungs_id serial NOT NULL,          -- booked exercise id
  user_id int NOT NULL,                 -- for which user
  uebung_id varchar(40) NOT NULL,       -- which exercise
  datum date NOT NULL,                 -- when/date
  startzeit time NOT NULL,              -- when/time
  freigegeben varchar(30) DEFAULT 'nein' NOT NULL, -- exercise set up?
  endzeit time,                          -- when/ended
  at_id int,                             -- setup at(1) job id
  at_id_end int,                         -- uebung_ende job id
  ip varchar(20),                       -- where user came from
  PRIMARY KEY (buchungs_id),
  UNIQUE (buchungs_id),
  CONSTRAINT fk_uebung
    FOREIGN KEY (uebung_id)
    REFERENCES uebungen (uebung_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT fk_user_id
    FOREIGN KEY (user_id)
    REFERENCES benutzer (user_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

### B.7.2 Example records

```
vulab> select * from buchungen;
```

buchungs_id	user_id	uebung_id	datum	startzeit	freigegeben	endzeit	at_id	at_id_end	ip
114	33	nfs	2004-05-03	15:00:00	nicht-mehr	15:06:18	234	235	132.199.213.37
115	33	nfs	2004-05-11	21:00:00	nicht-mehr	23:30:05	236	237	
116	33	nfs	2004-05-18	11:45:00	nicht-mehr	12:05:21	238	239	194.95.108.21
117	34	nfs	2004-05-20	18:00:00	nicht-mehr	20:30:05	241	247	
123	35	nfs	2004-05-21	15:00:00	nicht-mehr	16:20:57	248	250	194.95.108.32
124	35	nfs	2004-05-21	21:00:00	nicht-mehr	22:28:11	249	251	194.95.108.32



122	37	nfs	2004-05-22	12:00:00	nicht-mehr	13:07:14	246	252	194.95.108.32
120	35	nfs	2004-05-22	15:00:00	nicht-mehr	16:10:11	244	253	194.95.108.38
121	35	nfs	2004-05-23	12:00:00	nicht-mehr	12:51:23	245	255	194.95.108.38
126	35	nfs	2004-05-23	15:00:00	nicht-mehr	15:01:08	256	257	194.95.108.32
130	34	nfs	2004-05-25	21:00:00	nicht-mehr	21:16:40	261	262	194.95.108.32
127	37	nfs	2004-05-26	15:00:00	nicht-mehr	15:11:11	258	263	194.95.108.38
129	38	netbsd	2004-05-26	18:00:00	nicht-mehr	19:29:04	260	266	82.83.169.114
131	44	nfs	2004-05-27	18:00:00	nicht-mehr	19:25:03	264	271	132.199.227.122
134	43	nfs	2004-05-28	12:00:00	nicht-mehr	14:30:04	268	273	194.95.108.68
136	38	netbsd	2004-05-28	21:00:00	nicht-mehr	23:30:05	270	274	
138	38	netbsd	2004-05-29	12:00:00	nicht-mehr	12:37:46	275	277	194.95.108.32
128	35	nfs	2004-05-29	15:00:00	nicht-mehr	16:28:02	259	280	194.95.108.32
139	37	nfs	2004-05-29	18:00:00	nicht-mehr	19:38:45	276	281	194.95.108.38
132	35	nfs	2004-05-29	21:00:00	nicht-mehr	22:25:28	265	283	194.95.108.32
141	38	netbsd	2004-05-30	12:00:00	nicht-mehr	14:30:04	279	284	
140	38	nfs	2004-05-30	15:00:00	nicht-mehr	17:30:04	278	285	
143	39	nfs	2004-05-31	21:00:00	nicht-mehr	23:30:03	286	287	
144	38	nfs	2004-06-01	21:00:00	nicht-mehr	22:27:26	288	289	194.95.108.32
142	37	nfs	2004-06-02	15:00:00	nicht-mehr	16:13:50	282	290	194.95.108.38
137	44	nfs	2004-06-02	18:00:00	nicht-mehr	20:30:04	272	291	132.199.227.122
146	44	nfs	2004-06-02	21:00:00	nicht-mehr	23:30:03	293	294	132.199.227.122
147	38	nfs	2004-06-03	12:00:00	nicht-mehr	13:36:55	295	296	194.95.108.132
149	50	nfs	2004-06-07	15:00:00	nein		298		
145	44	nfs	2004-06-03	18:00:00	nicht-mehr	18:34:36	292	300	132.199.227.122
150	48	nfs	2004-06-03	21:00:00	nicht-mehr	22:24:17	299	302	132.199.227.122
155	44	nfs	2004-06-07	21:00:00	nein		306		
152	33	nfs	2004-06-04	09:00:00	nicht-mehr	11:30:05	303	307	
153	33	nfs	2004-06-04	12:00:00	nicht-mehr	14:30:17	304	308	194.95.108.65
156	38	nfs	2004-06-05	00:00:00	nicht-mehr	01:01:08	309	310	194.95.108.32
157	34	nfs	2004-06-07	18:00:00	nein		311		
154	48	nfs	2004-06-07	09:00:00	nicht-mehr	10:29:55	305	312	132.199.227.122
158	48	nfs	2004-06-08	12:00:00	nein		313		
159	44	nfs	2004-06-08	15:00:00	nein		314		
148	50	nfs	2004-06-07	12:00:00	nicht-mehr	13:29:05	297	315	194.95.108.159
160	52	nfs	2004-06-08	21:00:00	nein		316		
161	34	nfs	2004-06-10	15:00:00	nein		317		

(42 rows)

## B.8 Table: ergebnis\_checks

This table lists the results from the checks belonging to a certain booked exercise.

### B.8.1 Definition

```
CREATE TABLE ergebnis_checks (
    buchungs_id int NOT NULL,           -- booked exercise id
    check_id int NOT NULL,             -- check id
    erfolg boolean NOT NULL,          -- result
    CONSTRAINT pk_ergebnis_checks
        PRIMARY KEY (buchungs_id,
                    check_id),
    CONSTRAINT fk_check_id
        FOREIGN KEY (check_id)
        REFERENCES uebungs_checks (check_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    CONSTRAINT fk_buchungs_id
        FOREIGN KEY (buchungs_id)
        REFERENCES buchungen (buchungs_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

## B.8.2 Example records

```
vulab-> select * from ergebnis_checks where buchungs_id=129;
```

buchungs_id	check_id	erfolg
129	908	t
129	909	f
129	910	f
129	911	f
129	912	f
129	913	f
129	914	f
129	915	f

(8 rows)

```
vulab-> select buchungs_id, ergebnis_checks.check_id, bezeichnung, erfolg
vulab-> from ergebnis_checks, uebungs_checks
```

```
vulab-> where buchungs_id=129 and ergebnis_checks.check_id=uebungs_checks.check_id;
```

buchungs_id	check_id	bezeichnung	erfolg
129	908	tcsh installiert? (pkg_info -e tcsh)	t
129	909	bash installiert? (pkg_info -e bash)	f
129	910	Benutzer angelegt? (getpwnam(3))	f
129	911	Home-Directory richtig gesetzt?	f
129	912	Shell auf tcsh gesetzt? (getpwnam(3))	f
129	913	Shell auch in /etc/master.passwd (via vipw(1)) gesetzt?	f
129	914	Passwort richtig gesetzt? (getpwnam(3))	f
129	915	Shell des Users vulab auf bash gesetzt?	f

(8 rows)

# Appendix C

## Evaluation data and code

### C.1 Questionnaire: questions — raw format

The following data was used as input for txt2survey. A L<sup>A</sup>T<sub>E</sub>X'd form of the questions including evaluation and extended statistical value are displayed in appendix C.2.

```
Fragebogen: Akzeptanzuntersuchung zum Virtuellen Unix Labor SS2004
+ Wodurch haben Sie über das Thema "Systemadministration" gelernt?
. Besuch der Vorlesung
  (( Sehr viel / Einiges / Geht so / Wenig / Nichts ))
. Script zur Vorlesung
  (( Sehr viel / Einiges / Geht so / Wenig / Nichts ))
. Übungen zur Vorlesung
  (( Sehr viel / Einiges / Geht so / Wenig / Nichts ))
. Virtuelles Unix Labor
  (( Sehr viel / Einiges / Geht so / Wenig / Nichts ))
. Analyse der FH-Rechner
  (( Sehr viel / Einiges / Geht so / Wenig / Nichts ))
. Analyse eigener Rechner
  (( Sehr viel / Einiges / Geht so / Wenig / Nichts ))
. Bücher
  (( Sehr viel / Einiges / Geht so / Wenig / Nichts ))
. Online-Informationen
  (( Sehr viel / Einiges / Geht so / Wenig / Nichts ))
+ Einbindung des Labors in die Vorlesung
. Ist das Virtuelle Unix Labor generell eine sinnvolle
  Ergänzung zur Vorlesung?
  ((Sehr sinnvoll / Sinnvoll / Geht so / Wenig sinnvoll / Unsinnig))
. Wie empfanden Sie den Nutzen des Virtuellen Unix Labors?
  ((sehr positiv / positiv / neutral / negativ / sehr negativ))
+ Benutzung des Virtuellen Unix Labors
. War das System einfach zu benutzen?
  (( Sehr einfach / Einfach / Umstaendlich / Sehr umstaendlich ))
. Waren genügend Übungstermine zur Auswahl?
  (( Zu viele / Genügend / Zu wenige ))
. Von wo aus haben Sie auf die Übungsrechner zugegriffen
  (( Zuhause / FH oder Uni / Sonstige ))
. Von welchem Betriebssystem aus haben Sie die Übungen
  gemacht? (( Windows / Unix (Linux, ...) / Sonstiges ))
. War die Ausgangskonfiguration der Rechner ausreichend,
  damit Sie die Übung bearbeiten konnten?
  (Mussten Sie viele Vorbereitungen treffen, um mit der
```

```

eigentlichen Übung beginnen zu können oder
war die vorhandene Übungsumgebung nach all Ihren Wünschen
eingerrichtet?)
(( Zu spartanisch / etwas spartanisch / Geht so / Komfortabel
/ Sehr komfortabel))
+ Allgemeines zum Übungsverlauf:
. Haben Sie die Übung alleine oder in einer Gruppe
absolviert? (Bitte jedes Mitglied der Gruppe diesen
Fragebogen ausfüllen!)
(( Alleine / Zu zweit / Zu dritt / Zu viert ))
. War die Zeit für das absolvieren der Übung zu kurz/zu lang?
((Viel zu kurz / Zu kurz / Genau richtig / Zu lang / Viel zu lang))
. Fanden Sie die Aufgabenstellung zu detailliert oder hätten
sie sich mehr Informationen zum Bearbeiten der Aufgabe gewünscht?
(( Viel zu viel Information / Zuviel Information / Genau richtig
/ Bitte etwas mehr Informationen / Bitte viel mehr Informationen ))
. Hätten Sie sich während der Übung gewünscht, um Hilfe
anfragen zu können, um weiterzukommen?
(( Ja, Hilfe wäre gut gewesen / Nein, bin alleine klargekommen))
. Hätten Sie sich gewünscht dass das System automatisch Probleme erkennt
und Hilfestellungen anbietet? (( ja / nein ))
+ Übungsverlauf: Wieviel haben die folgenden Hilfsmittel zum Bearbeiten
der Übungen des Virtuellen Unix Labors beigetragen?
. Besuch der Vorlesung
(( Nicht genutzt / Sehr viel / Einiges / Geht so / Wenig / Nichts ))
. Script zur Vorlesung
(( Nicht genutzt / Sehr viel / Einiges / Geht so / Wenig / Nichts ))
. Übungen zur Vorlesung
(( Nicht genutzt / Sehr viel / Einiges / Geht so / Wenig / Nichts ))
. Analyse der FH-Rechner
(( Nicht genutzt / Sehr viel / Einiges / Geht so / Wenig / Nichts ))
. Analyse eigener Rechner
(( Nicht genutzt / Sehr viel / Einiges / Geht so / Wenig / Nichts ))
. Bücher
(( Nicht genutzt / Sehr viel / Einiges / Geht so / Wenig / Nichts ))
. Online-Informationen
(( Nicht genutzt / Sehr viel / Einiges / Geht so / Wenig / Nichts ))
+ Übungsverlauf: Wieviel hat der Besuch der Vorlesung zum Bearbeiten
der Übungen des Virtuellen Unix Labors beigetragen?
. War die Vorlesung hilfreich bei der Bearbeitung der Aufgabe zum
NIS Server: ((Sehr / Etwas / Geht so / Wenig / Nichts))
. War die Vorlesung hilfreich bei der Bearbeitung der Aufgabe zum
NIS Client: ((Sehr / Etwas / Geht so / Wenig / Nichts))
. War die Vorlesung hilfreich bei der Bearbeitung der Aufgabe zum
NFS Server: ((Sehr / Etwas / Geht so / Wenig / Nichts))
. War die Vorlesung hilfreich bei der Bearbeitung der Aufgabe zum
NFS Client: ((Sehr / Etwas / Geht so / Wenig / Nichts))
. War die Vorlesung hilfreich für den Umgang mit Solaris allgemein?
((Sehr / Etwas / Geht so / Wenig / Nichts))
. War die Vorlesung hilfreich für den Umgang mit NetBSD allgemein?
((Sehr / Etwas / Geht so / Wenig / Nichts))
. Konnten Sie den nicht direkt vermittelten Stoff aus den
bereitgestellten Informationen (Vorgehensweisen, allgemeine
Informationen über Systeme, Vorgehen zur Analyse) ermitteln?
((Sehr / Etwas / Geht so / Wenig / Nichts))
+ Übungsverlauf: Wieviel hat die Benutzung des Scripts zum Bearbeiten
der Übungen des Virtuellen Unix Labors beigetragen?
. War das Script hilfreich bei der Bearbeitung der Aufgabe zum
NIS Server: ((Sehr / Etwas / Geht so / Wenig / Nichts))
. War das Script hilfreich bei der Bearbeitung der Aufgabe zum
NIS Client: ((Sehr / Etwas / Geht so / Wenig / Nichts))
. War das Script hilfreich bei der Bearbeitung der Aufgabe zum
NFS Server: ((Sehr / Etwas / Geht so / Wenig / Nichts))
. War das Script hilfreich bei der Bearbeitung der Aufgabe zum
NFS Client: ((Sehr / Etwas / Geht so / Wenig / Nichts))
. War das Script hilfreich für den Umgang mit Solaris allgemein:
((Sehr / Etwas / Geht so / Wenig / Nichts))
. War das Script hilfreich für den Umgang mit NetBSD allgemein:
((Sehr / Etwas / Geht so / Wenig / Nichts))

```

```

+ Feedback nach der Übung
. Waren die Informationen der Auswertung detailliert genug, um etwaige
  Fehler nachvollziehen zu können?
  (( Ja, ich konnte aus meinen Fehlern lernen
    / Nein, ich weiss immer noch nicht was falsch war ))
+ Angaben zur Person
. Interesse am Studium allgemein
  ((Sehr gross / Gross / Mittel / Weniger / Gar nicht))
. Interesse am Thema "Systemadministration"
  ((Sehr gross / Gross / Mittel / Weniger / Gar nicht))
. Interesse an "Unix" (Linux, Solaris, NetBSD, ...)
  ((Sehr gross / Gross / Mittel / Weniger / Gar nicht))
. Wie schätzen Sie die Wichtigkeit des Teilgebiets "NIS" ein?
  ((Sehr gross / Gross / Mittel / Weniger / Gar nicht))
. Wie schätzen Sie die Wichtigkeit des Teilgebiets "NFS" ein?
  ((Sehr gross / Gross / Mittel / Weniger / Gar nicht))
. Wieviele von 10 Vorlesungsstunden haben Sie besucht?
  (( 0-3 / 4-8 / 9-10 ))
. Haben Sie bisher ausserhalb der Vorlesung mit Systemverwaltung
  zu tun (Praktikum, Rechner zu Hause, ...)? (( Ja / Nein ))
. Wenn ja, mit welchen Betriebssystemen ?
  (( Windows / Linux / BSD / Solaris / AIX / Novell / sonstige))
. Studiensemester (( unter 4 / 4 / 5 / 6 / 7 / 8 / über 8 ))
. Geschlecht (( m / w ))
. Haben Sie sonstige Anmerkungen? Bitten geben Sie ihr
  Feedback per EMail oder hier ab! ((5x_____))

```

## C.2 Questionnaire: questions and results

This section lists the questionnaire that students who used the Virtual Unix Lab in the summer semester 2004 were asked to fill out, and students' answers. The evaluation of the results can be found in section 7.3. The questions (and possible answers) presented here are printed in a SMALL CAPITALS font. For each question, the answers given to each item are displayed as absolute and relative number, as well as a simplified bar-graph. For the evaluation, the modus<sup>1</sup> is given for all results, and the median<sup>2</sup> is printed for questions whose answers are represented an ordinal scale:

### WODURCH HABEN SIE ÜBER DAS THEMA "SYSTEMADMINISTRATION" GELERNT?

1. BESUCH DER VORLESUNG  
(( SEHR VIEL / EINIGES / GEHT SO / WENIG / NICHTS ))

---

<sup>1</sup> [Fahrmeir, 2003] pp. 53

<sup>2</sup> [Fahrmeir, 2003] pp. 55

Sehr viel:	16	(57%)	oooooooooooooooooooo
Einiges:	8	(28%)	oooooooooooo
Geht so:	4	(14%)	oooo
Wenig:	0	(0%)	
Nichts:	0	(0%)	
Summe:	28	(100%)	

Modus: Sehr viel (16)  
Median: Sehr viel (14)

## 2. SCRIPT ZUR VORLESUNG

(( SEHR VIEL / EINIGES / GEHT SO / WENIG / NICHTS ))

Sehr viel:	11	(39%)	oooooooooooo
Einiges:	13	(46%)	oooooooooooooooooooo
Geht so:	4	(14%)	oooo
Wenig:	0	(0%)	
Nichts:	0	(0%)	
Summe:	28	(100%)	

Modus: Einiges (13)  
Median: Einiges (14)

## 3. ÜBUNGEN ZUR VORLESUNG

(( SEHR VIEL / EINIGES / GEHT SO / WENIG / NICHTS ))

Sehr viel:	10	(35%)	oooooooooooo
Einiges:	13	(46%)	oooooooooooooooooooo
Geht so:	5	(17%)	oooooo
Wenig:	0	(0%)	
Nichts:	0	(0%)	
Summe:	28	(100%)	

Modus: Einiges (13)  
Median: Einiges (14)

## 4. VIRTUELLES UNIX LABOR

(( SEHR VIEL / EINIGES / GEHT SO / WENIG / NICHTS ))

Sehr viel:	9	(32%)	oooooooooooo
Einiges:	12	(42%)	oooooooooooooooooooo
Geht so:	7	(25%)	oooooooooooo
Wenig:	0	(0%)	
Nichts:	0	(0%)	
Summe:	28	(100%)	

Modus: Einiges (12)  
Median: Einiges (14)

## 5. ANALYSE DER FH-RECHNER

(( SEHR VIEL / EINIGES / GEHT SO / WENIG / NICHTS ))

Sehr viel:	3	(10%)	ooo
Einiges:	12	(42%)	ooooooooooooo
Geht so:	10	(35%)	ooooooooooooo
Wenig:	3	(10%)	ooo
Nichts:	0	(0%)	
Summe:	28	(100%)	

Modus: Einiges (12)

Median: Einiges (14)

## 6. ANALYSE EIGENER RECHNER

(( SEHR VIEL / EINIGES / GEHT SO / WENIG / NICHTS ))

Sehr viel:	15	(53%)	ooooooooooooooooo
Einiges:	10	(35%)	ooooooooooooo
Geht so:	1	(3%)	o
Wenig:	1	(3%)	o
Nichts:	1	(3%)	o
Summe:	28	(100%)	

Modus: Sehr viel (15)

Median: Sehr viel (14)

## 7. BÜCHER

(( SEHR VIEL / EINIGES / GEHT SO / WENIG / NICHTS ))

Sehr viel:	6	(21%)	oooooo
Einiges:	9	(32%)	oooooooooo
Geht so:	4	(14%)	oooo
Wenig:	7	(25%)	ooooooo
Nichts:	2	(7%)	oo
Summe:	28	(100%)	

Modus: Einiges (9)

Median: Einiges (14)

## 8. ONLINE-INFORMATIONEN

(( SEHR VIEL / EINIGES / GEHT SO / WENIG / NICHTS ))

Sehr viel:	12	(42%)	ooooooooooooo
Einiges:	12	(42%)	ooooooooooooo
Geht so:	2	(7%)	oo
Wenig:	2	(7%)	oo
Nichts:	0	(0%)	
Summe:	28	(100%)	

Modus: Sehr viel (12)  
Median: Einiges (14)

#### EINBINDUNG DES LABORS IN DIE VORLESUNG

##### 9. IST DAS VIRTUELLE UNIX LABOR GENERELL EINE SINNVOLLE ERGÄNZUNG ZUR VORLESUNG?

((SEHR SINNVOLL / SINNVOLL / GEHT SO / WENIG SINNVOLL / UNSINNVOLL))

Sehr sinnvoll:	15	(53%)	ooooooooooooo
Sinnvoll:	13	(46%)	ooooooooooooo
Geht so:	0	(0%)	
Wenig sinnvoll:	0	(0%)	
Unsinnig:	0	(0%)	
Summe:	28	(100%)	

Modus: Sehr sinnvoll (15)  
Median: Sehr sinnvoll (14)

##### 10. WIE EMPFANDEN SIE DEN NUTZEN DES VIRTUELLEN UNIX LABORS?

(( SEHR POSITIV / POSITIV / NEUTRAL / NEGATIV / SEHR NEGATIV ))

sehr positiv:	8	(28%)	ooooooo
positiv:	15	(53%)	ooooooooooooo
neutral:	4	(14%)	oooo
negativ:	1	(3%)	o
sehr negativ:	0	(0%)	
Summe:	28	(100%)	

Modus: positiv (15)  
Median: positiv (14)

#### BENUTZUNG DES VIRTUELLEN UNIX LABORS

##### 11. WAR DAS SYSTEM EINFACH ZU BENUTZEN?

(( SEHR EINFACH / EINFACH / UMSTAENDLICH / SEHR UMSTAENDLICH ))



Sehr einfach:	4	(14%)	oooo
Einfach:	17	(60%)	ooooooooooooooooooooo
Umstaendlich:	7	(25%)	ooooooo
Sehr umstaendlich:	0	(0%)	
Summe:	28	(100%)	

Modus: Einfach (17)  
 Median: Einfach (14)

12. WAREN GENÜGEND ÜBUNGSTERMINE ZUR AUSWAHL?  
 (( ZU VIELE / GENÜGEND / ZU WENIGE ))

Zu viele:	0	(0%)	
Genügend:	26	(92%)	ooooooooooooooooooooo
Zu wenige:	2	(7%)	oo
Summe:	28	(100%)	

Modus: Genügend (26)  
 Median: Genügend (14)

13. VON WO AUS HABEN SIE AUF DIE ÜBUNGSRECHNER ZUGEGRIFFEN?  
 (( ZUHAUSE / FH ODER UNI / SONSTIGE ))

Zuhause:	20	(71%)	ooooooooooooooooooooo
FH oder Uni:	8	(28%)	ooooooo
Sonstige:	0	(0%)	
Summe:	28	(100%)	

Modus: Zuhause (20)  
 Median: n/a

14. VON WELCHEM BETRIEBSSYSTEM AUS HABEN SIE DIE ÜBUNGEN GEMACHT?  
 (( WINDOWS / UNIX (LINUX, ...) / SONSTIGES ))

Windows:	7	(25%)	ooooooo
Unix (Linux, ...):	21	(75%)	ooooooooooooooooooooo
Sonstiges:	0	(0%)	
Summe:	28	(100%)	

Modus: Unix (Linux, ...) (21)  
 Median: n/a

15. WAR DIE AUSGANGSKONFIGURATION DER RECHNER AUSREICHEND, DAMIT SIE DIE ÜBUNG BEARBEITEN KONNTEN? (MUSSTEN SIE VIELE VORBEREITUNGEN TREFFEN, UM MIT DER EIGENTLICHEN ÜBUNG BEGINNEN ZU KÖNNEN ODER WAR DIE VORHANDENE UEBUNGSUMGEBUNG NACH ALL IHREN WÜNSCHEN EINGERICHTET?)  
 (( ZU SPARTANISCH / ETWAS SPARTANISCH / GEHT SO / KOMFORTABEL / SEHR KOMFORTABEL ))

Zu spartanisch:	0	(0%)	
etwas spartanisch:	11	(39%)	ooooooooooooo
Geht so:	14	(50%)	ooooooooooooooooo
Komfortabel:	3	(10%)	ooo
Sehr komfortabel:	0	(0%)	
Summe:	28	(100%)	

Modus: Geht so (14)  
Median: Geht so (14)

#### ALLGEMEINES ZUM ÜBUNGSVERLAUF:

16. HABEN SIE DIE ÜBUNG ALLEINE ODER IN EINER GRUPPE ABSOLVIERT?  
(BITTE JEDES MITGLIED DER GRUPPE DIESEN FRAGEBOGEN AUSFÜLLEN!)  
(( ALLEINE / ZU ZWEIT / ZU DRITT / ZU VIERT ))

Alleine:	17	(62%)	ooooooooooooooooooooo
Zu zweit:	5	(19%)	ooooo
Zu dritt:	5	(19%)	ooooo
Zu viert:	0	(0%)	
Summe:	27	(100%)	

Modus: Alleine (17)  
Median: Alleine (14)

17. WAR DIE ZEIT FÜR DAS ABSOLVIEREN DER ÜBUNG ZU KURZ/ZU LANG?  
(( VIEL ZU KURZ / ZU KURZ / GENAU RICHTIG / ZU LANG / VIEL ZU LANG ))

Viel zu kurz:	1	(3%)	o
Zu kurz:	15	(53%)	ooooooooooooooooooooo
Genau richtig:	11	(39%)	ooooooooooooo
Zu lang:	0	(0%)	
Viel zu lang:	0	(0%)	
Summe:	27	(100%)	

Modus: Zu kurz (15)  
Median: Zu kurz (14)

18. FANDEN SIE DIE AUFGABENSTELLUNG ZU DETAILIERT ODER HÄTTEN SIE SICH MEHR INFORMATIONEN ZUM BEARBEITEN DER AUFGABE GEWÜNSCHT?  
(( VIEL ZU VIEL INFORMATION / ZUVIEL INFORMATION / GENAU RICHTIG / BITTE ETWAS MEHR INFORMATIONEN / BITTE VIEL MEHR INFORMATIONEN ))

Viel zu viel Information:	0	(0%)	
Zuviel Information:	2	(7%)	oo
Genau richtig:	9	(32%)	oooooooooooo
Bitte etwas mehr Informationen:	16	(57%)	oooooooooooooooooooo
Bitte viel mehr Informationen:	0	(0%)	
Summe:	27	(100%)	

Modus: Bitte etwas mehr Informationen (16)  
 Median: Bitte etwas mehr Informationen (14)

19. HÄTTEN SIE SICH WÄHREND DER ÜBUNG GEWÜNSCHT, UM HILFE ANFRAGEN ZU KÖNNEN, UM WEITERZUKOMMEN?  
 (( JA, HILFE WÄRE GUT GEWESEN / NEIN, BIN ALLEINE KLARGEKOMMEN ))

Ja (...):	19	(70%)	oooooooooooooooooooo
Nein (...):	8	(30%)	oooooooo
Summe:	27	(100%)	

Modus: Ja (...) (19)  
 Median: n/a

20. HÄTTEN SIE SICH GEWÜNSCHT DASS DAS SYSTEM AUTOMATISCH PROBLEME ERKENNT UND HILFESTELLUNGEN ANBIETET?  
 (( JA / NEIN ))

ja:	22	(81%)	oooooooooooooooooooo
nein:	5	(19%)	ooooo
Summe:	27	(100%)	

Modus: ja (22)  
 Median: n/a

**ÜBUNGSVERLAUF: WIEVIEL HABEN DIE FOLGENDEN HILFSMITTEL ZUM BEARBEITEN DER ÜBUNGEN DES VIRTUELLEN UNIX LABORS BEIGETRAGEN?**

21. BESUCH DER VORLESUNG  
 (( NICHT GENUTZT / SEHR VIEL / EINIGES / GEHT SO / WENIG / NICHTS ))

Nicht genutzt:	0	(0%)	
Sehr viel:	8	(28%)	ooooooooo
Einiges:	12	(42%)	ooooooooooooo
Geht so:	6	(21%)	oooooo
Wenig:	1	(3%)	o
Nichts:	0	(0%)	
Summe:	27	(100%)	

Modus: Einiges (12)  
Median: Einiges (14)

## 22. SCRIPT ZUR VORLESUNG

(( NICHT GENUTZT / SEHR VIEL / EINIGES / GEHT SO / WENIG / NICHTS ))

Nicht genutzt:	0	(0%)	
Sehr viel:	7	(25%)	ooooooo
Einiges:	14	(50%)	ooooooooooooo
Geht so:	5	(17%)	ooooo
Wenig:	1	(3%)	o
Nichts:	0	(0%)	
Summe:	27	(100%)	

Modus: Einiges (14)  
Median: Einiges (14)

## 23. ÜBUNGEN ZUR VORLESUNG

(( NICHT GENUTZT / SEHR VIEL / EINIGES / GEHT SO / WENIG / NICHTS ))

Nicht genutzt:	3	(10%)	ooo
Sehr viel:	5	(17%)	ooooo
Einiges:	8	(28%)	ooooooooo
Geht so:	9	(32%)	ooooooooo
Wenig:	2	(7%)	oo
Nichts:	0	(0%)	
Summe:	27	(100%)	

Modus: Geht so (9)  
Median: Einiges (14)

## 24. ANALYSE DER FH-RECHNER

(( NICHT GENUTZT / SEHR VIEL / EINIGES / GEHT SO / WENIG / NICHTS ))

Nicht genutzt:	3	(10%)	ooo
Sehr viel:	2	(7%)	oo
Einiges:	8	(28%)	oooooooooo
Geht so:	8	(28%)	oooooooooo
Wenig:	5	(17%)	ooooo
Nichts:	1	(3%)	o
Summe:	27	(100%)	

Modus: Einiges (8)  
Median: Geht so (14)

## 25. ANALYSE EIGENER RECHNER

(( NICHT GENUTZT / SEHR VIEL / EINIGES / GEHT SO / WENIG / NICHTS ))

Nicht genutzt:	2	(7%)	oo
Sehr viel:	4	(14%)	oooo
Einiges:	10	(35%)	oooooooooooo
Geht so:	6	(21%)	oooooo
Wenig:	4	(14%)	oooo
Nichts:	1	(3%)	o
Summe:	27	(100%)	

Modus: Einiges (10)  
Median: Einiges (14)

## 26. BÜCHER

(( NICHT GENUTZT / SEHR VIEL / EINIGES / GEHT SO / WENIG / NICHTS ))

Nicht genutzt:	11	(39%)	ooooooooooooo
Sehr viel:	2	(7%)	oo
Einiges:	4	(14%)	oooo
Geht so:	6	(21%)	oooooo
Wenig:	3	(10%)	ooo
Nichts:	1	(3%)	o
Summe:	27	(100%)	

Modus: Nicht genutzt (11)  
Median: Einiges (14)

## 27. ONLINE-INFORMATIONEN

(( NICHT GENUTZT / SEHR VIEL / EINIGES / GEHT SO / WENIG / NICHTS ))

Nicht genutzt:	0	(0%)	
Sehr viel:	19	(67%)	oooooooooooooooooooo
Einiges:	5	(17%)	ooooo
Geht so:	2	(7%)	oo
Wenig:	1	(3%)	o
Nichts:	0	(0%)	
Summe:	27	(100%)	

Modus: Sehr viel (19)  
Median: Sehr viel (14)

**ÜBUNGSVERLAUF: WIEVIEL HAT DER BESUCH DER VORLESUNG ZUM BEARBEITEN DER ÜBUNGEN DES VIRTUELLEN UNIX LABORS BEIGETRAGEN?**

28. WAR DIE VORLESUNG HILFREICH BEI DER BEARBEITUNG DER AUFGABE ZUM NIS SERVER:

(( SEHR / ETWAS / GEHT SO / WENIG / NICHTS ))

Sehr:	4	(14%)	oooo
Etwas:	12	(42%)	oooooooooooooooo
Geht so:	8	(28%)	ooooooooo
Wenig:	2	(7%)	oo
Nichts:	0	(0%)	
Summe:	26	(100%)	

Modus: Etwas (12)  
Median: Etwas (14)

29. WAR DIE VORLESUNG HILFREICH BEI DER BEARBEITUNG DER AUFGABE ZUM NIS CLIENT:

(( SEHR / ETWAS / GEHT SO / WENIG / NICHTS ))

Sehr:	4	(14%)	oooo
Etwas:	13	(46%)	oooooooooooooooo
Geht so:	8	(28%)	ooooooooo
Wenig:	1	(3%)	o
Nichts:	0	(0%)	
Summe:	26	(100%)	

Modus: Etwas (13)  
Median: Etwas (14)

30. WAR DIE VORLESUNG HILFREICH BEI DER BEARBEITUNG DER AUFGABE ZUM NFS SERVER:

(( SEHR / ETWAS / GEHT SO / WENIG / NICHTS ))

Sehr:	6	(21%)	oooooo
Etwas:	13	(46%)	ooooooooooooo
Geht so:	5	(17%)	ooooo
Wenig:	1	(3%)	o
Nichts:	0	(0%)	
<hr/>			
Summe:	25	(100%)	

Modus: Etwas (13)  
Median: Etwas (14)

31. WAR DIE VORLESUNG HILFREICH BEI DER BEARBEITUNG DER AUFGABE ZUM NFS CLIENT:

(( SEHR / ETWAS / GEHT SO / WENIG / NICHTS ))

Sehr:	7	(25%)	ooooooo
Etwas:	12	(42%)	ooooooooooooo
Geht so:	6	(21%)	ooooooo
Wenig:	1	(3%)	o
Nichts:	0	(0%)	
<hr/>			
Summe:	26	(100%)	

Modus: Etwas (12)  
Median: Etwas (14)

32. WAR DIE VORLESUNG HILFREICH FÜR DEN UMGANG MIT SOLARIS ALLGEMEIN?

(( SEHR / ETWAS / GEHT SO / WENIG / NICHTS ))

Sehr:	10	(35%)	ooooooooooo
Etwas:	9	(32%)	ooooooooooo
Geht so:	4	(14%)	ooooo
Wenig:	2	(7%)	oo
Nichts:	0	(0%)	
<hr/>			
Summe:	25	(100%)	

Modus: Sehr (10)  
Median: Etwas (14)

33. WAR DIE VORLESUNG HILFREICH FÜR DEN UMGANG MIT NETBSD ALLGEMEIN?

(( SEHR / ETWAS / GEHT SO / WENIG / NICHTS ))

Sehr:	7	(25%)	ooooooo
Etwas:	12	(42%)	ooooooooooooo
Geht so:	6	(21%)	ooooooo
Wenig:	1	(3%)	o
Nichts:	0	(0%)	
<hr/>			
Summe:	26	(100%)	

Modus: Etwas (12)  
Median: Etwas (14)

34. KONNTEN SIE DEN NICHT DIREKT VERMITTELTEN STOFF AUS DEN BEREITGESTELLTEN INFORMATIONEN (VORGEHENSWEISEN, ALLGEMEINE INFORMATIONEN ÜBER SYSTEME, VORGEHEN ZUR ANALYSE) ERMITTELN?

(( SEHR / ETWAS / GEHT SO / WENIG / NICHTS ))

Sehr:	1	(3%)	o
Etwas:	14	(50%)	ooooooooooooo
Geht so:	9	(32%)	ooooooooo
Wenig:	2	(7%)	oo
Nichts:	0	(0%)	
<hr/>			
Summe:	26	(100%)	

Modus: Etwas (14)  
Median: Etwas (14)

**ÜBUNGSVERLAUF: WIEVIEL HAT DIE BENUTZUNG DES SCRIPTS ZUM BEARBEITEN DER ÜBUNGEN DES VIRTUELLEN UNIX LABORS BEIGETRAGEN?**

35. WAR DAS SCRIPT HILFREICH BEI DER BEARBEITUNG DER AUFGABE ZUM NIS SERVER:

(( SEHR / ETWAS / GEHT SO / WENIG / NICHTS ))

Sehr:	3	(10%)	ooo
Etwas:	14	(50%)	ooooooooooooo
Geht so:	6	(21%)	ooooooo
Wenig:	4	(14%)	oooo
Nichts:	0	(0%)	
<hr/>			
Summe:	27	(100%)	

Modus: Etwas (14)  
Median: Etwas (14)

36. WAR DAS SCRIPT HILFREICH BEI DER BEARBEITUNG DER AUFGABE ZUM NIS CLIENT:

(( SEHR / ETWAS / GEHT SO / WENIG / NICHTS ))



Sehr:	4	(14%)	oooo
Etwas:	12	(42%)	oooooooooooooo
Geht so:	8	(28%)	oooooooooo
Wenig:	3	(10%)	ooo
Nichts:	0	(0%)	
<hr/>			
Summe:	27	(100%)	

Modus: Etwas (12)  
Median: Etwas (14)

37. WAR DAS SCRIPT HILFREICH BEI DER BEARBEITUNG DER AUFGABE ZUM NFS SERVER:  
( ( SEHR / ETWAS / GEHT SO / WENIG / NICHTS ) )

Sehr:	4	(14%)	oooo
Etwas:	15	(53%)	oooooooooooooooooo
Geht so:	5	(17%)	oooooo
Wenig:	3	(10%)	ooo
Nichts:	0	(0%)	
<hr/>			
Summe:	27	(100%)	

Modus: Etwas (15)  
Median: Etwas (14)

38. WAR DAS SCRIPT HILFREICH BEI DER BEARBEITUNG DER AUFGABE ZUM NFS CLIENT:  
( ( SEHR / ETWAS / GEHT SO / WENIG / NICHTS ) )

Sehr:	4	(14%)	oooo
Etwas:	15	(53%)	oooooooooooooooooo
Geht so:	6	(21%)	oooooo
Wenig:	2	(7%)	oo
Nichts:	0	(0%)	
<hr/>			
Summe:	27	(100%)	

Modus: Etwas (15)  
Median: Etwas (14)

39. WAR DAS SCRIPT HILFREICH FÜR DEN UMGANG MIT SOLARIS ALLGEMEIN:  
( ( SEHR / ETWAS / GEHT SO / WENIG / NICHTS ) )

Sehr:	8	(28%)	oooooooo
Etwas:	11	(39%)	oooooooooo
Geht so:	6	(21%)	oooooo
Wenig:	2	(7%)	oo
Nichts:	0	(0%)	
<hr/>			
Summe:	27	(100%)	

Modus: Etwas (11)  
Median: Etwas (14)

40. WAR DAS SCRIPT HILFREICH FÜR DEN UMGANG MIT NETBSD ALLGEMEIN:  
( ( SEHR / ETWAS / GEHT SO / WENIG / NICHTS ) )

Sehr:	4	(14%)	oooo
Etwas:	13	(46%)	oooooooooo
Geht so:	7	(25%)	oooooo
Wenig:	3	(10%)	ooo
Nichts:	0	(0%)	
<hr/>			
Summe:	28	(100%)	

Modus: Etwas (13)  
Median: Etwas (14)

### FEEDBACK NACH DER ÜBUNG

41. WAREN DIE INFORMATIONEN DER AUSWERTUNG DETAILIERT GENUG, UM  
ETWAIGE FEHLER NACHVOLLZIEHEN ZU KÖNNEN?  
( ( JA, ICH KONNTE AUS MEINEN FEHLERN LERNEN / NEIN, ICH WEISS IM-  
MER NOCH NICHT WAS FALSCH WAR ) )

Ja (...):	17	(60%)	oooooooooooooooooo
Nein (...):	9	(32%)	oooooooooo
<hr/>			
Summe:	26	(100%)	

Modus: Ja (...) (17)  
Median: n/a

### ANGABEN ZUR PERSON

42. INTERESSE AM STUDIUM ALLGEMEIN  
( ( SEHR GROSS / GROSS / MITTEL / WENIGER / GAR NICHT ) )

Sehr gross:	21	(75%)	oooooooooooooooooooooo
Gross:	6	(21%)	oooooo
Mittel:	0	(0%)	
Weniger:	0	(0%)	
Gar nicht:	0	(0%)	
<hr/>			
Summe:	27	(100%)	

Modus: Sehr gross (21)  
 Median: Sehr gross (14)

43. INTERESSE AM THEMA "SYSTEMADMINISTRATION"  
 (( SEHR GROSS / GROSS / MITTEL / WENIGER / GAR NICHT ))

Sehr gross:	15	(53%)	ooooooooooooooooooooo
Gross:	7	(25%)	ooooooo
Mittel:	5	(17%)	ooooo
Weniger:	0	(0%)	
Gar nicht:	0	(0%)	
<hr/>			
Summe:	28	(100%)	

Modus: Sehr gross (15)  
 Median: Sehr gross (14)

44. INTERESSE AN "UNIX" (LINUX, SOLARIS, NETBSD, ...)  
 (( SEHR GROSS / GROSS / MITTEL / WENIGER / GAR NICHT ))

Sehr gross:	16	(57%)	ooooooooooooooooooooo
Gross:	10	(35%)	oooooooooo
Mittel:	1	(3%)	o
Weniger:	0	(0%)	
Gar nicht:	0	(0%)	
<hr/>			
Summe:	27	(100%)	

Modus: Sehr gross (16)  
 Median: Sehr gross (14)

45. WIE SCHÄTZEN SIE DIE WICHTIGKEIT DES TEILGEBIETS "NIS" EIN?  
 (( SEHR GROSS / GROSS / MITTEL / WENIGER / GAR NICHT ))

Sehr gross:	2	(7%)	oo
Gross:	6	(21%)	oooooo
Mittel:	12	(42%)	oooooooooooooo
Weniger:	6	(21%)	oooooo
Gar nicht:	1	(3%)	o
<hr/>			
Summe:	27	(100%)	

Modus: Mittel (12)  
 Median: Mittel (14)

46. WIE SCHÄTZEN SIE DIE WICHTIGKEIT DES TEILGEBIETS "NFS" EIN?  
 (( SEHR GROSS / GROSS / MITTEL / WENIGER / GAR NICHT ))

Sehr gross:	3	(10%)	ooo
Gross:	14	(50%)	oooooooooooooooooo
Mittel:	8	(28%)	ooooooooo
Weniger:	2	(7%)	oo
Gar nicht:	0	(0%)	
<hr/>			
Summe:	27	(100%)	

Modus: Gross (14)  
 Median: Gross (14)

47. WIEVIELE VON 10 VORLESUNGSSTUNDEN HABEN SIE BESUCHT?  
 (( 0-3 / 4-8 / 9-10 ))

0-3:	1	(3%)	o
4-8:	3	(10%)	ooo
9-10:	23	(82%)	ooooooooooooooooooooo
<hr/>			
Summe:	27	(100%)	

Modus: 9-10 (23)  
 Median: 9-10 (14)

48. HABEN SIE BISHER AUSSERHALB DER VORLESUNG MIT SYSTEMVERWALTUNG ZU TUN (PRAKTIKUM, RECHNER ZU HAUSE, ...)?  
 (( JA / NEIN ))

Ja:	22	(78%)	ooooooooooooooooooooo
Nein:	5	(17%)	ooooo
<hr/>			
Summe:	27	(100%)	

Modus: Ja (22)  
 Median: n/a

49. WENN JA, MIT WELCHEN BETRIEBSSYSTEMEN ?  
 (( WINDOWS / LINUX / BSD / SOLARIS / AIX / NOVELL / SONSTIGE ))

Windows:	5	(17%)	ooooo
Linux:	16	(57%)	ooooooooooooooooooooo
BSD:	0	(0%)	
Solaris:	1	(3%)	o
AIX:	0	(0%)	
Novell:	1	(3%)	o
sonstige:	0	(0%)	
<hr/>			
Summe:	23	(100%)	

Modus: Linux (16)  
 Median: n/a

## 50. STUDIENSEMESTER

(( UNTER 4 / 4 / 5 / 6 / 7 / 8 / ÜBER 8 ))

unter 4:	1	(3%)	o
4:	23	(82%)	ooooooooooooooooooooooooooooo
5:	0	(0%)	
6:	0	(0%)	
7:	0	(0%)	
8:	2	(7%)	oo
über 8:	1	(3%)	o
<hr/>			
Summe:	27	(100%)	

Modus: 4 (23)

Median: n/a

## 51. GESCHLECHT

(( M / W ))

m:	26	(92%)	ooooooooooooooooooooooooooooo
w:	1	(3%)	o
<hr/>			
Summe:	27	(100%)	

Modus: m (26)

Median: n/a

## 52. HABEN SIE SONSTIGE ANMERKUNGEN? BITTEN GEBEN SIE IHR FEEDBACK PER EMAIL ODER HIER AB!

- Die Zeit ist teilweise etwas kurz bemessen, gerade wenn man vor einem Problem steht. Bei den Diensten die zu starten sind wären nähere Informationen hilfreich (z.B. dass der RPCBIND erforderlich ist für NIS)  
Bei der NIS Übung wäre ein Hinweis auf den korrekten Ablauf trotz Fehler beim make im Skript/Übungsaufgabe hilfreich.  
Die Geschwindigkeit der Rechner lässt zu wünschen übrig :)  
Dieses Formular lässt keine Mehrfachauswahl bei Betriebssystemen zu :)
- Bei der Auswertung des Virtuellen Unix Labors wäre es evtl. hilfreich, wenn bei den falschen Antworten ein kleiner Lösungshinweis vorhanden wäre.
- Ok, der NetBSD Rechner hat a bisserl oft gehangen, aber des kann ja vorkommen, schön wärs wenn die bash vorinstalliert wär, ansonsten is des Labor wunderbar.
- wird lang :)  
hier im fragebogen wären manchmal mehrfachnennungen hilfreich (z.b. 49).  
zu 17: für nfs reichen auch beim ersten mal, wenn man sich vorbereitet hat, 60min aus, für nis wären 120min nicht schlecht, da diese übung schon etwas komplizierter ist, und auch länger in der bearbeitung.

zu 19: vielleicht eine idee, da es aber das internet als nachschlagewerk gibt, sollte es auch dabei bleiben, damit bleibt die übung sehr realistisch.

zu 20: auf keinen fall. damit würde man, wie ich finde, dieses realszenario zu einem art geführten tutorial herabsetzen. dadurch würde erstens der lerneffekt etwas verlorengehen, und zweitens wäre es auch nicht mehr so interessant, weil man dann einfach mal trial&error machen kann, und das würde nicht mehr zu einem "real-live-szenario" passen.

hängt natürlich davon ab, wieweit diese live-unterstützung geht ...

zu 41: manchmal dachte ich bei der nis, daß die auswertung etwas obskur wirkt, manche punkte wurden auf nein gesetzt, obwohl wir sie eigentlich gemacht hatten ...

problem hierbei ist natürlich folgendes: umso mehr informationen man hier bei der auswertung "preisgibt", desto eher kann man beim zweiten versuch einer übung die lösung zum teil darauf ausrichten, d.h. genau das erfüllen, wonach die automatisch auswertung sucht. obwohl die auswertung an sich schon eine recht diffizile sache ist ...

sonst hat die übung schon spaß gemacht ... obwohl bei nis schon manchmal etwas verzweigung dabei war :) jedenfalls beim ersten versuch ...

- I don't know if this is possible, but it would be good having the virtual servers allways available to access as root whenever we want in order to be able to practise more and when we wanted.
- - Verfüegbarkeit des vulab laesst sehr zu wuenschen uebrig.
  - Zeit fuer die NIS-Uebung ist etwas zu knapp (Reboot erfolgte waehrend des finalen 'make' in /var/yp zum Update der group und hosts Maps fuer Eintrag von ypuser in Group 'wheel' und 'tab' in NIS hosts \*grrr\*).
  - NIS Client unter NetBSD; fehlendes /etc/domainname ist nicht falsch wenn der NIS Domainname z.B. via 'domainname="vulab"' in /etc/rc.conf gesetzt wird.
- Anmerkung zum Skript: Eine Druckversion wäre super, bei der keine Grafiken abgeschnitten werden.
 

zum VULAB:

Für die ersten Übungsdurchläufe wäre mehr Zeit nötig, wenn man keinen Plan hat (so wie ich), dann muss man ewig viel googeln um herauszufinden, was genau man machen muss.

Vielleicht wäre es noch möglich, eine Überprüfung einzubauen, ob auf die Rechner zugegriffen werden kann und die mögliche Fehler gleich weiter gibt.

Sonst finde ich das Konzept von VULAB echt genial.
- bei frage 49. sollte ne mehrfach auswahl möglich sein. eine andwort (Linux und Windows)
- Im Großen und Ganzen eine sehr lehrreiche Vorlesung.
 

Eine Verbesserung der Übungen und des VULAB's wäre jedoch wünschenswert.

- Beim Fragebogen den Abschnitt “Allgemeines zum Übungsverlauf” in NIS und NFS aufteilen, da (nicht nur bei mir) Unterschiede bei Bearbeitungszeit/-aufwand usw. waren.  
Der (BSD)Teufel steckt im Detail ;)
- Das VULab ist grundsätzlich eine sehr gelungene Einrichtung, weil man hier mit mehr oder weniger Begleitung (allein schon durch die Aufgaben) den Umgang mit UNIX lernt. Für die Zukunft wäre noch hilfreich, dass die Rechner während der Übung resetted, also in ihren Ausgangszustand gebracht werden können (sofern dies möglich ist), da es mir auch passiert ist, dass ich nach einer Fehlkonfiguration nicht mehr auf den Client zugreifen konnte. es weiteren könnte ich mir auch noch weitere Übungen vorstellen, die häufige Tätigkeiten im UNIX-Administrations-Bereich zum Thema haben. (Selbst Kompilieren und Installieren von Software, Einrichten einer neuen Kernel-Version,...)  
Ich weiß allerdings nicht inwieweit das auf dem System verwirklicht ist.

### C.3 Exercise results: selected SQL queries and results

The following list of PL/SQL queries to the PostgreSQL database are used in evaluation of Virtual Unix Lab exercise results in section 7.2:

1. Determine number of valid NIS exercises:

```
SELECT count(*)
FROM ( SELECT extract(hours from endzeit-startzeit)*60
        +extract(minutes from endzeit-startzeit) AS dauer
      FROM buchungen, benutzer
      WHERE buchungen.user_id=benutzer.user_id
            AND login != 'feyrer'
            AND NOT (endzeit-startzeit>='1:40'
                    OR endzeit<startzeit)
            AND uebung_id IN ('nis')
            AND buchungen.datum >= '2004-03-15'
            AND buchungen.datum <= '2004-07-25'
      ORDER BY dauer
    ) AS x;
```

2. Determine ending times of NIS exercises:

```
SELECT extract(hours from endzeit-startzeit)*60
        +extract(minutes from endzeit-startzeit) AS dauer
FROM buchungen, benutzer
WHERE buchungen.user_id=benutzer.user_id
      AND NOT (endzeit-startzeit>='1:40'
              OR endzeit<startzeit)
      AND uebung_id in ('nis')
      AND login != 'feyrer'
      AND datum >= '2004-03-15'
      AND datum <= '2004-07-25'
ORDER BY dauer;
```

## 3. Count number of valid NFS exercises:

```

SELECT count(*)
FROM ( SELECT extract(hours from endzeit-startzeit)*60
        +extract(minutes from endzeit-startzeit) AS dauer
      FROM buchungen, benutzer
      WHERE buchungen.user_id=benutzer.user_id
            AND NOT (endzeit-startzeit>='1:40'
                    OR endzeit<startzeit)
            AND uebung_id IN ('nfs')
            AND login != 'feyrer'
            AND datum >= '2004-03-15'
            AND datum <= '2004-07-25'
      ORDER BY dauer
    ) AS x;

```

## 4. Determine ending times of NFS exercises:

```

SELECT extract(hours from endzeit-startzeit)*60
        +extract(minutes from endzeit-startzeit) AS dauer
FROM buchungen, benutzer
WHERE buchungen.user_id=benutzer.user_id
      AND NOT (endzeit-startzeit>='1:40'
              OR endzeit<startzeit)
      AND uebung_id IN ('nfs')
      AND login != 'feyrer'
      AND datum >= '2004-03-15'
      AND datum <= '2004-07-25'
ORDER BY dauer;

```

## 5. Which user booked most exercises:

```

SELECT login, count(*)
FROM buchungen,benutzer
WHERE buchungen.user_id=benutzer.user_id
      AND login != 'feyrer'
      AND datum >= '2004-03-15'
      AND datum <= '2004-07-25'
GROUP BY login
ORDER BY count desc;

```

## Results:

login	count
pap34148	12
punky@schweinemarmelade.de	11
andreas.fischer@stud.fh-regensburg.de	10
wes35369	7
urk35769	7
walter.kern@stud.fh-regensburg.de	7
martina.heindl@stud.fh-regensburg.de	6
meindlth@asamnet.de	6
marius.strobl@stud.fh-regensburg.de	6
ramon@pangea.org	5
josef.scheuer@stud.fh-regensburg.de	5
benjamin.grundstein@stud.fh-regensburg.de	5
markus@fuchsi.de	5
gep31844	4
ham32330	4
trm35740	4
luf33607	4



### C.3. EXERCISE RESULTS: SELECTED SQL QUERIES AND RESULTS 373

wem35832	4
petach@gmx.de	4
wachenroeder@gmx.de	3
Dragoonsmail@gmx.de	3
ch.marchl@gmx.de	3
tdirscherl@onlinehome.de	3
Klausl.Rathmacher@stud.fh-regensburg.de	2
andreas.pollinger@stud.fh-regensburg.de	2
bernhard.gammel@stud.fh-regensburg.de	2
jingjing	1
(27 rows)	

#### 6. Which user booked most exercises, split by exercise:

```
SELECT login, uebung_id, count(*)
FROM buchungen,benutzer
WHERE buchungen.user_id=benutzer.user_id
AND login != 'feyrer'
AND datum >= '2004-03-15'
AND datum <= '2004-07-25'
GROUP BY login, uebung_id
ORDER BY count desc;
```

#### Results:

login	uebung_id	count
andreas.fischer@stud.fh-regensburg.de	nis	8
pap34148	nis	6
pap34148	nfs	6
walter.kern@stud.fh-regensburg.de	nis	5
urk35769	nis	5
wes35369	nis	4
meindlth@asamnet.de	nfs	4
markus@fuchsi.de	nis	4
punky@schweinemarmelade.de	netbsd	4
punky@schweinemarmelade.de	nis	4
wes35369	nfs	3
benjamin.grundstein@stud.fh-regensburg.de	nfs	3
josef.scheuer@stud.fh-regensburg.de	nfs	3
gep31844	nfs	3
ramon@pangea.org	nis	3
ham22330	nis	3
martina.heindl@stud.fh-regensburg.de	nfs	3
wem35832	nfs	3
marius.strobl@stud.fh-regensburg.de	nfs	3
trm35740	nis	3
punky@schweinemarmelade.de	nfs	3
petach@gmx.de	nis	3
walter.kern@stud.fh-regensburg.de	nfs	2
ramon@pangea.org	nfs	2
andreas.fischer@stud.fh-regensburg.de	nfs	2
Dragoonsmail@gmx.de	nis	2
josef.scheuer@stud.fh-regensburg.de	nis	2
ch.marchl@gmx.de	nis	2
benjamin.grundstein@stud.fh-regensburg.de	nis	2
wachenroeder@gmx.de	nfs	2
meindlth@asamnet.de	nis	2
martina.heindl@stud.fh-regensburg.de	nis	2
luf33607	nfs	2
urk35769	nfs	2
tdirscherl@onlinehome.de	nfs	2
luf33607	nis	2
marius.strobl@stud.fh-regensburg.de	nis	2
marius.strobl@stud.fh-regensburg.de	netbsd	1

ch.marchl@gmx.de	nfs	1
wachenroeder@gmx.de	nfs	1
trm35740	nfs	1
bernhard.gammel@stud.fh-regensburg.de	nfs	1
martina.heindl@stud.fh-regensburg.de	netbsd	1
tdirscherl@onlinehome.de	nfs	1
markus@fuchsi.de	nfs	1
andreas.pollinger@stud.fh-regensburg.de	nfs	1
petach@gmx.de	nfs	1
Klausl.Rathmacher@stud.fh-regensburg.de	nfs	1
wem35832	nfs	1
bernhard.gammel@stud.fh-regensburg.de	nfs	1
Klausl.Rathmacher@stud.fh-regensburg.de	nfs	1
ham32330	nfs	1
andreas.pollinger@stud.fh-regensburg.de	nfs	1
jingjing	nfs	1
gep31844	nfs	1
Dragoonsmail@gmx.de	nfs	1

(56 rows)

7. Which exercise was booked most, per exercise:

```
SELECT count(*), uebung_id
FROM buchungen, benutzer
WHERE buchungen.user_id=benutzer.user_id
AND login!='feyrer'
AND datum >= '2004-03-15'
AND datum <= '2004-07-25'
GROUP BY uebung_id;
```

8. Display how often the NIS exercise was booked:

```
SELECT count(*)
FROM ( SELECT distinct buchungen.user_id, uebung_id
      FROM buchungen, benutzer
      WHERE buchungen.user_id=benutzer.user_id
        AND uebung_id='nis'
        AND login!='feyrer'
        AND datum >= '2004-03-15'
        AND datum <= '2004-07-25'
      ) AS foo;
```

9. Display how often the NFS exercise was booked:

```
SELECT count(*)
FROM ( SELECT distinct buchungen.user_id, uebung_id
      FROM buchungen, benutzer
      WHERE buchungen.user_id=benutzer.user_id
        AND uebung_id='nfs'
        AND login!='feyrer'
        AND datum >= '2004-03-15'
        AND datum <= '2004-07-25'
      ) AS foo;
```

10. Display and compare first and last exercise results of every user/exercise:

```
CREATE FUNCTION vulab_count(integer) RETURNS bigint AS
'SELECT count(*)
FROM ergebnis_checks
WHERE buchungs_id=$1' LANGUAGE sql;
CREATE FUNCTION vulab_score(integer) RETURNS bigint AS
```

### C.3. EXERCISE RESULTS: SELECTED SQL QUERIES AND RESULTS 375

```

'SELECT count(*) FROM ergebnis_checks
WHERE buchungs_id=$1 and erfolg=TRUE)' language sql;
CREATE FUNCTION vulab_score_perc(integer) RETURNS bigint AS
'SELECT 100*vulab_score($1)/vulab_count($1)' LANGUAGE sql;
CREATE FUNCTION vulab_score_diff(integer, integer) RETURNS bigint AS
'SELECT vulab_score_perc($2) - vulab_score_perc($1)' LANGUAGE sql;
SELECT min(buchungs_id) AS first_id,
vulab_score_perc(min(buchungs_id)) AS f_pscore,
max(buchungs_id) AS last_id,
vulab_score_perc(max(buchungs_id)) AS l_pscore,
vulab_score_diff(min(buchungs_id),max(buchungs_id)) AS dpscore,
uebung_id,
substring(login from 1 for 12)
FROM buchungen,benutzer
WHERE buchungen.user_id=benutzer.user_id
AND (SELECT count(*)
FROM ergebnis_checks
WHERE ergebnis_checks.buchungs_id = buchungen.buchungs_id ) > 0
AND login!='feyrer'
AND datum >= '2004-03-15'
AND datum <= '2004-07-25'
GROUP BY login,uebung_id
ORDER BY login;

```

Results:

first_id	f_pscore	last_id	l_pscore	dpscore	uebung_id	substring
233	69	233	69	0	nfs	Dragoonsmail
234	46	241	79	33	nis	Dragoonsmail
148	41	148	41	0	nfs	Klausl.Rathm
149	4	149	4	0	nis	Klausl.Rathm
180	0	190	72	72	nfs	andreas.fisc
191	46	276	88	42	nis	andreas.fisc
247	41	247	41	0	nfs	andreas.poll
249	46	249	46	0	nis	andreas.poll
218	41	235	41	0	nfs	benjamin.gru
267	23	281	39	16	nis	benjamin.gru
237	69	237	69	0	nfs	bernhard.gam
238	16	238	16	0	nis	bernhard.gam
168	80	168	80	0	nfs	ch.marchl@gm
179	39	186	88	49	nis	ch.marchl@gm
117	27	157	61	34	nfs	gep31844
193	44	193	44	0	nis	gep31844
209	66	209	66	0	nfs	ham32330
150	44	158	83	39	nis	ham32330
134	47	134	47	0	nfs	jingjing
160	55	283	69	14	nfs	josef.scheue
278	60	286	95	35	nis	josef.scheue
162	55	196	50	-5	nfs	luf33607
202	39	208	90	51	nis	luf33607
225	0	225	0	0	netbsd	marius.strob
226	44	253	94	50	nfs	marius.strob
255	86	255	86	0	nis	marius.strob
272	44	272	44	0	nfs	markus@fuchs
257	44	273	27	-17	nis	markus@fuchs
224	0	224	0	0	netbsd	martina.hein
223	44	263	0	-44	nfs	martina.hein
222	60	245	53	-7	nis	martina.hein
182	75	217	94	19	nfs	meindlth@asa
200	41	216	81	40	nis	meindlth@asa
159	94	239	75	-19	nfs	pap34148
131	44	240	81	37	nis	pap34148
280	91	280	91	0	nfs	petach@gmx.d
260	4	262	76	72	nis	petach@gmx.d
129	12	141	0	-12	netbsd	punky@schwei
140	16	147	86	70	nfs	punky@schwei

156	27	171	86	59	nis	punky@schwei
231	100	236	100	0	nfs	ramon@pangea
220	46	228	95	49	nis	ramon@pangea
269	16	274	63	47	nfs	tdirscherl@o
275	48	275	48	0	nis	tdirscherl@o
122	86	122	86	0	nfs	trm35740
127	6	142	90	84	nis	trm35740
242	27	251	50	23	nfs	urk35769
243	18	284	81	63	nis	urk35769
143	16	176	55	39	nfs	wachenroeder
177	58	177	58	0	nis	wachenroeder
120	100	123	72	-28	nfs	walter.kern@
121	41	132	93	52	nis	walter.kern@
173	44	184	69	25	nfs	wem35832
185	53	185	53	0	nis	wem35832
201	69	214	69	0	nfs	wes35369
181	46	205	79	33	nis	wes35369

(56 rows)

11. List check scripts (primitives) and how often each is used in the various checks:

```
SELECT count(script), script
FROM uebungs_checks
WHERE uebung_id = 'nis'
OR uebung_id = 'nfs'
GROUP BY script
ORDER BY count(*) DESC;
```

12. List all checks that use the check-file-contents check script, and describe what they test:

```
SELECT check_id, bezeichnung
FROM uebungs_checks
WHERE script='check-file-contents';
```

Results:

check_id	bezeichnung
790	passwd-Information wird in NIS gesucht (/etc/nsswitch.conf)?
791	group-Information wird in NIS gesucht (/etc/nsswitch.conf)?
792	hosts-Information wird in NIS gesucht (/etc/nsswitch.conf)?
793	Domainname in /etc/defaultdomain gesetzt?
810	ypuser in wheel-Gruppe in /etc/group?
864	'share nfs /usr/homes' in /etc/dfs/dfstab?
882	Passender Eintrag in /etc/fstab?
885	'root=' Eintrag in dfstab?
774	Domäne in /etc/defaultdomain gesetzt?
783	PWDIR in /var/yp/Makefile auf /var/yp gesetzt?

(10 rows)

13. Determine checks that test if a certain program is running, the operating system image that was used for the machine the test was running on (i.e. what operating system the test was performed on) and list the description for the test:

```
SELECT check_id, image, bezeichnung
FROM uebungs_checks, uebung_setup
WHERE script='unix-check-process-running'
AND uebungs_checks.uebung_id=uebung_setup.uebung_id
AND uebungs_checks.rechner=uebung_setup.rechner;
```

### C.3. EXERCISE RESULTS: SELECTED SQL QUERIES AND RESULTS 377

Results:

check_id	image	bezeichnung
798	netbsd162.img.gz	rpcbind läuft?
799	netbsd162.img.gz	ypbind läuft?
865	solaris29.img.gz	Läuft rpcbind?
866	solaris29.img.gz	Läuft mountd?
867	solaris29.img.gz	Läuft nfsd?
868	solaris29.img.gz	Läuft statd?
869	solaris29.img.gz	Läuft lockd?
878	netbsd162.img.gz	Läuft rpcbind?
879	netbsd162.img.gz	Läuft rpc.lockd?
880	netbsd162.img.gz	Läuft rpc.statd?

(10 rows)

14. Determine checks that use the netbsd-check-rcvar-set script:

```
SELECT check_id, bezeichnung
FROM uebungs_checks
WHERE script='netbsd-check-rcvar-set';
```

Results:

check_id	bezeichnung
795	/etc/rc.conf: rc_configured gesetzt?
796	/etc/rc.conf: rpcbind gesetzt?
797	/etc/rc.conf: ypbind gesetzt?
874	/etc/rc.conf: rc_configured gesetzt?
875	/etc/rc.conf: lockd gesetzt?
876	/etc/rc.conf: statd gesetzt?
877	/etc/rc.conf: nfs_client gesetzt?

15. Find out places that deal with file ownership:

```
SELECT check_id, bezeichnung
FROM uebungs_checks
WHERE script='unix-check-file-owner';
```

Results:

check_id	bezeichnung
890	Gehört /usr/homes/nfsuser dem Benutzer 'nfsuser' auf vulab1?
891	Gehört /usr/homes/nfsuser dem Benutzer 'nfsuser' auf vulab2?
892	hallo-von-vulab1 gehört nfsuser auf vulab1?
893	hallo-von-vulab1 gehört nfsuser auf vulab2?
894	hallo-von-vulab2 gehört nfsuser auf vulab1?
895	hallo-von-vulab2 gehört nfsuser auf vulab2?

(6 rows)

16. Find out about places that check for existence of certain files (either created manually or via some setup procedure):

```
SELECT check_id, bezeichnung
FROM uebungs_checks
WHERE script='check-file-exists';
```

Results:

check_id	bezeichnung
776	Existiert /var/yp/Makefile?
777	Existiert /var/yp/binding/vulab/ypservers?
778	Existiert /var/yp/passwd.time?
784	Existiert /var/yp/passwd?
870	NFS-Server wird im Runlevel 3 gestartet?

(5 rows)

#### 17. Determine checks that regard package installation on NetBSD:

```
SELECT check_id, bezeichnung
FROM uebungs_checks
WHERE script='netbsd-check-installed-pkg'
AND uebung_id IN ('nis', 'nfs');
```

Results:

check_id	uebung_id	bezeichnung
900	nis	tcsh auf NetBSD installiert? (pkg_info -e tcsh)
901	nis	bash auf NetBSD installiert? (pkg_info -e bash)
904	nfs	tcsh auf NetBSD installiert? (pkg_info -e tcsh)
905	nfs	bash auf NetBSD installiert? (pkg_info -e bash)

(4 rows)

#### 18. Determine checks that regard package installation on Solaris:

```
SELECT check_id, uebung_id, bezeichnung
FROM uebungs_checks
WHERE script='solaris-check-installed-pkg';
```

Results:

check_id	uebung_id	bezeichnung
898	nis	tcsh auf Solaris installiert? (pkginfo SUNWtcsh)
899	nis	bash auf Solaris installiert? (pkginfo SUNWbash)
902	nfs	tcsh auf Solaris installiert? (pkginfo SUNWtcsh)
903	nfs	bash auf Solaris installiert? (pkginfo SUNWbash)

(4 rows)

#### 19. Checks that test for existence of a user account:

```
SELECT check_id, uebung_id, rechner, bezeichnung
FROM uebungs_checks
WHERE script='unix-check-user-exists'
AND uebung_id IN ('nis', 'nfs');
```

Results:

### C.3. EXERCISE RESULTS: SELECTED SQL QUERIES AND RESULTS 379

check_id	uebung_id	rechner	bezeichnung
789	nis	vulab1	User existiert (getpwnam(3))?
804	nis	vulab2	Existiert Benutzer ypuser?
888	nfs	vulab1	Benutzer 'nfsuser' existiert auf vulab1?
889	nfs	vulab2	Benutzer 'nfsuser' existiert auf vulab2?

(4 rows)

20. Determine usage of checks that test for existence of directories:

```
SELECT check_id, uebung_id, rechner, bezeichnung
FROM uebungs_checks
WHERE script='check-directory-exists'
AND uebung_id IN ('nis', 'nfs');
```

Results:

check_id	uebung_id	rechner	bezeichnung
785	nis	vulab1	Verzeichnis /usr/homes/ypuser existiert?
805	nis	vulab2	Existiert Home-Verzeichnis?
887	nfs	vulab1	Existiert Verzeichnis /usr/homes/nfsuser?

(3 rows)

21. An overview of date, start- and endtime as well as duration of exercises:

```
SELECT datum AS date,
startzeit AS starttime,
endzeit AS endtime,
endzeit-startzeit AS duration,
uebung_id,
substring(login from 1 for 20) AS Login
FROM buchungen, benutzer
WHERE buchungen.user_id=benutzer.user_id
AND login != 'feyrer'
AND datum >= '2004-03-15'
AND datum <= '2004-07-25'
ORDER BY duration;
```

Results:

date	starttime	endtime	duration	uebung_id	login
2004-06-08	21:00:00	00:03:19	-20:56:41	nfs	josef.scheuer@stud.f
2004-06-08	18:00:00	00:04:23	-17:55:37	nfs	luf33607
2004-07-15	18:00:00	17:13:20	-00:46:40	nfs	tdirscherl@onlinehom
2004-07-08	18:00:00	17:13:29	-00:46:31	nis	andreas.fischer@stud
2004-06-17	18:00:00	18:00:51	00:00:51	nfs	pap34148
2004-05-23	15:00:00	15:01:08	00:01:08	nis	walter.kern@stud.fh-
2004-06-15	00:00:00	00:04:19	00:04:19	nfs	wem35832
2004-07-07	15:00:00	15:08:57	00:08:57	nfs	urk35769
2004-05-26	15:00:00	15:11:11	00:11:11	nis	trm35740
2004-05-25	21:00:00	21:16:40	00:16:40	nfs	gep31844
2004-07-16	09:00:00	09:24:33	00:24:33	nis	urk35769
2004-07-17	15:00:00	15:27:10	00:27:10	nis	markus@fuchsi.de
2004-07-09	09:00:00	09:31:28	00:31:28	netbsd	martina.heindl@stud.
2004-06-03	18:00:00	18:34:36	00:34:36	nis	pap34148
2004-05-29	12:00:00	12:37:46	00:37:46	netbsd	punky@schweinemarmel
2004-07-14	15:00:00	15:41:24	00:41:24	nis	urk35769
2004-07-16	15:00:00	15:45:50	00:45:50	nis	benjamin.grundstein@
2004-07-04	18:00:00	18:45:52	00:45:52	nfs	ramon@pangea.org

2004-07-18	12:00:00	12:45:55	00:45:55	nis	urk35769
2004-07-12	12:00:00	12:45:59	00:45:59	nfs	martina.heindl@stud.
2004-07-18	18:00:00	18:47:37	00:47:37	nis	andreas.fischer@stud
2004-07-14	12:00:00	12:47:57	00:47:57	nfs	urk35769
2004-07-08	12:00:00	12:50:18	00:50:18	nfs	martina.heindl@stud.
2004-07-05	21:00:00	21:51:02	00:51:02	nfs	benjamin.grundstein@
2004-05-23	12:00:00	12:51:23	00:51:23	nis	walter.kern@stud.fh-
2004-07-02	09:00:00	09:52:30	00:52:30	nfs	meindlth@asamnet.de
2004-07-17	18:00:00	18:52:54	00:52:54	nfs	pap34148
2004-07-14	18:00:00	18:57:09	00:57:09	nis	andreas.fischer@stud
2004-07-03	21:00:00	21:58:03	00:58:03	nis	ramon@pangea.org
2004-07-08	09:00:00	09:58:44	00:58:44	nis	martina.heindl@stud.
2004-06-29	12:00:00	12:59:53	00:59:53	nfs	wes35369
2004-07-17	21:00:00	22:01:06	01:01:06	nis	pap34148
2004-06-05	00:00:00	01:01:08	01:01:08	nis	punky@schweinemarmel
2004-07-17	00:00:00	01:02:18	01:02:18	nfs	petach@gmx.de
2004-06-13	18:00:00	19:02:33	01:02:33	nis	punky@schweinemarmel
2004-06-26	21:00:00	22:03:20	01:03:20	nfs	ham32330
2004-07-16	18:00:00	19:06:31	01:06:31	nis	urk35769
2004-07-05	18:00:00	19:07:02	01:07:02	nis	andreas.fischer@stud
2004-05-22	12:00:00	13:07:14	01:07:14	nfs	trm35740
2004-06-23	18:00:00	19:08:23	01:08:23	nis	gep31844
2004-05-22	15:00:00	16:10:11	01:10:11	nfs	walter.kern@stud.fh-
2004-07-05	09:00:00	10:10:12	01:10:12	nis	Dragoonsmail@gmx.de
2004-06-10	12:00:00	13:11:38	01:11:38	nfs	ch.marchl@gmx.de
2004-07-03	12:00:00	13:13:15	01:13:15	nis	ramon@pangea.org
2004-06-07	18:00:00	19:13:18	01:13:18	nfs	gep31844
2004-06-02	15:00:00	16:13:50	01:13:50	nis	trm35740
2004-06-28	21:00:00	22:14:18	01:14:18	nfs	wes35369
2004-06-21	12:00:00	13:15:01	01:15:01	nfs	wem35832
2004-07-04	12:00:00	13:15:07	01:15:07	nfs	ramon@pangea.org
2004-06-24	21:00:00	22:15:20	01:15:20	nis	wes35369
2004-06-08	12:00:00	13:17:34	01:17:34	nis	ham32330
2004-06-17	21:00:00	22:20:44	01:20:44	nis	ch.marchl@gmx.de
2004-05-21	15:00:00	16:20:57	01:20:57	nfs	walter.kern@stud.fh-
2004-07-11	15:00:00	16:22:02	01:22:02	nis	markus@fuchsi.de
2004-06-21	21:00:00	22:22:40	01:22:40	nis	wes35369
2004-07-05	00:00:00	01:23:08	01:23:08	nis	Dragoonsmail@gmx.de
2004-06-08	15:00:00	16:23:38	01:23:38	nfs	pap34148
2004-06-03	21:00:00	22:24:17	01:24:17	nis	ham32330
2004-06-20	12:00:00	13:24:20	01:24:20	nis	ch.marchl@gmx.de
2004-05-27	18:00:00	19:25:03	01:25:03	nis	pap34148
2004-06-14	21:00:00	22:25:15	01:25:15	nfs	wem35832
2004-05-29	21:00:00	22:25:28	01:25:28	nis	walter.kern@stud.fh-
2004-06-28	09:00:00	10:25:40	01:25:40	nis	wes35369
2004-07-05	15:00:00	16:27:26	01:27:26	nfs	Dragoonsmail@gmx.de
2004-06-01	21:00:00	22:27:26	01:27:26	nfs	punky@schweinemarmel
2004-07-18	21:00:00	22:27:52	01:27:52	nis	benjamin.grundstein@
2004-07-15	15:00:00	16:28:02	01:28:02	nfs	markus@fuchsi.de
2004-05-29	15:00:00	16:28:02	01:28:02	nis	walter.kern@stud.fh-
2004-06-26	18:00:00	19:28:08	01:28:08	nis	andreas.fischer@stud
2004-05-21	21:00:00	22:28:11	01:28:11	nis	walter.kern@stud.fh-
2004-06-20	21:00:00	22:28:16	01:28:16	nfs	pap34148
2004-07-08	06:00:00	07:28:51	01:28:51	nis	andreas.pollinger@st
2004-07-16	21:00:00	22:28:54	01:28:54	nis	tdirscherl@onlinehom
2004-07-15	21:00:00	22:28:55	01:28:55	nfs	tdirscherl@onlinehom
2004-07-07	21:00:00	22:28:57	01:28:57	nfs	andreas.pollinger@st
2004-07-18	09:00:00	10:28:59	01:28:59	nis	josef.scheuer@stud.f
2004-07-17	12:00:00	13:29:01	01:29:01	nis	josef.scheuer@stud.f
2004-07-18	00:00:00	01:29:02	01:29:02	nfs	josef.scheuer@stud.f
2004-06-28	18:00:00	19:29:03	01:29:03	nis	andreas.fischer@stud
2004-06-23	15:00:00	16:29:03	01:29:03	nfs	meindlth@asamnet.de
2004-06-28	15:00:00	16:29:03	01:29:03	nis	meindlth@asamnet.de
2004-06-30	09:00:00	10:29:03	01:29:03	nis	meindlth@asamnet.de
2004-07-02	12:00:00	13:29:03	01:29:03	nis	andreas.fischer@stud
2004-07-02	18:00:00	19:29:03	01:29:03	nis	ramon@pangea.org
2004-07-06	15:00:00	16:29:04	01:29:04	nfs	bernhard.gammel@stud
2004-07-05	12:00:00	13:29:04	01:29:04	nis	martina.heindl@stud.
2004-05-26	18:00:00	19:29:04	01:29:04	netbsd	punky@schweinemarmel



### C.3. EXERCISE RESULTS: SELECTED SQL QUERIES AND RESULTS 381

2004-07-10	12:00:00	13:29:05	01:29:05	nis	markus@fuchsi.de
2004-06-24	18:00:00	19:29:05	01:29:05	nfs	luf33607
2004-06-07	12:00:00	13:29:05	01:29:05	nfs	Klausl.Rathmacher@st
2004-07-17	09:00:00	10:29:09	01:29:09	nfs	josef.scheuer@stud.f
2004-06-13	15:00:00	16:29:21	01:29:21	nis	punky@schweinemarmel
2004-06-12	21:00:00	22:29:21	01:29:21	nis	punky@schweinemarmel
2004-06-25	18:00:00	19:29:32	01:29:32	nis	luf33607
2004-07-06	18:00:00	19:29:37	01:29:37	nis	bernhard.gammel@stud
2004-06-07	09:00:00	10:29:55	01:29:55	nis	ham32330
2004-07-09	18:00:00	19:30:05	01:30:05	nis	marius.strobl@stud.f
2004-07-08	21:00:00	22:30:06	01:30:06	nfs	marius.strobl@stud.f
2004-06-07	21:00:00	22:30:07	01:30:07	nis	pap34148
2004-06-26	15:00:00	16:30:13	01:30:13	nis	luf33607
2004-06-20	18:00:00	19:30:33	01:30:33	nis	wachenroeder@gmx.de
2004-06-19	18:00:00	19:31:26	01:31:26	nfs	wachenroeder@gmx.de
2004-06-14	15:00:00	16:31:48	01:31:48	nfs	pap34148
2004-07-01	12:00:00	13:35:49	01:35:49	nfs	benjamin.grundstein@
2004-06-03	12:00:00	13:36:55	01:36:55	nfs	punky@schweinemarmel
2004-06-21	15:00:00	16:37:34	01:37:34	nis	wem35832
2004-05-29	18:00:00	19:38:45	01:38:45	nis	trm35740
2004-07-08	15:00:00	17:13:23	02:13:23	nfs	marius.strobl@stud.f
2004-06-22	15:00:00	17:30:03	02:30:03	nfs	andreas.fischer@stud
2004-07-02	21:00:00	23:30:03	02:30:03	netbsd	marius.strobl@stud.f
2004-06-02	21:00:00	23:30:03	02:30:03	nis	pap34148
2004-05-31	21:00:00	23:30:03	02:30:03	nfs	wachenroeder@gmx.de
2004-06-25	15:00:00	17:30:03	02:30:03	nis	wes35369
2004-07-11	18:00:00	20:30:04	02:30:04	nis	petach@gmx.de
2004-06-15	18:00:00	20:30:04	02:30:04	nfs	pap34148
2004-05-28	12:00:00	14:30:04	02:30:04	nfs	jingjing
2004-05-30	12:00:00	14:30:04	02:30:04	netbsd	punky@schweinemarmel
2004-06-02	18:00:00	20:30:04	02:30:04	nis	pap34148
2004-05-30	15:00:00	17:30:04	02:30:04	nfs	punky@schweinemarmel
2004-06-07	15:00:00	17:30:04	02:30:04	nis	Klausl.Rathmacher@st
2004-07-01	18:00:00	20:30:04	02:30:04	nfs	meindlth@asamnet.de
2004-07-03	15:00:00	17:30:04	02:30:04	nfs	benjamin.grundstein@
2004-07-11	21:00:00	23:30:04	02:30:04	nis	petach@gmx.de
2004-05-28	21:00:00	23:30:05	02:30:05	netbsd	punky@schweinemarmel
2004-07-15	09:00:00	11:30:05	02:30:05	nis	markus@fuchsi.de
2004-07-12	18:00:00	20:30:05	02:30:05	nis	petach@gmx.de
2004-06-21	18:00:00	20:30:05	02:30:05	nfs	andreas.fischer@stud
2004-05-20	18:00:00	20:30:05	02:30:05	nfs	gep31844
2004-07-16	12:00:00	14:30:06	02:30:06	nfs	martina.heindl@stud.
2004-06-25	12:00:00	14:30:30	02:30:30	nfs	meindlth@asamnet.de
2004-07-07	12:00:00	17:13:32	05:13:32	nis	urk35769
2004-07-07	09:00:00	17:13:25	08:13:25	nis	andreas.fischer@stud
2004-07-03	00:00:00	13:13:16	13:13:16	nfs	marius.strobl@stud.f
2004-07-09	21:00:00			nis	marius.strobl@stud.f
2004-07-19	00:00:00			nfs	wes35369

(135 rows)

22. Determine number of exercises with “sane” duration:

```

SELECT count(*)
FROM ( SELECT extract(hours from endzeit-startzeit)*60
      +extract(minutes from endzeit-startzeit) as dauer
      FROM buchungen, benutzer
      WHERE buchungen.user_id=benutzer.user_id
      AND login != 'feyrer'
      AND NOT (endzeit-startzeit>='1:40'
      OR endzeit<startzeit)
      AND uebung_id IN ('nis', 'nfs')
      AND datum >= '2004-03-15'
      AND datum <= '2004-07-25'
      ORDER BY dauer
    ) AS x;

```

23. Determine ending times of all exercises:

```

SELECT extract(hours from endzeit-startzeit)*60
       +extract(minutes from endzeit-startzeit) AS dauer
FROM buchungen, benutzer
WHERE buchungen.user_id=benutzer.user_id
      AND login != 'feyrer'
      AND NOT (endzeit-startzeit>='1:40'
              OR endzeit<startzeit)
      AND uebung_id IN ('nis', 'nfs')
      AND datum >= '2004-03-15'
      AND datum <= '2004-07-25'
ORDER BY dauer;

```

24. Exercise start times and number of exercises booked at that time:

```

SELECT count(*),startzeit
FROM buchungen, benutzer
WHERE buchungen.user_id=benutzer.user_id
      AND uebung_id in ('nis', 'nfs')
      AND login!='feyrer'
      AND datum >= '2004-03-15'
      AND datum <= '2004-07-25'
GROUP BY startzeit
ORDER BY startzeit;

```

25. Exercise start times and number of exercise booked at that time for histogram plotting, NIS and NFS exercises from students only:

```

SELECT extract(hours from startzeit)
FROM buchungen, benutzer
WHERE buchungen.user_id=benutzer.user_id
      AND uebung_id IN ('nis', 'nfs')
      AND login!='feyrer'
      AND datum >= '2004-03-15'
      AND datum <= '2004-07-25'
ORDER BY startzeit;

```

## Appendix D

# A theory of bugs — attempt of a reconstructive approach

The following list of PL/SQL queries to the PostgreSQL database and their results are used in evaluation of Virtual Unix Lab exercise results in section 9.3.2.2:

### 1. Query:

```
SELECT rechner, uebung_id, count(*)
FROM ergebnis_checks, uebungs_checks
WHERE ergebnis_checks.check_id = uebungs_checks.check_id
GROUP BY rechner, uebung_id
ORDER BY uebung_id, rechner;
```

### Results:

rechner	uebung_id	count
vulab1	netbsd	120
vulab1	nfs	3078
vulab2	nfs	3038
vulab1	nis	3780
vulab2	nis	3960

(5 rows)

### 2. Query:

### Results:

rechner	uebung_id	%
vulab1	netbsd	80
vulab1	nfs	42
vulab2	nfs	61
vulab1	nis	36
vulab2	nis	68

### 3. Query:

```
SELECT count(*), rechner, uebung_id
FROM uebungs_checks
GROUP BY uebung_id, rechner
ORDER BY uebung_id, rechner;
```

## Results:

count	rechner	uebung_id
8	vulabl	netbsd
18	vulabl	nfs
18	vulab2	nfs
21	vulabl	nis
22	vulab2	nis
1	localhost	update-solaris
1	vulabl	update-solaris

(7 rows)

## 4. Query:

```
CREATE VIEW test1 AS SELECT script
FROM ergebnis_checks, uebungs_checks
WHERE ergebnis_checks.check_id = uebungs_checks.check_id
GROUP BY script
ORDER BY script
;
CREATE FUNCTION count_all(uebungs_checks.script%TYPE) RETURNS bigint AS
'
SELECT count(*)
FROM ergebnis_checks, uebungs_checks
WHERE ergebnis_checks.check_id = uebungs_checks.check_id
AND script = $1
AND erfolg IN ( true, false )
GROUP BY script
ORDER BY script
' LANGUAGE sql;
CREATE FUNCTION count_fail(uebungs_checks.script%TYPE) RETURNS bigint AS
'
SELECT count(*)
FROM ergebnis_checks, uebungs_checks
WHERE ergebnis_checks.check_id = uebungs_checks.check_id
AND script = $1
AND erfolg in ( false )
GROUP BY script
ORDER BY script
' LANGUAGE sql;
SELECT count_all(script), count_fail(script),
100 * count_fail(script) / count_all(script)
AS perc,
script
FROM test1
ORDER BY perc DESC;
DROP FUNCTION count_fail(uebungs_checks.script%TYPE);
DROP FUNCTION count_all(uebungs_checks.script%TYPE);
DROP VIEW test1;
```

## Results:

count_all	count_fail	perc	script
15	13	86	netbsd-check-user-shell
180	152	84	unix-check-user-ingroup
195	161	82	unix-check-user-password

724	588	81	netbsd-check-installed-pkg
180	143	79	unix-check-user-fullname
15	11	73	unix-check-user-home
1002	724	72	unix-check-file-owner
167	114	68	unix-check-mount
694	452	65	solaris-check-installed-pkg
210	123	58	unix-check-user-shell
709	408	57	unix-check-user-exists
527	296	56	check-directory-exists
3722	2010	54	check-program-output
1769	940	53	check-file-contents
1224	650	53	netbsd-check-rcvar-set
895	188	21	check-file-exists
1748	357	20	unix-check-process-running

(17 rows)

## 5. Query:

```

CREATE VIEW test1 AS SELECT script, parameter
FROM ergebnis_checks, uebungs_checks
WHERE ergebnis_checks.check_id = uebungs_checks.check_id
GROUP BY script, parameter
ORDER BY script, parameter
;
CREATE FUNCTION count_all(uebungs_checks.script%TYPE,
                        uebungs_checks.parameter%TYPE)
RETURN bigint AS
,
SELECT count(*)
FROM ergebnis_checks, uebungs_checks
WHERE ergebnis_checks.check_id = uebungs_checks.check_id
AND script = $1
AND parameter = $2
AND erfolg in ( true, false )
GROUP BY script, parameter
ORDER BY script, parameter
' LANGUAGE sql;
CREATE FUNCTION count_fail(uebungs_checks.script%TYPE,
                        uebungs_checks.parameter%TYPE)
RETURN bigint AS
,
SELECT count(*)
FROM ergebnis_checks, uebungs_checks
WHERE ergebnis_checks.check_id = uebungs_checks.check_id
AND script = $1
AND parameter = $2
AND erfolg in ( false )
GROUP BY script, parameter
ORDER BY script, parameter
' LANGUAGE sql;
SELECT count_all(script, parameter), count_fail(script, parameter),
100 * count_fail(script, parameter) / count_all(script, parameter)
AS perc,
script, parameter
FROM test1
ORDER BY perc DESC;
DROP FUNCTION count_all(uebungs_checks.script%TYPE,
                        uebungs_checks.parameter%TYPE) ;
DROP FUNCTION count_fail(uebungs_checks.script%TYPE,
                        uebungs_checks.parameter%TYPE) ;
DROP VIEW test1;

```

## Results:

```
count | count | |
```

## APPENDIX D. A THEORY OF BUGS — ATTEMPT OF A RECONSTRUCTIVE APPROACH

_all	_fail	perc	script	parameter
180	180	100	check-file-contents	FILE=/etc/group CONTENT_SHOULD="" wheel:.ypuser"
15	14	93	unix-check-user-shell	LOGIN=vulab SHELL_SHOULD='./bash'
180	159	88	check-program-output	PROGRAM='ypcat hosts' OUTPUT_SHOULD='194.95.108.32.*tab'
15	13	86	netbsd-check-user-shell	LOGIN=test SHELL_SHOULD='./tcsh'
15	13	86	unix-check-user-shell	LOGIN=test SHELL_SHOULD='./tcsh'
15	13	86	unix-check-user-password	LOGIN=test PASSWD_SHOULD='vutest'
180	153	85	check-program-output	PROGRAM='/sbin/ping -c 1 tab 2>&  ; echo result:\$?' OUTPUT_SHOULD='result:0\$'
180	152	84	unix-check-user-ingroup	LOGIN=ypuser GROUP_SHOULD=benutzer
180	150	83	check-program-output	PROGRAM='ypcat group' OUTPUT_SHOULD='benutzer:'
180	148	82	unix-check-user-password	LOGIN=ypuser PASSWD_SHOULD=myntspw
362	298	82	netbsd-check-installed-pkg	PKG=tcsh
362	290	80	netbsd-check-installed-pkg	PKG=bash
180	143	79	unix-check-file-fullname	LOGIN=ypuser FULLNAME_SHOULD='NIS Testbenutzer'
167	130	77	check-program-output	PROGRAM='mount   grep nfs' OUTPUT_SHOULD='^vulabl:/usr/homes on /usr/homes'
167	130	77	check-program-output	PROGRAM='df -k   grep : ' OUTPUT_SHOULD='^vulabl:/usr/homes.*usr/homes\$'
167	129	77	check-file-contents	FILE=/etc/fstab CONTENT_SHOULD='vulabl:/usr/homes.*usr/homes.*nfs.*rw'
334	245	73	unix-check-file-owner	FILE=/usr/homes/nfsuser/hallo-von-vulabl OWNER_SHOULD=nfsuser
15	11	73	unix-check-user-home	LOGIN=test HOME_SHOULD='/home/test'
15	11	73	unix-check-user-exists	LOGIN=test
334	245	73	unix-check-file-owner	FILE=/usr/homes/nfsuser/hallo-von-vulab2 OWNER_SHOULD=nfsuser
347	246	70	solaris-check-installed-pkg	PKG=SUNWtcsh
334	234	70	unix-check-file-owner	FILE=/usr/homes/nfsuser OWNER_SHOULD=nfsuser
167	114	68	unix-check-mount	MOUNT_FROM=10.0.0.1:/usr/homes MOUNT_ON=mnt
171	117	68	check-program-output	PROGRAM='showmount -e vulabl' OUTPUT_SHOULD='/usr/homes'
180	118	65	netbsd-check-rcvar-set	RCVAR=rpcbind
180	114	63	netbsd-check-rcvar-set	RCVAR=ypbind
171	107	62	netbsd-check-rcvar-set	RCVAR=lockd
171	107	62	netbsd-check-rcvar-set	RCVAR=statd
360	217	60	unix-check-user-exists	LOGIN=ypuser
347	206	59	solaris-check-installed-pkg	PKG=SUNWbash
167	100	59	check-program-output	PROGRAM='share' OUTPUT_SHOULD='/usr/homes.*root='
171	102	59	netbsd-check-rcvar-set	RCVAR=nfs_client
167	98	58	check-file-contents	FILE=/etc/dfs/dfstab CONTENT_SHOULD='root='
360	207	57	check-directory-exists	DIR=/usr/homes/ypuser
180	96	53	unix-check-user-shell	LOGIN=ypuser SHELL_SHOULD='./ksh'
334	180	53	unix-check-user-exists	LOGIN=nfsuser
167	89	53	check-directory-exists	DIR=/usr/homes/nfsuser
360	185	51	check-program-output	PROGRAM=ypwhich OUTPUT_SHOULD='vulabl'
180	93	51	check-file-contents	FILE=/etc/nsswitch.conf CONTENT_SHOULD='hosts:.nis'
180	89	49	check-file-contents	FILE=/etc/nsswitch.conf CONTENT_SHOULD='passwd:.nis'
180	89	49	check-file-contents	FILE=/etc/nsswitch.conf CONTENT_SHOULD='group:.nis'
180	84	46	check-program-output	PROGRAM='ypcat passwd   grep ypuser:   wc -1' OUTPUT_SHOULD=1
360	160	44	check-program-output	PROGRAM='ypcat passwd   wc -1' OUTPUT_SHOULD='[0]*'
360	158	43	check-program-output	PROGRAM='ypcat group   wc -1' OUTPUT_SHOULD='[0]*'
360	154	42	check-program-output	PROGRAM='ypcat hosts   wc -1' OUTPUT_SHOULD='[0]*'
180	73	40	check-program-output	PROGRAM='cat /var/yp/passwd   grep ypuser:   wc -1' OUTPUT_SHOULD=1
360	135	37	check-file-contents	FILE=/etc/defaultdomain CONTENT_SHOULD='vulab'
360	136	37	check-program-output	PROGRAM=domainname OUTPUT_SHOULD='vulab'
175	64	36	check-file-contents	FILE=/etc/dfs/dfstab CONTENT_SHOULD='share.*nfs.*usr/homes'
180	63	35	check-file-exists	FILE=/var/yp/passwd
175	62	35	check-program-output	PROGRAM='showmount -e localhost' OUTPUT_SHOULD='/usr/homes'
180	63	35	check-file-contents	FILE=/var/yp/Makefile CONTENT_SHOULD='^PWDIR.*=/var/yp'
175	59	33	check-program-output	PROGRAM='share' OUTPUT_SHOULD='/usr/homes'
351	102	29	netbsd-check-rcvar-set	RCVAR=rc_configured
180	52	28	unix-check-process-running	PROCESS=ypbind
180	45	25	check-file-exists	FILE=/var/yp/binding/vulab/ypservers
171	44	25	unix-check-process-running	PROCESS=rpc.statd
171	44	25	unix-check-process-running	PROCESS=rpc.lockd
180	42	23	check-file-exists	FILE=/var/yp/passwd.time
526	121	23	unix-check-process-running	PROCESS=rpcbind
175	24	13	unix-check-process-running	PROCESS=nfsd
175	24	13	unix-check-process-running	PROCESS=mountd
175	24	13	unix-check-process-running	PROCESS=lockd
175	24	13	check-file-exists	FILE='/etc/rc3.d/S15nfs.server'
175	24	13	unix-check-process-running	PROCESS=statd
180	14	7	check-file-exists	FILE=/var/yp/Makefile

(66 rows)

# Appendix E

## Analysis of exercises under tutorial and adaptive aspects

The following text displays the NIS exercise's text with the single questions decomposed as outlined in section 9.3.1.

```
<h1> NIS Master und Client Setup</h1>
```

```
In dieser Übung soll auf den beiden vulab-Rechner der Network Information Service (NIS) installiert werden. Dabei wird auf dem Rechner "vulab1" der NIS-Master, auf dem Rechner "vulab2" der NIS-Client installiert.
```

```
<p>
```

```
<h2>1. Master (Solaris): vulab1</h2>
```

```
<ul>
```

```
<li> Stellen Sie sicher dass die nötigen Pakete (SUNWypr, SUNWypu, SUNWsprot, ...) installiert sind.
```

```
- What does the student have to do?
```

```
Use pkginfo(1) to verify if the named packages are installed (they are all installed by default!)
```

```
- What problems can occur, how can they be identified?
```

```
pkginfo(1) doesn't show packages  
pkginfo not in search path  
User doesn't know about pkginfo, and tries other commands (pkg_info, rpm, ...)
```

```
- Help: behavioristic, epistemic, ...
```

```
Give exact commands: pkg_info | egrep '(SUNWypr|SUNWypu|...)'  
Refer to pkginfo(8)  
Give general information on software management:  
http://www.feyrer.de/SA/12-sw.html
```

```
- What wrong thinking can cause problems? (Believes)
```

```
wrong: The packages are not installed and have to be installed first  
Student may know *some* packaging system, but not this
```

```

one

- What viewpoints may exist:

    pkginfo(1) view on installed packages
    View on /var/sadm/pkg for installed packages

- Ways of data acquisition: checks, keyboard tracing, ...

checks:
verify that everything is (still!) installed, and
that the student did not remove a package by accident

keyboard tracing:
Recognize if student tries to run several related
commands that are not relevant here (pkg_info, rpm,
...)

<li> Setzen Sie den NIS-Domänenname auf "vulab" (/etc/defaultdomain &
domainname(1))

- What does the student have to do?

Need to edit the file, plus make system read it:
1) put domain into /etc/defaultdomain
2) set domainname in system either via domainname(1),
   or by some init/rc.d script
   or by rebooting

- What problems can occur, how can they be identified?

Student doesn't know how to write data info file
Student doesn't know format of /etc/defaultdomain
Student doesn't know how to set domain in system
Student doesn't know about domainname(1) or how to use it
Student doesn't know how to use init/rc.d script to
   set domainname from /etc/defaultdomain

Detect by no data in /etc/domainname and domainname(1)
after some time

- Help: behavioristic, epistemic, ...
(Classification of help e.g. via URL or chapter)

Give proper commands:
    echo vulab >/etc/defaultdomain
    domainname `cat /etc/defaultdomain`
    or: sh /etc/init.d/inetinit stop/start
Hint at documentation for commands and files:
    domainname(1), defaultdomainname(4)
Give background on NIS:
    http://www.feyrer.de/SA/08-networking.html

- What wrong thinking can cause problems?

Expect some GUI program to be needed to set the
domainname (e.g. Yast, smc, admintool)
System recognizes changes to /etc/defaultdomain
automatically
Need special program to set /etc/defaultdomain (or domain
in general)

- What viewpoints may exist:

    only "from inside"

- Ways of data acquisition: checks, keyboard tracing, ...

    1) check /etc/defaultdomain

```



2) check domainname(1)

Or monitor commands entered, recognize

- 1) setting /etc/defaultdomain (via echo, vi, ...)
- 2) running domainname(1), init/rc.d script or reboot (harder than via checks!)

<li> Setzen Sie den Rechner mit "ypinit -m" als NIS Master auf

- What does the student have to do?

Hostname and domainname need to be set first  
No file backend for all NIS maps may be present, non fatal!

Run "ypinit -m",  
enter valid NIS server (10.0.0.1, or hostname from /etc/hosts)  
Judge errors printed as not critical

- What problems can occur, how can they be identified?

hostname given for NIS master, but not in /etc/hosts  
errors not identified as non-critical

- Help: behavioristic, epistemic, ...

give exact command and all data  
hint at programs: ypinit(8)  
give background: <http://www.feyrer.de/SA/08-networking.html>

- What wrong thinking can cause problems? (Believes)

All files must be present to create NIS maps  
Must use GUI program to setup NIS master (smc, ...)  
Name of NIS master is automatically known (instead of  
verifying that it's in /etc/hosts)  
ypinit runs 'make' in /var/yp automatically (true on  
Solaris, false on NetBSD)  
ypinit prints no errors  
errors printed by ypinit are fatal

- What viewpoints may exist:

from inside  
from outside (when service daemons run!):  
rpcinfo, grep yp /etc/rc

- Ways of data acquisition: checks, keyboard tracing, ...

keyboard tracing:  
catch 'ypinit -m' and check if prerequisites are met

checks:  
/var/yp/Makefile was setup  
/var/yp/binding/vulab exists (dir)  
/var/yp/binding/vulab/ypservers (file) exists  
/var/yp/passwd.time was generated

<li> Sorgen Sie dafür dass die nötigen Serverprozesse (ypbind, ypserv,  
...) beim booten gestartet werden.

- What does the student have to do?

Solaris: nothing, processes are started automatically  
Solaris' rc.d scripts look at existing config files, and  
start daemons then, no need to edit config files

- What problems can occur, how can they be identified?

```

In theory, processes could not be started.
Detected by missing processes after booting,
resulting in services (mostly RPC) not available

- Help: behavioristic, epistemic, ...

Name processes that need to run, and show how they are
started automatically by pointing at the right rc.d
script
Hint at rc.d scripts
Give Background on how services start up, and how the
process may be influenced:
http://www.feyrer.de/SA/06-booting.html

- What wrong thinking can cause problems? (Believes)

Need to edit some config file for services to start up
(Solaris: things just work; different for other
operating systems!)
wrong: need to edit some file
right: things just work

- What viewpoints may exist:

from inside: ps
from outside: rpcinfo vulabl

- Ways of data acquisition: checks, keyboard tracing, ...

check:
if daemons / services run

typescript:
see if user tries to find/edit some file to enable
daemons (e.g. the BSD /etc/rc.conf)

<li> Starten Sie die Serverdienste!

- What does the student have to do?

Solaris: reboot,
or run /etc/init.d/{inetinit,rpc}

- What problems can occur, how can they be identified?

Student doesn't know proper rc.d scripts or
commands to start services manually

- Help: behavioristic, epistemic, ...
(Classification of help e.g. via URL or chapter)

Advise to simply reboot
Hint at proper rc.d scripts / commands
Give Background on how services start up, and how the
process may be influenced:
http://www.feyrer.de/SA/06-booting.html

- What wrong thinking can cause problems? (Believes?)

wrong: need to edit some config file before starting
processes
wrong: editing some config file starts processes automatically

- What viewpoints may exist:

from inside: ps
from outside: rpcinfo

```

- Ways of data acquisition: checks, keyboard tracing, ...

checks: ps, rpcinfo  
typescript: see if student tries to edit some rc.conf  
file, warn if so

<li> Welcher NIS-Server wird verwendet?

- What does the student have to do?

Run 'ypwhich' to see if a the NIS server is found

- What problems can occur, how can they be identified?

If no service daemons are started (ypbind, ypserv), then nothing will be printed.

- Help: behavioristic, epistemic, ...

ypwhich  
State that if ypwhich doesn't show a server, there's something fundamentally wrong, either on the network layer, on the NIS client or server part.  
Hint at list for troubleshooting NIS setup at <http://www.feyrer.de/SA/08-networking.html>

- What wrong thinking can cause problems? (== Believes?)

Student doesn't know the proper command  
Student tries to find some (GUI?) program to show used nis server

- What viewpoints may exist:

from inside: ypwhich

- Ways of data acquisition: checks, keyboard tracing, ...

checks:  
verify the proper server is returned from ypwhich

typescript analysis:  
see if student knows the 'ypwhich' command  
see if student tries to guess some other commands  
see if student runs 'apropos' or 'man -k' to find commands

<li> Welche Datei wird für die Gruppen-Daten verwendet?

- What does the student have to do?

Look through /var/yp/Makefile to see what source is used for the 'groups' map(s)

- What problems can occur, how can they be identified?

Student doesn't know where to start looking  
Student names NIS map file instead of its source  
Student may mix up source of NIS map (/etc/group) and NIS map file (== binary DB file, used for ypcat)

- Help: behavioristic, epistemic, ...

Hint at /etc/group  
Hint at how the NIS map is produced  
Hint at the process performed by /var/yp/Makefile  
Outline general operation of NIS at <http://www.feyrer.de/SA/08-networking.html>

```

- What wrong thinking can cause problems? (== Believes?)

  Student misses parts in the chain of files used between
  server and client (/etc/group -> var/yp/group.byname
  -> ypserv -> ypbind -> getgrent/ypcat)

- What viewpoints may exist:

  from inside

- Ways of data acquisition: checks, keyboard tracing, ...

  checks: n/a
  typescript: not easily doable either

  To improve, tell user to put the filename into some file,
  then check the contents of that file later on.

<li> Welche Datei wird für die Passwort-Daten verwendet?

- What does the student have to do?

  same as for group file above
  think about shadow passwords and their (non)use in NIS

- What problems can occur, how can they be identified?

  No technical problems can arise
  Student may not understand use of shadow passwords with
  NIS; may manifest in student looking at /etc/shadow

- Help: behavioristic, epistemic, ...

  Hint at files
  Hint at overall process
  Hint at shadow passwords, and their (non)use in NIS

- What wrong thinking can cause problems? (== Believes?)

  wrong: shadow passwords are being used

- What viewpoints may exist:

  from inside (only)

- Ways of data acquisition: checks, keyboard tracing, ...

  checks: n/a
  typescript: see user browser /var/yp/Makefile,
  warn when touching /etc/shadow or /etc/master.passwd?

<li> Überprüfen Sie ob Gruppen- und Passwort-Informationen über NIS
abgefragt werden können.

- What does the student have to do?

  Run 'ypcat passwd' and 'ypcat group'

- What problems can occur, how can they be identified?

  No data or some error is printed,
  or the output doesn't contain anything that's in the
  relevant NIS maps (e.g. no 'root' in the passwd map,
  etc.)

```

Student uses finger or anything that needs nsswitch  
 setup, which is not done yet  
 ypsserv and ypbind need to run for this to work

- Help: behavioristic, epistemic, ...

Hint at 'ypcat passwd', 'ypcat group' etc.  
 Give general hint on NIS  
<http://www.feyrer.de/SA/08-networking.html>

- What wrong thinking can cause problems? (== Believes?)

User may try to go one step further and use system  
 interfaces (getpwent(3), getent(1), ...), but that's  
 not configured yet.  
 User may not be aware of ypcat, it's existence and  
 functionality

- What viewpoints may exist:

from inside and from outside the same (but no client  
 setup yet, and the test would be pretty much the  
 same)

- Ways of data acquisition: checks, keyboard tracing, ...

checks: not doable  
 typescript: see if user invokes ypcat with appropriate  
 arguments

<li> Vergleichen Sie den Passwort-Eintrag des Benutzers "vulab" im NIS  
 und in den /etc-Dateien. Was stellen Sie fest?

- What does the student have to do?

Run 'ypcat passwd' and determine the password (2nd) field  
 compare against the password (2nd) field in /etc/passwd  
 and /etc/shadow  
 Determine that NIS has the encrypted password (public),  
 while the password used to be in /etc/shadow (private)

No persistent change has to be made upon comparison.

- What problems can occur, how can they be identified?

Student doesn't understand/notice that the formerly  
 private (encrypted) password is now publically  
 available in NIS.  
 No change to the system reflects this possible  
 non-perception. Solution: ask/verify by asking?

- Help: behavioristic, epistemic, ...

Display both passwords.  
 Ask why the two entries may be different.  
 Ask about availability of shadow passwords on all Unix systems  
 Ask about security implications.

- What wrong thinking can cause problems? (== Believes?)

Shadow passwords are used with NIS in general

- What viewpoints may exist:

from inside only, to read /etc/passwd

- Ways of data acquisition: checks, keyboard tracing, ...

```

checks: n/a
typescript:
  see if commands 'ypcat passwd' are ran,
  see if /etc/passwd is observed
  see if /etc/shadow is observed

```

<li> Sorgen Sie dafür, dass die Passwort-Informationen künftig in der Datei /var/yp/passwd gehalten werden. Die existierenden Logins sollen dabei nicht übernommen werden.

- What does the student have to do?

```

create empty file /var/yp/passwd
change /var/yp/Makefile to set PWDIR=/var/yp
run 'make' in /var/yp
restart yppasswdd via /etc/init.d/rpc stop/start,
  /usr/lib/netsvc/yp/yp{stop,start}, or manually (kill,
  /usr/lib/netsvc/yp/rpc.yppasswdd)

```

- What problems can occur, how can they be identified?

```

Student doesn't know about the PWDIR switch in
  /var/yp/Makefile
student copies /etc/passwd to /var/yp/passwd, and gets
  duplicate, possibly non-identical accounts
syntax of PWDIR in /var/yp/Makefile is not followed as
  exactly as it has to, resulting in yppasswdd not
  starting up properly
new NIS map is not built, resulting in difference between
  /var/yp/passwd and the NIS maps
yppasswdd is not restarted, and will continue to update
  /etc/passwd

```

- Help: behavioristic, epistemic, ...

```

Give steps to perform: change PWDIR entry in
  /var/yp/Makefile, touch /var/yp/passwd,
  etc. etc. (see above)
Outline by what processes the passwd file is
  used, and how they know its location (PWDIR setting
  in /var/yp/Makefile)
Give general information on NIS components and handling

```

- What wrong thinking can cause problems? (== Beliefs?)

```

wrong: ypsserv re-reads /var/yp/Makefile automatically.
wrong: ypsserv rebuilds NIS maps automatically
wrong: yppasswdd re-reads /var/yp/Makefile automatically

```

- What viewpoints may exist:

```

from inside

```

- Ways of data acquisition: checks, keyboard tracing, ...

```

checks:
proper entry in /var/yp/Makefile for PWDIR
see if /var/yp/passwd exists
compare output of 'ypcat passwd' and passwd file to see
  if NIS maps were rebuilt
see if yppasswdd works, to determine if the daemon was
  restarted

```

```

typescript:
see if yppasswdd was restarted (either way!)
see if ypsserv is restarted
see if NIS map is rebuilt (make in /var/yp)

```

<li> Legen Sie im NIS eine Kennung "ypuser" mit eindeutiger UID, Home-Verzeichnis "/usr/homes/ypuser", Korn-Shell als Login-Shell, und Passwort "ypuser" an.

- What does the student have to do?
  - Put appropriate line into /var/yp/passwd
  - Rebuild NIS maps
- What problems can occur, how can they be identified?
  - Student doesn't know where to put the data
  - Student doesn't know proper format
  - Student doesn't know how to rebuild the NIS map
- Help: behavioristic, epistemic, ...
  - Give proper line, file and procedure to rebuild
  - Hint at file and give a general hint on rebuilding the NIS map
  - Explain what to do about the UserID (UID)
  - Explain how password encryption works
  - Give URL of documentation for NIS and user management
- What wrong thinking can cause problems? (== Beliefs?)
  - useradd(8) or adduser(8) will know how to handle NIS
  - Incomplete entries (e.g. empty fields) in the passwd file are ok
  - User doesn't know that Korn-Shell == /bin/ksh
  - User doesn't know how to determine UID
- What viewpoints may exist:
  - from inside and outside machine: check NIS map
  - from inside only: check home directory
- Ways of data acquisition: checks, keyboard tracing, ...
  - Checks:
    - passwd file must contain proper fields
    - home directory must exist and have proper permissions
    - NIS map must be updated

<li> Stellen Sie sicher dass der User "ypuser" via finger(1) sichtbar ist

- What does the student have to do?
  - enable nis as source for nsswitch in /etc/nsswitch.conf
  - verify that "finger ypuser" works properly
- What problems can occur, how can they be identified?
  - fingerd is not started/enabled in inetd.conf
  - nsswitch.conf is not setup properly
  - NIS user doesn't exist (see previous steps)
- Help: behavioristic, epistemic, ...
  - give exact steps: enable fingerd, fix nsswitch, run finger
  - hint at files that may need changing
  - explain how finger works
  - explain role of inetd.conf for finger(d)
  - explain role of nsswitch/getpwnam for finger(d)
  - give general information on NIS, networking, enabling network services via inetd.conf, usermanagement

- What wrong thinking can cause problems? (== Beliefs?)
    - wrong: assume fingerd works per default
    - wrong: finger needs fingerd
    - wrong: system will check NIS automatically (instead of needing nsswitch)
  - What viewpoints may exist:
    - from inside system: finger ypuser
    - from outside system: finger ypuser@vulabl (needs fingerd)
  - Ways of data acquisition: checks, keyboard tracing, ...
    - checks:
      - run finger and expect proper output
- <li> Stellen Sie sicher dass sich der User "ypuser" via telnet, ssh und ftp einloggen kann!
- What does the student have to do?
    - log into vulabl via telnet, ssh, ftp
    - possibly start sshd first
    - possibly enable telnet and ftp in inetd.conf (and restart inetd)
  - What problems can occur, how can they be identified?
    - ftp and/or telnet not enabled in inetd
    - user does not know how to use telnet, ftp, ssh - detecting?
    - logging in with / as home directory -> doesn't exist
    - can't login due to password unknown
  - Help: behavioristic, epistemic, ...
    - discuss using services (ftp, telnet, ssh)
    - discuss service setup (sshd, telnetd, ftpd)
    - Explain possible problems during account creation and login, and how to solve them
  - What wrong thinking can cause problems? (== Beliefs?)
    - services are enabled by default
    - login will succeed without making sure password is set properly
    - login will succeed flawlessly with no home directory
  - What viewpoints may exist:
    - from inside:
      - check if processes run and all other pre-requirements are met to do the login
    - from outside:
      - perform logins, and verify it works (easy!)
  - Ways of data acquisition: checks, keyboard tracing, ...
    - checks: perform automated logins from outside
    - typescript logging: n/a
- <li> Stellen Sie sicher, dass der User "ypuser" sein Passwort mit



ypasswd(1) ändern kann.

- What does the student have to do?
  - login as ypuser
  - change password using 'ypasswd'
  - see it changed in the output of 'ypcat passwd'
- What problems can occur, how can they be identified?
  - ypasswdd wasn't restarted after changing PWDIR in /var/yp/Makefile
- Help: behavioristic, epistemic, ...
  - First, note doen (encrypted) password
  - Run ypasswd, logout & login with new password
  - Use 'ypcat passwd' to verify if password was changed (and NIS map updated)
- What wrong thinking can cause problems? (== Beliefs?)
  - passwd works always for NIS
  - ypasswdd catches up PWDIR-change automatically
- What viewpoints may exist:
  - from inside & outside:
  - same test, as not only the local passwd file, but also the update of the NIS map needs to be verified, which needs to be done via the network interface (ypcat)
- Ways of data aquisition: checks, keyboard tracing, ...
  - checks: change password manually
  - typescripts: nothing sensible

</ul>

<h2>2. Client (NetBSD): vulab2</h2>

<ul>

<li> Setzen Sie den Domainnamen auf den selben Namen wie beim NIS-Master oben.

- [See above]

<li> Ist das aufsetzen des Clients mit "ypinit -c" nötig? Ist es sinnvoll? Warum (nicht)?

- What does the student have to do?
  - Think what 'ypinit -c' does
  - No changes to the system needed
- What problems can occur, how can they be identified?
  - n/a
- Help: behavioristic, epistemic, ...
  - ypinit manpage
  - general documentation on NIS
- What wrong thinking can cause problems? (== Beliefs?)

```

wrong: ypinit -c must always be run

- What viewpoints may exist:

n/a

- Ways of data acquisition: checks, keyboard tracing, ...

n/a

<li> Stellen Sie sicher dass die nötigen Dienste (ypbind, ...) beim
booten gestartet werden.

- [see above]
- NetBSD needs changes to rc.conf

<li> Starten Sie die Dienste!

- [see above]
- NetBSD has files in /etc/rc.d

<li> Welcher NIS-Server wird verwendet?

- What does the student have to do?

Run 'ypwhich' and see which NIS server gets listed

- What problems can occur, how can they be identified?

Student doesn't know how to determine the said NIS server
Student doesn't know that the command needed now is
'ypwhich'

- Help: behavioristic, epistemic, ...

Tell student to run 'ypwhich'
Outline how NIS clients are bound to servers (broadcast,
or fixed)
Give general introduction to NIS

- What wrong thinking can cause problems? (== Beliefs?)

Student may not know what the proper command is
Student may not know where to look for related
information

- What viewpoints may exist:

form inside: run command on client

- Ways of data acquisition: checks, keyboard tracing, ...

checks: n/a

typescripting:
see if student runs 'ypwhich'
recognize student trying various commands but failing

<li> Stellen Sie sicher dass die NIS Maps (group, hosts, ...) abgerufen
werden können

- What does the student have to do?

run "ypcat passwd", "ypcat group", "ypcat hosts" etc. and
see if they produce proper output

```

- What problems can occur, how can they be identified?
  - student doesn't know to use ypcat on the various NIS maps
  - tries poking around instead (niscat, ...)
- Help: behavioristic, epistemic, ...
  - give commands to run, and how to interpret output
  - General NIS documentation
- What wrong thinking can cause problems? (== Beliefs?)
  - n/a
- What viewpoints may exist:
  - from inside client only
- Ways of data acquisition: checks, keyboard tracing, ...
  - checks:
    - see if ypcat gives proper data
  - typescript:
    - see if student runs ypcat with passwd, group, hosts
    - see if student tries to guess some command names without knowing them

<li> Stellen Sie sicher dass die NIS-Benutzer mit finger(1) abgefragt werden können

- What does the student have to do?
  - Fix nsswitch.conf
- What problems can occur, how can they be identified?
  - Student doesn't know about /etc/nsswitch.conf
  - Student doesn't know what to put there
  - Student doesn't know proper order of fields
  - Student adds 'nis' to the group/password field in the nsswitch.conf file, but doesn't remove 'compat'. This basically locks up the system.
- Help: behavioristic, epistemic, ...
  - Print proper filename and contents
  - Outline how finger(1) gets its information
  - Refer to documentation about name resolving (which includes data on nsswitch)
- What wrong thinking can cause problems? (== Beliefs?)
  - wrong: finger(1) works out of the box when ypcat works
  - "the operating system is broken!"
- What viewpoints may exist:
  - from inside: finger
  - from outside: finger ypuser@client -> fingerd
- Ways of data acquisition: checks, keyboard tracing, ...
  - checks:
    - see if nsswitch.conf lists 'nis' as data source
    - see if finger(1) returns proper data
    - see if getpwnam(3) works properly

```

typescript:
  see if student edits nsswitch.conf
  see if he uses finger properly for testing

```

<li> Stellen Sie sicher dass sich der oben angelegte Benutzer "ypuser" auf dem Client einloggen kann. Erstellen Sie das Home-Verzeichnis dazu vorerst manuell.

- What does the student have to do?
 

```

setup home directory
login as 'ypuser' (via telnet, ssh, ftp, login, su)

```
- What problems can occur, how can they be identified?
 

```

user forgets to create home
user doesn't know how to log in
user can't login due to incompatible password formats
(des vs. blowfish/md5, ...) - unlikely

```
- Help: behavioristic, epistemic, ...
 

```

give exact commands
refer to documentation for setting up user accounts and
  their home directories
refer to NIS documentation
refer to documentation on using su/ssh/telnet/ftp

```
- What wrong thinking can cause problems? (== Beliefs?)
 

```

wrong: home directory is needed to login

```
- What viewpoints may exist:
 

```

From inside: su / telnet localhost
from outside: ssh, telnet

```
- Ways of data aquisition: checks, keyboard tracing, ...
 

```

Checks:
check if homedir is present
use su from localhost
use ssh/telnet from outside

Typescript:
expect mkdir
expect some login attempts (via telnet, ssh, ...)
  either from inside or outside

```

<li> Betrachten Sie das Passwort-Feld der Passwort-Datei des Users "ypuser" auf dem NIS Master!.

- What does the student have to do?
 

```

Run 'ypcat passwd' and remember the password field
of the ypuser

```
- What problems can occur, how can they be identified?
 

```

Student doesn't know how to retrieve the NIS password file
  or where to find the password file
Student doesn't know what the password field is

```
- Help: behavioristic, epistemic, ...
 

```

Give command to run and what field to extract
Refer to ypcat and passwd(5) manpage
Refer to general documentation on NIS

```

Refer to general documentation on user management

- What wrong thinking can cause problems? (== Beliefs?)

Students expects to make changes to system  
Student doesn't know where to locate the requested information

- What viewpoints may exist:

from inside (only)

- Ways of data acquisition: checks, keyboard tracing, ...

Checks:  
n/a

Typescript:  
see if student looks at /var/yp/passwd file on NIS server

<li> Ändern Sie das Passwort von "ypuser" vom Client aus im NIS auf  
'`mynlspw`'.

- What does the student have to do?

run 'yppasswd' on NIS client

- What problems can occur, how can they be identified?

yppasswd hasn't caught up with PWDIR change, will  
print error about user not existing  
student mis-types password -> error  
student doesn't know how to change password via NIS,  
tries 'passwd' (w/o proper option)

- Help: behavioristic, epistemic, ...

Hint at exact command (yppasswd, passwd -y)  
Refer to yppasswd and/or passwd manpage  
Give general information about working in a NIS environment

- What wrong thinking can cause problems? (== Beliefs?)

wrong: the passwd command changes the NIS password automatically

- What viewpoints may exist:

from inside only

- Ways of data acquisition: checks, keyboard tracing, ...

Checks:  
check for new password afterwards

Typescript:  
check if user runs the right command successfully  
Recognize if user tries to run the wrong program

<li> Betrachten Sie das Passwort-Feld der Passwort-Datei des Users  
"ypuser" auf dem NIS Master erneut. Was stellen Sie fest?

- What does the student have to do?

Observe the password field in /var/yp/passwd  
(or the corresponding NIS map) again, as above.  
Note it has changed after running yppasswd from the client

- What problems can occur, how can they be identified?

```

        Password hasn't changed. If so, an error was indicated
        in the previous step.

- Help: behavioristic, epistemic, ...

    Ask user to check if yppasswdd was restarted after
        PWDIR check in /var/yp/Makefile
    Hint at troubleshooting NIS setups
    Give general documentation on how to operate NIS

- What wrong thinking can cause problems? (== Beliefs?)

    n/a

- What viewpoints may exist:

    From inside only:
    either on NIS server (-> check /var/yp/passwd)
    or on NIS client (-> run ypcat passwd)

- Ways of data acquisition: checks, keyboard tracing, ...

    Checks: n/a

    Typescript:
    see if user looked at /var/yp/passwd (server) or
    output of 'ypcat passwd' (client) again, and
    guess that he's coming to proper conclusions.

    Asking the user questions here could help to
    verify the thesis of drawing proper conclusions.

```

</ul>

<h2>3. Diverses</h2>

<ul>

<li> Setzen Sie den "Full Name" des Benutzers "ypuser" auf "NIS Testbenutzer". Verifizieren Sie das Ergebnis mit finger(1). Welche Methoden zum setzen existieren auf dem NIS Master? Welche auf dem NIS Client?

```

- What does the student have to do?

    Client:
        use chfn(1)

    Server:
        edit /var/yp/passwd file directly,
        rebuild NIS map

    Then:
        Use finger(1) on client or server and see if the full
        name is set properly

- What problems can occur, how can they be identified?

    User doesn't know which program to use to set the
        fullname, tries/guesses some variations
    User doesn't know how to interpret finger(1) output
    Student forgets to rebuild NIS maps after editing the
        passwd file manually (no need for that when done via
        chpasswd(1))

- Help: behavioristic, epistemic, ...

    Give exact commands to run: chfn, vi passwd&&make, finger
    Hint at possible commands to run

```

Give information about where the full name is stored, hint at the relationship to NIS, and explain how `finger(1)` accesses information from the client to the server

- What wrong thinking can cause problems? (== Beliefs?)

wrong: After changing the `passwd` file, the NIS server will pick up the changes automatically, and no rebuild of the NIS maps is needed

wrong: There's no way to change the value from the client,

wrong: the editing must be done as root on the server

wrong: a 'In real life:' value of '???' on Solaris is ok

- What viewpoints may exist:

Change can be done from the client or the server

Verification can happen on both the client and the server

- Ways of data acquisition: checks, keyboard tracing, ...

checks:

verify if `fullname` was set properly

typechecking:

verify if the name was set from the server (by editing the `passwd` file and rebuilding the NIS map)

verify if the name was set from the client (by running `chfn(1)`)

verify that the `finger(1)` command was ran properly (`"finger ypuser"`)

<li> Legen Sie eine NIS-Gruppe "benutzer" an, und machen Sie diese zur (primären) Gruppe des Benutzers "ypuser". Welche Group-ID wählen Sie? Warum?

- What does the student have to do?

Edit `/etc/group` on the server

Create a new line with a distinct GID

Use new GID as primary GID for user 'ypuser'

Rebuild NIS maps

- What problems can occur, how can they be identified?

Student doesn't know about `/etc/group`

Student doesn't know that `/etc/group` is used for the corresponding NIS map, and uses `/var/yp/group`

Student doesn't recognize format of the group file, and uses broken syntax (e.g. leaving out single fields!)

Student doesn't find a distinct GID, re-uses a pre-existing one

Student doesn't use new GID for `ypuser`, leaves it unchanged or doesn't use that of group "benutzer"

Student forgets to rebuild both group and `passwd` NIS maps, files and NIS maps are inconsistent

- Help: behavioristic, epistemic, ...

Give exact entries for `/etc/group` and `/var/yp/passwd` on the server and instructions to rebuild NIS maps

Give hints on procedure for rebuilding NIS maps

Give background information on `group(5)` and `passwd(5)`

Explain groups (primary, supplementary)

- What wrong thinking can cause problems? (== Beliefs?)

wrong: no need to update NIS maps after editing  
group+passwd  
wrong: NIS server rebuilds maps automatically  
wrong: groups file is under /var/yp  
wrong: primary group membership is in group(5) file  
instead of the GID in the passwd(5) file

- What viewpoints may exist:

from inside:  
changes need to be made from inside the server  
checking for proper settings and consistency can be made  
on the server  
checking for updated NIS maps can be made on the server

from outside:  
the client can verify proper settings esp. in the NIS maps

- Ways of data acquisition: checks, keyboard tracing, ...

checks:  
see if group 'benutzer' exists w/ proper values  
see if GID in group map and passwd field is consistent

typescript:  
see if user edits wrong file (e.g. /var/yp/passwd)  
see if user did not run 'make' in /var/yp on server

<li> Legen Sie im Home-Verzeichnis des Benutzers "ypuser" auf dem  
Master und dem Client eine Datei an, und überprüfen Sie, welcher  
Gruppe sie gehört.

- What does the student have to do?

Create file on client and server  
Observe group

- What problems can occur, how can they be identified?

File may not belong to group 'benutzer' on both client  
and server, if user didn't login after changing the  
group ownership

- Help: behavioristic, epistemic, ...

Tell user to login again before doing this, on both  
client and server  
Reference login process, which sets a process'  
credentials (group!)

- What wrong thinking can cause problems? (== Beliefs?)

wrong: changing a user's entry in the passwd file  
automatically updates any process' credentials  
wrong: changing a user's GID automatically takes his  
files to the new group

- What viewpoints may exist:

from inside:  
for each of the corresponding machines (server for the  
file created on the server, dito for client)

from outside:  
for each other of the corresponding machines (from the  
server for the file created on the client, and vice  
versa)



- Ways of data acquisition: checks, keyboard tracing, ...

Checks:

see if the file belongs to the right group for each of the machines

Typescript:

see if/how user creates file (there are many possible ways!). Probably better to just check effect...

<li> Sorgen Sie dafür dass der Benutzer "ypuser" auf dem NetBSD-System mittels su(1) root-Rechte erhalten kann. Er muss dazu (unter NetBSD) zusätzlich Mitglied der Gruppe "wheel" sein.

- What does the student have to do?

Add user 'ypuser' to group 'wheel' in the client's /etc/group possibly try "su"

- What problems can occur, how can they be identified?

the corresponding user's primary group is set to wheel (in the passwd file; which won't work!)  
user is added to the NIS group file - this may not work!

- Help: behavioristic, epistemic, ...

Tell user to put add ",ypuser" to wheel-line of client's /etc/group file, then run su(8)  
refer to su(8) and group(5) manpages  
explain difference between primary and supplementary groups  
refer to login process for when process credentials are set

- What wrong thinking can cause problems? (== Beliefs?)

wrong: su(8) will always work  
wrong: setting primary group to wheel (0) is ok  
wrong: need to change NIS "group" file/map  
wrong: change is in effect immediately afterwards, instead of after next login

- What viewpoints may exist:

from inside client:  
change client's /etc/group file  
login, run su(8), see if it works

from inside server:  
TRY to change the group file, may not work...

- Ways of data acquisition: checks, keyboard tracing, ...

checks:

see if su(8) succeeds on the client  
see if user is in /etc/group on the client

typescript:

see if /etc/group was edited (difficult)  
see if user successfully ran su(8)

<li> Wie bewerten Sie die Tatsache dass das root-Passwort alleine nicht reicht, sondern auch die richtige Gruppenzugehörigkeit Voraussetzung für einen su(1) auf root ist? Vergleichen Sie zwischen NetBSD, Solaris und Linux!

- What does the student have to do?
  - think - no modification to the system required
  - recognize that proper group membership may not be easy to achieve by a malicious user, thus increasing security by requiring proper group membership
- What problems can occur, how can they be identified?
  - user does not recognize security benefit, sees step as extra hassle (see old linux su(8) manpage WRT system administrators); can't be detected easily
- Help: behavioristic, epistemic, ...
  - tell user that security is enhanced because a (already) privileged account needs to "invite" others
  - give general information on system security, user credentials and how to achieve "system" privileges
  - discuss password security
- What wrong thinking can cause problems? (== Beliefs?)
  - wrong: this is too much effort
  - wrong: this works the same everywhere
  - wrong: no special group membership is needed for su(8) - this is special on BSD
- What viewpoints may exist:
  - from inside client only
  - from outside: for NIS map delivery at best
- Ways of data acquisition: checks, keyboard tracing, ...
  - checks:
    - nothing to check
  - typescript:
    - nothing to check

<li> Der Rechner "tab" (IP-Nummer: 194.95.108.32) soll via NIS bekannt gemacht werden. Tragen Sie den Rechner auf dem Server in die entsprechende Hosts-Datei ein, aktualisieren Sie die NIS-Map und verifizieren Sie das Ergebnis mittels ypcat(1) und ping(1) sowohl auf dem NIS-Master als auch auf dem NIS-Client.

- What does the student have to do?
  - add line '194.95.108.32 tab' to server's /etc/hosts
  - rebuild NIS map(s)
  - use 'ypcat hosts' on client and server, verify that 'tab' is listed
  - run 'ping tab' on client and server'
- What problems can occur, how can they be identified?
  - user doesn't know proper format for /etc/hosts
  - user doesn't remember to rebuild the NIS map
  - /etc/nsswitch.conf not configured to grab 'hosts' entries via 'nis' (on both client and server)
- Help: behavioristic, epistemic, ...
  - give exact steps (see above)

hint at server's /etc/hosts and NIS  
 explain how name resolution works: gethostbyname(),  
 nsswitch, NIS, ypbind, ypserv, /etc/hosts on server

- What wrong thinking can cause problems? (== Beliefs?)

wrong: this is related to DNS  
 wrong: no need to rebuild NIS map

- What viewpoints may exist:

from "inside" NIS domain: both for client & server

from "outside" NIS domain: not doable

- Ways of data acquisition: checks, keyboard tracing, ...

checks:  
 see if "ping tab" works  
 see if "ypcat hosts" brings proper entry for 'tab'  
 see if gethostbyname("tab") returns a proper value,  
 e.g. with Perl

typescript:  
 see if /etc/hosts is modified ("vi /etc/hosts")  
 see if NIS map is rebuilt ("cd /var/yp ; make")  
 see if ping is ran on client and server

</ul>

<h2>Hinweise:</h2>

<ul>

- <li> Solaris-Pakete für bash und tcsh liegen in /cdrom, Installation mit pkgadd(1M).

- What does the student have to do?

```
cd /cdrom
pkgadd -d . SUNWtcsh
pkgadd -d . SUNWbash
```

- What problems can occur, how can they be identified?

student doesn't recognize this step as optional  
 student doesn't know how to add binary packages, tries  
 various commands (rpm, pkg\_add, pkgadd with varying  
 syntax, reads manpages, ...)

- Help: behavioristic, epistemic, ...

give exact commands to run (see above)  
 hint at pkgadd(1M) manpage  
 give overview on package management

- What wrong thinking can cause problems? (== Beliefs?)

wrong: bash & tcsh are installed by default  
 wrong: pkgadd just uses a package name/directory as  
 argument, similar to pkg\_add / rpm  
 wrong: package installation can be done w/o root  
 privileges

- What viewpoints may exist:

from inside only

- Ways of data acquisition: checks, keyboard tracing, ...

```

checks:
see if SUNW{bash,tcsh} are installed

typescript:
recognize problems as outlines above

<li> NetBSD-Pakete für bash und tcsh (und weitere) liegen auf
ftp://ftp.de.netbsd.org/pub/NetBSD/packages/1.6.1/sparc/All,
Installation mit pkg_add(1).

- What does the student have to do?

use ftp(1) to retrieve file, or use pkg_add(1) directly
use pkg_add(1) to install package

- What problems can occur, how can they be identified?

student doesn't recognize this step as optional
student doesn't know how to add binary packages, tries
various commands (rpm, pkg_add, pkgadd with varying
syntax, reads manpages, ...)
Student doesn't know how to fetch files
Student doesn't know that pkg_add can fetch the files
automatically
Packages may be moved from the above-named place(!)
Student may not have the necessary directories in $PATH
to run the binaries installed, resulting in "bash:
command not found" etc.

- Help: behavioristic, epistemic, ...

Give exact commands for installation and running
Reference manpages
Hint at adjusting PATH if necessary
Give general information on package handling

- What wrong thinking can cause problems? (== Beliefs?)

wrong: pkg_add knows where to find binary packages
automatically (w/o setting PKG_PATH)
wrong: Installed executables may reside in $PATH

- What viewpoints may exist:

from inside only

- Ways of data acquisition: checks, keyboard tracing, ...

checks:
see if both packages are installed

typescript:
recognize any problems in operating the commands properly
detect $PATH problems (esp. looking at the output of
commands ran)

</ul>

```