

The Heuristic Evaluation Manager (HEM)

An Online Collaborative Environment for Heuristic Evaluation

Martin Loitzl

The Heuristic Evaluation Manager (HEM)

An Online Collaborative Environment for Heuristic Evaluation

Master's Thesis

at

Graz University of Technology

submitted by

Martin Loitzl

Institute for Information Systems and Computer Media (IICM),
Graz University of Technology
A-8010 Graz, Austria

February 2006

© Copyright 2006 by Martin Loitzl

Advisor: Ao.Univ.-Prof. Dr. Keith Andrews

Der Heuristic Evaluation Manager (HEM)

Eine online Umgebung für kooperative heuristische Evaluierung

Diplomarbeit
an der
Technischen Universität Graz

vorgelegt von

Martin Loitzl

Institut für Informationssysteme und Computer Medien (IICM),
Technische Universität Graz
A-8010 Graz

Februar 2006

© Copyright 2006, Martin Loitzl

Diese Arbeit ist in englischer Sprache verfasst.

Betreuer: Ao.Univ.-Prof. Dr. Keith Andrews

Abstract

Heuristic evaluation is a usability inspection method, in which a small team of expert evaluators working alone review a user interface according to a small set of principles (or heuristics) and use their experience and judgement to assemble a list of potential usability problems. An evaluation manager then typically synthesises an aggregate list of problems from the individual lists of each inspector.

The Heuristic Evaluation Manager (HEM) is a collaborative web application which assists in every aspect of heuristic evaluation. Evaluators are given accounts in an evaluation project and enter their findings and supporting screenshots online. The evaluation manager uses HEM to assemble an aggregate merged list of findings. The evaluators then enter severity ratings for each usability problem into HEM. Finally, HEM supports sorting the findings in decreasing order of severity and generating a draft evaluation report in XHTML. HEM is implemented in PHP4 with XHTML and CSS style sheets and uses a relational database such as MySQL.

This thesis gives an overview of the methods of usability engineering in general and usability inspection methods in particular, including the technique of heuristic evaluation and various proposed sets of heuristics. The architecture, implementation, and use of HEM are discussed in detail. The thesis includes HEM user guides and a HEM developer guide as appendices.

Kurzfassung

Heuristische Evaluierung ist eine Methode zur Überprüfung der Usability einer Mensch-Maschine Schnittstelle. Ein Team von Experten erstellt mit Hilfe einer kleinen Menge von Richtlinien (Heuristiken) und persönlichen Erfahrungen eine Liste potenzieller Usability Probleme. Ein Evaluierungsleiter fasst diese individuellen Listen in eine Sammlung von Problemen zusammen.

Der Heuristische Evaluierungsmanager (HEM) ist eine Web Applikation die alle Aspekte einer gemeinschaftlichen heuristischen Evaluierung unterstützt. Die einzelnen Evaluatoren bekommen Zugang zu einem Evaluierungsprojekt um online ihre individuellen Problemlisten, welche mit Bildern illustriert werden können, zu erstellen. Der Evaluierungsleiter benutzt HEM um die individuellen Problemlisten zu einer Gesamtliste zusammenzufassen, welche danach von den einzelnen Evaluatoren nach Wichtigkeit der Probleme mittels HEM gereiht wird. HEM erstellt aus diesen Daten einen XHTML Bericht, der die Probleme sortiert nach der Wichtigkeit beinhaltet. HEM ist in PHP4 implementiert, benutzt XHTML und CSS, sowie eine relationale Datenbank zur Datenhaltung.

Diese Arbeit bietet eine allgemeine Übersicht über Usability Engineering Methoden und speziell über Inspektionsmethoden, einschließlich heuristischer Evaluierung und dabei verwendeter Heuristiken. Weiters präsentiert diese Diplomarbeit detailliert die Softwarestruktur, die Implementierung und Anwendungsbeispiele von HEM. Ein Benutzerhandbuch und eine Anleitung für Softwareentwickler befinden sich im Anhang.

I hereby certify that the work presented in this thesis is my own and that work performed by others is appropriately cited.

Ich versichere hiermit, diese Arbeit selbständig verfaßt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient zu haben.

Acknowledgements

I would like to thank:

- Keith Andrews for his kind advice and patience,
- Harald Auer, Alois Dengg, and Joe Zehetner for participating in the TUGraz HEM project,
- Gerald Aichholzer for sharing his knowledge on \LaTeX with me,
- and all the countless other people who made this thesis possible.

Martin Loitzl
Graz, Austria, February 2006

Credits

- This thesis was written using Keith Andrews' wonderful skeleton thesis [Andrews, 2004].
- The Gang-of-Four pattern map in Chapter 7 is from the Gamma et al. [1995] book.

Contents

1	Introduction	1
2	Usability Engineering	3
2.1	Usability Engineering	3
2.2	Usability Engineering Lifecycle	6
2.3	Usability Evaluation Methods	10
2.4	Discount Usability	12
3	Usability Inspection Methods	15
3.1	Heuristic Evaluation	15
3.2	Cognitive Walkthrough	16
3.3	Heuristic Walkthrough	18
3.4	Participatory Heuristic Evaluation	20
3.5	Guideline Reviews	21
3.6	Pluralistic Walkthroughs	21
3.7	Perspective-Based Inspection	23
3.8	Consistency Inspections	26
3.9	Formal Usability Inspections	27
4	Heuristic Evaluation	31
4.1	Conducting a Heuristic Evaluation	31
4.2	Number of Evaluators	34
4.3	Experience of Evaluators	35
4.4	The Kind of Problems Found by Heuristic Evaluation	36
4.5	Result Generation	36
5	Sets of Heuristics	39
5.1	Molich and Nielsen 1991 (10 Heuristics)	39
5.2	Nielsen 1994 (10 Heuristics)	40
5.3	Muller and McClard 1998 (15 Heuristics)	41
5.4	Online Computer Library Center (14 Heuristics)	42
5.5	Instone: Web Heuristics 1997 (10 Heuristics)	43
5.6	Skinner and McMullin: Rich Internet Application Heuristics 2003 (10 Heuristics)	44
5.7	Shneiderman: Eight Golden Rules of Interface Design 1997 (8 Heuristics)	45
5.8	W3C: Accessibility Heuristics 2001 (6 Heuristics)	45
5.9	Purho: Documentation Heuristics 2000 (10 Heuristics)	46
6	Computer-Supported Heuristic Evaluation	49
6.1	uzReview: A Computer Supported Heuristic Evaluation Tool	50
6.2	Review Assistant	51

7	Pattern-Oriented Software Development	57
7.1	Properties of a Pattern	57
7.2	Definition of a Pattern	58
7.3	Classification of Patterns and Pattern Systems	59
7.4	Patterns Applied in HEM	61
7.5	Software Frameworks	69
8	PHP Web Application Design	73
8.1	Important Features of Web Applications	73
8.2	PHP Security Issues	75
8.3	Best Practises	82
9	The Heuristic Evaluation Manager (HEM)	85
9.1	Components of HEM	85
9.2	The Application Architecture of HEM	107
9.3	The HEM Application Framework	109
9.4	The HEM Application Programming Interface (API)	112
10	Selected Details of the Implementation	113
10.1	The HEM Application Programming Interface (API)	113
10.2	The HEM Application Framework	122
11	HEM in Use for the TUGraz Project	127
11.1	The TUGraz Project Heuristics	127
11.2	The TUGraz Project	127
11.3	The TUGraz Project Evaluation Environment	127
11.4	The TUGraz Project Rating Scheme	130
11.5	Evaluation Procedure	130
11.6	Discovered Results	132
11.7	Feedback from Users of HEM	134
12	Outlook and Future Work	135
12.1	Generation of Statistics	135
12.2	Advanced Report Editing	135
12.3	XML-RPC or SOAP Interface	136
12.4	Increase the Speed of the Finding Merger	136
12.5	New Authentication Mechanisms	136
12.6	Evaluating Various Kinds of Interfaces	136
13	Concluding Remarks	137
A	HEM Manager Guide	139
A.1	The HEM Window	139
A.2	Heuristic Sets	140
A.3	Rating Schemes	142
A.4	Rating Scales	143
A.5	Evaluation Environments	143
A.6	Finding Merger	144
A.7	User Manager	145
A.8	Project Manager	146
A.9	Report Generation	148

B HEM Evaluator Guide	149
B.1 The HEM Window	149
B.2 The Home Application	151
B.3 Entering the Evaluation Environment	151
B.4 The Finding Collector	151
B.5 The Rating Collector	151
C HEM Developer Guide	155
C.1 Application Architecture	155
C.2 Template Architecture	159
C.3 Directory Structure of HEM	160
C.4 A HEM Example Application	162
C.5 A HEM Example Box	167
D HEM Report Example	173
E Wikipedia Entries	189
Bibliography	197

List of Figures

2.1	The human-centred design model	4
2.2	Three different dimensions of user experience	7
2.3	Learning curves for novice and expert users	8
3.1	The Human-Computer Interaction Model proposed by Zhang et al. [1998] based on the seven stages of action in Norman [1988].	25
3.2	A human task performance model used in formal usability inspections.	28
4.1	The relation between the number of evaluators and the number of usability problems found.	34
4.2	The relation between the benefits and costs of a heuristic evaluation for a different number of evaluators.	35
6.1	Creating an issue in Review Assistant	52
6.2	The list of collected issues in Review Assistant	53
6.3	The Checklist screen of Review Assistant	54
6.4	The Report tab of Review Assistant	55
7.1	The Gang-of-Four pattern map in Gamma et al. [1995]	62
7.2	UML diagram of an adapter using multiple inheritance	63
7.3	UML diagram of an adapter using object composition	64
7.4	UML diagram of a singleton	65
7.5	A layered software structure	67
7.6	The layers of the TCP/IP protocol family	68
9.1	A typical HEM workflow	86
9.2	A list of users shown in the User Manager	87
9.3	The form to change a user's attributes	88
9.4	The overview of all sets of heuristics	89
9.5	The form to edit a set of heuristics	90
9.6	An environment shown in the HEM Environment Manager	91
9.7	A rating scale shown in the HEM Rating Scale Manager	92
9.8	The form to create a rating scheme	93
9.9	An overview of projects in the Project Manager	94
9.10	The form to change project settings	96
9.11	The form to import a HEM project file	97
9.12	A view of the Environment Collector	98
9.13	A list of findings shown in the Finding Collector	99
9.14	The form to change a finding's attributes	101
9.15	A list of findings shown in the Finding Merger	102
9.16	Assigning an evaluator's finding to a manager's finding	103
9.17	An overview of projects in the merging phase of a heuristic evaluation	104

9.18	The list of screenshots submitted by the evaluators	104
9.19	The appearance of the form to change a manager finding if evaluator screenshots have been chosen	105
9.20	A view of the Ratings Collector	106
9.21	The software architecture of HEM interpreted as a layered structure	108
9.22	The software architecture of HEM interpreted as a Model-View-Controller (MVC) design pattern	109
11.1	Instone's heuristics in the HEM Heuristic Set Manager	128
11.2	The TUGraz project settings	129
11.3	The Environment Collector for the TUGraz Project	131
11.4	A five point rating scale used in the TUGraz project.	132
11.5	The number of findings discovered individually by each evaluator on the <i>www.tugraz.at</i> web site	133
A.1	The Home application of HEM	140
A.2	The form to edit a set of heuristics	141
A.3	The form to create a rating scheme	142
A.4	A rating scale shown in the HEM Rating Scale Manager	143
A.5	The merge view of the Finding Merger	144
A.6	Assigning an evaluator's finding to a manager's finding	145
A.7	A list of users shown in the User Manager	146
A.8	The form to change a project settings.	147
B.1	The Home application of HEM	150
B.2	The evaluator's add finding form	152
B.3	The overview of findings in the finding collector	153
B.4	The Ratings Collector	154
C.1	The HEM Template Structure	160

List of Tables

3.1	Six categories of design guidelines.	22
3.2	The members and responsibilities in a formal usability inspection team.	28
4.1	Two five point rating scales used for heuristic evaluation.	33
7.1	Gamma et al. [1995] design pattern classification.	60
A.1	Icons used in HEM	141
B.1	Icons used in HEM	150
C.1	Parameters of the HEM Application Framework	156
C.2	The HEM directory structure	161

Listings

8.1	A web form using a relative action address containing a drop-down field.	75
8.2	A web form with a text field to submit arbitrary values.	75
8.3	The HTTP Request of a form submission.	76
8.4	Spoofing a HTTP request using PHP.	76
8.5	An entry to a website which sends the victims cookies to the attackers server.	77
8.6	A PHP script which assembles an SQL insert statement.	79
8.7	An SQL query which contains an SQL injection.	79
8.8	An example for a method which makes session hijacking more difficult.	81
8.9	An example of a method call with many parameters.	83
8.10	An example of a method call with an associative array as parameter.	83
10.1	Data retrieval using the DBObject class	114
10.2	An example for an SQL <i>UPDATE</i> query	115
10.3	A helper method for updating data	115
10.4	Record update using the DBObject class	116
10.5	The Heuristic class extending the DBObject class	117
10.6	Usage of DBObject functionality in a derived class	118
10.7	An example for the data structure returned by the <code>getHeuristic</code> method	119
10.8	An example for a simple database filter	120
10.9	An example for a complex database filter	120
10.10	The implementation of the filter method	121
10.11	An example for an application derived from a framework	122
10.12	The constructor of a small example software framework	124
10.13	The <code>setDefault</code> method of the framework	124
C.1	Deriving an application from the HEM application framework.	162
C.2	Running a HEM application	163
C.3	An application configuration file.	163
C.4	Controlling user interaction with <i>GET</i> request fields.	164
C.5	Usage of the HEM API.	165
C.6	A dump of an example HEM object data array.	166
C.7	Usage example of the HEM Translator.	166
C.8	An example HEM language file	167
C.9	The Stats template	168
C.10	Usage of the <code>showScreen</code> method.	169
C.11	Registering a box in the global configuration file.	170
C.12	Configuration file of a HEM box.	170
C.13	The constructor of the <code>StatsBox</code> class.	171
C.14	The <code>getStatsBox</code> method of the <code>StatsBox</code> class.	172

Chapter 1

Introduction

This thesis describes the Heuristic Evaluation Manager (HEM) a web-based application which supports experts in conducting heuristic evaluation.

Chapters 2 to 8 embed this work into the research context. In Chapter 2 the term of usability engineering is introduced and the place of usability inspections in the usability engineering lifecycle is illustrated. Chapter 3 discusses common usability inspection methods, the class of methods to which heuristic evaluation belongs. Chapter 4 describes heuristic evaluation in more detail and outlines the work flow, benefits, and draw backs of the method. Chapter 5 presents different sets of heuristics which are used for heuristic evaluation. Chapter 6 describes characteristics of computer supported heuristic evaluation tools and discusses two projects in particular. Chapter 7 gives an overview of pattern-oriented software development, a technique used for the implementation of HEM. Chapter 8 discusses web application development with PHP, a commonly used programming language for implementing web applications.

Chapters 9 to 12 present the work done within the practical part of this thesis. Chapter 9 provides an introduction to the Heuristic Evaluation Manager, its application architecture and its components, and how it supports the process of a heuristic evaluation. Chapter 10 describes selected details of the implementation. In Chapter 11 an example of using HEM is presented.

This thesis concludes with Chapter 12, which discusses possible future extensions of HEM and usage scenarios which may be useful for usability engineering research.

Appendix B provides a guide for evaluators and Appendix A provides a guide for HEM managers. Appendix C provides a comprehensive inside for HEM developers. Appendix D contains an heuristic evaluation report created with HEM.

Chapter 2

Usability Engineering

Usability is defined in ISO 9241 part 11 [ISO 9241-11, 1998] as the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use. The three attributes stated in the definition are:

Effectiveness: How well can users achieve their goals using the system?

Efficiency: How many resources are consumed to achieve the goals?

Satisfaction: How does a user feel when using a system?

Jakob Nielsen defines usability in the context of the overall system acceptability in Nielsen [1993] and associates usability with the following five attributes:

Learnability: A system should be easy to learn.

Efficiency: A system should be efficient to use. Once a user has learnt to use the system, a high level of productivity should be possible.

Memorability: A user should be able to use the system after a longer period of not using it without learning again.

Errors: The system should be designed in a way that prevents users from making errors. If a user makes errors, it should be easy to recover from them. The system should not allow catastrophic errors.

Satisfaction: Users should be subjectively satisfied when using the system.

2.1 Usability Engineering

Usability engineering is a methodological way to improve the usability of a human-computer interface. The ISO standard 13407 [ISO 13407, 1999] provides guidance for a human-centred design process for interactive systems. The standard provides a framework for user-centred development activities, which can be adopted to different kinds of development models, starting from the waterfall model to iterative design process models. Five essential human-centred design activities are presented in the standard. These activities should take place during the whole development project. The human-centred design process should be started at the beginning of a project and should be repeated iteratively. The four stages of the ISO 13407 human-centred design model are illustrated in Figure 2.1.

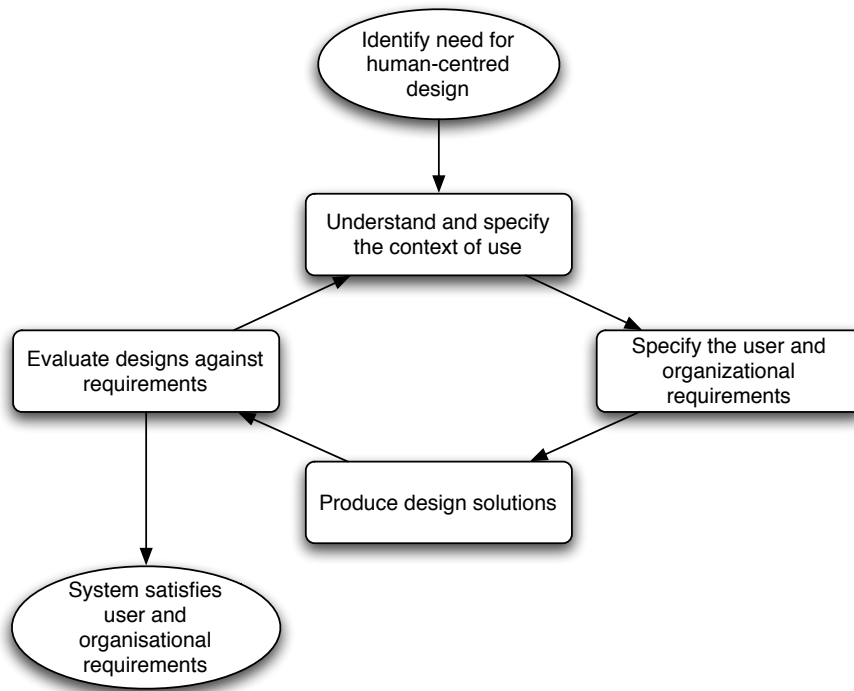


Figure 2.1: The four stages of the ISO 13407 [1999] human-centred design model are repeated iteratively until the product satisfies the specified user and organisational requirements.

2.1.1 Context of Use

The first stage in the ISO 13407 design model is to understand and specify the context of use, which is defined by the characteristics of the future users, their tasks, and the organisational and physical environment. A detailed understanding of the context will guide early design decisions and provides a basis for evaluation. The following factors identify the context of use of a system:

The characteristics of the future users: Important characteristics of users are knowledge, skill, experience, education, training, physical attributes, habits, preferences, and capabilities.

The tasks to be performed by the users: Tasks should not be described in terms of functions or features provided by the system. The task descriptions should contain: the overall goals of the tasks, characteristics that influence usability, implications for health and safety, activities and organisational steps between the human and technological resources.

The environment in which the system is used: This includes the used hardware, software, and materials, as well as the physical and social environment.

2.1.2 Specification of the User and Organisational Requirements

Most design processes include major activities of specifying functional and other requirements. This activity should be extended for a human-centred design to explicitly define user and organisational requirements. The ISO 13407 standard advises considering the following aspects:

- The required performance of the new system in operational and financial terms.

- Relevant legislative requirements, including safety and health.
- Cooperation and communication between users and other involved groups.
- The users tasks, including allocation of tasks, the user's well-being, and the user's motivation.
- Task performance.
- Work design and organisation.
- Change management.
- Operability and maintenance.
- The human-computer interface and the workplace.

The defined requirements are used to derive usability criteria and to set targets that consider the trade-offs identified between different requirements. The result should be in a form that allows subsequent testing. The standard recommends using the objectives defined in ISO 9241-11 [1998]: efficiency, effectiveness, and satisfaction.

2.1.3 Production of Design Solutions

Design solutions should be built upon the established state of the art. Existing scientific knowledge and theory, such as ergonomics, psychology, cognitive science, should be used as input for design solutions. User interface guidelines, existing standards, product knowledge of past developments, and marketing information will also provide useful information.

Simulations, models and other types of prototypes provide an effective way of communication with users and within the development team. Prototyping makes design decisions more explicit, it allows evaluation of several concepts before a decision is made. User feedback can be gained early in the development process and the quality and completeness of functional design specifications is enhanced. Prototypes can be produced very early in the design process. If it is impractical to confront users with prototypes early in the development process, evaluations can also be conducted by experts. Expert reviews should not replace user testing. Results of inspections and user comments about the prototype provide important input for design changes which can improve the usability of the product.

2.1.4 Evaluation Against Requirements

The evaluation stage is essential in human-centred design. Evaluations can deliver formative feedback used to improve the design, or summative feedback to assess if user and organisational requirements have been met. Evaluation results are as good as the context in which the system has been tested. Formative feedback can be obtained early in the development process. Summative feedback can be obtained later when a realistic prototype is available. The used evaluation technique depends on the environment in which the evaluation is performed, financial and time constraints, as well as the stage in the development cycle. Expert evaluation is usually faster and cheaper than user-based evaluation and will discover major usability problems. Expert evaluation must not replace user-based evaluation entirely. User-based evaluations can provide feedback at any stage of the of the design and are based on progressively more complete and concrete versions of the product.

The major benefit of iterative design is that the cost of change is low at early development stages. It is important to start evaluation as early as possible and to conduct evaluations throughout all development stages.

Usability evaluation should also be continued after the product release. Field validation is used to test the final system for conformance with the requirements of users, tasks, and the environment. The

main techniques suggested in the standard are using help-desk data and real user feedback, field reports, performance data, reports of health impacts, design improvements, and change requests. Means for long-term monitoring should be included in the product to collect data of the system usage. Long-term monitoring helps to reveal issues which are only recoverable after a longer period of system usage, for example changes in working places. It is also useful to see if performance requirements are met.

2.2 Usability Engineering Lifecycle

Usability measures should be taken as early as possible in the development of a product. Usability activities should be part of the whole development phase and cannot be seen in isolation. The earlier a usability problem is detected in a design, the cheaper it is to correct the problem.

The following list illustrates the usability engineering lifecycle. The phases are summarised in Andrews [2006] based on the usability engineering lifecycle published in Nielsen [1993].

- Know the User
- Usability Benchmarking
- Goal-Oriented Interaction Design
- Iterative Design:
 - Prototyping
 - Usability Evaluation (Inspection and Testing)
- Follow-Up Studies

Usability effort can still be useful, even if not every possible measure is taken. Little usability engineering is better than none. This concept is referred to as “Discount Usability” and is discussed in Section 2.4. The following sections give a short outline of each step of the usability engineering lifecycle.

2.2.1 Know the User

The first step in the usability engineering lifecycle is to identify the future users of the developed product. A user is every person whose work may be affected by the product.

There are several ways to classify the user population: experience, educational level, age, amount of prior training, etc. According to Nielsen [1993] there are three main dimensions in which user experience may differ: experience with the system, experience with computers in general, and experience with the task domain. Figure 2.2 illustrates the three dimensions in the “user cube”, as it is denoted in Nielsen [1993].

The amount of prior training needed to use a system takes learnability into account (Figure 2.3). A system may be designed for ease of learning. Such a system may be easy to learn, but may slow down users who have reached a certain level of expertise. Systems may be used rarely, for example an information system in a museum should require no prior learning. A system may also be developed with a focus on expert users. The learning curve is steep at the beginning, but if a user reaches a higher level of experience the system can be used very efficiently. Another way of system design is to support both novice and expert users. Features such as accelerators, useful for experienced users, may be hidden from novice users at the beginning. This allows a user who has reached a certain level of experience to switch to an expert mode speeding up their work.

Another essential input of system design is to study the users’ overall goals. The users’ model of their tasks should be examined to identify metaphors for the user interface. Identifying the weaknesses

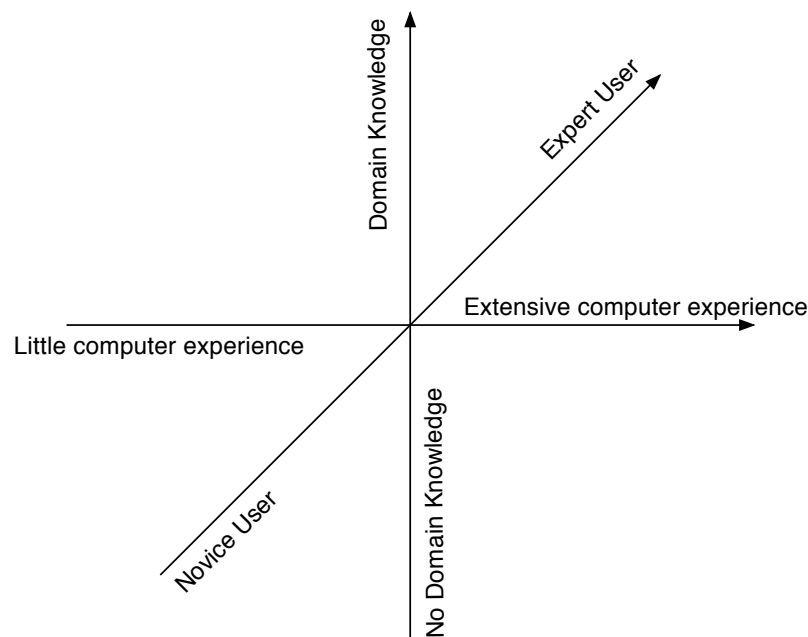


Figure 2.2: User experience differs in three dimensions: computer experience, experience with the system, and knowledge about the task domain [Nielsen, 1993, pages 43–48].

of the current situation provides the chance for substantial improvements in a new product. A task analysis will provide information about the users' goals, the information needed to achieve the goals, the steps that need to be performed, dependencies within the steps, and criteria that identify the quality and acceptability of a task's outcome.

The next step is to analyse the underlying functional reasons for a task. This identifies what really has to be achieved and can reveal other and better ways to achieve a goal.

Users will change their behaviour when using the system. The users will learn to use the system in new ways [Nielsen, 1993]. A user will likely never become an absolute expert in a particular system, but will also not remain a beginner. Hence, users are often perpetual intermediates [Cooper, 2004].

Although various ways to get to know the user are presented in Nielsen [1993], it is also stated that there is no general solution for the problem. The best way is to directly contact future users. Hackos and Redish [1998] describe various techniques for gathering data, analysis, and for moving from analysis to design. Kuniavsky [2003] provides a detailed coverage of 13 user experience research techniques for web, desktop, and mobile applications.

2.2.2 Usability Benchmarking

Usability benchmarking considers the current state of the art, competing products, usability targets, and the return of investment on usability procedures [Andrews, 2006; Nielsen, 1993].

The best prototypes available are already released products of competitors. Existing products can usually be easily analysed empirically with user testing, or heuristically with usability inspections. Defining usability targets allows the specification of how much better the product should be in terms of usability [Nielsen, 1993].

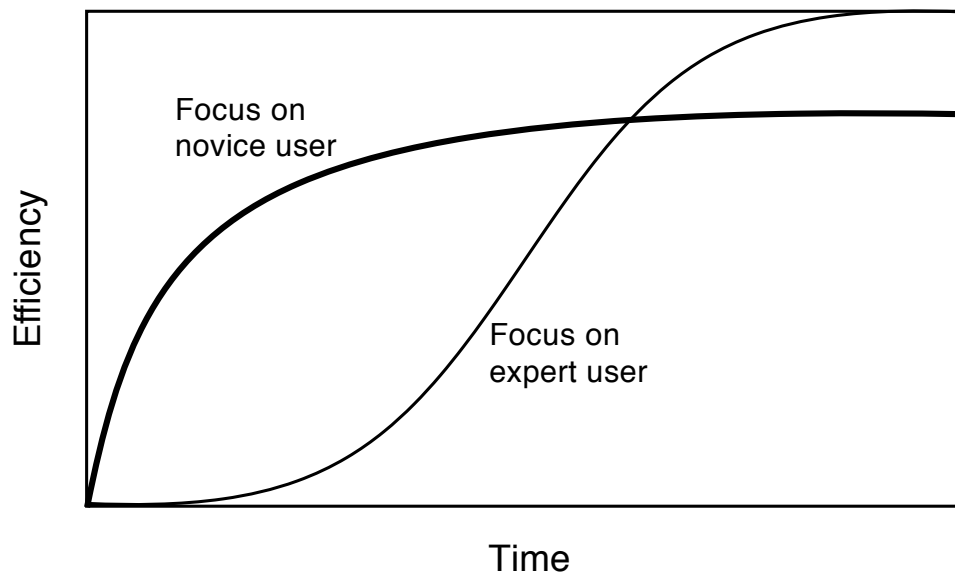


Figure 2.3: A system which is designed for ease of learning may be less efficient to use. A system designed for expert users may be hard to learn, but is highly efficient for expert users. A well-designed system should support both learning curves [Nielsen, 1993, pages 27–30].

Applying usability measures throughout the whole development process returns many benefits. Marcus [2002] presents key benefits of usability and defines appropriate value propositions. Nielsen [2003] states that current best practise is to invest 10% of the project budget in usability measures. Bias and Mayhew [2005] provide a comprehensive guide on how to provide economic arguments for usability procedures.

2.2.3 Goal-Oriented Interaction Design

The method of Goal-Oriented Interaction Design was developed by Alan Cooper [Cooper, 2004]. The foundation of the method is to define a pretend user and to design for that particular user. The pretend user is called a *persona*. Personas are not actual users but hypothetical prototypes of a particular type of user. Personas are used to check if the design meets the requirements of the user represented by the persona.

Personas are used to define goals. A goal is an end condition reached by performing a sequence of tasks. The kind of tasks performed to reach a certain goal may change, but the goal stays the same. A feature of a user interface only has meaning if it exists for a purpose. The purpose, or goal, cannot exist without the person having it, the persona.

Personas and goals are used to create usage scenarios which are used to validate the design during the design process. A scenario describes a persona using the user interface to achieve a goal. It is important to develop scenarios that support the design process. Cooper [2004] describes three kinds of scenarios: daily-use, necessary use, and edge-case scenarios. Edge-case scenarios will not be as influential for the design as daily-use scenarios.

2.2.4 Iterative Design

A design should be tested, evaluated and redesigned using the feedback from the evaluation. A design could be a verbal description, a paper prototype, a working prototype, or an implementation of the final design. Each prototype should be assessed for usability.

Prototyping

In order to obtain feedback as early as possible prototypes can be used to assess the usability of the design. As the design process proceeds, the complexity of the prototype increases [Andrews, 2006]:

Verbal description: A simple textual description of the interface, the dialogue elements and the choices.

Paper mock-ups: The interface elements are sketched on paper. Each screen can be drawn on a separate page, including menus, dialogue elements, and choices. Hand-drawn sketches can be replaced by more detailed print outs later. Hand-drawn prototypes can be used very early and are cheap. They provide maximum feedback for minimal effort.

Interactive sketches: Hand-drawn sketches are scanned and linked together using prototyping software products. The casual look of a hand-drawn interface encourages criticism and discussion.

Working prototypes: Working prototypes vary in complexity. Vertical prototypes contain all interface features, but no in-depth functionality. Horizontal prototypes contain in-depth functionality for a small, selected set of features. Scenario based prototypes provide the full functionality and user interface for a specified task scenario. Various products are available to build working prototypes, such as Macromedia Director [Macromedia, 2004], or Denim [Landay, 2005]. Using of existing components saves time and money. Wizard-of-Oz prototypes are driven by persons behind the scenes. The test user interacts with the system and an operator simulates interface responses.

Usability Evaluation

Each iteration of a design is assessed for its usability. The feedback of the evaluation is used to refine the design. A redesign may introduce new usability defects. The two major classes of usability evaluation methods, inspection methods and testing methods, are summarised in Section 2.3.

2.2.5 Follow-Up Studies

Important data for future product versions can be obtained by studies performed after the product release. Existing and competing products are often used as prototypes for a new product. The released product can also be seen as a prototype for future releases or new products. Feedback can be gathered by conducting marketing studies, or specific field studies such as questionnaires, interviews, and observation. Including logging facilities in the product may also provide important feedback. Logging facilities must only be used with permission of the user. Analysing user complaints and questions from the user hotline can also give important feedback for future products.

2.3 Usability Evaluation Methods

Within each iterative design cycle, the design of the user interface is assessed for its usability. There are two major classes of methods: empirical usability tests with real test users, and usability inspections conducted by usability specialists.

2.3.1 Usability Testing

Usability testing involves carrying out experiments with real test users to obtain specific information about an interface. Usability tests come from experimental psychology, which were primarily used to obtain statistical measures about a test object. Usability testing is usually more about interpretation of results than generating statistical data, although performance measurements are also made using testing methods [Hom, 1998]. Usability tests are usually performed in a test lab or using mobile equipment. The users are observed while using the system and the collected data is later analysed. Usability testing should not be used to measure the efficiency or productivity of the participating users.

According to Dumas and Redish [1993], usability testing has the following characteristics:

- The primary goal of a test is to improve the usability of the test object. Specific goals and concerns have to be defined when the test is planned.
- The test is conducted with real users.
- The users perform real tasks.
- The team observes and records what the users do and say.
- The team analyses the data, discovers problems, and recommends possible solutions.

Rubin and Hudson [1994] provide practical step-by-step guidelines on planning, designing, and conducting effective user tests. Important usability testing methods are summarised in the following sections.

Thinking Aloud

A thinking aloud test is performed using test users who are asked to use a system while continuously verbalising their thoughts. A thinking aloud test helps to gain insight into the users' view of a system and identifies many usability problems. The method yields much qualitative data while using only a small number of test users.

The method was originally used as a psychological research method. There are two main disadvantages. First, there is no sense in collecting performance data during a thinking aloud test, because the act of thinking aloud slows down the user. Second, thinking aloud is a somewhat unnatural behaviour for users. The facilitator will often need to encourage the user to keep thinking out loud [Nielsen, 1993].

Co-Discovery

During a co-discovery test a pair of users is asked to perform tasks while being observed. The co-discovery method originates from learning theory, where learners work together on a computer. In co-discovery for usability testing, users will talk naturally to one another about what has to be accomplished. If a problem occurs the users will discuss and try out possible solutions. In thinking aloud tests the user often stops talking when a problem occurs. The behaviour of test users in a co-discovery is more natural than in thinking-aloud tests. A disadvantage of co-discovery is that twice as many test users are required. Co-discovery has also a validity problem: it is uncommon in real life that two persons work together with the same user interface [Andrews, 2006; Usability Glossary, 2005a].

Formal Experiment

Formal experiments are controlled experiments with test users based on unambiguous specifications. Formal experiments are used for statistical analysis of objective measurements with fully implemented interfaces, or for objective comparisons of two or more alternative interfaces. Formal experiments produce fundamental statistical data. Many test users are required to ensure credible values of statistical significance.

It is important to ensure validity of the results. Problems with validity often arise by testing with the wrong kind of users, by testing the wrong tasks, or by not including time constraints or environmental influences.

The absolute performance of an interface can be tested by objectively running an experiment and determining if the interface meets specific requirements. A comparison of several interfaces can be conducted either with between-group testing, where test users only use one of the interfaces, or with within-group testing, where users use two or more of the interfaces [Field and Hole, 2002; Andrews, 2006].

Query Techniques

Asking users about a system after they have used it provides subjective data about the users' impressions. There are two possibilities: an interview, or a questionnaire. An interview is more flexible, since the facilitator can direct questions to interesting issues. Interview data is harder to compare than questionnaire data. A questionnaire is a structured form that is filled out by the user. It usually provides more quantitative data than an interview. Online or e-mail questionnaires can be used to reach large groups of users. Questionnaires are less flexible than interviews, but are easy to repeat and can be used to examine trends [Andrews, 2006; Hom, 1998].

Usage Study

A usage study examines the usage of a system in its actual work setting over a longer period of time. Other usability testing methods, such as thinking aloud, or formal experiments, bring people into a lab to examine specific usability questions. Usage studies may discover unexpected usability problems under unexpected circumstances. A usage study also provides a sense of how the work environment affects usability issues [Usability Glossary, 2005b]. For software, usage usually includes which applications are used, and how long, how often, and under what circumstances they are used [Modes, 2005].

Usage studies are based on automatically collected data. Event logging software can be used to record raw low-level data, for example, mouse movements, active applications, or key typing. The test equipment and software used for data collection should have minimal impact to the user's work and should be as unintrusive as possible. Another possibility to collect data is to record the user during work. The collected materials (video, audio, screen grab,...) are analysed afterwards, which is a highly time consuming task; one hour of video takes about 10 hours of manual coding. The advantage of this approach is that the user's actual context is available, for example incoming telephone calls, or interrupting colleagues. Manually evaluated materials contain more qualitative information [Modes, 2005].

2.3.2 Usability Inspection

Usability inspection methods utilise the knowledge and judgement of experienced usability specialists to inspect and analyse a user interface. Common usability inspection methods are discussed in Chapter 3.

2.4 Discount Usability

Nielsen [1994a] discusses a method for overcoming a phenomenon known as the intimidation barrier. Nielsen presents a number of studies that show that development teams rarely use recommended usability engineering methods. The most important reason will always be the total cost of using usability engineering methods in a development project. Unfortunately, this is partly due to several publications from the usability community. An often contemplated number has its origin in Mantei and Teorey [1988]. The case study presented in that publication estimated the total cost of USD 128,330 for introducing human factors elements in a software project. Nielsen and Bergman [1995] presents a research project about iterative design where 99 subjects has been employed. The total cost was estimated with USD 62,786. Such publications might suggest that usability engineering methods are highly elaborate and expensive procedures. Bias and Mayhew [2005] provide guidelines on how to justify the expense of usability procedures.

Usability studies usually use confidence levels of five percent. This means if a study shows that an interface A is significantly better than interface B, for example faster, at a confidence level of five percent, then of course interface A would be chosen, but there is still a 5 percent chance that interface B is actually faster. In practical applications the chance to choose the better of two interfaces is 50 percent if no usability measures are taken. Using much less effort for usability measures might yield a confidence level of, for example, 20 percent, while insufficient for a publication quality result it would enhance the chance of choosing the better of two interfaces from 50 percent to 80 percent in practical applications.

Another important reason is the perceived complexity of usability engineering methods. The discussion of usability inspection methods in Chapter 3 provides an overview. They vary much in their complexity and required effort. More complex methods like formal inspections might suggest that it is not possible to utilise usability engineering without having usability experts available in the project team.

Usability engineering has to be slowly propagated in a software development company. Nielsen uses the term "Guerrilla HCI" to describe the process of slowly sensitising development companies to usability engineering methods. Most projects would benefit significantly from using a little usability en-

gineering compared to using none. The discount usability approach discussed in Nielsen [1994a] is to apply some usability engineering to a project by not using perfect methods. The proposed components are much cheaper than the original methods, because they are simplified in many ways and require less expertise:

Scenario prototypes: Scenario prototypes are a special kind of prototype. Prototypes are used to reduce the complexity of an implementation. Horizontal prototypes reduce the level of functionality, vertical prototypes implement just one feature completely of a broad interface. Scenario prototypes reduce both the level of detail and the functionality. Since they are small, the effort to change them frequently is also small.

Simplified thinking aloud: Thinking aloud is usually conducted with a psychologist or usability expert as facilitator. The facilitator runs a test with a number of test users which is videotaped. Simplified thinking aloud is performed by a developer with just one or two test persons. Videotaping could also be substituted by a written log. This kind of simplified method reduces the cost for equipment and personnel.

Heuristic evaluation: Heuristic evaluation includes only a small set of heuristics rather than the large number of interface guidelines used in guideline reviews. These heuristics are very easy to teach in a short session. The heuristics can explain large categories of problems. As discussed in Section 4.3, even a team of unexperienced evaluators will perform fairly well in a heuristic evaluation. If the budget allows, it would be advisable to hire a usability expert to perform the prior training and to supervise the evaluation.

Chapter 3

Usability Inspection Methods

The term usability inspection was introduced by Nielsen [1993]. It is a generic name for a set of methods. These methods involve inspection of a user interface to assess its usability. The inspection is conducted by a number of usability experts and in some cases also by representative users.

Usability inspection differs from empirical usability testing in the following characteristics:

- Evaluators are usually not recruited from the user community, but are usability specialists.
- Evaluations takes usually less time than usability testing.
- The evaluation cost is usually lower.

This chapter presents nine usability inspection methods. They vary strongly in their formality and complexity. The least formal method is heuristic evaluation.

3.1 Heuristic Evaluation

Heuristic evaluation was first described in Nielsen and Molich [1990] and is conducted by a number of evaluators who inspect a user interface using a set of usability guidelines, called *heuristics*. The idea behind the method is that it is unlikely that a single evaluator will find more than a small fraction of the usability problems of a user interface. Chapter 4 describes heuristic evaluation in detail, this section provides an outline of the method.

3.1.1 The Heuristic Evaluation Procedure

The first part of the heuristic evaluation is the evaluation part. During an evaluation session lasting between one and two hours, each evaluator inspects the user interface individually. If the user interface is very complex, or has a large number of dialogues, the inspection should be split into more than one session. Additionally, evaluators can save time by having an assistant sit next to them to log their findings for them.

The inspection is usually performed in several passes. The evaluator should work through the interface at least twice. The first pass is to become acquainted with the interface and its possibilities. In the second pass, the evaluator scans through the interface and compares its elements with a list of usability heuristics. Various sets of heuristics are available. Some of them are described in more detail in Chapter 5. One set of heuristics was published in the paper where heuristic evaluation was first described [Nielsen and Molich, 1990]. The variety of heuristics is not limited in any way. In Nielsen and Mack [1994] various approaches are discussed for obtaining new heuristic sets for specific kinds of user interfaces.

The second part of a heuristic evaluation is where the individual usability problems discovered by the evaluators are discussed and compiled into one list of findings. This part can be conducted by an evaluation manager who combines the individual findings. This step can also be a group activity where all evaluators meet and discuss their findings. The result of the second step is a list of usability problems with reference to a corresponding usability principle.

The third part of the heuristic evaluation is to prioritise the discovered usability problems by severity. The list can then be used to identify and fix the most important problems.

3.1.2 Scope and Limitations of Heuristic Evaluation

Heuristic evaluation is a method which is very easy to use. It does not require a high degree of expertise, although the number of problems discovered increases with the experience of the evaluator [Nielsen, 1992]. The method can be learnt in half a day [Nielsen and Mack, 1994]. Heuristic evaluation does not consider the context in which the user is operating. Three new heuristics were added to Nielsen and Molich's original set of heuristics to address this problem [Muller and McClard, 1995]. The method primarily produces a list of usability problems rather than recommendations, although it is possible to generate recommendations from the list of findings.

3.1.3 Position of Heuristic Evaluation in the Development Cycle

Since the evaluators do not need to perform a real task it is possible to use an interface description, paper mock-up, or a simple prototype as the subject of evaluation. This enables the method to be used early in the development cycle.

3.2 Cognitive Walkthrough

A cognitive walkthrough is a usability inspection method which focuses on the ease of learning of a new interface. Most users prefer to learn how to use software by exploring it. Instead of formal training they prefer to explore new features while conducting their normal tasks. The cognitive walkthrough method was developed by Clayton Lewis, Peter Polson, Cathleen Wharton, and John Riemann [Lewis et al., 1992]. A cognitive walkthrough has a similar structure to other types of design walkthrough, such as code and requirement walkthroughs. A number of reviewers evaluate a proposed user interface in the context of specific user tasks.

The user interface does not have to be fully implemented, a paper mock-up or a working prototype can be used. It is even possible to use a detailed description of the design of the user interface. The target user population is also taken into account. The context in which the interface is used and task scenarios have to be defined before evaluating the interface.

3.2.1 The Cognitive Walkthrough Procedure

A cognitive walkthrough has two phases: the first to prepare the necessary materials and the second where the walkthrough is performed. In the preparation phase the reviewers have to prepare the inputs for the analysis: the tasks, the sequence of actions the user takes, the user population, and the user interface. During the analysis phase the reviewers work through each action of every task.

The walkthrough can be performed by a single person or by a team of experts. When evaluating as a team, the members may include other designers and software engineers, and representatives of other disciplines such as marketing and training. Each expert in the team should have a specific role during the evaluation: a scribe, a facilitator and roles according to the expertise of the members.

Defining the Inputs

Before the walkthrough four questions have to be answered:

- *Who are the users of the system?* A definition of who the users of the system will be. The outcome may be more revealing when specific background knowledge is added to the user descriptions.
- *What tasks will be analysed?* A detailed description of the tasks that will be analysed. The set of tasks should be representative of the core functionality of the system.
- *What is the correct sequence of actions to accomplish each task?* A description of the concrete steps the user is expected to take when first looking at the interface. The granularity of the appropriate steps depends on the prior knowledge the user is expected to have.
- *How is the interface defined?* This question is answered with a detailed description of the interface. What the user will see before taking the next action, a description how the user can take the action, and the feedback the system gives after the user has taken the action. If a prototype is available the description is replaced by the prototype.

Conducting the Walkthrough

All actions along the correct action path are examined in sequence. For each step of a task it is assumed that the user acts according to the problem-solving model described by the CE+ theory of Polson and Lewis [1990].

The CE+ problem-solving model makes the assumption that users behave roughly as follows when placed in an unfamiliar situation:

1. Start with a rough description of the task they want to accomplish.
2. Explore the interface and select actions they think will accomplish the task.
3. Observe the interface response to see if their actions had the desired effect.
4. Determine what action to take next.

As the walkthrough proceeds the analysts have to answer the following questions in order to create a success or failure story for each step in the correct action sequence:

- Will the user try to achieve the right action?
- Will the user notice what correct action is available?
- Will the user associate the correct action with the effect trying to be achieved?
- If the correct action is performed, will the user see that progress is being made toward the solution of the task?

3.2.2 Scope and Limitations of the Method

Cognitive walkthroughs analyse only the ease of learning attribute of usability. The use of cognitive walkthroughs to decide design trade-offs may lead to a tendency towards ease of learning. This may yield false results, for example when evaluating productivity features which may need some prior learning. Ease of learning is not the key attribute of such an interface.

The method of cognitive walkthrough finds differences between the users' and developers' views of a task, poor wording of button labels and menu titles, and inadequate feedback about the consequences of an action.

3.2.3 Position of Cognitive Walkthrough in the Development Cycle

Since cognitive walkthrough is an inspection method which evaluates an interface for learnability, it can be used very early in the development process. It can be used when the requirements are analysed and the definition of desired functionality is finished. A walkthrough can be performed by using a paper mock-up or a rudimentary prototype. A cognitive walkthrough can also be beneficial in all other stages of the development process, but the cost of making changes increases as a project progresses.

3.3 Heuristic Walkthrough

The heuristic walkthrough is a hybrid method of heuristic evaluation (Section 3.1) and cognitive walkthrough (Section 3.2) published by Sears [1997]. The method is designed to utilise the advantages of the incorporated methods and to suppress their disadvantages.

The process of a heuristic evaluation is unstructured and the only guidance is provided by the usability heuristics. The heuristic walkthrough method uses the free-form nature of heuristic evaluation and the idea to use a list of usability heuristics as guidance for the evaluation.

Cognitive walkthroughs provide a very detailed structure which may discourage evaluators from exploring the user interface freely. The heuristic walkthrough method uses the idea from cognitive walkthroughs to guide the evaluation process by asking questions along the way. The evaluator uses four questions which are similar to the ones used in the cognitive walkthrough method to allow the evaluator to focus on the thoughts of a future user exploring the interface. Additionally, the heuristic walkthrough method uses detailed task descriptions like in cognitive walkthroughs.

A heuristic walkthrough is conducted as a two-pass process. The first pass is similar to a cognitive walkthrough guided by a prioritised list of task descriptions and so called “thought-focusing” questions. The evaluator gains an insight into the user interface by considering the future user’s typical tasks. The second pass is similar to a heuristic evaluation. The evaluators are guided by their task-oriented knowledge of the system from the first pass.

3.3.1 Materials

The following materials are used during the heuristic walkthrough.

Task List

The task list is a prioritised collection of frequent and important tasks which are typically performed using the interface. This list may also include extra tasks in order to help expose evaluators to every part of the system. Evaluators are free to explore every aspect of the interface, but should consider the priorities when choosing appropriate portions of the interface for the evaluation.

“Thought-focusing” Questions

The first pass of a heuristic walkthrough is essentially a cognitive walkthrough. Four questions should help the evaluators in focusing on the users’ thoughts while they are exploring the system, with a focus on learnability like in the cognitive walkthrough method.

Will the user know what they need to do next? Does the system allow a situation where the user will simply not know what to do next.

Will the users notice that there is a control available that will allow them to accomplish the next part of their task? Is there a possibility that the next action is hidden, or does not match the user's terminology, or does not match what the user is looking for.

Once users find the control, will they know how to use it? For example, a user will not know how to use a drop down list that looks like a button. Users may be able to find an icon that corresponds to the desired action, but may not be able to use it because it requires a triple-click to activate it.

If users perform the correct action, will they see that progress is being made toward completing the task? The system has to provide appropriate feedback. Otherwise, users may not be sure that an action was performed correctly.

Usability Heuristics

The second pass of the heuristic walkthrough is essentially a heuristic evaluation. Every established list of usability heuristics can be used for the second pass.

3.3.2 The Heuristic Walkthrough Procedure

An heuristic walkthrough is conducted in two phases. First, the evaluator steps through a task list. The second phase is for free exploration. The evaluator uses the list of usability heuristics as guideline like in heuristic evaluations.

Pass 1: Task-Oriented Evaluation (Cognitive Walkthrough)

The list of tasks discussed in the previous section is used to guide the evaluator through this step. The priority of each task is usually related to the importance of the task and how often the users will perform that task. The evaluators are free to perform other tasks as well, as long as they want, and in any ordering. The priorities should just assist the evaluator in choosing appropriate sections of the interface to explore. The first pass ensures that the first experience of the evaluator with the interface is task oriented, just like how new users would learn to use the system. The evaluators are furthermore guided by the "thought-focusing" questions which have also been discussed in the previous section.

Pass 2: Free-Form Evaluation (Heuristic Evaluation)

During the second pass the evaluators freely explore any aspect of the system. Guidance is available through the knowledge of the interface the evaluator gained in the first pass and through a list of usability heuristics.

3.3.3 Conclusions

The method of heuristic walkthrough is designed to combine the advantages of heuristic evaluation and cognitive walkthrough. The heuristic walkthrough method contains of cognitive walkthrough as the first pass and a heuristic evaluation as the second pass. In Sears [1997] several assumptions are presented about heuristic walkthroughs which are supported by empirical results.

Heuristic walkthroughs and heuristic evaluations both discover a large number of usability problems. Heuristic evaluation produces many more false positives than heuristic walkthrough. Time is wasted in identifying false positives and removing them from the list of problems. Due to the task-oriented nature of heuristic walkthroughs the evaluators tend to focus much less on unimportant parts of the interface and thus produce fewer false positives.

Heuristic walkthroughs and cognitive walkthroughs both identify serious usability problems. Heuristic walkthroughs are well-suited to generate a comprehensive list of real usability problems without wasting time on unimportant issues.

3.4 Participatory Heuristic Evaluation

Participatory heuristic evaluation is a variant of the heuristic evaluation method described in Section 3.1. The Muller et al. [1998] paper contains a review of heuristic evaluation and presents reasons for modifying the technique.

3.4.1 Attributes of Heuristic Evaluation

The authors of Muller et al. [1998] found it useful to describe usability methods using the following attributes: an *object model*, a *process model*, a *participation model*, the *result* of the method, and the *position of a method within the product lifecycle*. According to Muller et al. [1998], heuristic evaluation has the following attributes:

Object model (Materials used): A group of evaluators is asked to look for usability defects using a generalised list of common usability problems, the *heuristics*.

Process model: Heuristic evaluation may be conducted as a free exploration of a design or as a series of task-guided scenarios. If heuristic evaluation is conducted as discount usability method the evaluators are usually recruited from the product team's workplace. The discovered usability problems do not have to be described in terms of heuristics, they are provided as aid for discovering problems.

Participation model: The inspectors of a heuristic evaluation may be experts in usability engineering, in some cases also in software engineering, or both.

Results: The result of a heuristic evaluation is a collection of summarised usability problems.

Position in the product lifecycle: Heuristic evaluation is often used for formative evaluation of design iterations, with the assumption that it is followed by more formal usability testing.

3.4.2 Extension of Heuristic Evaluation to Participatory Heuristic Evaluation

Muller et al. [1998] extended heuristic evaluation in two ways:

- by adding new task-based heuristics and by
- including users as evaluators.

the participation model and the object model.

Process Model

Usually, no real users participate in a heuristic evaluation. Under certain circumstances users may be near the project team. It may be valuable to include users in the evaluation team because of their knowledge of the work domain. The participation model of participatory heuristic evaluation is thus extended by work domain experts.

Participation Model

The second modification deals with the heuristics used in heuristic evaluation. Muller et al. [1998] differentiate between product-oriented and process-oriented paradigms in software engineering. The product-oriented paradigm regards to the design or computer system to be evaluated. The process-oriented paradigm looks at a system in the context of the user's work. The heuristics of Jakob Nielsen are primarily product oriented. Muller et al. [1998] introduce additional process-oriented heuristics.

The authors also argue that the wording in Nielsen's heuristics is too specific to usability professionals. A refined set of heuristics is presented with respect to principles of technical writing and user-oriented documentation to enable experts of the work domain to use the heuristics. The set of heuristics introduced in the paper is also categorised into five groups which represent a higher level of abstraction of the heuristics. Section 5.3 presents the heuristics used in participatory heuristic evaluation.

3.4.3 Conducting the Inspection

Participatory heuristic evaluation is conducted in the same way as heuristic evaluation, usually guided by a task list or scenarios. The participants are evaluating the interface individually. The participation of users as work domain experts may require additional prior training. If users are near the product team or easily recruited, participatory heuristic evaluation can also be used as a discount usability method.

3.5 Guideline Reviews

A guideline review is a usability inspection where a user interface is evaluated for conformance against a list of usability guidelines. Guidelines are available in a broad variety of abstraction levels, but collections of guidelines tend to be very large. Guideline reviews can be considered as a compound of heuristic evaluation (Section 3.1) and standards inspection [Nielsen and Mack, 1994]. Standards inspections are conducted using industry standards, for example platform-dependent conventions, or country-specific standards.

One of the first popular collections of guidelines was compiled for the U.S. Air force by Smith and Mosier [1986]. 944 guidelines are organised into six categories (see Table 3.1). Such a large collection is very hard to apply "rule-by-rule" in an inspection, unless the conducting inspector has a high degree of expertise [Nielsen and Mack, 1994]. These guidelines are quite old and primarily focused on text-interfaces and high-level issues of dialogues and functionality. The basic advice of these set of guidelines is still valuable, for example many web site forms would benefit from applying the guidelines [Cockton et al., 2003].

More recent guideline collections are available from the International Standardisation Organisation. ISO 9241-11 [1998] is a multi-part collection that covers ergonomic requirements for office work and visual display terminals. An example of a category covered in this standard is dialogue styles, which are separated into: menu (part 14), command (part 15), direct manipulation (part 16), and form-filling dialogues (part 17). Sun Microsystems, Microsoft, and Apple Computer have also published design guidelines for their software development products [Sun Microsystems Inc., 1999; Apple Computer Inc., 1992; Microsoft Corporation, 1999].

3.6 Pluralistic Walkthroughs

The pluralistic walkthrough [Bias, 1994] is a technique which utilises different kinds of expertise. The walkthrough is conducted by a group of representative users, product developers, and usability specialists. In contrast to other methods where specialists evaluate the whole interface on their own before

Section	Functional Area	Guidelines
1	Data entry	199
2	Data display	298
3	Sequence control	184
4	User guidance	110
5	Data transmission	83
6	Data protection	70

Table 3.1: The guideline collection created by Smith and Mosier [1986] for the U.S. Airforce contains 944 guidelines organised into six categories.

discussing the outcome, the pluralistic walkthrough is a method conducted as a group. Each single step of a certain task is discussed after finishing the step.

3.6.1 The Pluralistic Walkthrough Procedure

Materials

Each participant receives written instructions with the following ground rules at the beginning of the walkthrough. The participants also receive a hard-copy of task scenarios which contain a description of every task covered in the walkthrough including the data that will be manipulated. The particular steps of each task are printed on separate pages, called panels, to allow each participant to record responses and comments. Each panel contains a short description of the step and the data needed for this step, for example any assumed system parameters.

- The participants have to assume the role of the future user.
- The responses and additional comments have to be written on the hard-copies of the panels representing the steps of the tasks. The group finishes one step at a time. The discussion is conducted after each panel.
- The participants must not start the discussion until all have finished their responses and the administrator starts the discussion.
- The participants are not allowed to advance to the next step until the discussion of the current panel is finished.

Participants

The pluralistic walkthrough not only consults representative future users, but also solicits the participation of the product developers, designers, and coders. The product developers support the walkthrough by providing extensive knowledge of the product implementation and background. They can gain direct feedback from the representative users.

The broad variety of expertise available by the members of the evaluation team also enables some on-the-fly redesign of the evaluated interface; furthermore it facilitates collective or shared ownership of the product as proposed in extreme programming [Beck, 2000].

The human factors professionals bear an important role as walkthrough administrators. Besides their expertise in usability issues they can help to express the future users' suggestions. They also help to

defend the users' concerns against the developers. Programmers tend to play down usability issues spotted by users.

Conducting the Walkthrough

Participants are first instructed with the ground rules, and receive the package with task descriptions and scenarios. The product expert explains the key concepts and interface features of the product to be inspected. This introduction ensures that the participants have all prior knowledge that future users are assumed to have.

Then for every step of each task the same procedure is conducted. The administrator asks the participants to write down their responses for the specified task. After all participants are finished the administrator announces the correct answer. Next, the representative users present their responses and potential usability problems. The product experts are not allowed to intervene at this point of discussion. The human factors experts should try to facilitate the discussion amongst the representative users. This ensures that the information received from the users is not influenced by the product experts. The representative users will tend to make less critical comments due to the developer's presence. The product developers should thus facilitate an attitude of welcoming comments. The usability engineer should underline that point at the beginning of the walkthrough.

After the discussion of the representative users has finished, the product experts can join in. The product experts should take the users' comments very seriously, but not every user comment must lead to a redesign.

3.6.2 Scope and Limitations of the Method

The walkthrough's progress is as slow as the slowest person that participates. This means that it is nearly impossible to gain an impression of the flow of the interface.

Due to the fact that every step is evaluated separately it is hard to cover all possible actions that yield to the same goal. Usually just one path through the interface is evaluated. This prevents the participants from exploring the interface on their own, which could also give some important feedback.

Another problem related to the above is if a participant chooses a path that was not planned for the walkthrough. The administrator has then to stop the participant after writing down the response and bring the participant back to the correct path. The administrator announces the right action after all participants have finished the step.

The walkthrough inspects one step after another of a task, the method is thus most appropriate for full screen interfaces which have clearly divided distinct screens for each subtask.

3.6.3 Position of Pluralistic Walkthroughs in the Development Cycle

Usability walkthroughs are usually conducted very early in the development process. Manuals, help systems, and other documentation are likely not to be available in an early development stage of a system.

System designers and developers who participate in the walkthrough can act as "living manuals", allowing the representative users to ask questions for which they usually would consult the manual or the help system [Nielsen, 1993].

3.7 Perspective-Based Inspection

Perspective-based inspections were developed by Zhang et al. [1998] to address the problem that the percentage of usability problems found individually by each evaluator is rather low. They developed a

human-computer interaction model and used it to define usability perspectives. For each perspective they developed goals and generated a list of questions about whether the goals can be achieved.

3.7.1 Human-Computer Interaction Model

The human-computer interaction model is based on the “Seven Stages of Action” proposed by Norman [1988]. The model describes how a user interacts with a computer in two phases: in the *execution phase* users plan and perform actions, in the *evaluation phase* users perceive and interpret the outcome of their actions. Norman’s model originally had seven steps, but lacked a step of error handling, which was added to form the human-computer interaction model by Zhang et al. [1998].

The complete human-computer interaction model contains the following steps (see also Figure 3.1) grouped into two phases:

Execution Phase

1. **Form the goal:** Specify what should be achieved, which can be very imprecise.
2. **Form the intentions:** Develop intentions out of the goals. What has to be done to satisfy the goal?
3. **Identify the action:** How can this be transformed to the real world? What sequence of actions does the user have to perform in order achieve the goal?
4. **Execute the action:** Perform the planned action sequence in the real world.

Evaluation Phase

5. **Perceive the system response:** Perception of what actually happened.
6. **Interpret the results:** Understanding of what happened.
7. **Evaluate the outcome:** Comparison of the desired outcome and the actual response.
8. **Deal with errors that may have occurred:** If the evaluation yielded a great difference from the original goals, the cause has to be examined in order to form a new goal.

The two phases can be used to categorise two classes of usability problems: the difference between the user’s intentions and the actions available and, second, the difference between the systems response and the user’s expectation. These two classes are referred to as the *gulf of execution* and the *gulf of evaluation*.

3.7.2 Usability Perspectives

Using perspectives the evaluator can focus on a particular subset of usability problems during each inspection. In order to find as many usability problems as possible these perspectives should be mutually exclusive.

A user will usually be in one of the following situations:

Novice use: The users’ prior knowledge and experience do not support the users in achieving their goals.

Expert use: The user has an in-depth knowledge of the system, knows how to achieve certain goals and is also able to adjust the system to speed up the user’s work.

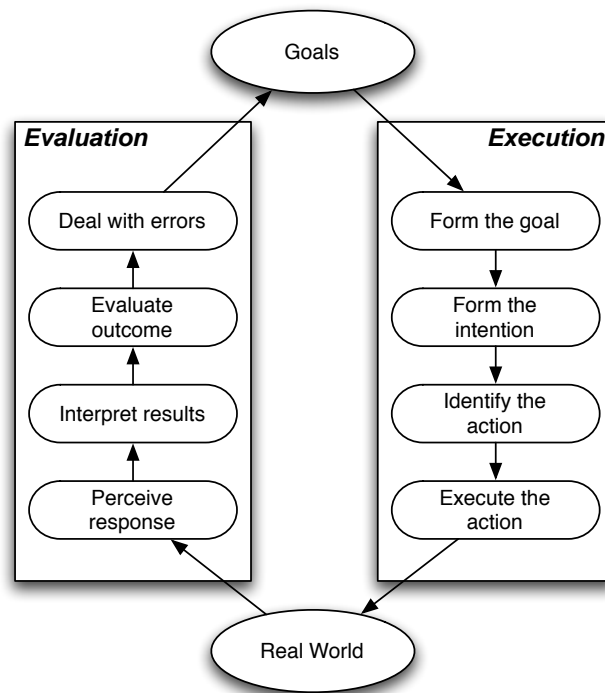


Figure 3.1: The Human-Computer Interaction model published in Zhang et al. [1998] based on the seven stages of action published in Norman [1988]. The *evaluation* and *execution* phases can be used to categorise two classes of usability problems. First, the difference between the user's intentions and the actions which are available and, second, the difference between the system's response and the user's expectation.

Error handling: The user encountered a difference between the desired outcome and the achieved effect of his action and needs to resolve the problem.

The publication [Zhang et al., 1998] also mentions the possibility to use other perspectives. Zhang et al. [1998] state that perspectives should be mutually exclusive which is ambiguous to the actual perspectives presented in the paper.

3.7.3 Conducting the Inspection

Each evaluator receives a hard copy with a description of the inspection procedure to help evaluators organise their inspection process and reduce the risk of overlooking usability problems. The inspection procedure also includes directions for each perspective:

Novice use: The inspectors are advised to think about users who are unfamiliar with the interface. The users will need some help from the interface in order to find the right action and make progress towards the goal.

Expert use: The inspectors are asked to consider a user who is familiar with the interface. The inspectors should evaluate the interface in terms of efficiency, flexibility, and consistency in supporting the user's tasks. The inspector has to become familiar to the interface prior to the inspection. The inspectors should also consider available short-cuts which help to increase efficiency when using the interface.

Error handling: The inspectors need to examine all possible user errors and system failures. For each error the inspector has to examine how the interface minimises the chance of an error, how informative the error feedback is, and how the user can recover from the error.

3.8 Consistency Inspections

Consistency inspection is a method discussed in Wixon et al. [1994]. It ensures consistency of the user interface across multiple components or applications of the same organisation.

3.8.1 Background

There is a trend towards highly integrated applications. For example in an office product suite every common feature should work in the same way independently of which part of the suite is currently in use. Usually a consistency inspection starts with usability experts. First, the various ways a particular user interaction is implemented throughout the system are evaluated. With these results a team of system designers tries to find the best way that should be used for all system components.

3.8.2 Conducting the Inspection

A user interface expert first reviews all elements of the application and produces a document which describes the current state of the interface. A team of representatives of each of the component's development teams is formed. Each member should be empowered to decide about changes to the team's component. The members receive the document produced by the UI expert prior to the meeting. During the meeting every element is reviewed step by step by the team of representatives. All discovered inconsistencies should be resolved in a way that satisfies all representatives.

Agreements and changes are noted in a document. Issues that cannot be resolved easily are noted for later discussion and may require a more focused meeting.

3.8.3 Position of Consistency Inspections in the Development Cycle

This method is ideally applied early in the development process to avoid extensive changes due to inconsistencies across the product. The best time is after all components' design documents are finished.

3.8.4 Position of Standards Inspections in the Development Cycle

The best time to conduct a standards inspection is when the design has been finished. A standards inspection can be used as a milestone allowing an initial design to be evaluated against some standards before further engineering work and manufacturability analysis are conducted.

3.9 Formal Usability Inspections

Formal usability inspection is a method described in Kahn and Prail [1994] and Hom [1998], based on the ideas of software or code inspection. Ideas from other usability inspection methods are also included: heuristics are used to support novice inspectors in finding usability defects and a cognitive model is used to help inspectors to understand the user's goals and purposes.

3.9.1 Defect Detection and Description

Kahn and Prail [1994] refer to usability problems as *usability defects* and defined them as a product characteristic which makes it difficult or unpleasant for a user to accomplish the tasks which are supported by the product. Since usability defects are not recognised during development, a process providing structure for the design review became necessary. The method provides facilities to formulate usability defects in a way that supports the discovery of best solutions in the following redesign phase.

A task performance model is provided to the inspectors to model the human task flow and guide where to look for defects. This model contains four phases (see Figure 3.2): perceiving, planning, selecting, and acting. The inspectors apply the model as they step through the tasks. Each phase of the model can be checked with a set of questions similar to the cognitive walkthrough method.

The task performance model explains where to look for problems. A set of usability heuristics helps the inspectors to identify what they have to look for. Any set of usability heuristics may be used, similar to heuristic evaluation. Chapter 5 discusses commonly used sets of heuristics.

The discovered defects have to be described from the user's perspective. This forces an inspector to fully understand the user's problem. This will later help to communicate with others about the problem and also helps finding solutions when the group discusses the problem. The inspector can use the task performance model and questions to formulate the discovered defects.

3.9.2 Conducting the Inspection

A formal usability inspection is conducted as a sequence of six phases:

1. Planning

The moderator and owners of the system component compile the information needed for the inspection. They also choose and recruit the appropriate usability engineers. The information they assemble consists of the following parts:

- Definition of the goals of the inspection.
- List of team members.

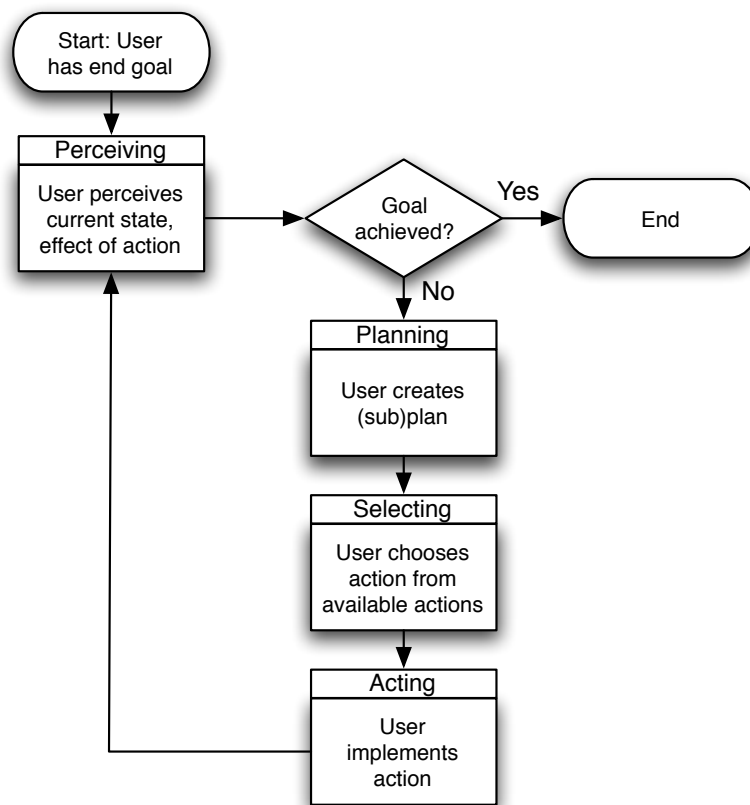


Figure 3.2: The human task performance model used in formal usability inspections [Kahn and Prail, 1994]. The model contains four phases: perceiving, planning, selecting, and acting. The inspectors apply the model as they step through the tasks similar to the cognitive walk-through method.

Responsibility	Description	Persons
Moderator	Manages the process, collects and distributes the materials that support the inspection, schedules and facilitates meetings. The moderator should be chosen from outside the project team.	1
Owner(s)	Representatives of the design teams whose components are inspected. They have to be able to understand the defects, contribute to solutions, solve the defects when updating the product.	1 per component
Inspectors	They find and report the defects, contribute to solutions. The moderator and owner can also act as inspectors.	3–5
Scribe	Records all defects discovered during the inspection publicly. Scribe responsibility can be rotated amongst the team members.	1

Table 3.2: A formal usability inspection team consists of a number of members who are assigned to specific responsibilities [Kahn and Prail, 1994].

- The inspection package given to all inspectors during the kickoff meeting.

The inspection package includes the following items:

- Inspection instructions: a description of the parts of the inspection packet and how they are used.
- Product description: the representation of the inspected interface, for example a prototype, paper mock-ups, or a description.
- Supporting documents: background information such as user demographics, standards, or related products.
- User profiles: one user profile for each target user group.
- Task scenarios: a description of each user task and to what user profiles the task applies.
- Task performance model: the task performance model shown in Figure 3.2.
- Heuristics: principles that help to identify usability defects.
- Defect logging form: the form used by the inspectors to record their defects.
- Logistics: schedule and locations of the meetings.

2. Kickoff

The kickoff meeting is the first meeting of the whole inspection team. The moderator and owner hand out the inspection package and explain its contents. The moderator discusses how the inspectors should look for defects using the user profiles, the task scenarios, the task performance model, and the usability heuristics.

3. Preparation

During the preparation phase the inspectors look for usability defects, working independently on their own. For each task, they take the role of the user described in each corresponding user profile. If a task step occurs where the inspector cannot proceed without confusion a usability defect is noted. The defect is logged and described from the user's perspective. This defect detection process is repeated for each user and task profile combination. The inspector may also note a possible solution, but this is not mandatory.

4. Logging Meeting

During the logging meeting the inspectors aggregate their lists of defects. The moderator steps through all steps of each user profile and task combination.

At each step the inspectors report the defects they discovered. During this process the correct sequence of steps for each task is disclosed. If an inspector took a path that deviates from the correct path this is also logged as defect.

The moderator has to manage the meeting in a user-oriented way. Suggested solutions could also be discussed, but this will take much more time. The logging should be conducted publicly so that everyone can see what has been logged. Less experienced inspectors can achieve great benefits from the logging meeting, because they can learn how other inspectors find defects and how to put themselves in the user's perspective.

5. Rework

During this phase solutions for the discovered defects are discussed and implemented. The earlier the inspection took place in the software development cycle, the easier this will be. If the situations and defects are more complicated the moderator and owner of the defects may form working groups or apply severity ratings to the defects. If it is necessary the moderator, inspectors, and owners can conduct another meeting to discuss complex defects and brainstorm for solutions.

6. Follow-Up

Finally, the moderator has to do some post-processing. The moderator collects all available data from the inspection process, compiles it, and sends it to all team members. This data can be used to inform the management and generate statistics. Interesting measures include, for example, the number of hours used per discovered defect, or the ratio of discovered defects to fixed defects.

3.9.3 Position of Formal Usability Inspections in the Development Cycle

The method is designed to reduce the time taken to discover usability defects. The method can be conducted using paper mock-ups of verbal descriptions, so the method can be applied very early in the development process.

Chapter 4

Heuristic Evaluation

Heuristic evaluation is a usability engineering method first published by Nielsen and Molich [1990]. The method is designed to find usability problems in a user interface so that they can be fed back into an iterative design life cycle. A heuristic evaluation is performed by a small number of evaluators who inspect the user interface for its compliance with a small set of usability principles, called the “heuristics”.

4.1 Conducting a Heuristic Evaluation

A heuristic evaluation usually consists of four phases which are discussed in the following sections.

4.1.1 Training

If the evaluators are new to heuristic evaluation they receive an introduction to the heuristic evaluation method. If the evaluators are usability specialists it may not be necessary to teach them the usability heuristics in much detail.

Depending on the kind of evaluation it may be necessary to provide some training about the domain knowledge needed to use the evaluated system. Such training should not try to make the evaluators experts in the domain, they should simply be given an idea about the purpose of the system. Another possibility to deal with missing domain knowledge is to provide an assistant who can give advice during the evaluation. The evaluators should have a fresh and unbiased perspective before starting with the evaluation. Any prior training should thus not expose any details about the user interface of the inspected system. If the evaluation only covers a small part of the system, a specific task scenario can be presented to the evaluators which can serve as the basis for their evaluation.

4.1.2 Evaluation

The evaluation is conducted by each evaluator individually and independently of the other evaluators. That means also that the evaluators should not discuss their results until all other evaluators have finished their evaluations. An evaluation typically lasts between one and two hours. If the interface has a very large number of dialogues or is very complicated, the evaluation should be split into multiple sessions of about the same time.

Multiple Evaluation Passes

The evaluator usually performs the evaluation in at least two passes. The first pass is to gain an overview of the interface to gain a feeling for the flow of interaction and the general scope of the system. If

scenarios are used, the first pass is used to step through them. The second pass is to perform a detailed analysis of the dialogues and their elements using the list of usability heuristics.

It is not mandatory to evaluate in two passes. The evaluators can decide on their own how to proceed with the evaluation. A first pass to gain an overview is very advisable. The evaluator can also proceed with a focus on a specific heuristic or interface element. The evaluators are asked to find as many usability problems as possible without considering their severity. The problem descriptions should be as precise as possible, should include a screenshot whenever possible, and should specify how to reproduce the problem.

Usability Heuristics

The problems identified usually refer to one or more usability heuristics. The evaluator is also allowed to consider any additional usability principle that may be relevant for the interface. The set of heuristics used for the evaluation is not fixed by the method. Different sets of heuristics may be used in different circumstances. Heuristic sets are presented in detail in Chapter 5.

Logging Discovered Problems

The description of the problems should be as specific as possible. Each problem should refer to any usability heuristics it violates. All problems should be logged separately even if they appear in the same interface element. It may not be possible to fix all of the discovered problems; the fixing of minor problems could be postponed to future releases. If all problems are logged separately the risk that a problem is lost is mitigated. The logged problems should also be documented with screenshots during the evaluation. A problem may not be reconstructible or the interface may not be available in the same version after the evaluation session. The screenshots may later be used to support discussion about the discovered usability problems. The evaluators should also log positive impressions, which should also include screenshots, whenever it is possible.

4.1.3 Merging

The problems discovered by each evaluator are merged into one long combined list of problems. This is a prerequisite to perform severity ranking. The combined list should also keep track of which evaluator found which problem in the list. This information can be used to determine the correlation between the number of evaluators and the number of problems discovered and can also be used to determine the ideal number of evaluators for a heuristic evaluation. This issue is discussed in more detail in Section 4.2.

Merging problem lists can be hard. The individual problem lists may overlap. A problem discovered by one evaluator may be described in three separate problems of another evaluator. The individual evaluator lists may be very long depending on the evaluator's experience and the evaluation subject. Cox [1998] discussed the problem of merging problem lists and developed a process for result synthesis, which is summarised in Section 4.5.

4.1.4 Severity Rating

The results of the heuristic evaluation can be used to determine the severity of the discovered problems. After the usability problems have been compiled into one list in the merging step, they can be presented to the evaluators. The evaluators are then asked to estimate the severity of each problem. It is difficult to obtain good severity ratings from the evaluators, if they are asked to rate during their individual evaluation session. One evaluator will find only a small subset of all usability problems of the interface (Section 4.2); the ratings would thus be incomplete. Rating the whole set of determined usability problems give evaluators a chance to rate problems they did not find in their own evaluation session. Severity ratings

Score	Frequency	Score	Severity
0	almost never (<1%)	0	No problem at all.
1	rarely (1-10%)	1	Cosmetic problem.
2	occasionally (11-50%)	2	Minor usability problem.
3	regularly (51-89%)	3	Major usability problem.
4	constantly (>90%)	4	Catastrophic usability problem.

Table 4.1: Two five point rating scales to prioritise discovered usability problems. *Frequency* is used to assess how often a problem occurs [Andrews, 2006, page 82]. *Severity* is used to assess the importance of the problem when it occurs [Nielsen, 1994b, page 49]. The two scores are added to form a combined result.

can be used to prioritise the resources to fix the usability problems. The project could be continued with minor usability problems in a tight project schedule. The severity of a problem has several components:

Frequency: The frequency with which the problem occurs.

Severity: The importance of the usability problem.

These factors are usually combined into a single severity rating. This can be calculated either by asking the evaluators to state only one rating for each problem, or by asking them for a rating for each factor and calculating a combined score. One final value of severity facilitates the prioritisation of the usability problems and decision-making. Table 4.1 illustrates a rating scheme containing two factors: *frequency* and *severity*. The combined scores of both factors can be used to prioritise the summarised list of findings.

4.1.5 Writing a Report

The Industry USability Reporting (IUSR) project created the Common Industry Format (CIF) [IUSR, 1999] for usability test reports. Andrews [2005] created a skeleton report which is similar to CIF and lists only the parts relevant for heuristic evaluation:

Title Page: The title page contains the name and version of the product which was evaluated, the name of the test leader, the date when the report was prepared, and the name of the person who prepared the report.

Executive Summary: This section provides a high-level overview for decision makers. The executive summary should state the identity and a description of the product, a summary of the used evaluation method, and a brief summary of results.

Evaluation Environment: This section contains relevant data of each evaluator and the hardware and software used by each evaluator for evaluation. This section is split into a “Participants” and “Test Facility” section in [IUSR, 1999].

Positive Impressions: This section appears only in Andrews [2005] and contains three to five things which the evaluators found positive, including a screenshots where reasonable.

Analysis of the Main Problems: This section appears in [Andrews, 2005] and contains a discussion of the five problems with the highest severity. This analysis section is split in IUSR [1999] into the “Data Analysis” and the “Presentation of Results” section.

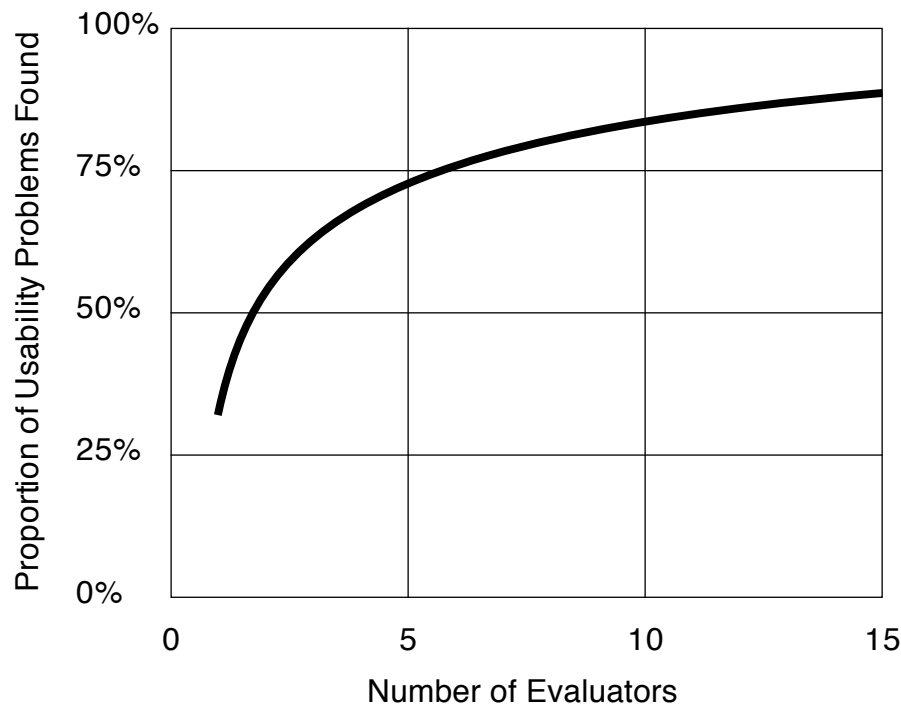


Figure 4.1: The number of usability problems found by heuristic evaluation with different numbers of evaluators. The data used for this diagram represents the average of six case studies of heuristic evaluation discussed in Nielsen [1994b].

List of Problems Found: This section in [Andrews, 2005] contains a comprehensive list of all discovered usability problems, including who of the evaluators discovered each, and the severity ranking of each evaluator. This section is included in the “Presentation of Results” section in IUSR [1999].

Appendix: The appendix in Andrews [2005] contains the individual logs from each evaluator. Including the raw data is not mandatory in IUSR [1999].

4.2 Number of Evaluators

A heuristic evaluation could be performed by only one evaluator. Several studies by Jakob Nielsen and Rolf Molich showed that a single evaluator finds only about 35 percent of known usability problems in a certain user interface [Nielsen and Molich, 1990]; [Molich and Nielsen, 1990]. To achieve better results, the aggregated evaluation results from several evaluators should be used. Figure 4.2 shows how the number of discovered usability problems increases with the number of evaluators conducting the evaluation. A good minimum number of evaluators would be three, because the number of discovered problems increases strongly until this point according to the Nielsen and Molich [1990] studies. More evaluators should be used in projects that are very critical.

To examine the ideal number of evaluators for a heuristic evaluation requires a cost-benefit analysis.

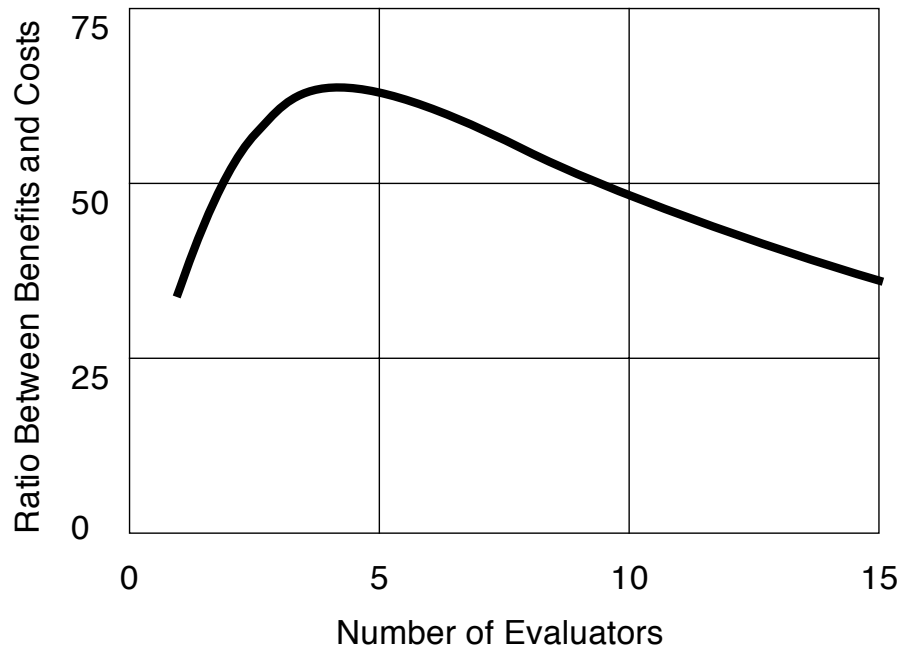


Figure 4.2: The curve shows how many times the benefits are greater than the costs of a heuristic evaluation with different numbers of evaluators. The data is based on Nielsen's cost-benefit model discussed in Nielsen [1994b].

Jakob Nielsen discussed a cost-benefit model in Nielsen and Mack [1994]. The model is based on the data of several heuristic evaluations. Assuming that model, the ratio between the benefits and cost for different numbers of evaluators can be visualised. Figure 4.2 shows that the ideal number of evaluators is four, when considering the data from Nielsen's cost-benefit model.

4.3 Experience of Evaluators

Two case studies which analysed the effect of evaluator experience in heuristic evaluations are summarised in Nielsen [1994b]. The first study [Nielsen and Molich, 1990] was conducted with 34 evaluators. They all had the same background and were asked to evaluate two different user interfaces. The number of usability problems found by individual evaluators in both user interfaces correlated with a value of 0.57. The ability of evaluators to find usability problems consistently is better than random, but there is also a substantial variability in performance between the evaluations. This variability could be mitigated by forming a group of evaluators which had good performances in past evaluations. This would anticipate that inexperienced evaluators could gain experience by conducting evaluations with an experienced team of evaluators.

The second study [Nielsen, 1992] was conducted by groups of evaluators with different degrees of expertise. The first group consisted of usability novices with general knowledge about computers. The

second group consisted of single experts who had usability expertise, but no knowledge about the domain of the interface. The third group consisted of double experts who had expertise with usability and knowledge about the domain of the inspected interface. Each evaluator in the group of usability novices found an average of 22 percent of the usability problems of the interface. The single experts found 41 percent which is 1.8 times better than the novices. Each the double experts found an average of 60 percent, which is 2.7 times better than the novices and 1.5 times better than the single experts.

Results of both studies showed systematical differences between the group performances. Although heuristic evaluation can be performed by evaluators with little expertise it is better to use experienced evaluators. Nielsen does not recommend employing users as evaluators, because of their lack of expertise in usability and interface design. Furthermore, he states that they would yield much better feedback in a user test.

4.4 The Kind of Problems Found by Heuristic Evaluation

Heuristic evaluation finds both major and minor usability problems. The probability of finding major problems is higher than finding minor problems. In the six case studies summarised in Nielsen [1994b] the probability of finding a given major usability problem by one single evaluator was 42 percent, and finding a minor problem was 32 percent. Although major problems are easier to discover Nielsen's case studies discovered many more minor problems (152) than major problems (59). Problems discovered by heuristic evaluation will tend to be minor ones. Severity ratings can be useful to balance this tendency. Although major problems are more important, the minor ones are also relevant. Minor usability problems are easier to discover with heuristic evaluation than with other methods.

In Nielsen [1994b] four different possibilities are described of where a usability problem can be located in an interface. First, the problem can have a single location. Second, the problem can also occur in multiple locations, and the locations have to be compared to find the problem. Third, the problem can be one of the overall interface structure, and, fourth, the problem can be a missing part of the interface.

Nielsen discussed a study in Nielsen [1994b] where 211 usability problems were analysed. The study found that the difference between the four locations was not statistically significant. Evaluators were approximately equally good at finding problems in all of the four locations. Problems of the category where something is missing in the interface were easier to find in running systems than on paper prototypes. Nielsen's explanation for that phenomenon is that the evaluator gets stuck when some functionality is missing in a running system. Using a paper prototype the evaluator can just proceed to the next page.

4.5 Result Generation

The data generated by heuristic evaluation is a comprehensive list of usability problems, which may have also been rated by severity. This kind of data may not be directly usable by software developers, because it does not state what the real problem cause is, or because no possible solutions are recommended. Cox [1998] introduces the term of result synthesis as:

”the process of transforming the raw inspection data from heuristic evaluation into a complete, concise, and coherent statement of the problems in the evaluated interface as well as recommended actions to address the problems identified.”

Cox [1998] states that not much research effort has been spent to result generation and developed a participatory model of result synthesis. Several groups of persons can be involved in the process, such as software developers, documentation engineers, or customer support engineers. A participatory approach produces the best trade-offs between competing demands and also educates participants, which are often

from different disciplines. The result of result synthesis is thus a well-supported set of recommendations and not a plain list of problems.

The result synthesis process contains four stages:

- *Preparation*: During this phase the participants ensure that the raw problem descriptions have been recorded in a suitable way. A raw problem consists of a problem description, an associated heuristic, and the name of the author of the problem.
- *Familiarising*: The participants are asked to review the whole set of problems, while talking to each other about the problems. Duplicate problems should be summarised. Each participant should gain an overview of the whole set of problems.
- *Emergence*: Ideas and interpretations are not obvious from raw data. The result of the emergence stage is to form a single, complete, concise, and coherent view of the interface problems amongst the participants.
- *Finalising*: The understanding of the problems which is partly available in the heads of the participants and partly represented by the workspace has to be recorded in a way that makes it usable for those who have not participated. The result of this stage is a report in a standardised format.

The result synthesis process can be paper-based, which is a flexible and familiar medium. The raw problem descriptions are noted on sticky notes which are organised on a suitable kind of workspace. Relations between problems are expressed by their spatial distance and lines connecting them. Working with sticky notes can be very tedious. The structures created during the emergence stage are also not suitable for the final report. Cox [1998] developed a prototype of an application which assists the participants during the result synthesis process. The prototype supports the arrangement of problems in a virtual workspace and the creation of relations between them. The structures created during the emergence stage can be directly used to create the final report. The prototype is implemented as groupware which allows the result synthesis process to be conducted with participants in different places.

Another approach to solve the communication problem between usability engineers and software developers was undertaken by Mahajan [2003]. Software developers are usually not part of a usability team and may thus not understand the list of problems generated during usability inspection or usability testing. Mahajan [2003] developed the Usability Problem Diagnosis tool based on the User Interaction Framework developed at Virginia Tech. The User Interaction Framework is a hierarchically structured knowledge base of usability concepts.

The Usability Problem Diagnosis tool allows usability engineers to store usability problems in a common database. The stored usability problems are classified using the taxonomy of the underlying User Interaction Framework. The classification assists usability engineers and software developers to find the root cause of a problem. The Usability Problem Diagnosis tool does not use heuristics for classification because they are too few in number, the User Interaction Framework provides hundreds of categories for classification. The stored and classified usability problem descriptions can later be retrieved by software developers in order to fix them. The Usability Problem Diagnosis tool is implemented as an online application, which allows collaboration of participants in different places.

Chapter 5

Sets of Heuristics

This chapter describes commonly used sets of heuristics. Heuristics are closely related to guidelines and distinguishing between them is not always easy. In general, heuristics are more general usability principles and are usually fewer in number. Each of the following sections describes a set of heuristics and, where documented, how they were developed.

5.1 Molich and Nielsen 1991 (10 Heuristics)

The original list of usability heuristics was developed by Rolf Molich and Jakob Nielsen [Molich and Nielsen, 1990]. The list represents general interface guidelines which every user interface designer should keep in mind. The principles are very broad and can be applied to nearly every type of user interface. The last one, Help and Documentation, was added in 1991. Each guideline is described in detail in Nielsen [1993].

Simple and Natural Dialogue: The interface should present exactly the information the user needs. The user has less to learn, less chance of getting something wrong, and less information which could distract from performing tasks. The information should appear in a natural order, related information should be graphically clustered. The information that the user needs should be provided in an order which the user expects. Irrelevant and rarely needed information should be hidden or removed. Navigation and window management should be as simple as possible.

Speak the User's Language: The terms used in a user-centred interface design should be based on the user's language. Developers tend to use system-oriented terminology. The user's language is not limited to words, it also includes the symbolic language used in icons. Speaking the user's language also means that the interface uses the perspective of the user, not the system. A user-oriented dialogue provides a good mapping between the computer model of information and the user's conceptual model.

Minimise the User's Memory Load: Users are not good at remembering things, the computer should take over this function. The human brain is better at recognising information than remembering it. A user-centred interface should thus present the user with a choice of pre-generated items. Technologies which can achieve this include menus, icons, and choice boxes. The user should also receive some hints if a special input format has to be used. Recognition-based systems will rely on the visibility of objects which are important for the user. This argument competes with the first heuristic in this list. The interface has to strike a balance between these factors.

Consistency: Consistency is a very important usability principle. The same actions must have the same effect throughout the whole system. Consistency applies to the names of interface elements,

commands, and actions. A consistent user interface will encourage the user to try out exploratory learning strategies. The user is able to transport concepts from one part of the system to explore new parts. Consistency can be achieved more easily by taking interface design standards into account. Consistency is not only a matter of screen design, but also the structure of steps in a task should be consistent across different tasks.

Feedback: The system should provide continuous feedback about what it is doing and how it is interpreting the user's input. The user should always know what is currently happening. Feedback should appear when information becomes available. The feedback should reflect the user's input to indicate what is happening with the user's data. Different types of feedback need different degrees of persistence. Feedback may only be important during the occurrence of a certain phenomenon. Other types of feedback may need the explicit acknowledgement of the user; such messages will have a higher degree of persistence. Feedback becomes more important when response times increase.

Clearly Marked Exits: The user should have the feeling of control over the system. Cancel buttons and escape facilities enable the user to return to the previous state. A more general approach is to provide an undo function. The undo function should be generic enough that it is available in every system state. A general undo function will encourage exploratory learning, because the user can recover from unwanted system states.

Shortcuts: An experienced user should be able to accelerate work flow by using shortcuts for frequent tasks. Common strategies for acceleration are abbreviations, function keys, command completion, menu shortcuts, or double clicking instead of menu selection. Pen interfaces or virtual reality interfaces may also use gestures as shortcuts.

Precise and Constructive Error Messages: Errors are critical for two reasons: the user is in a situation where it may be not possible to perform a task and, second, the system has the chance to provide feedback that will help the user to learn how to avoid that error. The system may have knowledge about what happened and can help the user. Good error messages are easy to understand, give a precise representation for what happened, provide constructive help for the user, and are polite.

Prevent Errors: Better than providing good error messages is preventing errors in the first place. Many situations are known to produce errors. A system should be designed to avoid users getting into erroneous situations. Error-prone situations occur for example in applications with different modes: with modes it is very likely that the user will attempt to use commands that do not apply to the mode the application currently is in.

Help and Documentation: Although it is better if a system does not need documentation because of its intuitive design, this may not always be possible. Users usually do not like to read documentation. Various types of help systems are available: tutorials and getting started guides, reference manuals, reminders, context sensitive help, wizards, tips and tool tips, etc.

5.2 Nielsen 1994 (10 Heuristics)

The list of usability heuristics discussed in Section 5.1 was revised after a factor analysis of 249 usability problems. The revised list was published in Nielsen [1994b]. The following list quotes the exact original formulation of the "10 Nielsen Heuristics":

Visibility of System Status: The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

Match Between System and Real World: The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

User Control and Freedom: Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

Consistency and Standards: Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

Error Prevention: Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

Recognition Rather than Recall: Minimise the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

Flexibility and Efficiency of Use: Accelerators – unseen by the novice user – may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

Aesthetic and Minimalistic Design: Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

Help Users Recognise, Diagnose, and Recover from Errors: Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

Help and Documentation: Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

5.3 Muller and McClard 1998 (15 Heuristics)

Muller and McClard [1995] argue that the heuristics from Molich and Nielsen do not consider the context in which a system is used. They initially introduced three new heuristics which take into account how the system meets the needs of the users and their environment.

The method of participatory heuristic evaluation [Muller et al., 1998], summarised in Section 3.4, uses the three new heuristics of Muller and McClard [1995] and introduces two further heuristics. The authors of the Muller et al. [1998] paper also criticise the language used in the 10 Nielsen heuristics for being too specific to software developers and usability specialists. Participatory heuristic evaluation includes users as evaluators, the paper presents a set of heuristics which users having no expertise in software development and usability engineering could use.

System Status: The system should keep the user informed with appropriate feedback within reasonable time.

Task Sequencing: The user should be able to select and sequence tasks. The system should not control the users' actions.

Emergency Exits: Users should be able to find a way out of functions used by mistake without having to go through complicated dialogues. Users should be able to make their own decisions.

Flexibility and Efficiency of Use: The system should support accelerators, which are unseen by novice users. Expert users should also be able to tailor frequent actions. The system should also be accessible for users who differ from the “average” user, for example in physical and cognitive ability, culture, or language.

Match Between System and Real World: The system should speak the users’ language with wording and concepts that are familiar to the user. Information should appear in a natural and logical order.

Consistency and Standards: Every interface element, such as words, phrases, or images, should have a single meaning throughout the whole interface. Conventions of the target system should be followed.

Recognition Rather than Recall: The user should not be forced to remember information from one part of the interface to another. Help should be easily retrievable whenever needed.

Aesthetic and Minimalist Design: The interface should not contain irrelevant or rarely needed information because it reduces the relative visibility of relevant parts.

Help and Documentation: The system should be intuitive to use, performing common tasks should not require reading documentation. When documentation is needed it should be easy to search, support the user’s task, and list concrete steps to be carried out.

Help Users Recognise, Diagnose, and Recover from Error: Error messages should be expressed in the user’s language and should precisely indicate the problem and suggest a constructive solution. The user should not be blamed for the error.

Error Prevention: The system design should prevent errors. Errors should be anticipated, the system should treat users’ errors as valid input or an ambiguous input to be clarified.

Skills: The system should support, extend, supplement, or enhance the users’ skills. The system should not try to replace the user.

Pleasurable and Respectful Interaction with the User: The system enhances the users’ quality of experience with the system. The user should be treated with respect.

Quality Work: The system should support the user in producing quality work. Attributes of quality are: timeliness, accuracy, aesthetic appeal, and appropriate levels of completeness.

Privacy: The system protects the users and the users’ clients personal or private information.

5.4 Online Computer Library Center (14 Heuristics)

The Online Computer Library Center [OCLC, 2005] conducts usability evaluations. A set of 14 heuristics is used for their evaluations. The first ten heuristics are from Jakob Nielsen’s list. The list contains four additional heuristics, but the OCLC web site gives no further details about how they were created:

Affordances: Are the interface elements intuitive enough, that the user understands their meaning before using them?

Use Chunking: Each piece of content should exactly cover one topic. The user should not be forced to view several documents to complete one single thought.

Provide Progressive Levels of Detail: Information should be structured hierarchically. The level of detail should increase the deeper the user browses the hierarchy.

Do Not Lie to the User: Misleading or erroneous links should be avoided. Missing information should not be referred to.

5.5 Instone: Web Heuristics 1997 (10 Heuristics)

In 1997 Keith Instone wrote an article about how Jakob Nielsen's heuristics apply to web sites [Instone, 1997]. Instone took Nielsen's revised list of heuristics presented in Section 5.2 and discussed how each of them apply in the world wide web:

Visibility of System Status: For web users two important questions are where they are and where they can go next. Each page should be branded so that the user can see which section the page belongs to. Links to other pages should be clearly visible. These characteristics are important so that users can navigate freely from one section to another.

Match Between System and Real World: On the web users come from different backgrounds, making it is hard to find the right level of language.

User Control and Freedom: Most of the emergency exits are provided by the browser, but there are still many possibilities to support the user's freedom. On the other hand there are many ways to take away user control by forcing the user to use certain fonts, colours, styles, and screen widths. New technologies should be treated with care.

Consistency and Standards: Wording, content, and buttons should be consistent on a web site. Page titles and page headers should be consistent to links pointing to the page. It is also necessary to conform to web standards and conventions.

Error Prevention: The limitations of web forms make it hard to check user input before they are submitted. Javascript should be used wisely, because browsers support deactivation of Javascript.

Recognition Rather than Recall: This heuristic is closely related to the system status in the web context. The user should be able to recognise a location on a web site without recalling the sequence of actions that ended in that location. Good labels, descriptive links, and examples for form fields support recognition.

Flexibility and Efficiency of Use: Bookmarks are very good accelerators. Make the web site easy to bookmark, by providing permanent URLs, avoiding frames, and using descriptive page titles.

Aesthetic and Minimalistic Design: Unnecessary or seldom used information distracts from the important content and slows down the web site. Rarely needed information can be sourced out and made accessible with a link. A good strategy for clearly structured information is to use progressive levels of detail. More general information should be provided at higher levels of the hierarchy. The user can navigate deeper into the hierarchy if more detail is required.

Help Users Recognise, Diagnose, and Recover from Errors: Error messages should provide a solution or a link to a solution.

Help and Documentation: Integrate help and documentation into the individual web pages.

5.6 Skinner and McMullin: Rich Internet Application Heuristics 2003 (10 Heuristics)

Based on the idea of Keith Instone, Grant Skinner and Jess McMullin created a commented list of usability heuristics for Rich Internet Applications (RIAs) based on Nielsen's 10 heuristics in 2003. The list focuses on the use of Macromedia Flash in web sites [Skinner and McMullin, 2003].

Visibility of System Status: The rich display capabilities of RIAs should be used to provide constructive feedback whenever the user has to wait. The user can be guided through sequential tasks using task steps. This helps the user to stay informed about what progress has been made towards finishing a task.

Match Between System and the Real World: RIAs enable the developer to introduce new metaphors. The developer should ensure that the metaphor acts consistently with its the real-world counterpart. The application should speak the users' language and meet their expectations. RIAs have introduced many new features, which the users will very likely not care about.

User Control and Freedom: Users are usually familiar with browser controls. A RIA should not break these controls, for example by deactivating the back and forward button of the browser. An RIA should also be aware of the browser's history and bookmarking features.

Consistency and Standards: The best way to provide a consistent user interface is by following interface standards. Macromedia maintains a usability topic web site for developers of Flash applications. Other interface guidelines [Sun Microsystems Inc., 1999], [Apple Computer Inc., 1992], [Microsoft Corporation, 1999] may also provide help in designing user-friendly RIAs.

Error Prevention: Forms should indicate required fields and formats with examples. The application should be able to accept various input formats, for example for dates, rather than forcing the user to a comply with a given format. The amount of data to be entered can be limited with auto-filling fields. Functions which may end up in dangerous results, like "delete all records", should be avoided.

Recognition Rather than Recall: The rich possibilities of Flash are often used to hide important interface elements using a fancy application menu. The user should not be forced to search for key elements of the user interface.

Flexibility and Efficiency of Use: The advanced functionality of RIAs can be used to provide powerful accelerators like, for example, keyboard shortcuts, or type-ahead auto-completion. The developer should not forget less powerful accelerators like bookmarks.

Aesthetic and Minimalistic Design: RIA design is often a trade-off between possibilities and user distraction. Providing a simple and elegant interface is often better than overwhelming the user with features and fancy design elements.

Help Users Recognise, Diagnose, and Recover from Errors: Error messages should be simple and should hide technical background from the user. Although RIAs have many possibilities to provide complex animations about the cause and the solution of an error, the response of the application should focus on a possible solution.

Help and Documentation: RIAs should contain a simple and concise help system. RIAs provide powerful mechanisms, like animation and sound, to create narrative and animated help for the user.

5.7 Shneiderman: Eight Golden Rules of Interface Design 1997 (8 Heuristics)

Shneiderman and Plaisant [2004] proposed this collection of rules as a guide for good interaction design. The principles are derived heuristically from experience. If properly extended, refined, and interpreted they should be applicable to the most interactive systems.

Strive for Consistency: The sequence of actions to take for a task should be consistent across the whole user interface. The terminology should be the same in prompts, menus, and help screens. Interface elements such as layout, colour, fonts, and capitalisation should be consistent throughout the whole user interface.

Cater to Universal Usability: The needs of different users should be taken into account, for example novice-expert differences, age ranges, and users with disabilities. The system should be designed for plasticity. Users who regularly use the user interface want to reduce the number of interactions. Features for novice users, such as explanations or assistants, should be added to the interface.

Offer Informative Feedback: Every action taken by a user should be followed by proper feedback of the application. Frequent and minor actions should provide moderate feedback, where infrequent and major actions should provide comprehensive feedback.

Design Dialogue to Yield Closure: Sequences of actions should be grouped. The feedback of completing a group of actions gives users the satisfaction of accomplishment and a feeling that progress has been made towards their goals.

Prevent Errors: The application should be implemented in a way that prevents the users from making errors. If the user has ended up in an erroneous situation, the system should offer simple and comprehensible methods to resolve the situation.

Permit Easy Reversal of Actions: Providing facilities which enable the user to undo steps will mitigate the users' concerns to make errors. This will encourage the user to explore unfamiliar options.

Support Internal Locus of Control: Experienced users will like systems which give them a feeling of control. The system should be designed to make users the initiators of actions and not responders to a system.

Reduce Short-term Memory Load: The human short term memory is limited. The user should not be forced to remember something. Screens should be simple, multiple page displays should be compact, and options should be clearly visible.

5.8 W3C: Accessibility Heuristics 2001 (6 Heuristics)

The Web Accessibility Initiative (WAI) developed several guidelines for accessibility. These guidelines have been extensively reviewed by different working groups of the World Wide Web Consortium (W3C). The following set of heuristics was developed to provide simple support for developers who have no deep knowledge of accessibility issues. Koivunen and McCathieNeville [2001] created this set of heuristics by generalising three different W3C accessibility recommendations: Web Content Accessibility Guidelines (WCAG), User Agent Accessibility Guidelines (UAAG), and Authoring Tools Accessibility Guidelines (ATAG).

Provide Alternative Equivalents: Text is easy to create and transform to different media. Provide textual equivalents of graphics, images, and video. This enables users to follow the material even if they have certain limitations, or a device that cannot handle certain types of media.

Provide Means to Select Equivalent Content: Users should be able to access equivalent content in a flexible way. Usually, web browsers provide the means to switch to alternative content.

Provide User Control for Presentation: Presentation and content should be separated. Style sheets can be used to render elements appropriate to the needs of the user.

Provide Device-Independent Interaction: Users with different input and output devices should be able to access all available functionality, for example through keyboard shortcuts as well as mouse clicks.

Provide Semantics for Structure: The user should be able to use alternative ways for navigation and orientation.

Provide Reusable Components: Sometimes a single graphic or video element is repeated several times on a web page. The same element should be reused rather than simply copied, so that it can be recognised by non-visual users.

5.9 Purho: Documentation Heuristics 2000 (10 Heuristics)

Vesa Purho conducted a study [Purho, 2000] on documentation usability at Nokia. Usability heuristics especially for documentation systems were not available at the time the study was conducted. One of the results of this study was a list of ten heuristics based on Nielsen's list but adapted to documentation systems:

Match Between Documentation and the Real World: The documentation should speak the users' language. Words, phrases, and concepts familiar to the user should be preferred to system-oriented terms. Information should appear in a natural and logical order.

Match Between Documentation and the Product: The application interface should match the manuals, online help system, and references. The same terminology should be used in all of them. This may disagree with "Match between Documentation and Real World" if the application interface uses bad terminology.

Purposeful Documentation: If the documentation consists of several materials, the purpose and intended use of each should be clear. The media of the documentation must meet the requirements of the user. For example, a construction worker would prefer a reference card rather than a CD-ROM.

Support for Different Users: The documentation should be convenient for users with different levels of knowledge of the domain. Unnecessary information should be hidden from novice users. This information can be put into a reference manual where it is available for expert users. Quick reference cards can be used to provide help for the most important tasks.

Effective Information Design: Information must be stored in an easily retrievable way. Graphics, tables, and lists are easy to scan and read. Unnecessary graphics slows down the user. The format of the documentation should be suitable for the media which transports it. The instructions should address the user directly by using active sentences.

Support for Various Methods for Searching Information: Documentation should support different ways of finding information. An index should contain the user's own language as well as system terminology, terms from international standards, and those used by competitors. The layout of documentation should support the user in scanning over pages; beginning of chapters and other important parts should be easily recognisable.

Task Orientation: Documentation should be structured according to the user's tasks and not around the tools the user will use. The job tasks remain the same although the tools may change. This strategy reduces the need to restructure documentation when a product is changed. The tasks should have the same level of granularity throughout the whole documentation.

Troubleshooting: The documentation should contain a troubleshooting section giving users guidance for common and known problem situations and how to analyse erroneous situations. Documentation related to errors must be quickly accessible to save time in case of emergency.

Consistency and Standards: If the documentation has several parts, they should be consistent in their structure and wording. The information in different documents should not overlap. Platform conventions should be followed when creating a help system.

Help on Using Documentation: Large documentation systems should provide instructions on how to use the documentation, and how the documentation is updated.

Chapter 6

Computer-Supported Heuristic Evaluation

In Nielsen [1993] the concept of Computer Aided Usability Engineering (CAUSE) is discussed. CAUSE tools could rapidly speed up the usability engineering process. The following list of categories was published in Nielsen [1993] and shows that the possibilities are manifold. There is a vast amount of CAUSE tools freely or commercially available. An overview is given in Masse et al. [1998].

Some examples of CAUSE tools are:

- Tools that support the quick construction of working prototypes or paper mock-ups for user interface testing.
- Tools for easier creation and use of specifications, models, and task analysis techniques to lower the barriers of their use.
- Support tools for user testing, especially logging tools, or eye-tracking facilities.
- Localisation and translation support tools for international user interfaces.
- Automated event and key loggers to support action analysis or allow to collect performance data.
- User feedback databases and data mining facilities in order to provide usable feedback for future developments.
- Tools for support and automation of usability evaluation techniques.

Tools for computer supported heuristic evaluation are very likely to exist in companies which provide heuristic evaluation as a service. Software development companies which use heuristic evaluation may also have their own in-house implementations.

Unfortunately, there are few software tools for heuristic evaluation available to the public. An open source project named uzReview by the uzilla group, was included into a commercial framework and was thus discontinued. The name of the commercial framework is *Uzilla.net* and is based on a customised web browser that logs user activity and an Internet-based data collection and aggregation server.

Uzilla.net describes the following features of their commercial application:

- Rapid setup with re-usability of previous test components for iterative testing
- Detailed user activity down to mouse motion paths and word level typing times.
- On-screen task instruction and surveys before, after, and during tasks.

- Customisable surveys for pre, during, and post-session feedback.

The original open source application from the uzilla group is described in Section 6.1.

Another tool can be tested using a trial version. It is called *Review Assistant* and is a commercial product provided by *Information&Design*¹. The trial version of Review Assistant is described in Section 6.2.

6.1 uzReview: A Computer Supported Heuristic Evaluation Tool

uzReview [2003] is built upon the eXtensible user interface definition language (XUL) used by the Mozilla² project. The first and also the last major release of uzReview dates back to 2003 and no longer works in versions of Mozilla above 1.4 or Mozilla Firefox. In order to use uzReview the Mozilla browser has to be installed. The Mozilla browser is available for several platforms, like Microsoft Windows, Linux, several Unixes, and Mac OS X. uzReview appears as a sidebar in Mozilla and is used to evaluate web based interfaces.

6.1.1 Conducting the Review

uzReview has two modes for evaluating a web site, *URL* mode and the *keyword* mode.

The URL mode can be used if the evaluation should be conducted on a per page basis. The evaluator can choose a number of corresponding heuristics for one page and enter comments for each heuristic. After the evaluator navigates to another web page the heuristics and comments are logged. If no comments were made and no heuristics were selected, nothing is logged.

The keyword mode can be used to evaluate work flows. A set of heuristics and comments can be assigned to a set of pages. The difference to the URL mode is that every page visit is logged, even if the evaluator did not enter a comment.

6.1.2 Rating

Ratings has to be done on a per heuristic basis during the individual evaluation session. uzReview does not support grouping of individual evaluator findings and thus lacks the ability to prioritise the summarised list of usability problems.

6.1.3 Reports

The results of the evaluation are stored in an XML (eXtensible Markup Language) file in the projects folder of the uzReview directory. A XHTML (eXtensible HyperText Markup Language) report can be generated where the collected issues are grouped either by location and keyword or by heuristic.

In location and keyword grouping, the view is ordered by the addresses that were visited during the evaluation. Only those locations are listed where the evaluator entered a severity ranking or entered a comment for one or more heuristics.

In heuristic grouping, the view shows all items where a comment has been entered for one or more heuristics or a severity ranking has been made. This view provides an overview of heuristics that were commonly violated.

¹<http://www.infodesign.com.au>

²<http://www.mozilla.org>

6.1.4 Advanced Configuration

uzReview allows the provision of custom heuristics using a configuration file in XML format. uzReview can also be used to evaluate different sites at a time using multiple browser windows. uzReview allows evaluation of multiple sites using the *tabbed browsing* feature of the Mozilla browser. The tab which is currently focused is used to log issues.

6.2 Review Assistant

Review assistant is a commercial Microsoft Windows application published in 2002 which can be used to conduct usability reviews and heuristic evaluations. The application supports the evaluator in logging issues to reduce the amount of rework after the evaluation. The application has four different screens arranged as tabs. The first tab is used to identify issues and to prioritise them by frequency and impact. The second tab supports reviewing and sorting the list of collected issues. The third tab supports overall assessment with a checklist of guidelines. The fourth tab is used to generate a skeleton review report, which is then reworked in a word processor.

6.2.1 Creating Issues

Figure 6.1 illustrates the issues tab where issues can be entered. The title and a detailed description of the issue can be entered. Beneath the title and description parts, the location of the issue can be specified. Next to the location field, a drop down field is provided where the violated usability principle can be selected. The next part of the screen allows the assessment of frequency and impact for each issue. Review Assistant calculates the resulting severity by combining both factors, which may also be overridden by the evaluator. In the last part of the screen a recommendation for the issue can be entered.

6.2.2 Reviewing Issues

The second tab of Review Assistant, the issue list, gives an overview of the collected issues and allows them to be sorted. Figure 6.2 shows a list containing three issues. The list can be used to change an existing issue by selecting it. The list can also be sorted by chronological order, name, usability principle, and severity.

6.2.3 Checklist

The third tab, shown in Figure 6.3, shows the Checklist screen of Review Assistant. This screen is used to assess the interface overall using a checklist of guidelines. The checklist is categorised by usability principle. Selecting a principle in the category drop down field shows the corresponding checklist. Each item in the checklist can be rated for compliance. Additionally, a comment can be added for each item of the checklist.

6.2.4 Reports

The fourth tab of Review Assistant supports the generation of draft reports. Review Assistant reports must be finished using a word processor (see Figure 6.4). The *Reports* tab allows to enter the title of the report and name of the evaluator. A preview of the report is shown in the user's default word processor application.

The screenshot shows a window titled "Review Assistant - Untitled" with a menu bar (File, Issue, Checklist, Report, Help) and a toolbar. Below the toolbar are tabs for "Issues", "Issue List", "Checklist", and "Report". A yellow instruction box says: "Use this screen to note issues you identify in your review."

The form contains the following fields and controls:

- Issue name (used as a heading in the report):** A text box containing "Different navigation across the site."
- Description of issue:** A larger text box containing "The navigation style changes across the site. Users may not be able to notice their actual position on the site."
- Specify where the issue occurs:** A text box containing "Across the whole site".
- Principle violated:** A dropdown menu showing "Consistency".
- Frequency:** Radio buttons for High (selected), Medium, and Low.
- Impact:** Radio buttons for High (selected), Medium, and Low.
- Frequency and Impact = Calculated severity:** A label showing "High".
- Recommendation:** A text box containing "The issue can be solved by implementing a consistent site navigation or by introducing a location indicator, for example a breadcrumb navigation." Above this box is a label: "Use this field if you want to override the calculated severity: Override severity:" followed by a dropdown menu.

At the bottom, it says "Issue 1 of 1" and includes navigation buttons: a left arrow, "List issues", a right arrow, and a "New issue" button.

Figure 6.1: This screen of Review Assistant is used create issues. An issue consists of a title, a description, the location where the issue occurs, and the usability principle which is violated by the issue. The issue may also be prioritised by frequency and impact. The final text area can be used to enter a recommendation for the issue.

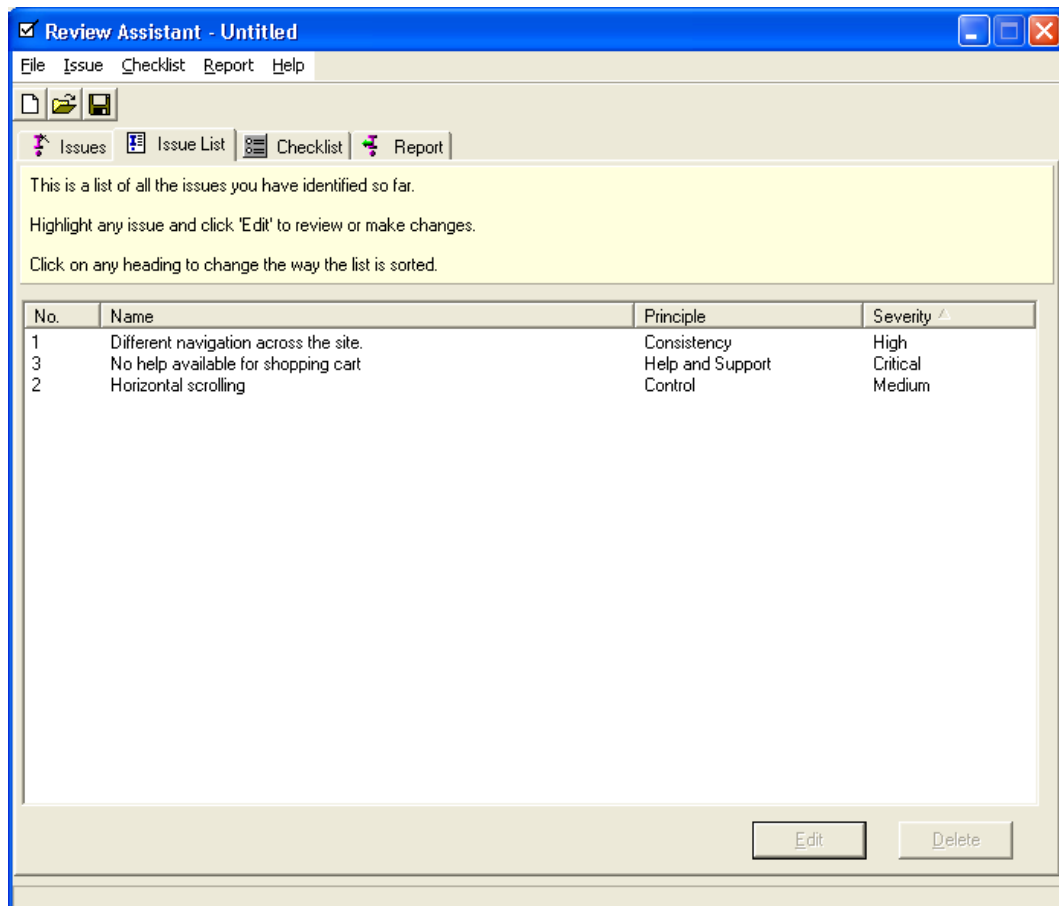


Figure 6.2: This screen is used to review the list of issues collected during a review. The list can be ordered by title, usability principle, or severity. An issue can be changed by pressing the *edit* button after selecting an issue from the list.

Review Assistant - Checklist

File Issue Checklist Report Help

Issues Issue List Checklist Report

Use the checklist to specify the extent to which the subject of your review complies with guidelines under various categories. The guidelines and categories are specified in the template file. The Checklist is saved automatically when you move from this screen.

Category: Control (There are 11 categories in total)

	Compliance					Comment
	Yes	No	Sometimes	Don't Know	Does not apply	
User can cancel interactions	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Comment
Clear Exit point from interaction pages	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	Comment
Page size is less than 50Kb	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	Comment
Alternatives to graphic links	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Comment
Site supports user's workflow	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	Comment
All appropriate browsers supported	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	Comment

Figure 6.3: The Checklist screen of Review Assistant is used to assess the interface overall using a checklist of guidelines. Selecting a principle from the drop down list shows a corresponding set of guidelines. Each guideline can be rated for compliance.

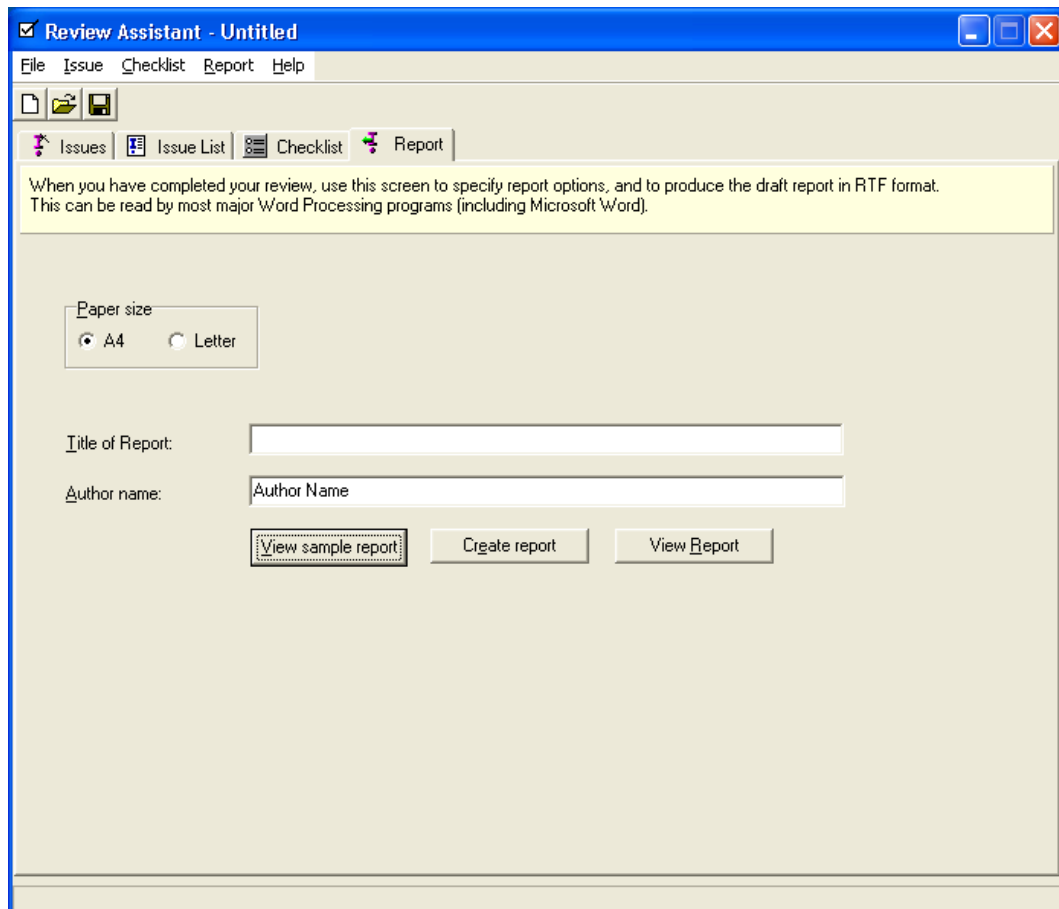


Figure 6.4: The Report tab of Review Assistant supports the generation of draft reports. Review Assistant reports must be finished using a word processor

6.2.5 General Considerations

Review Assistant can be used with different sets of usability heuristics. A template mechanism can be used to define various sets of heuristics as well as checklist guidelines.

Review assistant supports only reviews conducted by single evaluators. There are no facilities for managing reviews by multiple inspectors or for merging individual lists of issues into a combined list. Review Assistant has no facilities for providing one or more screenshots to illustrate an issue.

Chapter 7

Pattern-Oriented Software Development

The idea of using pattern-oriented design dates back to 1977 when Alexander et al. [1977] published their book about solving often recurring problems in architecture. The approach stated that every architectural design problem could be divided into smaller parts. The solutions of these parts are always of the same nature - so-called *design patterns*. Alexander described a set of patterns and built a pattern language using closely related patterns. This language could be used to solve complex architectural problems by applying appropriate design patterns at different levels of abstraction, such as towns, villages, buildings, and rooms. The approach was contentious because it reduces the creative aspect in architecture: by applying the correct patterns, everyone would be able to create sophisticated architecture.

Alexander's approach can also be applied to object-oriented software design, where the main goal is to build reusable and robust software structures. The design should be a specific solution to a given problem and should be flexible enough to address future problems or the requirements of a changing world. Experienced programmers tend to reuse designs, certain problems re-occur and are solved using similar designs, having the characteristics of a design pattern.

In 1987 Kent Beck and Ward Cunningham began to apply Alexander's ideas to software problems [Beck and Cunningham, 1987]. Soon after, Jim Coplien compiled a list of so-called *idioms*, which are similar to patterns, but more specific to a particular programming language. These idioms were published in *Advanced C++ Programming Styles and Idioms* [Coplien, 1992].

Design patterns gained much popularity in object-oriented software development after the book *Design Patterns: Elements of Reusable Object-Oriented Software* [Gamma et al., 1995] was published by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, better known under the synonym *gang-of-four*. In the same year, 1995, the first *Pattern Languages of Programs* conference took place and the *Portland Pattern Repository* was founded by Ward Cunningham to document design patterns.

7.1 Properties of a Pattern

A design pattern is not invented, it is discovered. The problem solved by a pattern is well-known and has been often solved in similar ways. Problems which can be solved using a design pattern often occur in a certain context: the pattern has been applied many times before and its advantages and disadvantages are known. A design pattern is not bound to a specific programming language, its implementation can vary. It is also possible that a particular pattern makes no sense in a certain language, or cannot be implemented. For example, PHP version 4 is not a fully object-oriented programming language, many features are not present. The resulting drawbacks and possible solutions are discussed in Section 7.4.

A design pattern definition identifies a set of problem forces or constraints and a solution which resolves these forces [Beck et al., 1996]. The definition of a design pattern has the following four essential elements [Gamma et al., 1995].

Name: The name is the handle for the pattern. The name provides a common vocabulary for developers. The name should be a representation of the design problem, the solution, and the consequences of the solution in one or two words. A representative name enables communication at a high level of abstraction and the use of the term in documentation. The name of the pattern is unique and can be cited.

Problem: A description of the circumstances in which the pattern can be applied and the problem itself.

Solution: The solution describes the elements which help in solving the problem. It is not a concrete implementation, but rather a model which can be used in appropriate problem situations. The solution is an abstract description of the design problem and the general arrangement of the elements involved.

Consequences: The limits and trade-offs when using the pattern. They are essential when evaluating design alternatives.

Patterns usually consist of a set of components. The pattern describes the involved parts and how they interact with each other. The level of abstraction of a design pattern is above that of the description of single classes and instances. A design pattern describes cooperating components, their responsibilities, and relationships [Gamma et al., 1995].

Since pattern names are carefully chosen, they introduce a vocabulary for a design language. This language allows an effective discussion of design problems. The name of a design pattern represents the problem and its solution. A design team where every member is aware of design patterns need not to discuss details of a problem which is already solved by a pattern [Gamma et al., 1995]. These ideas can also be used in documentation. A pattern identified by its name prevents the misuse of a design idea by another developer [Buschmann et al., 1996].

Pattern-oriented software development allows the creation of well-defined structures. A pattern's definition provides a description of its functional behaviour and its relation to other patterns. Related patterns often occur in the same problem context. By assembling a system of cooperating design patterns, fully-defined functional structures can be designed before the first line of code is written. This enables designers to create large heterogeneous application architectures [Buschmann et al., 1996]. Systems of patterns can be interpreted as building blocks for complex systems. This attribute of pattern-oriented software design enforces the speed and quality of a design. Pattern usage saves time, because there is no need to search for solutions [Gamma et al., 1995].

7.2 Definition of a Pattern

The first software design pattern book [Gamma et al., 1995] introduced a common scheme for the definition of a design pattern. The proposed scheme became widespread and is used in most of the design patterns literature [Gamma et al., 1995], [Buschmann et al., 1996], [Schmidt et al., 2000], [Völter et al., 2002]. The following description illustrates the Gamma et al. [1995] scheme and the variations introduced by Buschmann et al. [1996].

Pattern Name: The name of the pattern is essential. It should be a descriptive and unique name. The name helps in identifying and referring to the pattern.

Pattern Classification: The pattern should be classified corresponding to a classification scheme. Classification helps developers find the pattern in a context of use.

Intent: The aim of the pattern, including the problem solved by the design pattern.

Example: An example to illustrate the existence of the problem and the need for the pattern (added in Buschmann et al. [1996]).

Also Known As: Other names that have been used to identify the pattern.

Motivation: The design problem and how this pattern can be used to solve the design problem (*Problem* and *Context* in Buschmann et al. [1996]).

Applicability: A description of situations where this pattern can be applied, possibly including poor designs which could be improved by using this pattern.

Structure: A graphical representation of the pattern's structure based on some object modelling language, for example Unified Modelling Language (UML).

Participants: A list of classes and objects used in this design pattern and their roles.

Collaboration: A description of how the participants interact.

Dynamics: The typical run-time behaviour of the pattern. It is equivalent to the *Participants* and *Collaboration* section in Buschmann et al. [1996].

Consequences: A description of the results, side effects, and trade-offs caused by the application of the pattern.

Implementation: Guidelines for implementing the pattern. This part represents the solution of the problem forces. Buschmann et al. [1996] give examples in C++, Smalltalk, and Java, but patterns can also be applied in other programming languages if it makes sense.

Sample Code: Code fragments to illustrate how the pattern could be implemented.

Known Uses: A list of examples where the pattern has been successfully applied in real systems.

Related Patterns: A description of other patterns that may be used along with this pattern, or instead of this pattern. It also includes the differences between this pattern and other related patterns. This section is denominated as *See Also* in Buschmann et al. [1996]. The section helps to build systems of patterns which cooperate to solve a larger problem.

Variants: A description of known variants or specialisations of this pattern in Buschmann et al. [1996].

Example Resolved: This section, introduced by Buschmann et al. [1996], describes any aspects of the pattern not yet covered in other sections.

7.3 Classification of Patterns and Pattern Systems

Design patterns vary in their level of abstraction and granularity. The large number of patterns defined by the design pattern community made it necessary to introduce flexible classification schemes. A classification scheme helps find a design pattern for a specific problem and other patterns which can be used along with a pattern to solve a larger problem context.

7.3.1 Classification Methods

The first classification approach appeared in Gamma et al. [1995] by classifying the patterns described in the book. A scale with two parameters was used. The first parameter is the *purpose* of the pattern and has three occurrences: creational, structural and behavioural design patterns. The second parameter

		Purpose		
Scope	Class	Creational	Structural	Behavioural
		Factory Method	Adapter	Interpreter, Template Method
	Object	Abstract Builder, Singleton	Factory, Prototype,	Adapter, Bridge, Composite, Decorator, Facade, Proxy
				Chain of Responsibility, Command, Iterator, Mediator, Memento, Flyweight, Observer, State, Strategy, Visitor

Table 7.1: The design pattern classification scheme from Gamma et al. [1995] uses two categories: *purpose* and *scope*. The table shows design patterns as they are categorised in the Gamma et al. [1995] book.

represents the *scope* of the pattern, taking the values object or class. Table 7.1 illustrates that first classification approach with corresponding design pattern names. Due to the large number of newly discovered design patterns, this scheme soon became obsolete. A more sophisticated classification scheme was later published in Buschmann et al. [1996] using the following requirements:

- The scheme should be easy to learn and understand.
- It should consist of only a few classification criteria.
- Every category should reflect characteristic properties of the contained design patterns.
- The scheme should provide a road map that yields a whole set of design patterns for a particular problem scope.
- The scheme should be open to enable the integration of newly discovered patterns.

Two classification criteria were developed from these requirements: *pattern categories* and *problem categories*. The first criterion, pattern categories, reflects different stages in the software development process:

Architectural Patterns: Design patterns to be used for creating the fundamental structure of an application.

Design Patterns: Design patterns which help to refine the coarse grained structure of the application.

Idioms: These are used during the implementation phase to realise a software architecture in a specific programming language.

The second criterion is problem-oriented. Every pattern is the result of a specific problem solution. By generalising these problems, the following categories of problems were developed:

From Mud to Structure: Patterns which support the controlled decomposition of large unstructured task blocks into a set of smaller structured tasks.

Distributed Systems: Systems which have infrastructures with components located in different processes or in several subsystems.

Interactive Systems: Patterns which help structure systems for human-computer interaction.

Adaptable Systems: Patterns that provide infrastructure for extending applications to react to evolving and changing functional requirements.

Structural Decomposition: Patterns to decompose large systems into smaller cooperating components.

Organisation for Work: Patterns that cooperate in order to provide a complex service.

Access Control: Patterns for guarding and controlling access to components.

Management: Patterns for managing and controlling a large number of objects, services and components.

Communication: Patterns to organise communication between components.

Resource Handling: Patterns to coordinate the access to shared components.

A design pattern which cannot be assigned to a single category, because it solves problems from more than one category, is assigned to every category for which it applies.

7.3.2 Pattern Systems

Design patterns often appear along with others in a specific context. Cooperating patterns can be used to solve a larger problem context. Each pattern definition contains the section *Related Patterns* which describes the relation of a design pattern to other ones. This attribute of design patterns can be used to create pattern systems and pattern languages. Each design pattern is visualised in a map in their neighbourhood to related patterns. The patterns in the Gamma et al. [1995] book were visualised as the pattern map shown in Figure 7.1.

7.4 Patterns Applied in HEM

HEM is written in version 4 of the PHP programming language. Version 5 of PHP was not available at the time the HEM project was started. Since PHP version 5 is backwards compatible to PHP version 4, HEM can also be used with PHP 5. PHP version 4 does not have some important features of object-oriented programming languages. The following design patterns were used in the implementation of HEM. Each section describes how missing features of the PHP programming language were compensated. A more detailed discussion of web application development using PHP is presented in Chapter 8.

7.4.1 Adapter

The adapter is a creational design pattern introduced in Gamma et al. [1995].

Intent

An adapter converts a given interface to an interface expected by the client. Using an adapter, classes can work together which would otherwise be incompatible.

Also Known As

Another name for an adapter is a wrapper.

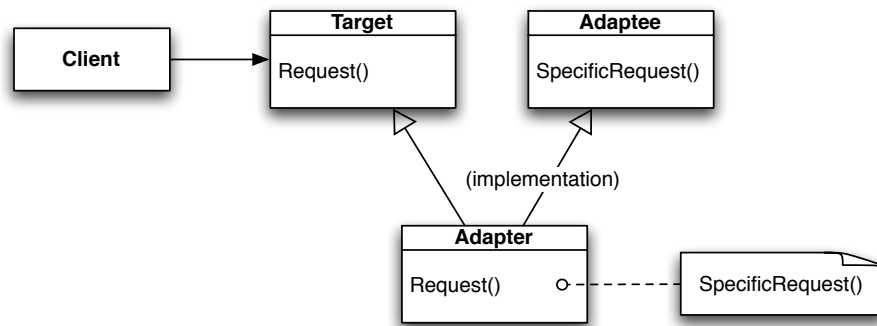


Figure 7.2: The adapter using multiple inheritance inherits from all adaptees and the target interface. The programming language has to support multiple inheritance. The client uses the target's interface.

Motivation

Sometimes a library is not reusable because the interface is not compatible with the interface an application requires.

Applicability

The adapter pattern is used when:

- an existing class with an incompatible interface should be used.
- a reusable class has to be created which should cooperate with unrelated or unforeseen classes.
- several existing subclasses should be used, but it is impractical to adapt their interfaces by subclassing. This applies only for the adapter using object composition.

Structure

An adapter can adapt in two different ways: by multiple inheritance or by object composition:

- *Adaption by multiple inheritance:* The adapter class inherits from all adaptees. The interface of the target is used by the client classes. The programming language has to support multiple inheritance in order to achieve this. Java and PHP do not support this feature, while C++ does. Figure 7.2 shows an UML diagram of adaptation by multiple inheritance.
- *Adaption by object composition:* The adapter class instantiates its adaptees and uses them as objects. The target's interface is used by the client and the adapter transforms the requests corresponding to the adaptees' interfaces. Figure 7.3 shows adaptation by object composition.

Participants

The *target* represents the domain-specific interface expected by the *client*. The *client* works with objects which conform to the *targets* interface. The *adaptee* is the existing interface that needs to be adapted. The *adapter* adapts the interface of the *adaptee* to the *target* interface.

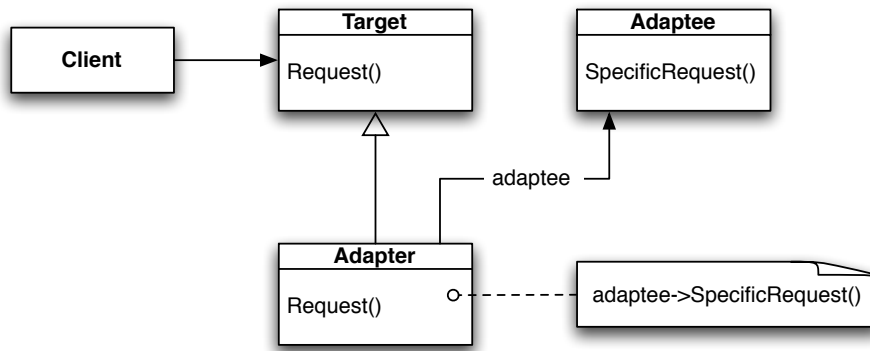


Figure 7.3: The adapter using object composition instantiates all its adaptees and uses them as objects. The adapter transforms requests corresponding to the adaptees' interfaces.

Collaborations

Clients call operations of the adapter interface. The adapter calls operations of the adaptee, which carries out the requests.

Consequences

The adapter design pattern is used if code written to be reusable cannot be reused because of incompatible interfaces. An adapter can also be used in the opposite way: a software component with a certain interface may be extended later with a better interface. In order to minimise the effort involved in using the new component, an adapter can be used to wrap the interface. The interface of the adapter can be defined corresponding to the application's requirements. If the adapted component is exchanged, only the adapter has to be rewritten.

Related Patterns

The bridge pattern has a similar structure, but a different intent: bridges separate interfaces from implementations to enable exchangeability of the implementations. The adapter changes the interface of an *existing* object.

The decorator pattern enhances another object interface without changing the interface. A decorator is more transparent to the application than an adapter. A decorator's aim is to support recursive composition, which is not possible with adapters.

The proxy design pattern provides a replacement for another object without changing the interface.

7.4.2 Singleton

The singleton is a design pattern in Gamma et al. [1995] and is an idiom in the terminology of Buschmann et al. [1996].

Intent

The singleton design pattern ensures that only one instance of a class exists, and provides a global point of access to it.

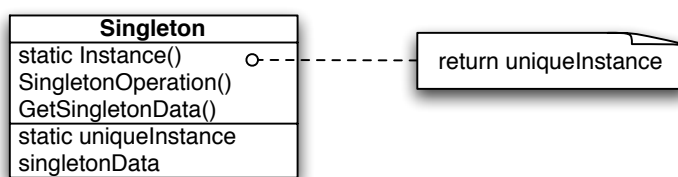


Figure 7.4: The singleton provides access to the unique instance it keeps through a custom instantiation mechanism. The singleton may also be responsible for creating its own unique instance.

Motivation

It can be important for many purposes that only one instance of an object exists. These are usually situations where shared resources are used, or where security concerns exist. A printer spooler, for example, should be used by only one person at a time.

Use of a global variable also makes an object accessible, but cannot be used to control access or to create further instances. A better way is to make the class itself responsible for keeping track of the number of instances.

Applicability

The singleton pattern should be used when:

- there must be exactly one instance of a class, and the instance must be accessible from a well-known point.
- when the instance should be accessible by sub-classing.

Structure

The access to the constructor has to be restricted. This is possible by declaring the constructor as private (C++), or by overriding the new keyword (Smalltalk). The only instance that is available should be referenced statically by the singleton class. A method provides another means of access to that reference.

PHP version 4 has no way to protect a constructor or to override the new keyword. A singleton can thus only be a convention that should be followed by developers. This is a good argument for using proper naming when implementing design patterns. If the object reference has the name *singleton* in its name, a pattern-aware developer knows how to use the object.

Participants

The singleton defines an instantiation operation which provides access to its unique instance. The singleton may also be responsible for creating the unique instance.

Collaborations

The singleton's instance can only be accessed by clients through the singleton's instantiation operation.

Consequences

The singleton pattern has the following benefits:

- *Controlled access to a single instance:* The singleton encapsulates its instance and controls how and when a client is allowed to access it.
- *Reduction of name space:* The singleton pattern is an improvement over global variables. The name space does not become not overpopulated by global references.
- *Variable number of instances:* The pattern can also be used to control a specific number of instances, not only one.

Related Patterns

The singleton pattern is used in many other patterns which require unique instances, for example abstract factory, builder, or prototype.

7.4.3 Layer

The layer pattern is an architectural design pattern described in Buschmann et al. [1996].

Intent

The layer design pattern helps to structure an application's architecture by decomposing it into groups of subtasks. Each subtask has a defined level of abstraction.

Problem and Context

Systems consisting of a mix of low-level and high-level functionality, where high-level modules rely on low-level functionality often have complex dependencies within the system's structure. The vertical structure can be defined using the level of abstraction of the different functionalities. Such systems often require also a horizontal subdivision, because of several different modules at the same level of abstraction.

Structure

A layered structure is illustrated in Figure 7.5. A particular layer is defined by its level of abstraction and the services it provides. A layer has to provide functionality for the next higher layer and is allowed to consume functionality from the underlying layer. No further dependencies are allowed between layers. Each layer shields all lower layers from direct access of higher layers.

Dynamics

Buschmann et al. [1996] describes five possible scenarios:

Scenario 1: the client requests functionality from layer N . Since layer N cannot fulfil the request, it calls layer $N - 1$ for support. Layer $N - 1$ also sends the request to the next layer $N - 2$. This sequence is repeated until layer 1 is reached, where the lowest level of service is performed. The answer for the client's request is passed up through all layers again, until layer N is reached.

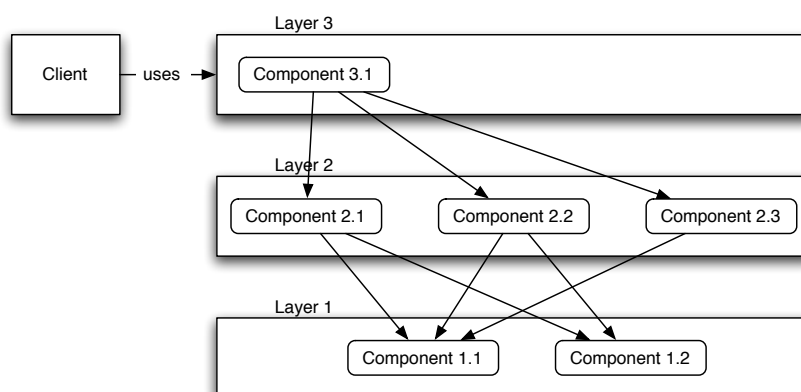


Figure 7.5: Each layer contains one or more components. Each layer provides functionality for the next higher layer and must only consume functionality from the next lower layer.

Scenario 2: the chain of actions is started at layer 1, for example when a device driver detects input. Layer 1 translates the request and sends it to layer 2, which interprets it. The request is passed through all layers until the highest layer N is reached.

Scenario 3: it is also possible that only a subset of layers is needed to fulfil a request, for example, if layer N sends a request to layer $N - 1$ and layer $N - 1$ is already able to fulfil the request. Such layers may be, for example, caching layers.

Scenario 4: similar to scenario 3 may be a situation where requests stop at a layer below N . For example, an impatient client may have re-sent a request, but the server has already sent the response in the meantime. A layer $N - 1$ may have noticed this, and stops re-sending the request.

Scenario 5: this scenario involves two stacks, which is commonly used for communication protocols. Layer N of the first protocol stack receives a request, which is passed down through all layers until it reaches layer 1. Layer 1 of the first stack sends the request to layer 1 of the second stack, where the request is moved through all layers until it reaches layer N of the second stack. A response of stack 2 is re-sent in the same way to stack 1.

Consequences

A layered structure enables the reuse of layers because they have a well-defined interface. Clearly defined and common levels of abstraction may be used for standardisation. For example, the TCP/IP protocol family (see Figure 7.6) is implemented as a layered structure and is standardised by the Internet Engineering Task Force [IETF, 1981].

Dependencies are kept local between neighbouring layers. A certain layer depends only on the functionality it consumes from the layer below. This reduces the risk of software defects due to code changes. A layer can be easily exchanged if the interfaces of the old and new layer are compatible. Even if they are not compatible, an adapter can be used to adjust the interface of the new layer.

Changing the behaviour of a layer may introduce problems. For example if the physical link layer of a network stack is exchanged with a layer with more bandwidth, other layers may not be able to operate quickly enough.

A layered structure is usually less efficient than a monolithic structure, because the requests may travel very long. Unnecessary work may be done by lower layers. The benefits of the layer design pattern cannot be fully exploited unless the granularity of layers is carefully chosen.

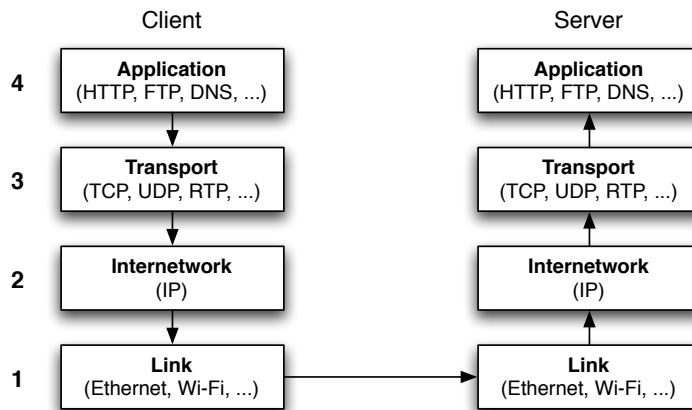


Figure 7.6: A client’s request is passed through all layers of the TCP/IP protocol stack until it reaches the physical *link* layer. At the server side the request is passed through all layers until the *application* layer is reached.

Related Patterns

The composite message pattern describes a possibility of object-oriented encapsulation of messages travelling through layers. The composite message pattern is a variation of the composite design pattern. The microkernel architecture is a variation of the layer design pattern.

7.4.4 Model-View-Controller (MVC)

The Model-View-Controller (MVC) pattern is an architectural design pattern presented in Buschmann et al. [1996].

Intent

The MVC design pattern is used to divide interactive applications into three cooperating parts. The *model*, which represents data and core functionality to access data, a number of *view* components to display information to the user, and the *controller* which reacts to user input and model changes. A change propagation mechanism is used to keep the model and any views consistent.

Problem and Context

The MVC design pattern is used in the context of flexible human-computer interfaces. User interfaces are usually subject to change and conflicting requirements. Building a flexible interface may be complicated if the user interface is intertwined with the functional core. The following *forces* influence the solution:

- Same information is presented differently.
- The display of information has to immediately reflect information manipulation.
- User interface changes should be easy and even possible at run-time.
- Support for different “look and feel” should not affect the underlying code.

Structure and Dynamics

The *model* encapsulates data and provides methods to access the stored data. Controllers can call these methods when data is required by a view. The model has to provide access for views to enable them to acquire the data to display.

A *change propagation* mechanism needs to keep track of all participating components. Controllers and views can announce their need to be informed of changes. Changes of the model trigger the propagation mechanism and all subscribed components are informed about the changes. The change propagation mechanism is the only connection between model, views, and controller.

The *view* components present the information provided by the model to the user. Different views present the information in different ways. A view has to subscribe to the change propagation mechanism. If the model changes, the views are notified and update their presentation.

The *controller* component interprets user input as events. The event handling mechanism depends strongly on the platform. The received events are translated to requests to the model or the associated views. If controller behaviour depends on the state of the model, the controller can register itself to the change propagation mechanism.

Consequences

Using the MVC pattern has the following benefits:

- *Multiple views of the same model:* Data and user interface are strictly separated in MVC. Multiple views can be used with a single data model.
- *Synchronised views:* The change propagation mechanism ensures that all subscribed components are notified of changes. This benefit is limited in web applications because the browser cannot be notified of model changes, since web applications use a client-server protocol. Only the browser is able to notify by sending a request.
- *Pluggable views and controllers:* The conceptual separation of the cooperating parts allows flexible interfaces by being able to easily exchange views and controllers.

Related Patterns

The *presentation-abstraction-control* pattern uses a different approach by coupling the views and the controller component to a presentation component.

7.5 Software Frameworks

A framework is not a design pattern. Frameworks are closely related to design patterns and object-oriented programming. A software framework is a reusable architecture developed for a class of problems and not for a specific task. Frameworks are open architectures. They provide a set of cooperating classes which help to solve a specific problem. The classes are abstractions that provide defined points of access, that enable the framework to be adapted and extended to fit the application's specific functionality. These points of access or "plug points" are usually realised with callbacks, polymorphism, or delegation. A framework is not a working application. A developer uses the framework by extending it with the necessary functionality to build the application.

7.5.1 General Considerations

The difference between a framework and a library is an *inverted flow of control*. Using a library means that the application developer uses the library components: Using a framework means extending the framework's functionality and implementing callback functions. The application has only to call one method of the framework, which takes over control. The framework invokes the necessary components and functionality at the right time and place, in a manner which is fully transparent for the developer.

Gamma et al. [1995] describe the following major differences between frameworks and design patterns:

- *Design patterns are more abstract than frameworks.* Frameworks can be expressed in code, patterns are descriptions of problems and their solution. A framework can be programmed once and be reused several times. A pattern has to be implemented afresh every time it is applied.
- *Design patterns are smaller architectural elements than frameworks.* A typical framework usually contains a number of design patterns
- *Design patterns are less specialised than frameworks.* Design patterns can be used in nearly every kind of application, frameworks are specialised to a particular problem set.

Model-View-Controller Design Pattern

A framework is often implemented using the Model-View-Controller (MVC) design pattern. Furthermore, many structural elements of this design pattern are already present in the request-response nature of web applications.

7.5.2 Open-Source Web Application Frameworks

This section describes some prominent open source web application frameworks and gives a short outline of their architecture. The presented information is based on the Web Application Framework entry from Wikipedia [2005a] and on the web sites of the various project groups. The list is organised corresponding to the programming language used by the frameworks.

Python

Zope is a fully object-oriented web application server. *Zope* is the abbreviation for “Z Object Publishing Environment”. Almost every aspect of *Zope* can be managed using the web-based user interface. *Zope* is based on Python objects which are stored in an object database maintained by the same development group. *Zope*'s strength lies in web publishing. Basic object types such as documents, images, and page templates are already included in *Zope*. Many more advanced third party objects exist, such as blogs and content focused content management systems (wikis).

Quixote also utilises the benefits of already available Python modules. The aim of *Quixote* is to enable experienced Python programmers to create web applications with their existing programming skills. *Quixote* uses the common gateway interface (CGI) and provides its own template language.

Ruby

Ruby on Rails is a framework closely related to the Model-View-Controller (MVC) design pattern. *Ruby on Rails* is designed for simplicity, minimising the amount of code needed for application implementation. The model in *Rails* represents an abstraction for relational databases. The view is the display logic and the controller represents the application logic. *Ruby on Rails* can be extended by many existing modules.

Perl

Maypole is also an MVC-oriented framework. It is very similar to Apache Struts. Maypole aims to have minimal coding effort, while being flexible enough to support enterprise web applications. The framework provides a generic way to use the MVC model. It provides a top level implementation of a controller to process user requests. The controller calls methods in the model, which collate the corresponding data. The model asks the view class to format the data and sends it back to the user.

Java

Apache Cocoon is a pipeline or component-based application framework. The central element of cocoon is the site map which represents the pipeline. The developer has to configure the site map in order to define how the pipeline components should handle different requests to produce a corresponding response.

Apache Struts is an open source framework for Java 2 Enterprise Edition (J2EE). It extends the Java Servlet language to encourage developers to adopt the MVC design pattern. Apache Struts features a broad variety of models (for example Enterprise Java Beans and Java Beans), and views (for example Java Server Pages, Extensible Stylesheet Language Transformation).

PHP

PRADO uses an event-driven component-based approach. It was originally inspired by Apache Tapestry, a Java web application framework. The approach is to split a web application into a set of pages each separated into components. *PRADO* is available in PHP5, the PHP4 version is no longer maintained.

HORDE is an application framework with a strong groupware background. It originates from one of the first web clients for the IMAP (Internet Message Access Protocol) e-mail protocol. The framework has now about 60 framework packages and maintains 5 core groupware applications. There are over 50 applications in development. The framework has no overall structure, but has important features like back-end independence using pluggable data sources, internationalisation, and a powerful template engine.

Chapter 8

PHP Web Application Design

According to Wikipedia [2005b], PHP was originally a small set of Perl scripts developed by Rasmus Lerdorf. They were later rewritten as a set of C programs using the common gateway interface (CGI) by the same author in 1994. The aim of these Personal Home Page tools (PHP) was to collect and present certain data, such as traffic analysis of the author's private home page. On the 8th of June 1995 Lerdorf released the first version of PHP combined with the Form Interpreter (FI) under the name PHP/FI.

The parser of the PHP interpreter was later rewritten by Zeev Suranski and Andi Gutmans from the Technion Israel Institute of Technology. The result of this work, PHP version 3, was published as a public beta version in 1997. In June 1998 PHP 3 was declared stable. Afterwards, Suranski and Gutmans rewrote the core of PHP. The result was published as the first Zend engine in 1999. In the year 2000 version 4 of PHP was released. It was powered by the Zend engine version 1.0.

PHP was not fully object-oriented until version 5 although it was capable of the most important object-oriented concepts such as inheritance, pass-by-reference and return-by-reference for objects. Complete object-orientation was implemented in version 5 which was released in July 2004. The most important enhancements in PHP 5 were a new object-model, private and protected member variables and methods, abstract classes and methods, interfaces, destructors, and exceptions. Although the above features were not available in PHP 4 it was possible to build robust object-oriented code.

8.1 Important Features of Web Applications

PHP became a very popular development platform for personal and corporate use because of its speed, ease of learning, and absence of license fees. These attributes enabled fast development cycles with PHP. This often ends up in poorly designed scripts, which is often criticised by developers of other platforms. To overcome that attribute of PHP, a modern web application should have the properties described in the following sections, which are summarised from Kabir [2003].

8.1.1 Object-Oriented Code Base

Object-oriented design offers a robust, maintainable, and reusable code base. Every application uses a basis of similar components. A web application, for example, will usually need database connectivity, authentication and authorisation facilities, and a presentation mechanism. The development of a framework which incorporates such components allows the reuse of tested and deployed code every time a new application is created. Reuse of code reduces development time. Using an object-oriented framework simplifies the process of error detection and correction. For example, if every part of the application uses the same database interface, errors concerning the database interface can be corrected at one point in the application framework. Software frameworks are described in more detail in Section 7.5.

8.1.2 External HTML Interface

According to Nielsen [1999] “The only web constant is change”, in order to meet this challenge it is necessary to implement presentation mechanisms which can easily be adapted. Another requirement is that users want to change the look and feel of their interface and companies want to adapt the design to their corporate identity. With a clever template mechanism, for example by using XHTML (Extensible Hypertext Markup Language) [W3C, 2000] as the data representation language and CSS2 [W3C, 1998] as the layout language, it is easy to build adaptable designs. Following standards also enables users with disabilities to use the web application. Another advantage of a separate presentation layer is that the design and logic of the application are separated. The application can be developed by a number of people simultaneously. The members of the development team do not interfere with one another as long as they work on logically separated parts, for example, business logic and presentation layout.

8.1.3 External Configuration

When an application is delivered, the end user has to be able to configure it easily. By placing the necessary configuration settings in a separate file, all configuration can be done in one place. External configuration files can also enhance security, because the place where they are stored can be changed according to the target system’s needs. This and other security topics are discussed in Section 8.2.

8.1.4 Multilingual User Interface

Web applications are very likely to be deployed for users with different languages. All translated parts of the application, such as messages, labels, and status information, should be stored in external files. Using a localisation strategy allows support for new languages by translating the language files. The application’s code is not altered, reducing the chance of introducing new software bugs.

8.1.5 External Data Storage

Using external data storage enhances manageability and security. Database systems usually maintain mechanisms for access control. This reduces the risk that data becomes corrupted by unauthorised access. Database systems usually also provide mechanisms for backing up and restoring data which are well-tested and robust. There is no need to create extra implementations for these tasks.

External data storage also ensures scalability. If the amount of data increases during the use of the application, the underlying database system could later be exchanged for a more capable system.

8.1.6 Access Control

Web applications which store sensitive data from multiple users will need authentication and authorisation facilities. Management of user accounts and privileges has to be implemented.

8.1.7 Portable Directory Structure

Web applications are usually deployed using the Internet. It is hard to predict the kind of environment in which the application will be used. A directory structure which can be adjusted allows the application to be adapted to a changing environment. Ideally, the application contains one directory. This may not always be possible, for example if external libraries are used which have to be stored in a special place outside the application’s directory.

```
1 <form action="/process.php" method="POST">
2   <select name="color">
3     <option value="red">red</option>
4     <option value="green">green</option>
5     <option value="blue">blue</option>
6   </select>
7   <input type="Submit" />
8 </form>
```

Listing 8.1: A web form with a drop-down field containing three options. The form is submitted back to the server from where it was fetched.

```
1 <form action="http://example.org/process.php" method="POST">
2   <input type="text" name="color" />
3   <input type="Submit" />
4 </form>
```

Listing 8.2: A web form with a text field. It can be located on any server because the address in the action attribute is an absolute address.

8.2 PHP Security Issues

The PHP Security Consortium maintains the PHP Security Guide [Fama, 2005] to provide documentation of common security problems that arise in PHP web application development. The most important of these are summarised below.

8.2.1 Form Processing

Spoofed Form Submit

This is one of the most simple methods to manipulate a web application. If the application which receives the form does not perform any checks, another form on a different server can be used to manipulate the application. Anyone can obtain the source code of the original form shown in Listing 8.1 from the server where it is stored, say from `http://example.org/form.html`. The form also exposes which script will receive the contents of the form through the action attribute of the form. This is enough information to create a form like that shown in Listing 8.2. An attacker can enter arbitrary values into the text field. The form will be submitted to the script which ought to receive the original form. Without proper filtering the application will accept all submitted values.

Spoofed HTTP Requests

A web browser communicates with a server using the Hypertext Transfer Protocol (HTTP). If a form is submitted the web browser sends a request to the server. The server receives the form values, processes them, and sends an answer back to the client's browser. Even if spoofed form submissions are not possible, for example because the host name of the submitted form is filtered, it is possible to manipulate the HTTP request. The HTTP request of a form submission containing one form field looks like that in

```
1 POST /process.php HTTP/1.1
2 Host: example.org
3 Content-Type: application/x-www-form-urlencoded
4
5
6 color=red
```

Listing 8.3: If a form is submitted by the browser a HTTP request is generated. This listing shows an example request of a form which submits one field, denoted by “color”. The request also contains a field where a host name is transmitted.

```
1 $http_response = '';
2
3 $fp = fsockopen('example.org', '80');
4 fputs($fp, "POST /process.php HTTP/1.1");
5 fputs($fp, "Host: example.org");
6 fputs($fp, "Content-Type: application/x-www-form-urlencoded\n\n");
7 fputs($fp, "color=red");
8
9 while(!feof($fp)) {
10     $http_response .= fgets($fp, 128); }
11
12 fclose($fp);
13
14 echo nl2br(htmlentities($http_response));
```

Listing 8.4: A HTTP request can easily be manipulated using a programming language which is capable of socket programming. This listing shows an example how PHP can be used to submit a manipulated HTTP request. The value in the *Host* field of the request can be freely chosen by an attacker.

```
1 <script>
2 document.location = 'http://DrEvil.org/steal_cookie.php?cookies='
   + document.cookie;
3 </script>
```

Listing 8.5: This entry sends the cookies of the victim to the server of the attacker using a javascript method which submits cookies.

Listing 8.3.

Manipulating the HTTP header of a request is harder than spoofing a form submission. The attacker needs to know the HTTP protocol. A terminal application could be used to directly communicate with the HTTP server. It is also possible to use a programming language that supports socket programming, like PHP, to write a custom client to manipulate HTTP request headers. Listing 8.4 shows an example of a custom HTTP client written in PHP. The *Host* field and any other attribute of the HTTP request can be freely defined by the attacker. Any filtering of HTTP request fields would fail.

Cross-Site Scripting

Cross-site scripting (XSS) is a very common vulnerability of web applications. XSS attacks have the following characteristics:

- The attacker exploits the trust that other users have for the site. Users usually do not have much trust in websites. However, the user's browser has more trust if it sends its cookies to the server. A cookie is used by the web server to store information in the client's browser. The behaviour of the browser regarding cookies depends heavily on the security settings of the browser. Many users do not care much about their browser settings.
- The web site displays external user data. These are web applications like e-mail clients, forums or other applications which display user entered content.
- The attacker has to inject malicious content to the web site. If the content is not properly filtered the attacker can submit any type of malicious data.

An example is shown in Listing 8.5. This is an entry from a web application which displays user submitted data without proper filtering. If another forum user displays the page containing this entry he will be redirected to the attackers web site. The cookies of the user are submitted as a parameter to the web site of the attacker. The risk of XSS vulnerability can be mitigated by implementing the following measures:

- All external data has to be filtered properly, firstly, when it enters the system, and, secondly, when it leaves the system.
- Existing PHP functionality should be used for security measures.
- Restrictions for entering foreign data should use a white list approach. A white list contains rules for what is allowed, everything else is forbidden. A black list is the reverse strategy. A white list approach is safer than a black list approach.
- Strict naming conventions should be used throughout the whole application source code. This makes it easy for developers to distinguish between unfiltered and filtered content.

Cross-Site Request Forgery

Cross-site request forgery (CSRF) is the opposite strategy of cross-site scripting (XSS). XSS exploits the trust that the user has in the web site, where CSRF exploits the trust a web site has in a user. A site will not trust an anonymous user, but if users have authenticated themselves to a web site, trust is transported using cookies. A CSRF has the following characteristics:

- Users have authenticated themselves to the web site and have higher privileges than anonymous users. Authenticated users are potential victims.
- The website relies on the identity of the user. The user is identified with session information stored in cookies, which are transmitted with every request.
- The attacker is able to add content which contains arbitrary HTTP requests.

CSRF requires faking of HTTP requests. HTTP is a client-server protocol: the web browser of the user acts as client and the HTTP server acts as server. The client starts a transaction with a HTTP request and the server finishes the transaction by sending an appropriate response. A typical HTTP request has the following sequence:

- The client's browser requests a document by sending a *GET* request with the desired document as parameter. Transmitting parameters with the *GET* method includes the value of the parameter in the URL (uniform resource locator) which is used to transmit the request to a web server.
- The server responds by sending the document.
- The client receives the document and processes its content. If the server's response contains an image the client needs to send another HTTP request to retrieve that image. The client also includes all session information to that request because the server cannot distinguish if the request was a direct user request or just a request which originates from an element within a document.

This request response scheme enables a CSRF attack. For example, if users have authenticated themselves to a web site of an online brokering service, this site also will provide a form for stock ordering. An example request for ordering stocks could look like this:

```
http://stocks.org/buy.php?stock_symbol=AAPL&quantity=1000
```

This request buys 1000 items of the APPL stock. A potential attacker could add the following code to the online brokering site. The content is later displayed on the site:

```

```

The browser will try to request the image and will include the session information into the request. However, the source of the image is not an image, but a request to order 1000 APPL stocks using the victims identity.

The following measures can be taken to protect an application from CSRF attacks:

- The HTTP specification considers the *GET* method to be safe. The above example shows that this is not true. All security critical forms should use the *POST* method. Parameters transmitted with the *POST* method are sent to the server within the HTTP request and are not visible in the URL.
- The application should not rely on the `register_globals` feature of PHP. If this feature is active all request variables are registered as global variables by default. Using the `$_POST` and `$_GET` arrays provided by PHP is considered to be safe.


```
1 $sql = "INSERT INTO users (username , password , email)
2     VALUES ( '$_POST['username ']',
3         '$_POST['password ']',
4         '$_POST['email ']' );
```

Listing 8.6: This PHP script assembles an SQL query which is used to insert a user account into a database. The script uses the values submitted by a form without checking them. This introduces an SQL injection vulnerability into the application.

```
1 INSERT INTO users (username, password, email)
2     VALUES ( 'bad_guy', 'mypass',
3         ('good_guy', 1234, 'good@guy.org');
```

Listing 8.7: After a successful SQL injection, the insert query contains two user accounts. The attacker has successfully injected SQL code.

- Security should not be neglected to increase convenience.
- The application should ensure that its own forms are used. This can be achieved by using tokens which are difficult to replicate and expire after a short period of time.

8.2.2 Databases and SQL

Exposed Access Credentials

If an application uses a database, the username and password for accessing the database must be stored somewhere. These are usually stored in a separate file which is included by all parts of the application requiring database connectivity. The file containing the access data is either stored inside the application's directory structure, or outside the document root of the web server. Both possibilities can expose the contents of the file.

If sensitive data is stored within the application's directory, it must be ensured that the file has a format which the web server will not deliver as plain text to the client. To store the access credentials outside the web server's document root may not always be possible in shared host environments. It must also be ensured on shared hosts that other users cannot read files containing sensitive data.

SQL Injection

An SQL injection is an attack method where the attacker tries to manipulate SQL queries by injecting extraneous SQL code through the web interface. For example, on a web site where the user has to provide a user name and password for authentication, the submitted data is used to create an SQL query to check if the provided data is correct. The attacker tries to enter data into the form fields to manipulate the SQL statement sent to the database.

The following example shows how an attacker could add user accounts to a web application. Listing 8.6 shows a PHP script which assembles an SQL query to add a user account to the database. The script uses the submitted form values directly. If an attacker enters `bad_guy', 'mypass')`, `('good_guy` into

the field for the user name, the SQL query would look like the query in Listing 8.7. The query adds two instead of one user accounts, enabling the attacker to add arbitrary user accounts.

In order to protect a web application from SQL injections, it has to have the following characteristics:

- The data submitted by a web form has to be filtered properly before it is used to create a database query.
- If the database system allows it, the data should be quoted. Quoting means enclosing variables with special characters which are used by the database system to distinguish between data and SQL statements.
- The data should be escaped. Escaping means that characters which have a special meaning within the SQL syntax are marked with a special escape character. PHP provides functions like `mysql_escape_string` and `addslashes` for this task.
- PHP provides also differentiates functions which submit only one SQL statement at a time and ones that are able to submit multiple statements per function call. Using single statement functions makes injecting nested queries impossible.

8.2.3 Sessions

Due to the request-response behaviour of a web application, a client can not be identified during subsequent page requests. To overcome this drawback a session identifier is used to identify the client. The session identifier is usually stored as a cookie in the client's browser. If the client's browser does not accept cookies the session identifier is transmitted as a parameter within the URL (uniform resource locator).

Session Fixation

Session fixation is the precondition for many session-related attacks. Session attacks have the goal of impersonation in most cases. There are three strategies how an attacker might examine the session identifier.

Prediction: The attacker tries to guess the session identifier. It is very unlikely that this method will work if the application uses PHP's own session identifier generation methods.

Capture: Session identifiers are usually transmitted either as cookies or as a GET parameter in the URL. Although there have been cases of cookie hijacking in Internet Explorer, the cookie method is now considered safe. The GET method is very risky because the session identifier is transmitted using the URL to the web application. A simple protocol level redirect could expose the session identifier.

Fixation: This method is very easy to defend against. If the PHP script does not explicitly generate a session identifier, the server can be forced to use a identifier transmitted by the user. For example, if the parameter `?PHPSESSID=1234` is added at the first request the server can be forced to use that identifier for the rest of the session.

The revelation of session identifiers can be prevented by not using the GET method to transmit the session identifier. Unfortunately, this is the only method which works with browsers having no cookie support or cookies deactivated.

A second consideration regards the usefulness of session identifiers. Session hijacking, which is discussed in the next section, is only useful if an attacker can gain some higher privileges on the target system. If the session identifier is regenerated every time the user gets higher privileges the risk of session fixation can be minimised.

```

1  session_start();
2
3  if ( isset( $_SESSION[ 'FINGERPRINT' ] ) ) {
4      if( $_SESSION[ 'FINGERPRINT' ] != md5( $_SERVER[ 'HTTP_USER_AGENT' ]
5          ) ) {
6          /* Possible hijacking attempt */
7          /* Prompt for password again */ }
8  } else {
9      $_SESSION[ 'FINGERPRINT' ] = md5( $_SERVER[ 'HTTP_USER_AGENT' ] ); }

```

Listing 8.8: Adding a hard-to-guess string as a session variable makes it hard to hijack a session identifier. This example adds the MD5 hash of the clients user agent to the session variables. If the hash changes during the session a hijacking attempt might have occurred.

Session Hijacking

Session hijacking is the common term for all attempts where an attacker tries to gain access to another user's session data. Gaining access to a session identifier is not impossible, as discussed in the previous section. The best strategy is to make the hijacking of a session identifier less dangerous. If no other security measures are taken, a session can simply be hijacked by examining the session identifier. It is necessary to use other mechanisms to make impersonation by an attacker harder. Relying on information at the protocol level is not a good choice, for example multiple clients can have the same address if they use the same proxy server.

A better choice is using the `User-Agent` HTTP header field, where the client's browser is stated. It is very unlikely that a user will change browser within the same session. This is rather a sign for a possible hijacking attempt. In order to use this property the user agent that was initially used when the session was started has to be stored as a session variable. Listing 8.8 shows how this can be achieved. The script stores the MD5¹ hash of the user agent string as a session variable. The name of the variable should not give any hint how this session variable has been generated. In line 3 and 4 of the listing the variable is checked for changes. If the value has changed, the user agent header has changed and a possible hijacking attempt has been identified. There is still a small chance that this was not a hijacking attempt, so placing any blame on the user should be avoided.

8.2.4 Shared Hosts

A shared host is a server where multiple users have web accounts. Sharing a server reduces the cost, but results in less security. The threat emerges from other users of the shared host.

Exposed Session Data

In a shared host environment different users have to share common resources. PHP stores session data for every user by default in the same directory. The session files cannot be read by a user, but the web server which created the files has sufficient rights to read them. A simple PHP file browser can be used to read the files. This vulnerability applies only to web servers where the *safe mode* of PHP is deactivated. The *safe mode* feature of PHP increases security, for example no files can be read by the web server outside the home directory of the owner of the web space. The drawback of an activated *safe mode*

¹MD5 is a one-way function that produces a hash over a string with a minimum chance that two different strings will yield the same hash.

is less flexibility. A web application which cannot rely on the *safe mode* setting has to implement an alternative way of storing session data.

PHP allows the methods for session handling to be rewritten by user-defined functions. An excellent way to solve the session data problem is to use a database to store session data. Using the PHP function `session_set_save_handler` custom functions for the handling of session data can be defined. These functions can use a database to store session variables. Access to a database can be restricted more easily than access to the file system.

Browsing the File System

PHP offers enough file system functions to create a file system browser. Since PHP runs as a module in the web server, this browser has the same access privileges as the web server itself. This is not a security vulnerability, but in a shared environment the developer should keep this in mind when designing an application.

8.3 Best Practises

In Kabir [2003] a set of best practises for PHP programming is proposed. The best practises vary from coding conventions to proper documentation inside the code. The most important practises are summarised in the following sections.

8.3.1 Coding Conventions

Coding conventions enhance the readability of code. If good names for classes, methods, and variables are chosen, novice developers can easily gain insight into the functionality. A good name is one which transports meaning about the contained functionality.

There are many coding conventions available, some of them differ only in small details. It is very important that the development team agrees on a specific coding convention before starting the project.

8.3.2 Methods

The return value of a method belongs to the definition of the interface of a module. A common problem arises from the use of database queries. Often a developer relies upon the fact that the database will return at least one row. PHP does not know type safety, so it is in the developer's hands to return proper variables types.

Another problem related with method calls are long lists of parameters. These lists often cause misplacement errors which are very hard to correct. A better approach for long parameter lists is to use associative arrays. This makes a method call a two step action. First, the associative array with parameters is defined, and second, the method is called using the array as a parameter. This is much more concise and errors are easier to find, because the variable names and their values are side by side within the code.

Listing 8.9 shows a method call with a long list of parameters and Listing 8.10 shows a method call with an associative array.

8.3.3 Database Access

Much flexibility in database access can be gained by introducing a further layer of abstraction. This layer can be implemented using the adapter design pattern presented in Section 7.4.1. The adaptee can be any implementation of a database access library. The advantage of using an adapter is that the database

```

1 function someBadFunction( $name = null,
2     $email = null,
3     $age = null,
4     $addr1 = null,
5     $addr2 = null.
6     $city = null,
7     $state = null,
8     $zip = null ) {
9
10    /* Code */
11 }
12
13 $return_value = someBadFunction('John Doe',
14     'john@doe.com',
15     '88',
16     '405 Main St',
17     'Parsons , KS 67357',
18     'Parsons',
19     'Kansas',
20     '67357',
21     );

```

Listing 8.9: This example shows a method call which has many parameters. A mistaken permutation of some parameters would be hard to find.

```

1 function aGoodFunction($params) {
2     if (is_array($params)) {
3         /* Code */
4     }
5 }
6
7 $params = Array(
8     'name' => 'John Doe',
9     'email' => 'john@doe.com',
10    'age' => '88',
11    'addr1' => '405 Main St',
12    'addr2' => 'Parsons , KS 67354',
13    'city' => 'Parsons',
14    'state' => 'Kansas',
15    'zip' => '67357',
16 );
17
18 $return = aGoodFunction($params);

```

Listing 8.10: This example shows a method which receives an associative array as its parameter. The variable names and their values are side by side within the code. Errors are easier to find.

library can be easily exchanged by rewriting the adapter. The application is not bound to a specific database system or library.

Good `SELECT` Statements

A developer should always keep in mind that the underlying database may be extended as the application grows. For example, developers often tend to just select every field available in the current table. This may increase memory load if further table columns are added to the table scheme. It can also cause errors if the application retrieves SQL data as an array, because the array is returned in a different order after a table has been changed.

Quoting and Escaping

No input should be inserted into an SQL statement which has not been filtered, quoted, and escaped. As already discussed in Section 8.2.2 this opens possibilities for SQL injection attacks.

Returning of Errors

It is often assumed that queries are executed without any errors. This is especially dangerous with queries which return no data, like `INSERT` statements, `UPDATE` statements, and `DELETE` statements. A well-designed method should return appropriate error codes if the query fails. The meaning of the error codes should be included in the method's documentation.

8.3.4 User Interface

Application logic and data presentation should be separated. Depending on the size of a web application, a more or less sophisticated presentation technique should be used. For smaller applications a HTML template mechanism can be used. The application opens the HTML template files which contain HTML code and template variables. The application defines the values to be put into the variables and the template engine creates a HTML page. The complexity and capability of the template mechanism depend on the size of the application and the amount of traffic to be expected.

8.3.5 Documentation

Kent Beck writes in his book about Extreme Programming [Beck, 2000] that good code should be self-explanatory. The high level design can still be hard to understand just by reading code. The design documentation should be detailed enough to enable new programmers to join the development team. The code should be written in a way that the functionality is reflected in the naming of classes, methods, and variables. More complex ideas should be documented with several lines of comment.

Chapter 9

The Heuristic Evaluation Manager (HEM)

The Heuristic Evaluation Manager (HEM) is a collaborative web application which assists in every aspect of heuristic evaluation. Evaluators are given accounts in an evaluation project and enter their findings and supporting screenshots online. The evaluation manager uses HEM to assemble an aggregate merged list of findings. The evaluators then enter severity ratings for each usability problem into HEM. Finally, HEM supports sorting the findings in decreasing order of severity and generating a draft evaluation report in XHTML. HEM is implemented in PHP4 with XHTML and CSS style sheets and uses a relational database such as MySQL.

9.1 Components of HEM

Figure 9.1 shows a typical sequence of tasks when performing a heuristic evaluation using HEM. The User Manager is used for creating accounts for each participant. Sets of heuristics can be created using the Heuristic Set Manager. The Environment Manager is used to create forms which are used by evaluators during heuristic evaluation to describe their evaluation environment. Several rating scales created with the Rating Scale Manager can be assembled to rating schemes using the Rating Scheme Manager. The Project Manager is used to create projects and to manage projects' phases. Evaluators describe their evaluation environment using the Environment Collector. Using the Finding Collector evaluators are conducting their individual inspections by entering their findings which can be illustrated with screenshots. The manager of an evaluation summarises the individual finding lists into one comprehensive list of findings using the Finding Merger. The summarised list is rated by evaluators using the Ratings Collector. The Report Generator is used to preview, generate, and export the report of an evaluation.

9.1.1 The User Manager

The HEM User Manager shows an overview list of users as shown in Figure 9.2. The overview contains the following elements:

User The full name and the HEM username, in brackets, of the user.

Operations: Buttons to edit a user's attributes and to delete a user's account.

Status: The current state of the user's account: active, or inactive.

The appearance of the User Manager depends on the role of the authenticated user. Managers are able to change the settings of every user, and evaluators can only change their own settings. The following attributes can be set (see Figure 9.3):

User Manager	Managing Participants
Heuristic Set Manager	Managing Heuristic Evaluation Attributes
Environment Manager	
Rating Scale Manager	
Rating Scheme Manager	
Project Manager	Conducting Heuristic Evaluation
Environment Collector	
Finding Collector	
Finding Merger	
Ratings Collector	
Report Generator	Creating Results

Figure 9.1: A typical sequence of tasks when performing heuristic evaluation with HEM. Accounts for the participants have to be created, the attributes of heuristic evaluation can be changed, the evaluation is conducted, and finally, reports can be generated.

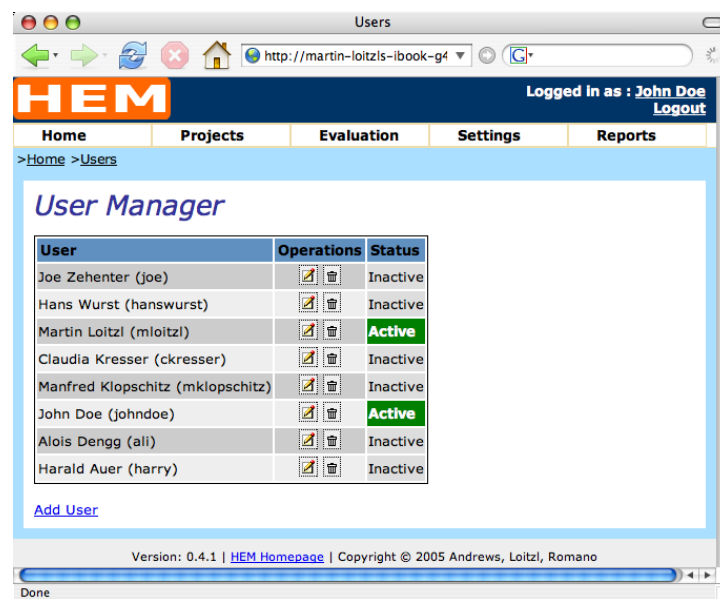


Figure 9.2: The User Manager shows an overview list of users available in a HEM installation. A user account can be changed and deleted by using the corresponding buttons. The last column of the table shows the current state of the user, active or inactive.

First Name, Last Name: The real name of the user.

Email: The email address of the user. This address is used to recover the user's password, if the user forgot it.

User Name: A string of characters used by the evaluator to log in. This field is not displayed if an evaluator uses this form.

Active: A user account can be blocked using this setting. No data is deleted, but the user cannot log in to HEM.

Street, Number, City, Post Code, Country: The postal address of the user.

Phone No.: The telephone number of the evaluator.

Old Password: The old password has to be supplied by the user in order to choose a new one.

New Password: The new password.

Retype Password: The new password has to be re-entered to prevent typing errors.

Language: The language of the user interface.

Member of Group: The user can be member of one of the following groups: administrator, manager, or evaluator. This field is not shown if an evaluator uses this form.

Comment: This field can be used by administrators and managers to store notes about the user. This field is not shown if an evaluator uses this form.

Change User

Logged In as : **John Doe**
[Logout](#)

HEM

Home Projects Evaluation Settings Reports

>Home >Users >johndoe >Change User

Change User: johndoe

First Name	John
Last Name	Doe
Email	hem@iicm.edu
Username	johndoe
Active	<input checked="" type="checkbox"/>
Street	Inffeldgasse
Number	16c
City	Graz
Post Code	8010
Country	Austria
Phone No.	
Old Password:	
New Password:	
Retype Password:	
Language	English
Member of Group:	Administrator
Comment	Working for UZSG, met him at CHI04.

Submit Cancel

Version: 0.4.1 | [HEM Homepage](#) | Copyright © 2005 Andrews, Loitzl, Romano

Done

Figure 9.3: The Change User form where user attributes can be changed. Evaluators, managers, and administrators see different sets of fields. Evaluators cannot see their group membership and username. The comment field is also not visible to evaluators.

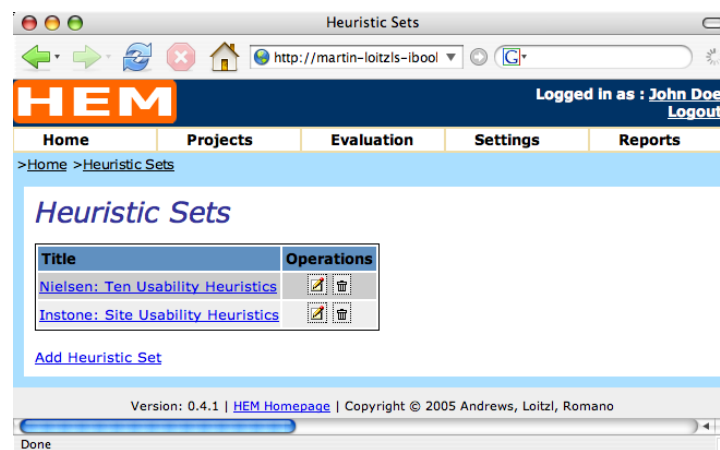


Figure 9.4: The Heuristic Set Manager shows one set per line. Every line contains buttons to Delete and Edit the set. New sets can be added using the “Add Heuristic Set” link at the end of the list.

9.1.2 The Heuristic Set Manager

Different sets of heuristics are used in heuristic evaluations depending on the kind of interface being evaluated. Sets of heuristics are discussed in Chapter 5. A heuristic evaluation can be conducted in HEM with any set of heuristics created with the Heuristic Set Manager. The heuristic set to be used for a project can be selected in the project settings form of the HEM Project Manager.

Figure 9.4 shows a list of all available sets of heuristics in the HEM Heuristic Set Manager. Each line contains an Edit and a Delete button. The Delete button removes the set. To change a set the Edit button is used.

A set of heuristics consists of a title section, a description section, and a list of heuristics. The description of a set is shown as tool tip whenever a set is listed in an overview. Each heuristic in the set also has a title and a description.

The Heuristic Set Manager provides an interface where every attribute of a heuristic set can be altered within one form. Figure 9.5 shows the interface with the set of 10 Nielsen heuristics. The upper part contains the title and description of the set. The lower part is used to enter several heuristics and their descriptions. Figure 9.5 shows a set that contains an empty field for one heuristic. New heuristics can be added by entering a number to the text field and pressing the “more heuristic entries” button. Every heuristic is followed by a Delete button which can be used to delete the corresponding heuristic and its description. All data should be entered in all available languages because titles and descriptions are delivered in the user’s language.

9.1.3 The Environment Manager

Evaluators describe the environment they used for their evaluation. This description is included in the evaluation report. The environment depends on the interface being evaluated. The Environment Manager allows the creation of custom forms where evaluators enter the description of their evaluation environment. The Environment Manager (Figure 9.6) provides the same interface functionality as the Heuristic Set Manager. Each environment consists of a title and a description in all languages and a number of form fields, one for each attribute of the evaluation environment.

Change Heuristic Set

Logged in as : [John Doe](#)
[Logout](#)

[Home](#) [Projects](#) [Evaluation](#) [Settings](#) [Reports](#)

>[Home](#) >[Heuristic Sets](#) >[Help and documentation](#) >Change Heuristic Set

Change Heuristic Set

Title of Heuristic Set:

English	German
Instone: Site Usability Heuristics	Instone: Web Usability Heuristiken
Jakob Nielsen's 10 usability heuristics with his description in each first paragraph and Instone's Web-specific comment following.	10 Heuristiken von Jakob Nielsen, darunter jeweils seine Beschreibung im ersten Absatz und Instones Kommentare darunter

Heuristics:

Order	English	German	
1	Visibility of system status	Rückmeldung des Systemzustandes (Feedback)	delete
	The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.	Das System sollte dem Benutzer immer (zum richtigen Zeitpunkt) Rückmeldung geben, womit es sich gerade beschäftigt.	
	Any user information should be easy to search, focused on the user's task, list	über ihre/ihrens Notwendig.	

0 more heuristic entries

Figure 9.5: The upper part of the form contains the title and a description of this set of heuristics. The lower part contains a title and description. All data should be entered in all languages, since the list is shown in the user's language.

Change Evaluation Environment

Authenticated as: **John Doe**
[Logout](#)

Home Projects Evaluation Settings Reports

>Home >Evaluation Environments >Change Evaluation Environment > Website Evaluation

Change Evaluation Environment

Title of Evaluation Environment:

English	German
Website Evaluation	Evaluierung von Webseiten

Description of Evaluation Environment:

English	German
Environment form used for evaluations of websites.	Umgebungsformular für die Evaluierung von Webseiten

Evaluation Environment Fields:

Order	English	German	
1	Age	Alter	Delete
2	Sex	Geschlecht	Delete
3	Web Browser	Webbrowser	Delete
4	Operating System	Betriebssystem	Delete
5	Connection	Verbindung	Delete
6	Monitor Colours	Monitor Farben	Delete
7	Monitor Resolution	Monitor Auflösung	Delete
8	Monitor Size	Monitor Größe	Delete
9	Date of Evaluation	Datum der Evaluierung	Delete
10	Time of Evaluation	Zeitraum der Evaluierung	Delete

0 more environment entries

Figure 9.6: Evaluators describe the environment they used for their evaluation. The Environment Manager allows the creation of custom forms where evaluators enter the description of their evaluation environment.

Title of Rating Scale:	
English	German
Frequency	Häufigkeit

Rating Scale Entries:			
Value	English	German	
0	Very seldom (<1%)	sehr selten (<1%)	Delete
1	Seldom 1-10%	selten 1-10%	Delete
2	Frequently (11-50%)	häufig (11-50%)	Delete
3	Often (51-89%)	sehr häufig (51-89%)	Delete
4	Almost always (>90%)	fast immer (>90%)	Delete

0 more value entries

Save Reset

Figure 9.7: A rating scale consists of a title and several values. Each value has a numerical value and a caption in each available language. The caption is presented to the evaluators in a drop-down field. Values can be added and deleted by using the corresponding buttons.

9.1.4 The Rating Scale Manager

Evaluators are asked to prioritise the finished list of manager findings. The priority of a finding may contain several factors. The Rating Scale Manager is used to create scales for every kind of factor. Rating scales can be combined to rating schemes using the Rating Scheme Manager described in Section 9.1.5. The interface has the same functionality as the Heuristic Set Manager (Section 9.1.2) and the Environment Manager (Section 9.1.3). Each rating scale consists of a title and several entries. Each entry consists of a numerical value and a short textual description (caption) of the value. Figure 9.7 shows an example with a factor for market impact. The scale consists of four values (0–3). Each value has descriptions in all available system languages. The descriptions are used as captions in the drop-down field presented to the evaluators when they submit their ratings.

9.1.5 The Rating Scheme Manager

HEM supports the usage of rating schemes containing of several scales. Nielsen [1994b] presents several factors which may influence the fixing priority of a usability problem. In order to support rating of findings using several factors the Rating Scheme Manager can be used to compile rating schemes containing several factors. Evaluators have to rate each finding on every factor of the rating scheme. HEM calculates the overall priority of the problem corresponding to the operation specified in the particular rating scheme.

Figure 9.8 shows an example of a scheme containing two factors: severity and frequency. The first part of the form contains the title of the rating scheme. The following part, denoted by “Associated Rating Schemes”, shows all factors which are currently assigned to the rating scheme. The Remove button (minus sign) can be used to remove the rating scale from the scheme. The list below contains all available rating scales frequency, market impact, and severity. Every scale can be added to the scheme

Figure 9.8: The first part of the form is used to enter the title of the rating scheme. A scheme consists of one or more scales. Scales can be added to the scheme by pressing the Add button (plus sign) in the list of available rating scales. Pressing the Remove button (minus sign) detaches a scale from the scheme. The drop-down field selects the operation used to calculate the overall result of the scheme: addition, multiplication, or average.

using the Add button (plus sign). Rating scales can be created using the Rating Scale Manager described in Section 9.1.4.

The last field in the form is to choose one of four possible ways to combine individual scales into an overall result. There are three possibilities:

Addition: The overall result is the sum of all scales.

Multiplication: The overall result is the product of all factors. This possibility may be used to combine factors that increase and factors that decrease the priority. For example, a scale with values from 1–10 may be an increasing factor, and a scale with values from zero to one may be the decreasing factors.

Average: The result of the scheme is the average over all scales.

A finding's rating on one scale is the average rating of all evaluators on that scale.

9.1.6 The Project Manager

The Project Manager assists the manager in running an evaluation project.

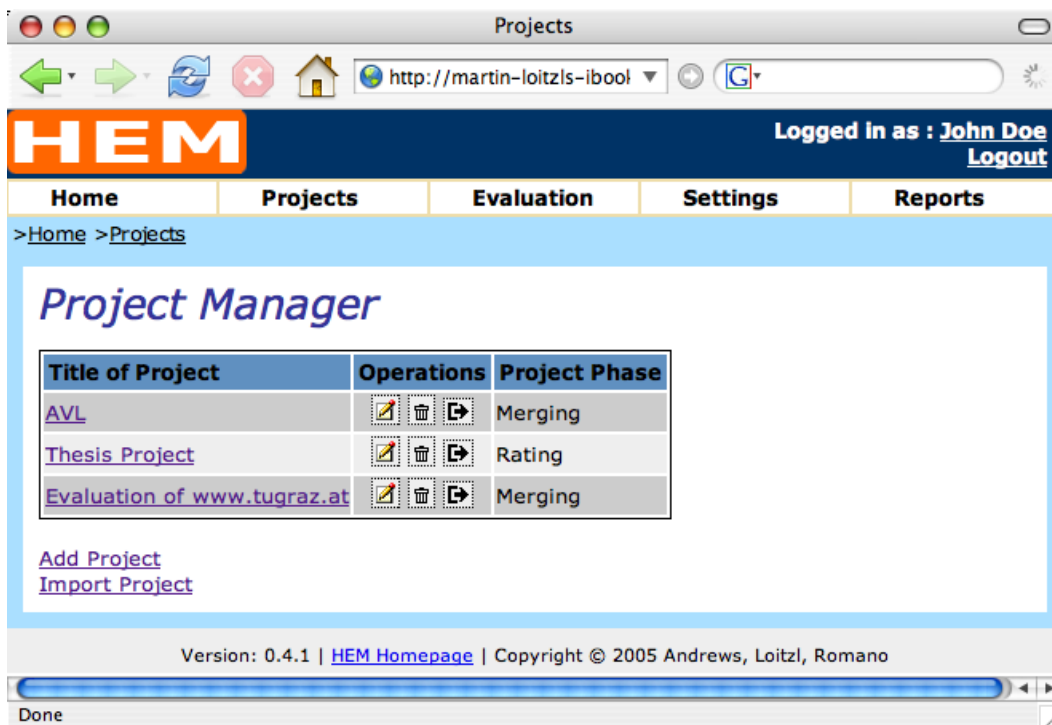


Figure 9.9: The overview screen of the HEM Project Manager shows all existing evaluation projects. A project can be edited, deleted, and exported as a single archive using the corresponding buttons. The last column of the table shows the project's current phase.

Project Overview and Export

The overview page of the project manager shows all projects of current HEM installation. Figure 9.9 shows the Project Manager containing three projects. The first column shows the title of each project, the second column contains buttons for all available operations:

Edit: This button opens the form to change a project's attributes.

Delete: The *Delete* button deletes the whole project. This function does not delete associated data to ensure database consistency. User accounts, rating schemes, environments, and heuristic sets are preserved. The operation does delete findings, screenshots, and project settings.

Export: Using this function a project can be exported as one single archive file. This file can be imported using the import function described in the next section. An exported project can be imported in any other HEM installation because all primary keys are globally unique. If a key already exists in a certain HEM installation it can be assumed that it represents existing data. The list of ten usability heuristics by Jacob Nielsen for example is already available in every HEM installation. This set will not be overwritten by a project import. The exported file contains all data of a project, including user accounts, heuristic sets, rating schemes, and environments.

The last column contains the current state of the project. The evaluation manager is able to gain an overview of the different projects' states.

Project Settings

Figure 9.10 shows the form where all project settings can be altered. The following list describes each form field and its purpose:

Title: The title of the project has to be provided in every language. The title of the project is presented in the evaluator's language.

Description: The description of the project is shown as a tool tip whenever the project appears in a list.

Users: Users can be assigned as evaluators for a project. A selected user name is added and removed from the project using the two arrow buttons between the lists. Users accounts can be created with the HEM User Manager described in Section 9.1.1

Set of Heuristics: A HEM project can be run with or without heuristics. HEM supports the use of different sets of heuristics, which can be created using the Heuristic Set Manager described in Section 9.1.2.

Environment: Each evaluator must describe the environment used for the evaluation. Each evaluator's evaluation environment is listed in the evaluation report. HEM offers an interface where any kind of environment form can be created using the Environment Manager described in Section 9.1.3.

Rating Scheme: The evaluators are asked to prioritise the finished list of manager findings using a rating scheme, which can be created using the Rating Scheme Manager described in Section 9.1.5. A rating scheme in HEM may contain one or more scales.

Project Phase: The project phase is used to define the different phases of the evaluation project. A HEM evaluation project has five phases:

Not started: The evaluation manager can add the accounts of the participating evaluators. The evaluators can enter their personal data. They can also enter their evaluation environment, but this can be revised until the Finished state. A briefing meeting may be used to create the user accounts.

Evaluation: In this phase evaluators conduct their individual evaluations using the Finding Collector.

Merging: After all evaluators have finished, the individual lists of findings have to be merged and summarised. This can be a group activity with several or all evaluators, or be conducted by the evaluation manager alone. For a group merging session the Finding Merger could be projected using a video projector to enable the evaluators follow the merging process.

Rating: After the findings have been summarised the evaluators are asked to prioritise the list of findings using the Ratings Collector.

Finished: The project is finished. The evaluators can no longer change any project-related information.

Project Import

A project can be imported into any HEM installation. A HEM project file is a single archive file which contains all data and screenshots. The archive file is a zip file and should not be extracted before importing. The import form (Figure 9.11) displays the maximum upload file size the server is configured to accept. The file browsing dialogue has to be used to point the browser to the file to be uploaded.

HEM Logged In as : **John Doe**
[Logout](#)

Home **Projects** **Evaluation** **Settings** **Reports**

>[Home](#) >[Projects](#) >[Evaluation of www.tugraz.at](#) >Change Project

Change Project

Title of Project	
English:	German:
Evaluation of www.tugraz.at	Evaluierung von www.tugraz.at

Description of Project	
English:	German:
The evaluation of www.tugraz.at using the HEM web application.	Die Evaluierung von www.tugraz.at mit HEM.

Users in Project	Available Users
Joe Zehenter Martin Loitzl Harald Auer Alois Dengg	Manfred Löpschitz Claudia Kresser Hans Wurst John Doe

Heuristics to use	Instone: Site Usability Heuristics
Environment	Website Evaluation
Ratingscheme	Severity
Project Phase	Merging
<input type="button" value="Save"/> <input type="button" value="Reset"/>	

Version: 0.4.1 | [HEM Homepage](#) | Copyright © 2005 Andrews, Loitzl, Romano

Done

Figure 9.10: The evaluation manager uses this form to run a project. The attributes of a project can be set by choosing a set of heuristics, a rating scheme, and an evaluation environment. Users can be added as evaluators for the project. The last drop-down field controls the current project phase.

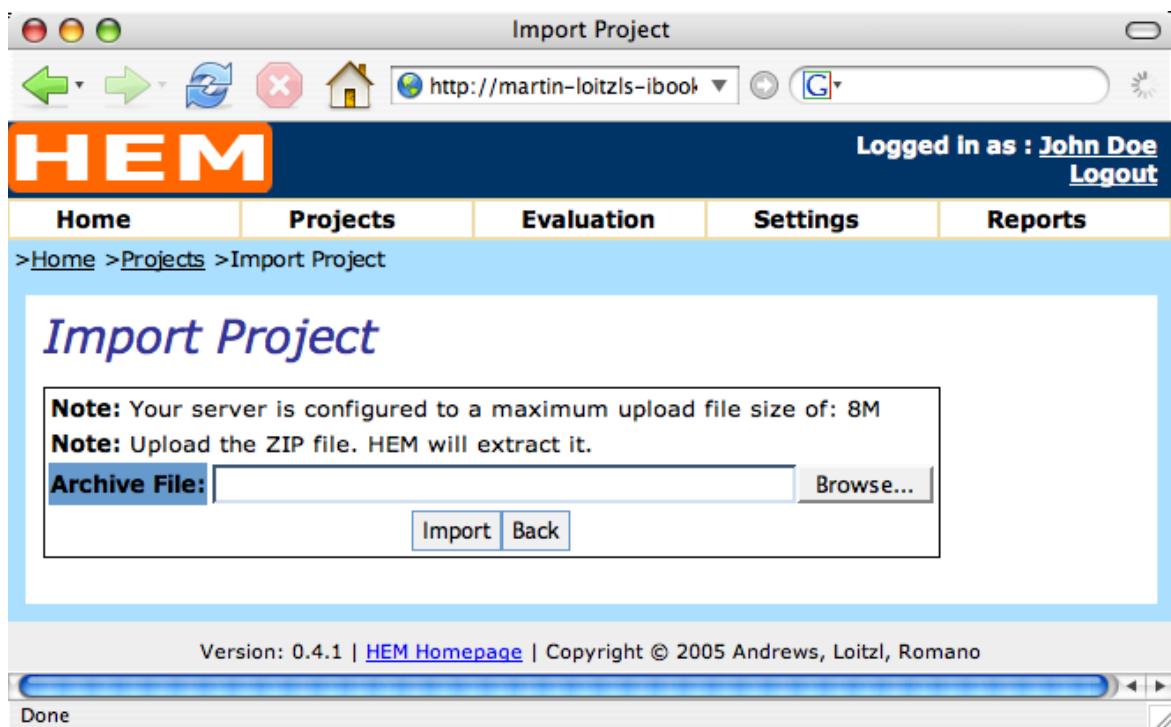


Figure 9.11: This form is used to import a HEM project. The user selects the location of the file to be imported. The screen displays the maximum upload file size.

After a HEM project file has been imported a summary of database operations is displayed: the number of inserted lines and the number of items that have not been imported because the data was already available in the database.

9.1.7 The Environment Collector

Depending on the kind of evaluation, evaluators will often use different equipment for the inspection. The evaluators are asked to complete a form to describe their evaluation environment. The Environment Collector (Figure 9.12) presents a form to the evaluators which can be configured using the Environment Manager described in Section 9.1.3. The evaluation environment of each evaluator is shown in the generated evaluation report.

9.1.8 The Finding Collector

The Finding Collector assists the evaluators in logging positive impressions (positive findings) and usability problems (negative findings) discovered during an evaluation session. The findings are organised as a list. The list can be re-arranged by evaluators as they work.

Heuristic evaluations can be conducted with or without using heuristics. If the evaluation is performed with heuristics, the evaluator can choose the heuristic corresponding to the logged finding from a drop-down list. The drop down list only contains the title of each heuristic. If the evaluator wishes to consult a more detailed description of a heuristic, the link beside the drop-down list can be used. A new window appears where the set of heuristics used for the evaluation is listed including a detailed description of each heuristic.

Figure 9.12: Evaluators will often use different equipment for the inspection. The evaluators are asked to complete a form to describe their evaluation environment.

Two images can be assigned to every finding, for example screenshots of a computer screen. These images are useful to discuss a usability problem after the evaluation, for example, if the problem cannot be reconstructed after the evaluation session, or if the interface is not available any more. One image can be edited and annotated by the evaluator in order to emphasise the finding, the second one should be the full-size unmodified original.

Figure 9.13 shows the finding overview of the Finding Collector. Each line represents one finding. The list can be arranged by using the up and down buttons in the operations column, which also contains Delete and Edit buttons. The third column denotes if the finding is a positive impression or a usability problem. The fifth column shows the heuristic that has been violated by the finding where appropriate. The last two columns show thumbnail images to help document the finding. The first is for the annotated version of the image, and the second for the original image. A full size view can be obtained by clicking on the thumbnail. To insert a new finding the arrow between two successive rows can be used. The position of the finding can later be changed by using the up and down arrows in the operations column.

Clicking on the Edit button brings the evaluator to a form where the attributes of a finding can be changed (Figure 9.14). The first item of the form is a text area where the description of the finding can be entered or edited. Below is a check box which denotes if the finding is a positive impression or a usability problem. The next field is a drop down box which can be used to choose the corresponding heuristic for the finding. That drop down field is only visible if the evaluation is conducted using heuristics.

The next part is for the images. Figure 9.14(b) shows the change finding form when images already have been added to the finding. After each thumbnail there is a delete button which can be used to remove the image. The first image can be a version annotated by the evaluator. The second image should be the original version without any modifications. Figure 9.14(a) shows the form if no images have been attached to the finding. The file browsing dialogue has to be used in order to select an image from the

The screenshot shows a web browser window titled "List of Findings" with the URL <http://martin-loitzls-ibook-g4.local/martin/hem-0.4/>. The page features the HEM logo and a navigation bar with "Home" and "Evaluation" tabs. The user is logged in as "Martin Loitzl" with a "Logout" link. The breadcrumb trail is ">Home >Collect Findings >Thesis Project".

Individual List of Findings

Finding Text	Positive Finding	Operations	Heuristic	Annotated Screenshot	Fullsize Screenshot
The logo is sometimes a link to the home page, sometimes not.	No		Consistency and standards		
Navigational elements are clearly visible.	Yes		Consistency and standards		
With every click inside the 'Job' part of the web site, a new window is opened. May be confusing for the user.	No		Error prevention		

Version: 0.4.1 | [HEM Homepage](#) | Copyright © 2005 Andrews, Loitzl, Romano

Figure 9.13: This is a list of findings shown in the HEM Finding Collector. Each row represents one finding. The first column shows the description of the finding, the next column denotes if the finding is a positive impression or a usability problem. The third column contains buttons for each operation which can be performed on that finding. The fourth column is optional and shows the corresponding heuristic. The last two columns contain thumbnails of the optional screenshots.

local file system.

9.1.9 The Finding Merger

The second step of a heuristic evaluation is to merge the individual problem lists of the evaluators into a single combined list. This activity can either be conducted as a group activity together with all evaluators and the evaluation leader, or only by the evaluation leader. The leader of an evaluation is called *manager* within HEM. The manager can also participate in an evaluation as an evaluator.

The Finding Merger supports the manager in creating the merged list of findings. Each finding the manager creates is a summary of all evaluators which discovered the same usability problem. Findings created by the manager are called *manager findings*. Each manager finding has to be connected with the individual evaluator findings which describe the same usability problem. The list of manager findings has the same properties as the list of evaluator findings within the Finding Collector described in Section 9.1.8. The descriptions of the findings should represent a summary of all individual evaluator findings. One part of the result of an heuristic evaluation is a list a list of findings. The relation of the manager finding to each evaluator finding is used to illustrate how many evaluators have identified a certain usability problem. This information could also be used to generate statistics about individual evaluator performance.

Figure 9.15 shows the part of the Finding Merger interface which is used to create a manager finding. The upper part of the interface is a form to change the attributes of a finding. It contains a form field for the summarised description of the usability problem, a drop-down field for a corresponding heuristic, where appropriate, and a check box which can be used to mark the finding as a positive impression. The lower part of the screen shows a tabbed list of all findings collected during the project. The first tab holds the list of manager findings. The list of manager findings has the same functionality as the list of the Finding Collector: findings can be ordered to assist the manager in organising the process of merging. Each row contains buttons to edit and delete a manager finding. Each further tab contains one list of individual findings for each evaluator participating in the project.

The Process of Merging

The manager starts to merge findings by inserting a new finding somewhere in the list using the Insert button (an arrow) below every finding line. The next step is to figure out which, if any, of the evaluator findings describe the same problem. Usually the manager will use the list of the evaluator containing the most findings for orientation.

The association of an evaluator finding with the Manager Finding is performed by clicking the Add button (a plus sign) at the beginning of every line within an evaluator tab (see Figure 9.16(a)). The background colour of the corresponding line changes colour to reflect the action of the manager. Also the icon that denoted the add operation will change to an icon containing a minus sign to denote the remove operation (see Figure 9.16(b)). By clicking the remove button the manager can detach the evaluator finding from the manager finding. Findings which have already been assigned to a manager finding have a darker background colour than findings that have not been used. This indicates the progress of merging findings.

Each evaluator finding has also an Edit and a Delete button (Figure 9.16). These can be used to correct the evaluators' findings after their evaluation session. The description of the Evaluator Finding can be reused by the manager by clicking the use text icon with the three dots. This button adds the description of the evaluator finding into the text area of the manager finding. The check box below the text area can be used to denote the finding as a positive impression. If heuristics are used within the project a drop-down field containing a set of heuristics is displayed. Next to the drop-down field is a link which opens a new window containing a detailed description of the set of heuristics in use.

(a) The form to change a finding's attributes where screenshots could be added by using the buttons to browse the local file system.

(b) The form to change a finding's attributes where screenshots have already been added. The Delete button can be used to remove a screenshot.

Figure 9.14: Two different states of the change finding form.

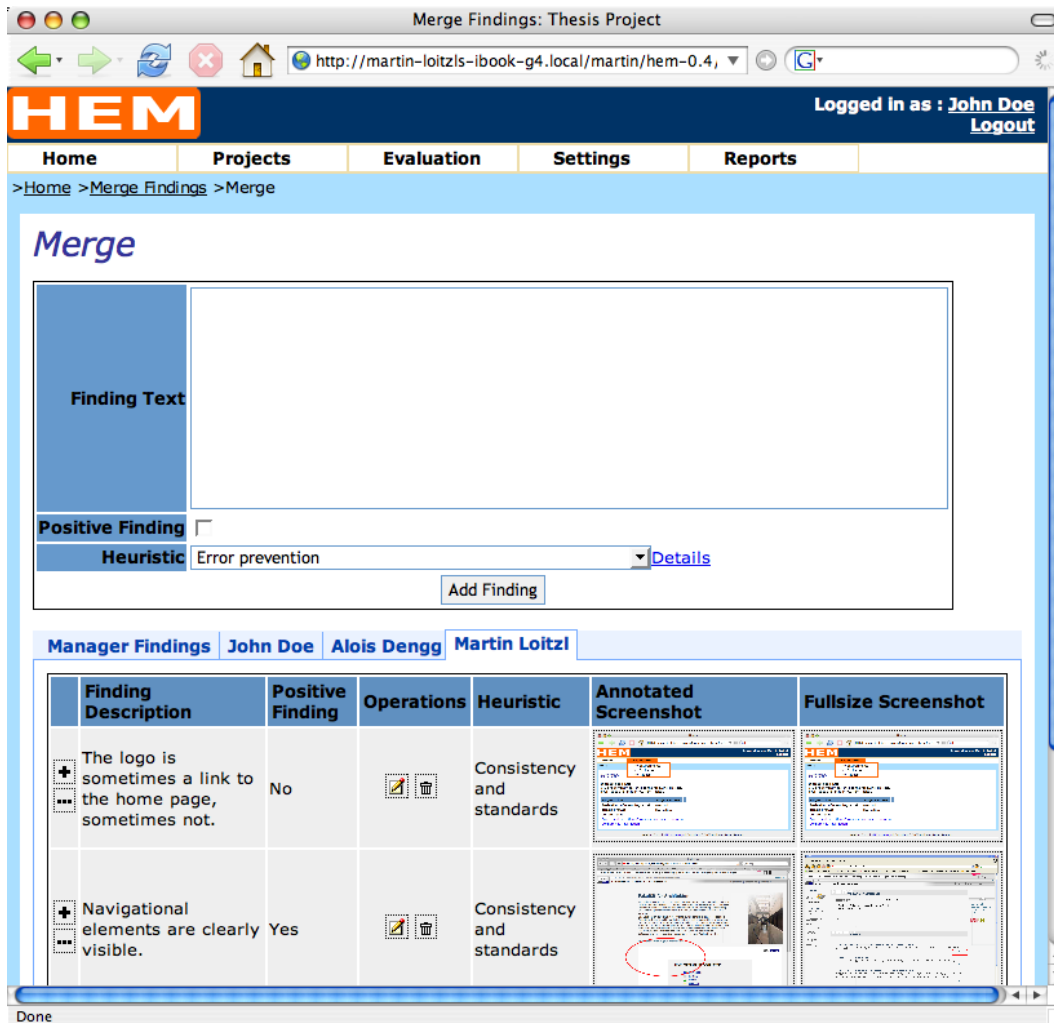


Figure 9.15: The user interface to merge the individual lists of findings from the evaluators has two parts. The upper part contains a form to add and change the attributes of a summarised manager finding. The lower part contains a list of already merged findings and tabs for each list of individual findings. The illustration shows the Finding Merger with the tab of the first evaluator activated.

Finding Description	Positive Finding	Operations
Home: Unklar was sind Links, was ist Werbung wie kommt man zu weiteren Seiten	No	[Edit] [Delete]
Home: Login Button nicht auf der Homeseite, aber auf Folgeseiten	No	[Edit] [Delete]
Home: AVI Logo ist		

(a) One or both of the findings from the evaluator can be assigned to the manager finding currently being edited by clicking the Add button denoted by a plus sign. The description of the evaluator can be reused by clicking the button with the *Use Text* button denoted by three dots.

(b) The first finding of the evaluator has been assigned to the manager finding. The add button changed to a remove button and the background colour turned red to reflect the action of the manager. The finding can be removed from the manager finding by clicking the remove button.

Figure 9.16: The manager of an evaluation summarises a finding by relating all corresponding findings from the evaluators. The descriptions of the evaluators' finding can be changed using the *Edit* button. The Delete button can be used to delete an evaluator's finding completely.

If the project contains many findings with attached screenshots, it can take some time to load the Finding Merger. In order to reduce loading time, the Finding Merger can be configured not to display screenshot thumbnails during the merging process. Figure 9.17 shows an overview of projects where findings can be merged. At the bottom of the overview is a check box which can be used to deactivate the display of screenshot thumbnails.

Image Selection

The manager can also choose the best screenshot to represent a summarised finding from among the screenshots submitted by the evaluators. In the best case, every evaluator will have submitted one annotated and one full size screenshot.

The first tab of the Finding Merger lists the already summarised manager findings. If no screenshot has been chosen, the column for the screenshots contains a link denoted by "Choose a Screenshot". After clicking this link a new window appears where all submitted screenshots are shown (Figure 9.18). The manager can then select one annotated and one full size screenshot. These screenshots are assigned to the manager finding. If screenshots have already been added to the manager finding the change form of the manager finding shows the selected screenshots (Figure 9.19). In order to detach the screenshots from the manager finding the "Detach Screenshots" check box can be used (see Figure 9.19).

9.1.10 The Ratings Collector

When the merged list of findings is finished, it is presented to the evaluators, who are asked to prioritise the list. HEM supports any combination of rating schemes. A rating scheme may contain several factors. The evaluators have to rate every finding by each factor. The rating scheme shown in Figure 9.20 contains two rating factors: severity and frequency. The result is later shown in the report of the evaluation. The

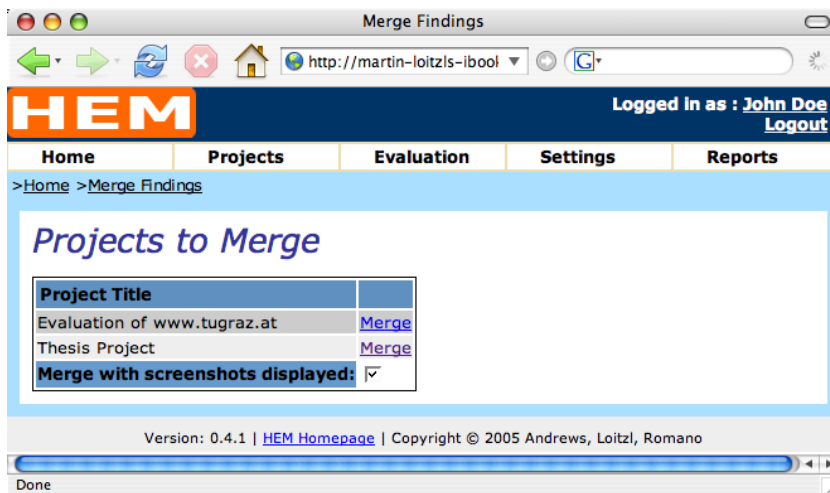


Figure 9.17: The overview of projects where findings are ready to be merged. Below the list of projects is a check box which can be used to deactivate the display of screenshot thumbnails to decrease the time needed to display the Finding Merger.

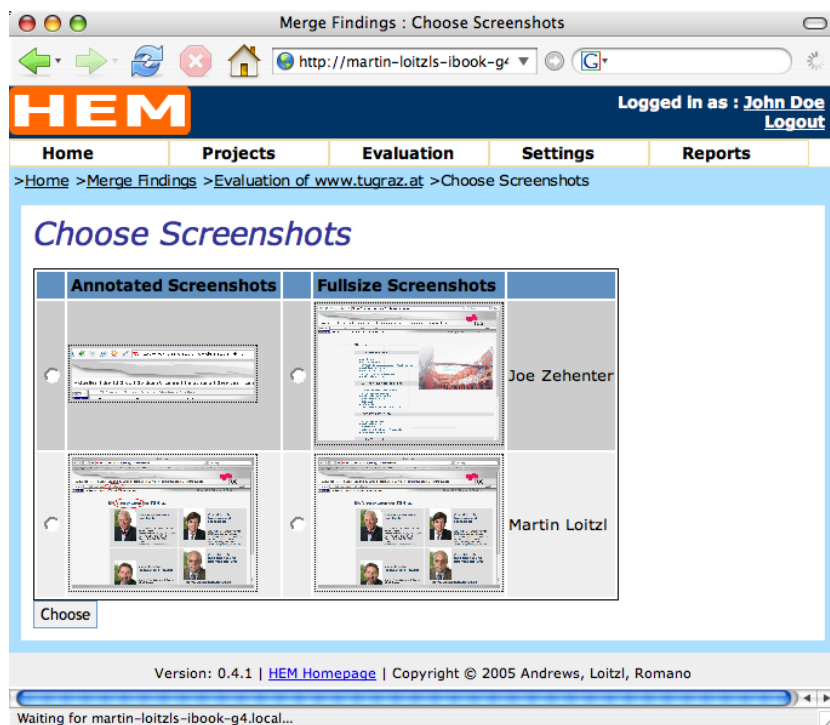


Figure 9.18: Each evaluator may have submitted one annotated and one unmodified screenshot for a finding. The manager can choose the best ones from the list for the summarised finding.

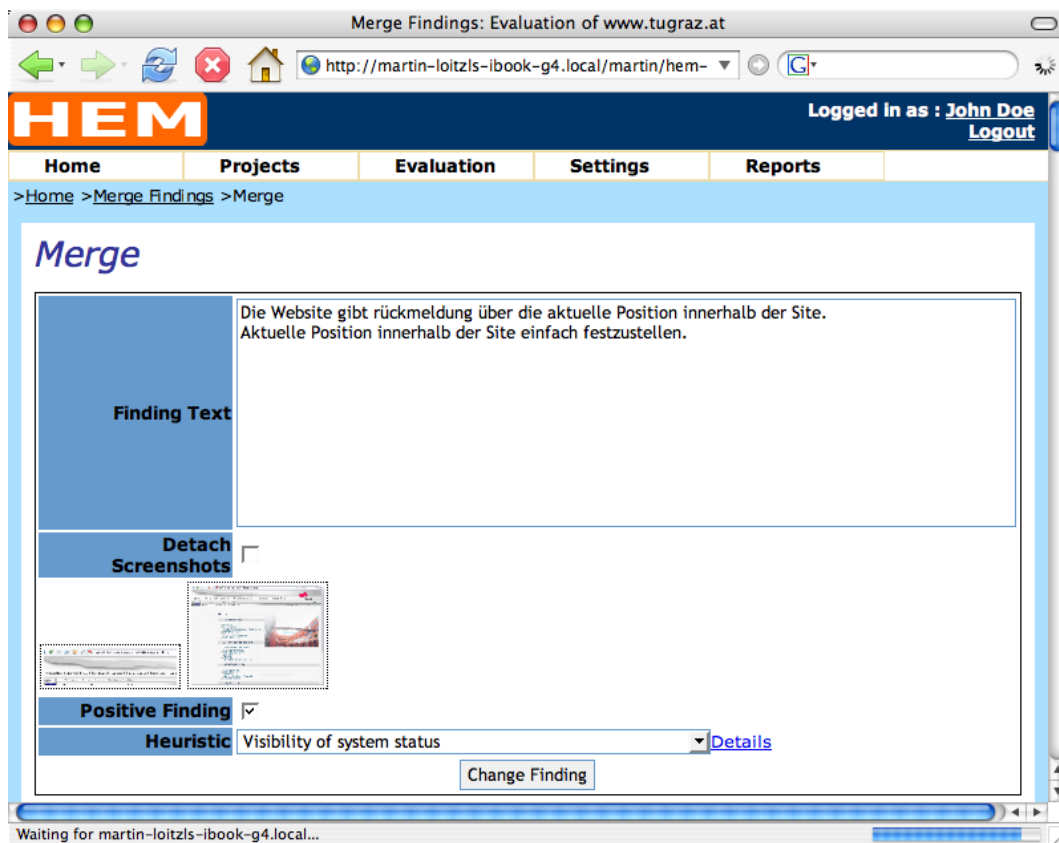


Figure 9.19: If the manager has already selected screenshots from the ones submitted by the evaluators, they can later be removed using the “Detach Screenshots” check box.

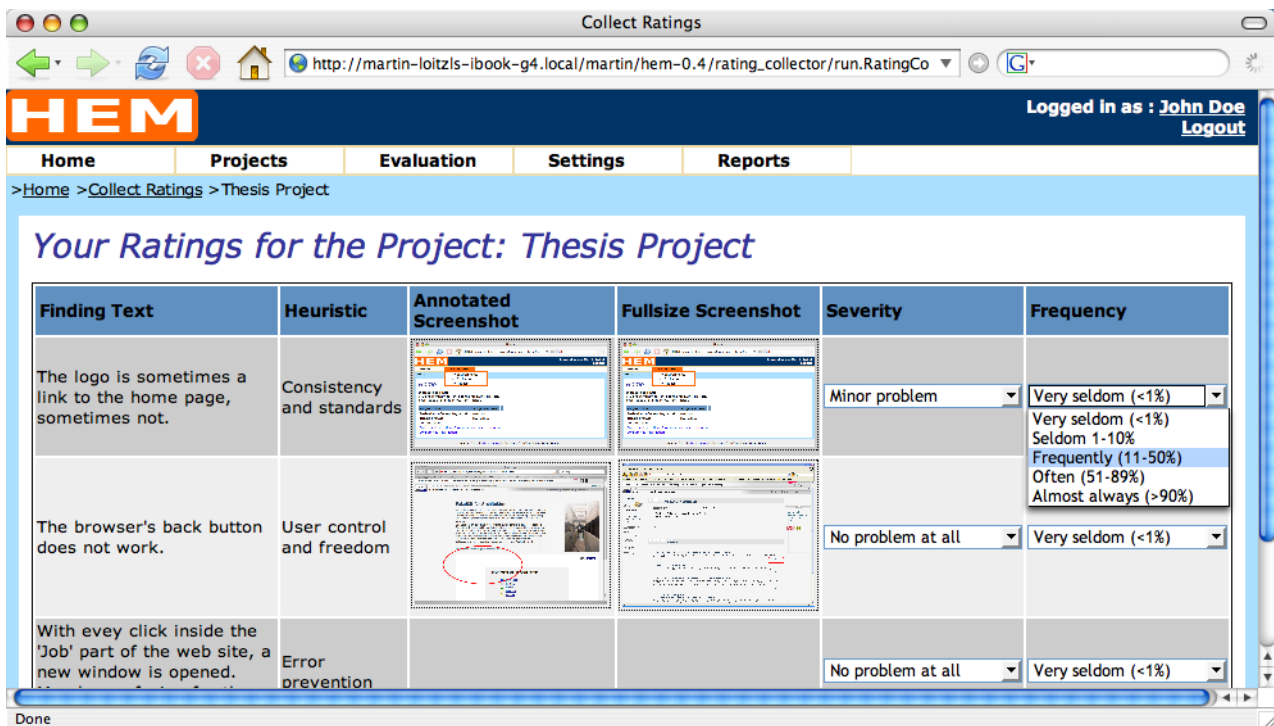


Figure 9.20: The finished list of manager findings is presented to the evaluators, who are asked to prioritise each finding. HEM supports rating schemes which may contain several factors. The scheme shown here contains of two factors: severity and frequency.

average rating is calculated for each finding. For the rating scheme shown in Figure 9.20 the overall rating result for one finding is the sum of both averages. Rating schemes are created using the Rating Scheme Manager described in Section 9.1.5.

9.1.11 The Report Generator

The Report Generator provides three important functions. First, it can be used to preview a report, which is useful to inspect a project's progress. Second, the report can be generated and stored to the file system for presentation by a web server. Third, a report can be stored as a single file archive for offline use. Reports can be generated in every language of the HEM user interface by clicking the corresponding button.

Report Preview

The Report Preview can be used to monitor the progress of an evaluation. The preview is generated instantaneously. Data entered by evaluators and managers is immediately visible within the report preview. Submitted screenshots are displayed in reduced size to reduce the page loading time. A full size view can be obtained by clicking on the reduced size screenshots.

Report Generation

This part generates a static version of the report using a customisable HTML template. The end user of HEM is able to easily adapt the layout to a particular corporate design. The static report files are stored in

the local file system of the server. If the directory where the reports are stored is located in the document tree of the web server the reports can be browsed online. The generated report contains screenshots: annotated versions are shown as inline images in the document, the full size versions can be obtained by clicking the annotated screenshots.

Report Export

In order to store a generated report on the local file system a HEM manager may download the report as single archive file using the export report feature. The archive has to be extracted before the report can be viewed.

9.2 The Application Architecture of HEM

HEM is based on a framework with a modular architecture. The application framework originally appeared in Kabir [2003] and has been rewritten and extended by the author. HEM is a collection of independent application modules which are all derived from the HEM application framework. The different HEM components were discussed in the previous section. The components of HEM communicate solely over a common database layer provided by the application framework. This ensures that no dependencies occur between the different HEM modules.

The HEM application framework uses several external libraries for authentication, database access, internationalisation of the user interface, and data presentation. The libraries are all under open source licenses. There is a chance that open source projects are discontinued, or that other, better projects become available. To address this possibility, HEM uses the adapter design pattern to adapt external libraries. The advantage of this approach is that new libraries can be inserted into HEM by simply rewriting the adapters. Otherwise every part of the application which used the replaced library would have to be rewritten.

The structure of HEM also provides an open architecture for future extensions of HEM. Some possible future extensions of HEM are discussed in Chapter 12. The HEM application architecture can be interpreted in two different ways: as a layered structure or as a Model-View-Controller structure. Both interpretations are design patterns. Pattern-oriented software development was discussed in Chapter 7.

9.2.1 HEM as Layered Structure

The layer design pattern describes an architecture where groups of subtasks can be decomposed into a layered structure. Each layer can only communicate with the underlying layer and the layer above. The layer design pattern was discussed in Section 7.4. Figure 9.21 illustrates the layered structure of HEM. The library layer contains all external libraries that are used by the application framework. In order to ensure exchangeability of libraries every library has its own adapter. These adapters represent the adaptation layer. The control layer contains the application framework which provides functionality for the application layer. The application layer contains the business logic of this HEM application which is plugged into the framework.

9.2.2 HEM as Model-View-Controller Structure

Figure 9.22 shows the software architecture of HEM interpreted using as Model-View-Controller (MVC) design pattern. Frameworks and their relationship to the MVC design pattern were discussed in Section 7.5. The MVC design patterns consists of three cooperating components:

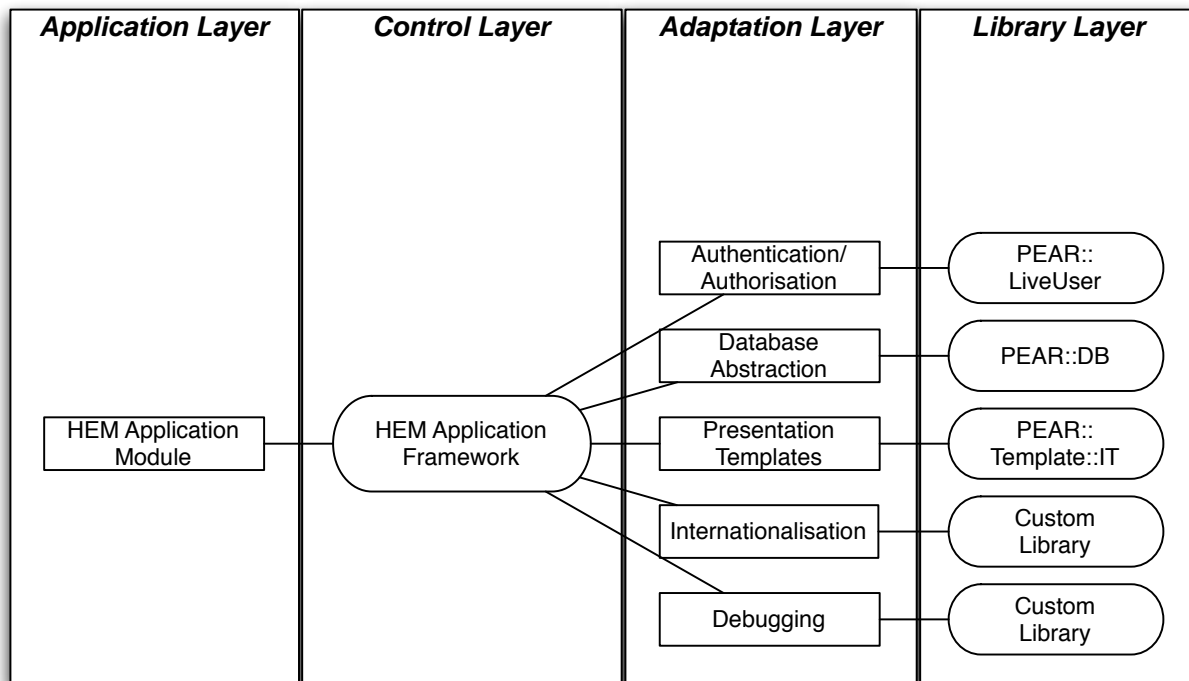


Figure 9.21: The software architecture of HEM interpreted as a layered structure. The *Library Layer* contains every library used by HEM. The library layer is adapted using the *Adaptation Layer*. The framework controls the execution of the application and thus represents the *Control Layer*. A HEM application, shown in the *Application Layer*, is implemented using the application framework.

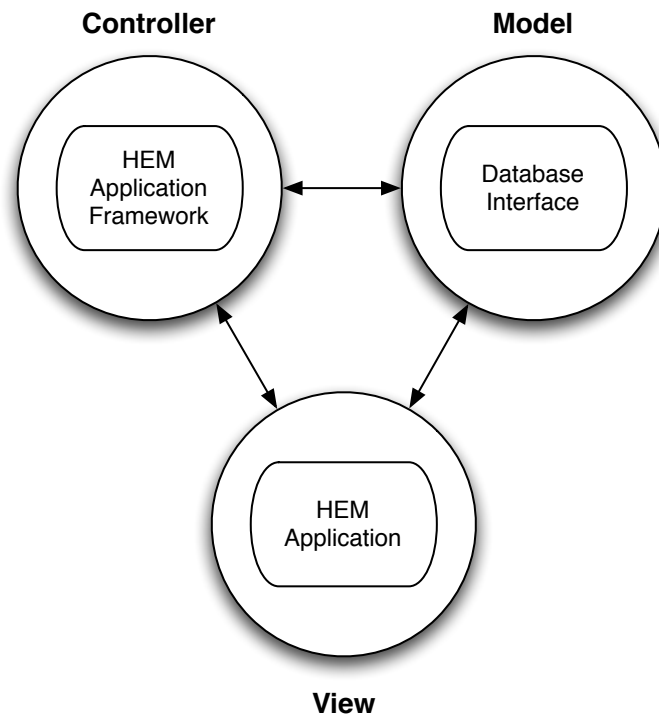


Figure 9.22: The HEM software architecture can be interpreted as a Model-View-Controller (MVC) design pattern. The database layer represents the model, the framework represents the controller, and the derived application models are the views in MVC.

Model: The model represents the data which is presented in one or several views. HEM uses a common database interface for all applications which is provided by the HEM framework. The database is the model of HEM. It stores all data and provides methods for controlled access.

Controller: The controller manages the way how the interface reacts to user input. A framework architecture maintains an inverted flow of control. The part of the framework that provides control facilities represents the controller of the MVC design pattern.

View: The view has to ensure that its appearance reflects the current state of the model. Whenever the model is changed the model has to notify the views that depend on the model. A presentation layer is provided by the HEM framework; the derived application modules consume presentation functionality from the framework. Each application module represents a view of the MVC design pattern.

9.3 The HEM Application Framework

The HEM Application framework uses several external libraries. The PHP community maintains an extension and application repository called PEAR. The purpose of PEAR is to provide a structured library of open-source code for PHP developers and users, as well as a system for code distribution and package maintenance [PEAR, 2005].

The HEM Application framework uses the PEAR::LiveUser package for authentication and authorisation purposes. The PEAR::DB package provides a very flexible database abstraction. A simple and easy

to learn template engine is provided by the PEAR::HTML_Template_IT package. Unlike other packages, it uses no special programming syntax inside the template definitions.

9.3.1 Authentication and Authorisation

The PEAR::LiveUser package has two major components. The first component is used for authentication. The package provides a container architecture which can be used to support several ways to authenticate in a heterogeneous environment. The LiveUser package can not only be used to authenticate against database tables, but also for example against e-mail servers or directory services. Existing systems for user management in enterprises can be used. The package's container architecture enables the developer to create custom authentication containers.

The second part of the LiveUser package can be used to address authorisation tasks. The package provides three authorisation models with increasing complexity. The *simple* model gives every user any right. The *medium* model has the features of the simple model and additionally a group mechanism. User groups can be defined and equipped with an arbitrary number of rights. Every user can be a member of an unlimited number of groups. The *complex* model has all features of the simple and medium model but also a possibility to define right implication, right levels, and different application areas.

HEM uses the medium model. There are three groups: evaluators, managers, and administrators.

An *evaluator* has the following rights:

- Entering and changing of *evaluator's* own evaluation environment.
- Adding and changing the *evaluator's* findings.
- Attaching and detaching screenshots to and from the *evaluator's* own findings.
- The submission of ratings for the finished list of the manager findings.

A *manager* has additional privileges:

- The *manager* can change the data of *evaluators* including environment descriptions, finding data, and ratings.
- The *manager* can add, change, or delete projects, sets of heuristics, rating schemes, rating scales, and evaluation environment definitions.
- The *manager* can export and import projects.
- The *manager* can generate and download evaluation reports.

The *manager* and the *administrator* group have the same privileges. The *administrator* role was added for future extensions of HEM.

9.3.2 Interface Internationalisation

In the current implementation of HEM two languages are available: English and German. The translation of the user interface is stored in external files. New languages can be easily added by translating the language files to the new language.

The HEM user interface has a language concept containing three components:

Labels: Labels are used for buttons, field descriptions, table captions, table headings, and application titles.

Messages: Messages are used to provide feedback for the user.

Errors: Error messages are used to signal that a problem has occurred and suggest a possible solution.

The internationalisation functionality is provided by a library created by the author. The language files are initialised when the HEM application framework is initialised. The library is included using an adapter. The framework cares for proper setup of the language. A developer who uses the internationalisation functionality needs not care about language settings of the users.

9.3.3 Content Translation

Content added to HEM has also be translated. Internationalised content is only entered by managers. The HEM application framework provides a translator class which allows the developer to create forms for every language. Every method of the translator class provides a programming interface which automatically adapts to the configured languages of a HEM installation. If another language is later added to HEM no application code has to be changed.

The translation class has a language transparent interface. The developer does not need to deal with language settings of the current user. The developer uses the translation class and the framework delivers the correct language.

9.3.4 Template Engine

A template engine separates data presentation from application logic. This reduces complexity and dependencies within the application code and enables parallel development of an application. The application logic can be implemented by a software engineer and the presentation can be realised by a web designer.

The HEM Application Framework uses a template engine with a simple syntax. The template is written in plain HTML with variables and blocks. Variables represent content which is later filled by the template parser. Blocks are used for iterations over repeated parts of HTML code, for example lists or select boxes.

The presentation of HEM is implemented in XHTML (eXtensible HyperText Markup Language) and CSS2 (Cascading StyleSheets) which introduces a further layer of separation. XHTML is a markup language which represents content. The layout is defined using a CSS2 stylesheet. The HEM application framework is able to handle multiple stylesheets which enables the developer to implement a user selectable look and feel for the application interface.

HEM is implemented using a layered template structure. A *master* template represents the first layer and defines the overall page structure. It contains header definitions, template variables for the page title and the stylesheet, and a variable that for the main content. The main content is provided by the application modules of HEM. This structure separates the parts of the user interface that are the same across the whole application from the parts which are provided by the application modules. The overall design can be changed at one point within the application code.

A different concept is the use of boxes. Boxes represent content that is independent from the application modules. By using boxes the developer is able to define further template variables in the master template. Content can be added to those variables with box applications. Every box has to be configured in the global configuration file. The framework needs to know where to find the class file, the name of the class, and the method that returns the content.

An example for a box is the authentication information. If a user has logged in the box shows the user's name and a logout button. This information is independent from the current application module in use. Another example is breadcrumb navigation. This box has to be visible in each application module.

The content depends on the current module in use. The box application collects the necessary data and renders it for the user.

9.3.5 Database Abstraction

HEM uses PEAR::DB as a database library, which provides database abstraction for relational database systems. Database abstraction allows the use of different database systems for data storage. Migration is even possible after the application has already been in use. PEAR::DB supports every database system that can be accessed by PHP functions and support some basic database features defined by the library developers. Examples for these systems are MySQL, Oracle, Microsoft SQL, ODBC sources, Sybase, SQLite, PostgreSQL, and Informix. The use of other systems may be possible, but has not been tested.

The database library is maintained by an adapter. The adapter ensures exchangeability of the library and a single point of data access and control. The database adapter also ensures proper quoting and escaping of the processed data.

9.3.6 Debugging

The HEM Application Framework also provides a debugging facility using a debugging library written by the author. The library supports dumps of varying complexity for every kind of PHP data structure. The debugger can be disabled using a central boolean variable. If a software defect is discovered the application can be debugged even if it is installed by enabling the debugging feature.

9.4 The HEM Application Programming Interface (API)

The application modules of HEM do not directly communicate with the database using SQL queries. A separate layer of abstraction is provided by an object-oriented Application Programming Interface (API). The API is separated into two parts: every class of the API inherits from a class representing a basic database object. The classes derived from the basic object have to define which table to use and of which columns the table consists. The database object provides the necessary methods for initialising, selecting, adding, updating, and deleting data. The classes which use the basic database object can implement further specialised functionality.

The API provides a high-level interface for developers to extend HEM. A developer need not care about details of the database structure unless the API should be extended. Details about the implementation of the API structure are discussed in Section 10.1. A very interesting opportunity to use the HEM API would be to create a layer for remote procedure calls. This would allow the construction of native client applications for operating systems. Chapter 12 discusses this possibility in more detail.

Chapter 10

Selected Details of the Implementation

This chapter describes several important details concerning the implementation of HEM. The application is written in PHP 4, while it can also be used with PHP version 5. The application comes with a flexible directory structure, which allows HEM to be deployed in various environments.

At the time of writing HEM contains 22.000 lines of PHP code. Although lines of code are no good measure of code quality it may give an idea of the complexity of HEM. This Chapter shows only a small fraction of noteworthy details of the implementation to provide an insight into the ideas used to implement HEM. The core parts of HEM are the application programming interface (API) described in Section 10.1 and the HEM application framework outlined in Section 10.2.

10.1 The HEM Application Programming Interface (API)

The different HEM application modules are independent of each other. Each application module could be run on its own. The modules communicate with a common database layer. This eliminates dependencies between different modules and highly improves the maintainability of the application code. HEM also provides a high-level API based on the database layer, which abstracts low level database interaction to high-level method calls.

10.1.1 Flexible Data Objects

The HEM API is based on a class, called `DBObject`, which provides basic functionality for all database transactions: *SELECT*, *INSERT*, *UPDATE*, and *DELETE*. The `DBObject` class has to work for every kind of database table, a derived class has to define which table to use, and which columns the table consists of. The `DBObject` class receives a reference to a database handle over the `DBObject` class constructor. The reference is provided by the HEM application framework, although other frameworks may also be used.

Retrieval of Database Entries

At the time the `DBObject` class is instantiated it receives the primary key of the entity it should represent. The `DBObject` creates the correct SQL query, sends it to the database handle, and receives a database result object from the database library. The database result is analysed and processed afterwards. Each attribute of the entity is then available for the derived class as a member variable and additionally as an associative array. Listing 10.1 shows the method that actually retrieves the data. The method is called by an initialisation method at object instantiation. The derived class has already set the columns

```

1 function getData() {
2     $fields = $this->getFieldList();
3     $fields_string = implode(',', $fields);
4
5     if(!is_null($this->id_)) {
6         $query = "SELECT $fields_string FROM $this->table_ WHERE $this
7             ->table_primary_key_ = '$this->id_'";
8
9         $result = $this->dbh->query($query);
10
11        if( (!$this->dbh->hasError()) && ($result->numRows() > 0) ) {
12            $row = $result->fetchRow();
13            foreach($fields as $f) {
14                $this->$f = htmlspecialchars($row->$f);
15                $this->data_array_[$f] = htmlspecialchars($row->$f); }
16
17            return $this->data_array_;}
18
19        elseif($this->dbh->hasError())
20            { /* Error handling */ }
21
22        else
23            { /* Error handling */ }
24    }

```

Listing 10.1: The DBOBJECT class translates high-level method calls to low-level SQL queries. The code example shown here retrieves data from a database table defined by a class derived from the DBOBJECT class.

corresponding to the database table. Line 2 retrieves the list of columns and line 3 converts it to the format that is used for the SQL query.

If the derived class has set the primary key of the desired entity, the query is performed in lines 5 and 6. There are cases where the derived class does not know the identity of the desired record at object instantiation time. In order to minimise the number of queries sent to the database connection no query is performed in such a case. The `getData` method can also be called later by the derived class. If the derived class already knows the record to retrieve, the query is performed in line 6. There is still a chance that the record does not exist or some database connection error occurred, so error handling is done in lines 18 to 23.

If the data existed in the database the DBOBJECT iterates over the table columns and sets the corresponding member variables and the associative array (lines 13 and 14). The method `htmlspecialchars` converts HTML tags to entities. This prevents the execution of malicious Javascript code entered by a user.

Insertion of Database Entries

The method to insert records into a database table constructs the query in the same way as the `getData` method retrieves data. The only difference regards the proper quoting of the submitted values. Content which is submitted to the method may contain characters or keywords which have special meaning in the syntax of the underlying database system. In order to prevent errors, the submitted values have to be

```

1 UPDATE heuristic_set SET hSetId = 'foo', hSetTitleId = 'bar',
  hSetDescriptionId = 'test' WHERE hSetId = 'io1cd6'

```

Listing 10.2: The structure of an SQL *UPDATE* query. Each field is followed by its new value.

```

1 function makeUpdateKeyValuePairs($fields = null, $data = null) {
2     $set_values = array();
3
4     if(($fields != null) && ($data != null)) {
5         while(list($k, $v) = each($fields)) {
6             if(!strcmp($v, 'text')) {
7                 if(isset($data[$k])) {
8                     $v = $this->dbh->quote($data[$k]);
9                     $set_values[] = "$k = $v"; } }
10            else {
11                if(isset($data[$k]) && !empty($data[$k])) {
12                    $set_values[] = "$k = $data[$k]"; }
13            }
14        }
15        return implode(', ', $set_values);
16    }
17    else return FALSE;
18 }

```

Listing 10.3: This method creates the field value pairs needed for an SQL *UPDATE* query. Textual data is properly quoted to prevent SQL injection vulnerabilities.

quoted and escaped. The escaping is performed by the database library. The quoting is handled by the `DBObject` class. Proper quoting is also very important to prevent SQL injection vulnerabilities, which were discussed in Section 8.2.

Updating a Database Record

Updating a database record is more complicated than insertion, because of the structure of the SQL statement. The `DBObject` class uses a helper method for this purpose, which is called `makeUpdateKeyValuePairs`. Each field of the query is directly followed by the new value. Listing 10.2 illustrates an *UPDATE* query. The `makeUpdateKeyValuePairs` method shown in Listing 10.3 creates these pairs using the list of fields and the submitted data. The method iterates over the field list and checks whether each field contains textual data, because textual data has to be quoted. The method returns a complete list of field value pairs.

Listing 10.4 shows the `updateData` method. Line 1 checks if data has been passed to the method. Line 4 retrieves the key value pairs from the helper method and line 6 creates the query which is passed to the database library in line 8. The rest of the method takes care of for proper error handling.

```

1 function updateData($data = null) {
2     if( ($data != null) ) {
3         $fields = $this->table_fields_;
4         $key_value_pairs = $this->makeUpdateKeyValuePairs($fields,
5             $data);
6
7         $query = "UPDATE $this->table_ SET $key_value_pairs WHERE
8             $this->table_primary_key_ = '". $data[$this->
9             table_primary_key_]. "'";
10
11        $result = $this->dbh_->query($query);
12
13        if($this->dbh_->hasError()) { /* Error Handling */ }
14        else
15            return TRUE; }
16        else { /* Error Handling */}
17    }

```

Listing 10.4: This method creates the SQL statement used to update a database record using a helper method to create the required key value pairs.

Benefits of the Flexible Data Object

A client class which inherits from the DBObject can use basic data retrieval and insertion functionality without using SQL statements. The data is exchanged exclusively using member variables and method calls. The DBObject class model provides a single point of access to the database layer which enables an easy way to extend functionality of the DBObject class and all derived classes. A single point of access also allows new security issues, that might not be known at the time of development, to be more easily addressed.

10.1.2 Example: The Heuristic Class

Every class in HEM which uses the database to store data inherits functionality from the DBObject class. The class that represents a heuristic has been chosen to illustrate how the basic functionality of the DBObject class can be used and extended.

Configuration

As already mentioned earlier in this section the class that derives from the DBObject has to provide proper configuration. Listing 10.5 shows how this is done. Line 3 defines which database table should be used. The array defined from line 5 to 11 declares the column names of the database table. The DBObject class also needs to know which field is used as primary key for the entity to be retrieved. The field which holds the primary key is defined in line 13. In lines 17 to 21 the member variables which will later contain the retrieved data are initialised to null. Starting in line 23 the constructor instantiates the translator class and calls the constructor of the DBObject class in line 34. Line 32 shows how the name of the database table can be overridden using a global configuration variable.

```

1 class Heuristic extends DBOBJECT {
2     var $dbh_ = null;
3     var $table_ = 'heuristic';
4     var $id_ = null;
5     var $table_fields_ = array(
6         'hId' => 'text',
7         'hTitleId' => 'text',
8         'hDescriptionId' => 'text',
9         'hSetId' => 'text',
10        'hOrder' => 'text',
11    );
12
13    var $table_primary_key_ = 'hId';
14    var $init_ok_ = FALSE;
15    var $data_array_ = array();
16
17    var $hId = null;
18    var $hTitleId = null;
19    var $hDescriptionId = null;
20    var $hSetId = null;
21    var $hOrder = null;
22
23    function Heuristic($heuristic_id, & $dbh) {
24        global $HEURISTIC_TABLE, $LANGUAGES;
25
26        if (isset($LANGUAGES) && is_array($LANGUAGES)) {
27            $this->languages_ = $LANGUAGES;
28            $this->translator_ = & new Translation(0, $dbh); }
29        else {
30            $this->error_ = 'Heuristic(): No languages defined, or not in
31                correct format'; }
32
33        $this->table_ = (isset($HEURISTIC_TABLE)) ? $HEURISTIC_TABLE :
34            $this->table_;
35
36        DBOBJECT::DBOBJECT($heuristic_id, $dbh);
37    }
38 }

```

Listing 10.5: The class which represents a heuristic extends the DBOBJECT class. The derived class must configure the database table fields, the primary key, and the table to be used.

```

1 function getHeuristic() {
2     if (!$this->init_ok_)
3         $this->init();
4
5     $return_array = array(
6         'hId' => $this->id_,
7         'title_translation' => array(
8             'trans_id' => $this->hTitleId,
9             ),
10        'description_translation' => array(
11            'trans_id' => $this->hDescriptionId,
12            ),
13        'hOrder' => $this->hOrder,
14        'hSetId' => $this->hSetId,
15    );
16
17    if (!$return_array['title_translation'] = $this->translator->
18        getTranslationArray($this->hTitleId))
19        $this->error_ = $this->translator->getError();
20    if (!$return_array['description_translation'] = $this->
21        translator->getTranslationArray($this->hDescriptionId))
22        $this->error_ = $this->translator->getError();
23
24    return $return_array;
25 }

```

Listing 10.6: This example shows how the functionality of the `DBObject` class can be used to create a high-level data structure containing a complete heuristic. The method does not contain any SQL code.

Usage

This section shows how a developer can obtain a high-level data structure which contains every attribute of a heuristic including a translation in all languages. The developer needs only to instantiate a heuristic object by passing the primary key of the corresponding heuristic and a database connection reference. A method call of the `getHeuristic` method returns the corresponding data structure. Listing 10.6 shows the implementation of the `getHeuristic` method. Since the method will definitely retrieve data, it checks if the object has been initialised correctly (lines 2 and 3), otherwise the initialisation method is called.

If everything has been properly set up, member variables are available which contain the corresponding heuristic data. In lines 5 to 15 these variables are assigned to the array that will later be returned. The last thing is to obtain the translations from the translator object. The translations are retrieved in lines 17 and 19. The array is now ready to be returned as an associative array in line 22. The resulting data structure is shown in Listing 10.7.

A Filter Extension

This example will show how the basic functionality of the `DBObject` can be extended. This example requires knowledge of SQL syntax. It would be beneficial to have a method which returns a list of identifiers which represent entities meeting certain criteria. For example, every heuristic record has a


```
1 Array
2 (
3   [hId] => 4ot0hthyub3fitrqzyeyyzlic5tjbne7
4   [title_translation] => Array
5     (
6       [US] => Error prevention
7       [DE] => Vermeidung von Fehlern
8       [trans_id] => 5jsnx8xbmjr690nfcyruliwlzrnajpgzl
9     )
10
11  [description_translation] => Array
12    (
13      [US] => Even better than good error messages is a
14        careful design which prevents a problem from
15        occurring in the first place.
16      [DE] => Besser als gute Fehlermeldungen ist ein
17        Konzept, das Fehler vermeidet.
18      [trans_id] => xeea3yzlvp076rj0rkqz18wmyhjau3ml
19    )
20  [hOrder] => 1
21  [hSetId] => io1cd6u3g7e6xp75hp6cpwvnd2ivvfvk
22 )
```

Listing 10.7: This example shows the data structure returned by the `getHeuristic` method. It contains the title and the description in both system languages.

```

1 $filter = array(
2   'hSetId' => 'io1cd6' ,
3   'hOrder' => '5' ,
4 );

```

Listing 10.8: This filter configuration will match every heuristic which belongs to a heuristic set with the number “io1cd6” and has an order of “5”. This is a *simple* filter configuration with no support for operators.

```

1 $filter = array(
2   'hSetId' => array('op' => '=', 'value' => 'io1cd6', 'cond' => '
   OR'),
3   'hOrder' => array('op' => '<', 'value' => '5', 'cond' => ''),
4 );

```

Listing 10.9: This filter configuration will match all heuristic identifiers which belong to the set with number “io1cd6” or which have an order lower than “5”. This is a *complex* filter configuration; it contains comparison and binary operators.

field which represents the set of heuristics it belongs to. In order to obtain the whole set a filter which returns all heuristic identifiers of a certain set is needed.

Listing 10.10 illustrates a method which takes an associative array containing a filter configuration and creates the corresponding SQL statement. The query is executed and a list of corresponding identifiers is returned. The filter array can use every column of the database to match entries. It is also possible to use binary and comparison operators within the filter configuration. The implementation is adapted from the PEAR::LiveUser library.

The filter can have two different styles: *simple*, without operator support, and *complex*, with support for binary and comparative operators. A simple filter configuration is shown in Listing 10.8. The filter will match every heuristic which belongs to the heuristic set number “io1cd6” and has an order of “5”. A complex filter configuration is shown in Listing 10.9. It will return all heuristic identifiers belonging to the set with number “io1cd6” or having an order lower than “5”.

The implementation of the `getHeuristicIds` filter method (Listing 10.10) has to convert the filter array into the corresponding *WHERE* clause. Line 5 decides if a filter has been passed. Between lines 7 and 14 the method iterates over all filter settings. For each filter entry it decides if it is the complex or simple configuration. In lines 9 and 10 the corresponding part of the *WHERE* clause is created. A *WHERE* clause consists of the variable name, the comparison operator, and the value. The last part is the boolean operation which is used to concatenate this part of the *WHERE* clause with the next one. The simple implementation in lines 12 and 13 uses “AND” as the boolean operation and “equals” as the comparison operator. Within a complex filter the operators which have been submitted through the filter configuration array are used. After the iteration has finished, the last boolean operator has to be cut off, because no further *WHERE* clause parts are following. The method can also be used to sort the returned identifiers by a custom table column. The sort direction is ascending by default. In line 23 the SQL query statement is created using the *WHERE* clause and the *ORDER* statement. The remaining lines of the method analyse the result and construct the return array of the method.

```

1  function getHeuristicIds($filters = array(), $order_column = null,
2     $order_direction = 'ASC') {
3     $where = '';
4     $order = '';
5     if (sizeof($filters) > 0) {
6         $where = ' WHERE';
7         foreach ($filters as $f => $v) {
8             if (is_array($v)) {
9                 $cond = ' ' . $v['cond'];
10                $where .= ' ' . $v['name'] . $v['op'] . $this->dbh->quote
11                    ($v['value']) . $cond; }
12            else {
13                $cond = ' AND';
14                $where .= " $f=" . $this->dbh->quote($v) . "" . $cond; }
15        }
16        $where = substr($where, 0, -(strlen($cond)));
17
18        if(!is_null($order_column)) {
19            $order = " ORDER BY $order_column ";
20            if($order_direction !== 'ASC')
21                $order.= 'DESC'; }
22
23        $query = "SELECT $this->table_primary_key_ FROM $this->table_
24            " . $where . $order;
25
26        $result = $this->dbh->query($query);
27
28        /* Analyse result and return ids */
29    }

```

Listing 10.10: The implementation of the method which analyses the submitted filter and builds a corresponding SQL statement out of the filter.

```

1 class sampleApp extends PHPApplication
2 {
3     function sampleApp($params) {
4         PHPApplication::PHPApplication($params); }
5
6     function run() {
7         $this->doSomething(); }
8
9     function doSeomthing() {
10        /* Application code goes here */ }
11 }

```

Listing 10.11: The *sampleApp* application extends the application framework. The application framework is instantiated within the constructor of *sampleApp*, which passes control to the framework. After the framework has been set up the `run` method is called to start *sampleApp*.

10.2 The HEM Application Framework

The HEM application framework is very flexible. A derived application can be used with or without authentication, database support, session support, or multilingual support. The framework constructor has numerous conditional statements to handle all the options. The concepts of the features are similar. The overall structure of the application and the framework is discussed in Section 9.2 and 9.3. This section describes only a small fraction of the whole functionality to give an insight into:

- How control is passed to the framework.
- How the framework initialises the database adapter.
- How automatic authentication is performed.

10.2.1 The Framework Mechanism

A software framework provides an inverted flow of control. An application using the framework uses its functionality by inheritance. The derived application has to implement a certain method, called `run` in HEM. The application passes control to the framework by calling the constructor of the super class. Every time an instance of the application constructor is created, the framework constructor is also executed because of the inheritance structure. The framework constructor takes care of proper setup of its functionality and finally calls the method that has been implemented by the derived application.

A simplified example of a derived application is shown in Listing 10.11. In line 1 the inheritance is performed by extending the super class named *PHPApplication*. In line 3 the constructor of the derived application is defined. The constructor has to call the constructor of the framework, because older versions of PHP do not automatically call the constructor of super classes. By calling the constructor of the super class, control is passed to the framework. The framework constructor performs the proper setup of all needed functionality and finally calls the `run` method to start the derived application.

A small example framework is shown in Listing 10.12. The framework provides database connectivity and automatic authentication using external libraries which are embedded using adapters. The configuration of the framework is performed using global variables and an array of parameters passed using the constructor. Line 4, 5, and 8 set the configuration as member variables to make them accessible for the

deriving application. The `setDefault` method represents a small piece of functionality provided by the framework.

Since configuration settings are optional, the framework cannot directly access a configuration variable, because this would yield a PHP error and terminate the application. Instead the framework uses the `setDefault` method shown in Listing 10.13. The method has two parameters: the first is the setting to examine, and the second is the default value to be used in a case where the variable does not exist. The method first checks if the setting is set before accessing it. If it is set the value is returned, and if not the passed default value is returned.

10.2.2 Database setup

After all parameters are set up correctly the framework constructor (Listing 10.12) configures the database handle using the database adapter in line 9. The `getHandle` method of the adapter returns a fully configured database connection that can be used by the derived application using the corresponding member variable, `$this->db_handle_`. Error handling has been skipped in the listing for clarity.

10.2.3 Automatic Authentication

The next task is to set up the authentication module and to check if a user is already authenticated. If no user is authenticated, or if the session has expired the user is directed to an application module displaying a login form. Line 15 of Listing 10.12 shows the application which provides the login form. Line 16 sets up the authentication adapter. In line 17 a method is called to retrieve the user identifier of the currently logged in user, if any. This user identifier is passed to the authentication handle, which checks if the user is logged in and if the session has not expired. If the authentication handle returns `FALSE` the user is redirected by the framework to an application which provides a form for authentication. The authentication application also uses the application framework. If the login was successful, the framework returns to the application module where the authentication error occurred.

After everything has been successfully set up, the framework constructor calls the `run` method implemented by the derived application. At that point the framework starts the application, which is now able to use a configured database connection and an authenticated user.

10.2.4 General Considerations

The framework actually provides much more functionality. The above description should clarify the mechanisms used within a framework. The HEM application framework additionally provides the following functionality:

Template engine: The framework takes care of the setup of the master template. It determines which stylesheet a logged in user has chosen, and configures the master template to use it. The content produced by each derived application can be generated using the template interface of the framework. This concept allows the embedding of a pluggable look and feel, for example to adapt HEM to a particular corporate design.

Multiple languages: If no user is logged in, the framework determines the language of the client's browser. After a user has authenticated the language is set corresponding to the user's setting. The framework provides a transparent language handling mechanism. A developer only has to provide translations of the language file. The correct language is delivered automatically by the framework.

PHP Annoyances: The HEM application framework also solves common problems which occur when developing for different versions of PHP:

```

1 function PHPApplication($params = null) {
2     global $DEFAULT_LANGUAGE;
3
4     $this->use_db_ = $this->setDefault($params[use_db], FALSE);
5     $this->auto_auth_ = $this->setDefault($params[auth], FALSE);
6
7     if ($this->use_db_) {
8         $this->dsn_=$this->setDefault($params[db_url], null);
9         $this->db_handle_ = DBAdapter::getHandle($this->dsn_);
10    } else {
11        $this->db_handle_ = FALSE;
12    }
13
14    if ($this->auto_auth_) {
15        $this->auth_app_ = $this->setDefault($params[auth_app], null);
16        $this->auth_handle_ = AuthAdapter::getHandle($this->db_handle_
17            );
18        $user_id = $this->getSessionUserId();
19        if (!$this->auth_handle_->isAuthenticated($user_id))
20            $this->redirectTo($this->auth_app_);
21    }
22    $this->run();
23 }

```

Listing 10.12: The constructor of an example software framework sets up a database connection and provides automatic authentication.

```

1 function setDefault($value, $default) {
2     return (isset($value)) ? $value : $default ;
3 }

```

Listing 10.13: A piece of functionality provided by the framework. A direct reference to a missing variable yields an error. The method shown here checks the existence of the variable prior to accessing it.

- Magic quotes is a feature which automatically quotes *POST* and *GET* variables, depending on the PHP configuration. This tends to interfere with quoting facilities of the application framework itself. The HEM application framework automatically disables the Magic Quotes feature of PHP.
- Character encoding may cause problems because of the transition from ISO (International Organisation for Standardisation) encodings to UTF-8 encoding. ISO used a different encoding of characters for every language, while UTF uses a common encoding scheme for all languages. The PHP engine, the web server, and the database server could be configured differently on the target system. The application framework addresses this problem by determining and converting the contents to the appropriate character encodings.

Chapter 11

HEM in Use for the TUGraz Project

The Heuristic Evaluation Manager was used to manage a heuristic evaluation of the web site of Graz University of Technology¹. The evaluation was performed in a group of four evaluators. The step of summarising the discovered findings was conducted as a group activity.

11.1 The TUGraz Project Heuristics

The evaluation was conducted using the list of 10 Nielsen heuristics annotated by Keith Instone, presented in Section 5.5. Each heuristic has a comment concerning how the heuristic applies to a web site. Since the evaluation subject was a web site these comments supported the evaluators in discovering usability problems. Figure 11.1 shows the HEM Heuristic Set Manager with Instone's heuristics.

11.2 The TUGraz Project

The HEM Project Manager is used for configuring and running a HEM project. Figure 11.2 shows the HEM Project Manager for the TUGraz project. The upper part is for the title and the description of the project, which should be added in both system languages, English, and German. The next part is for assigning users to the project. The next part of the form is for choosing a set of heuristics, an environment, and a rating scheme. The rating scheme, and the environment is already in HEM, the set of heuristics was newly created in Section 11.1.

11.3 The TUGraz Project Evaluation Environment

In order to describe the equipment used by the evaluators to conduct their evaluation, the evaluators were asked to complete a form consisting of the following attributes within HEM:

Age: The age of the evaluator at the time of evaluation provides feedback of any special demographic attributes that might influence the kind of findings discovered by the evaluator.

Sex: The sex of the evaluator might also influence the kind of findings discovered by the evaluator.

Web Browser: The web browser the evaluator used for the evaluation. Some usability problems only occur in certain browsers.

¹<http://www.tugraz.at>

The screenshot shows a web browser window titled 'Change Heuristic Set'. The URL is <http://martin-loitzis-ibook-g4.local/mar>. The user is logged in as 'John Doe'. The navigation menu includes Home, Projects, Evaluation, Settings, and Reports. The breadcrumb trail is >Home >Heuristic Sets >Help and documentation >Change Heuristic Set.

Change Heuristic Set

Title of Heuristic Set:

English	German
Instone: Site Usability Heuristics	Instone: Web Usability Heuristiken
Jakob Nielsen's 10 usability heuristics with his description in each first paragraph and Instone's Web-specific comment following.	10 Heuristiken von Jakob Nielsen, darunter jeweils seine Beschreibung im ersten Absatz und Instones Kommentare darunter

Heuristics:

Order	English	German	
1	Visibility of system status	Rückmeldung des Systemzustandes (Feedback)	delete
	The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.	Das System sollte dem Benutzer immer (zum richtigen Zeitpunkt) Rückmeldung geben, womit es sich gerade beschäftigt.	
	Any user information should be easy to search, focused on the user's task, list	über miteisens Notwendig.	

0 more heuristic entries

Save Reset

Figure 11.1: The TUGraz project was conducted using Keith Instone's commented list of Nielsen heuristics. The content should be entered in both system languages, English and German.

The screenshot shows a web browser window titled "Change Project" with the URL <http://martin-loitzls-ibook>. The page header features the HEM logo and the user "John Doe" logged in, with a "Logout" link. A navigation menu includes "Home", "Projects", "Evaluation", "Settings", and "Reports". The breadcrumb trail is ">Home >Projects >Evaluation of www.tugraz.at >Change Project".

Change Project

Title of Project	
English:	German:
Evaluation of www.tugraz.at	Evaluierung von www.tugraz.at

Description of Project	
English:	German:
The evaluation of www.tugraz.at using the HEM web application.	Die Evaluierung von www.tugraz.at mit HEM.

Users in Project	Available Users
Joe Zehenter Martin Loitzl Harald Auer Alois Dengg	Manfred Löpschitz Claudia Kresser Hans Wurst John Doe

Heuristics to use	Instone: Site Usability Heuristics
Environment	Website Evaluation
Ratingscheme	Severity
Project Phase	Merging
<input type="button" value="Save"/> <input type="button" value="Reset"/>	

Version: 0.4.1 | [HEM Homepage](#) | Copyright © 2005 Andrews, Loitzl, Romano

Figure 11.2: The HEM Project Manager is used for configuring and running a HEM project. The title and description should be added in both system languages.

Operating System: The operating system used for the evaluation. The operating system may also affect the kind of usability problems discovered.

Connection: The bandwidth of the Internet connection used by the evaluators. This might give an explanation for findings regarding to the speed of the web site.

Monitor Colours: How many different colours the monitor can display. This helps to explain certain usability problems. For example, systems with a redirected graphical user interface like Windows Terminal Service Clients or Unix X Windows system might use a small number of different colours to reduce bandwidth requirements.

Monitor Resolution: The number of pixels the evaluator's monitor is able to display.

Monitor Size: The size of the evaluator's monitor, typically in inches.

Date of Evaluation, Time of Evaluation: Discovered usability problems might not be reconstructible after the evaluation. The date and time of the evaluation session might help in determining the cause of the finding.

Evaluators complete the environment form using the HEM Environment Collector shown in Figure 11.3.

11.4 The TUGraz Project Rating Scheme

In order to prioritise the discovered usability problems the evaluators were asked to prioritise the discovered findings using a five point rating scale. The rating scheme consisted only of severity. The scale is shown in Figure 11.4. The first column shows the numerical values of severity, the second and the third column show the captions presented to the evaluators in English and German.

11.5 Evaluation Procedure

The evaluation was conducted in four consecutive steps. Each step is discussed below. Four evaluators participated in the evaluation. One also had the role of the manager.

Briefing

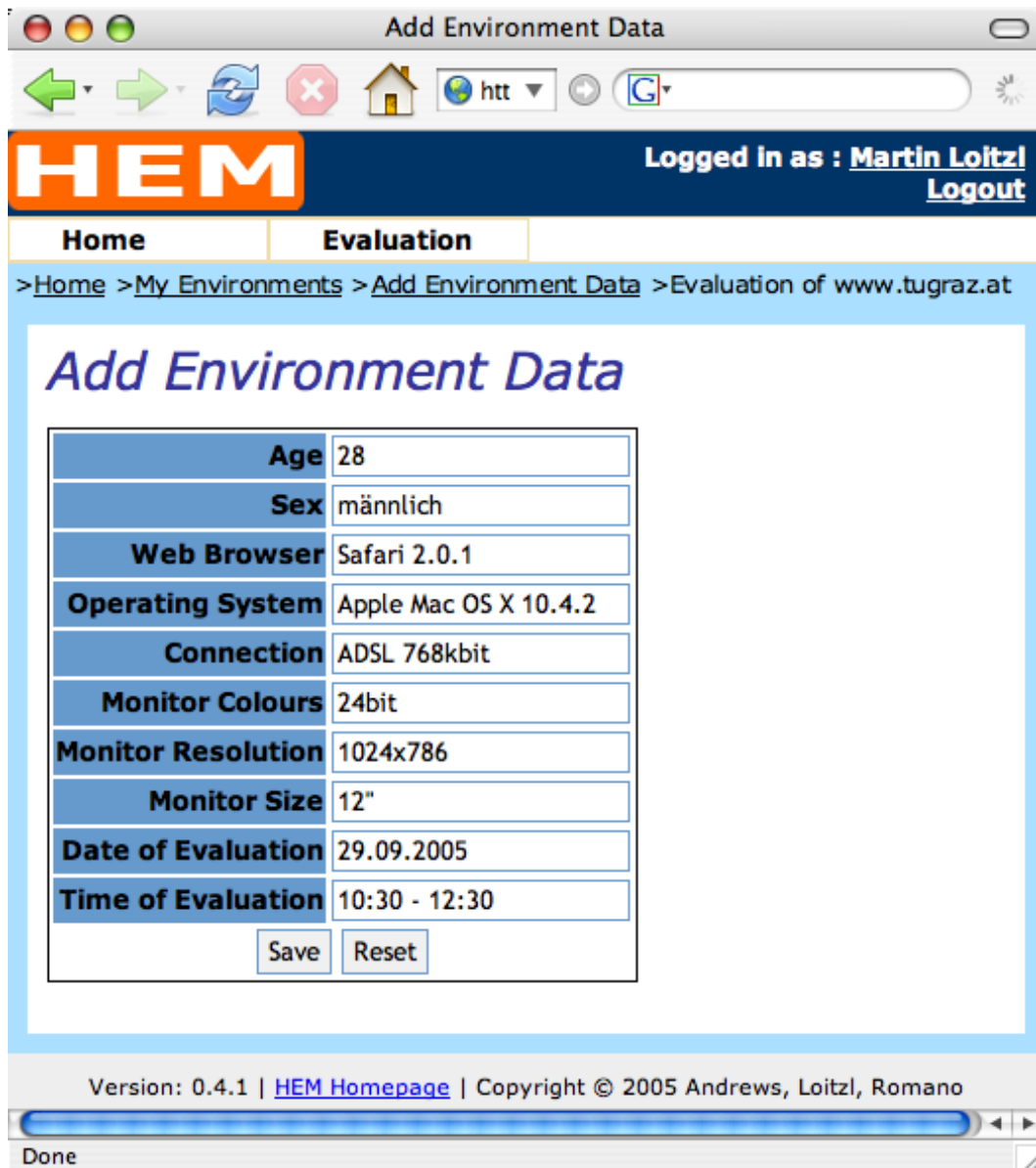
This step was conducted as a meeting which lasted about one and a half hours. The evaluators had some experience of heuristic evaluation from a practical exercise during a course on human-computer interaction. The evaluators received a short introduction to heuristic evaluation in order to refresh their knowledge about the methodology. The set of heuristics and its special relevance for the evaluation of web sites was also discussed.

Evaluation

Each evaluator evaluated the web site within a two-hours session individually. The evaluators were also asked to make notes about the HEM user interface.

Merging

The process of summarising the individual lists of findings was conducted as a group activity. The manager used a video projector to project the merging screen of HEM to enable the evaluators to follow



The screenshot shows a web browser window titled "Add Environment Data". The browser's address bar shows "http://www.tugraz.at". The page header includes the HEM logo and the text "Logged in as : Martin Loitzl" with a "Logout" link. Below the header are navigation tabs for "Home" and "Evaluation". The breadcrumb trail reads: ">Home >My Environments >Add Environment Data >Evaluation of www.tugraz.at".

Add Environment Data

Age	28
Sex	männlich
Web Browser	Safari 2.0.1
Operating System	Apple Mac OS X 10.4.2
Connection	ADSL 768kbit
Monitor Colours	24bit
Monitor Resolution	1024x786
Monitor Size	12"
Date of Evaluation	29.09.2005
Time of Evaluation	10:30 - 12:30

Save Reset

Version: 0.4.1 | [HEM Homepage](#) | Copyright © 2005 Andrews, Loitzl, Romano

Done

Figure 11.3: Evaluators complete the environment form using the HEM Environment Collector.

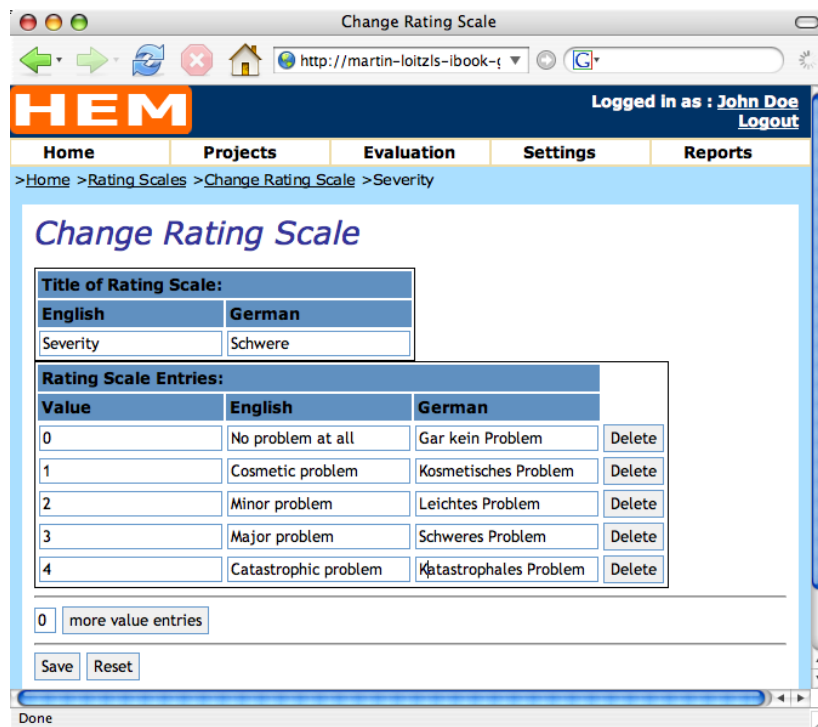


Figure 11.4: The rating scheme used for the TUGraz project consists of a single five point severity rating scale.

the progress of the merging process. The session lasted about 2 hours. The manager took the longest list of individual findings and proceeded through that list. The evaluators could easily remember if they discovered same problems because of the short time between the evaluation and the merging of problems. The generation of the merged summarised list thus proceeded very quickly. Conducting the merging process as a group activity also supported discussion about several usability problems and possible solutions. In some cases the manager had to ask the evaluators to focus on the merging process, because discussion was digressing.

Rating

The ratings were then submitted by each evaluator individually, which took about half an hour for each evaluator.

11.6 Discovered Results

The evaluation resulted in 31 findings, six positive impressions and 25 usability problems. Figure 11.5 shows the number of usability problems discovered by each evaluator individually. The number of all problems discovered is represented by the thick line at the value of 25. On average each evaluator individually discovered 38 percent. The total number of usability problems in a web site can not be determined with any degree of confidence.

The ratings of the evaluators prioritised the findings by severity. The result can be split into three main classes of usability problems: major, minor, and cosmetic. Major problems have an average rating of 2.51 and more, minor problems have an average rating between 1.51 and 2.5, and cosmetic problems

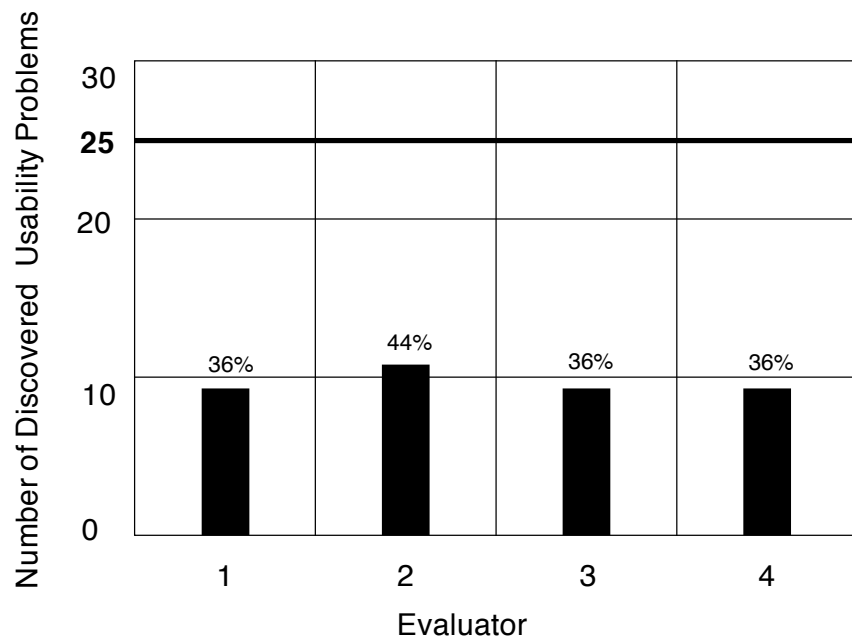


Figure 11.5: Each evaluator discovered an average of 38 percent of all 25 usability problems found on the *www.tugraz.at* web site.

have an average rating below 1.51. Using this classification the evaluation discovered 10 major problems, 12 minor problems, and 3 cosmetic problems. The final report is shown in Appendix D.

This differs slightly from the theory discussed in Section 4.4 that heuristic evaluation tends to discover major usability problems, but it can be assumed that one single evaluation with 25 findings is not going to be representative.

11.7 Feedback from Users of HEM

The users of HEM found the interface very simple and easy to understand. One evaluator had problems in connection with the browser (Opera) used for the evaluation. HEM had not been tested with this browser before. One evaluator mentioned the possibility of arranging the findings as positive. Three evaluators used the German user interface and one, the manager, used the English version. The manager complained about the long time of loading the merging interface if screenshots are displayed. A possible solution for that problem is discussed in Section 12.4.

Chapter 12

Outlook and Future Work

HEM provides an open architecture for future extensions. This chapter describes some possible features which could be useful enhancements for HEM.

12.1 Generation of Statistics

In Nielsen [1994b] several statistical properties of heuristic evaluations are discussed. One example deals with individual evaluator performance. The data collected during the process of heuristic evaluation already allows the generation of statistics about the performance of individual evaluators. The total number of discovered findings can be gathered from the number of summarised findings. The number of problems discovered by individual evaluators is also available through the individual finding lists. The ratio of the number of each evaluator's findings to the number of summarised findings is a value which can be interpreted as evaluator performance.

Another statistical approach would be to generate cost-benefit statistics. Information about the benefit of fixing usability problems discovered during the heuristic evaluation would have to be collected. The effort of conducting an evaluation has to be split into fixed and running costs. The fixed costs are primarily defined by running HEM. The running costs depend on the individual evaluator's hourly rate. Since HEM is an online environment, it would be easy to measure the individual evaluator's time for an evaluation. This amount of time could also be used as a measure of effort. The ratio between effort and benefit of a heuristic evaluation could then be visualised as a cost-benefit chart.

12.2 Advanced Report Editing

Currently, only a fairly simple report generator is available. This generator automatically generates a comprehensive list of findings, including the evaluators who discovered them, and their severity rankings. It also lists the individual findings of each evaluator. The report generator scales the submitted screenshots to a size that is appropriate for a printed document. In order to assist the evaluation manager in discussing the most relevant findings and positive impressions, the report generator also creates corresponding chapters by estimating the content as far as possible.

Nevertheless, the facilitator has to take the report out of the system and edit it using an editor or word processor to finish it. An environment for editing reports within HEM would be desirable.

12.3 XML-RPC or SOAP Interface

XML-RPC (eXtensible Markup Language - Remote Procedure Calls) is a technique which performs remote procedure calls. It uses XML markup for the transmission of data. It was originally developed by Dave Winer of UserLand Software in cooperation with Microsoft. SOAP (Simple Object Access Protocol) is based on XML-RPC and has been enhanced by Microsoft, UserLand, and DevelopMentor. SOAP has been passed to a working group of the World Wide Web Consortium [W3C, 2003] who now maintain it as a standard. These techniques allow the definition of open and flexible interfaces for distributed computing.

Although web applications are very usable nowadays, a native client application on a real operating system provides more possibilities for application development and interface design.

Various libraries exist for XML-RPC and SOAP client applications for languages such as C and C++, Perl, Java, and many more. If HEM had such an interface, native client applications could be written in any programming language where client libraries are available. An XML-RPC or SOAP interface could be built upon the existing HEM API.

12.4 Increase the Speed of the Finding Merger

The Finding Merger has the problem that it is very slow with large numbers of findings. A new technique for building flexible web interfaces became available through the AJAX (Asynchronous JavaScript And XML) methodology [Garrett, 2005]. The techniques used within AJAX are not new. The new idea is to transmit parts of a web page asynchronously. Usually a web page has to be reloaded completely after a user has pressed a button. Using AJAX only the necessary information has to be transmitted, without requesting the whole page. This concept can greatly increase the speed and degree of interactivity of a web application. Rewriting the HEM Finding Merger using the AJAX concept would greatly enhance the speed of the user interface.

12.5 New Authentication Mechanisms

PEAR::LiveUser supports other authentication mechanisms in addition to storing username and password in a database. It may be beneficial in intranet environments to use existing authentication systems. For example, PEAR::LiveUser can be configured to use various e-mail systems, Microsoft Windows domain controllers, and many other systems for authentication.

12.6 Evaluating Various Kinds of Interfaces

HEM provides a very flexible environment for conducting heuristic evaluations. It reduces the workload of the evaluation manager, so that new ideas can be tested with minimal effort. The following features of HEM support the evaluation of many kinds of interfaces, not just web sites or software products:

- The set of heuristics can be of any kind.
- The evaluator is not bound to submit screenshots of a computer screen. It would be possible to submit photos from a digital camera, for example.
- Severity ranking for discovered usability problems can also be freely combined using several factors (rating scales).

These features allow the evaluation a broad variety of interfaces, for example hand-held devices, consumer electronic devices, toys, or even architecture, using the infrastructure provided by HEM.

Chapter 13

Concluding Remarks

This thesis described the Heuristic Evaluation Manager (HEM) a web-based application which supports experts in conducting heuristic evaluation.

Chapters 2 to 8 embedded this work into the research context. In Chapter 2 the term of usability engineering was introduced and the place of usability inspections in the usability engineering lifecycle was illustrated. Chapter 3 discussed common usability inspection methods, the class of methods to which heuristic evaluation belongs. Chapter 4 described heuristic evaluation in more detail and outlined the work flow, benefits, and draw backs of the method. Chapter 5 presented different sets of heuristics which are used for heuristic evaluation. Chapter 6 described characteristics of computer supported heuristic evaluation tools and discussed two projects in particular. Chapter 7 gave an overview of pattern-oriented software development, a technique used for the implementation of HEM. Chapter 8 discussed web application development with PHP, a commonly used programming language for implementing web applications.

Chapters 9 to 12 presented the work done within the practical part of this thesis. Chapter 9 provided an introduction to the Heuristic Evaluation Manager, its application architecture and its components, and how it supports the process of a heuristic evaluation. Chapter 10 described selected details of the implementation. In Chapter 11 an example of using HEM was presented.

This thesis concluded with Chapter 12, which discussed possible future extensions of HEM and usage scenarios which may be useful for usability engineering research.

Appendix A

HEM Manager Guide

The appearance of the HEM user interface depends on the role of the currently logged in user. There are three roles available: evaluators, managers, and administrators. The interfaces of the manager and the administrator do not differ. An administrator or manager can also participate in an evaluation, thus all parts the evaluator's interface are also available for administrators or managers. A user guide for the HEM evaluator interface is presented in Appendix B.

A heuristic evaluation is conducted in three steps:

Evaluation: The evaluators conduct their evaluation by entering discovered usability problems. The evaluator can also add screenshots to these findings.

Merging: The manager of the evaluation summarises findings using the individual finding lists of the evaluators and creates a single merged list of findings.

Rating: The summarised list generated by the manager is rated by the evaluators.

Section A.1 provides a general overview of the HEM window. An evaluation manager can manage every project, and can also be part of an evaluation. With HEM every aspect of a heuristic evaluation is freely configurable. Custom sets of heuristics can be entered using the Heuristic Set Manager (Section A.2), the ten usability heuristics by Nielsen [1993] are already provided by HEM. Rating schemes can be defined by combining several rating scales (Section A.3). Rating scales can be created using the Rating Scale Manager (Section A.4). The severity rating scheme is already provided HEM. Depending on the interface to be evaluated, forms for describing evaluation environments can be created (Section A.5). Typically web sites or software products are evaluated, but it is also possible to define environments for other interfaces or objects, such as hand-held devices or home entertainment gadgets.

The manager of an evaluation has to summarise the individual lists of evaluator findings. This task is performed using the Finding Merger described in Section A.6.

A.1 The HEM Window

As illustrated in Figure A.1 the HEM window is divided into five areas:

Title Bar: The topmost element of the HEM interface is the title bar. On the left side there is the HEM logo, the right side shows the name of the currently logged in user and a logout link. The user's attributes can be changed by clicking on the user's name.

Menu Bar: Below the title bar is the menu bar. The menu bar holds a drop-down menu. Figure B.1 shows the home application with the evaluation menu expanded.

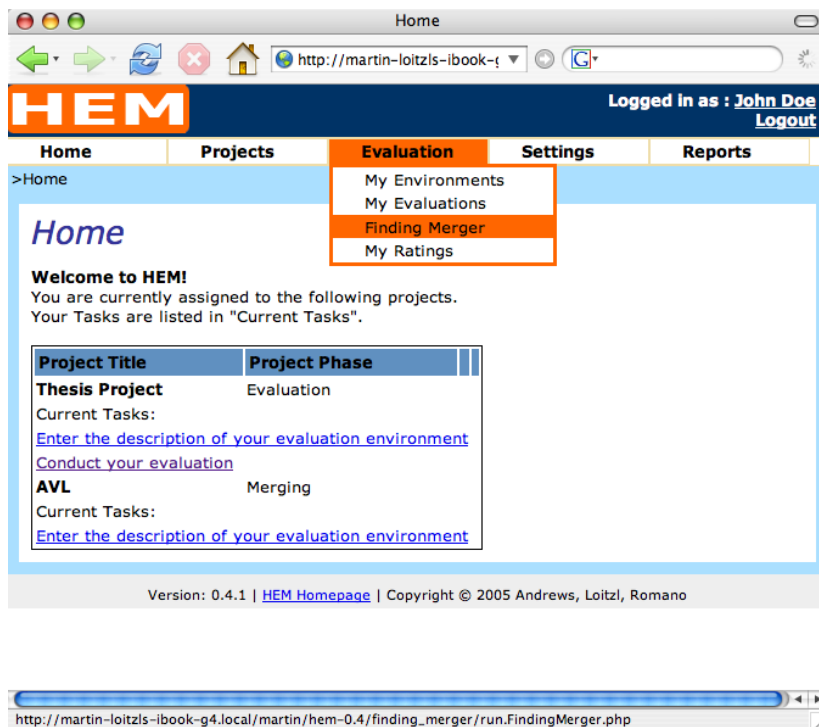


Figure A.1: This image shows the Home application of HEM with the evaluation menu expanded. The Home application contains the tasks that currently have to performed by the user.

Breadcrumb Navigation: Below the menu bar breadcrumb navigation is shown. It displays the current location of the user. If an item has a function, it is displayed as hyperlink leading to the corresponding part of the application.

Main Application Area: The content of the main area depends on the part of the application currently in use.

Footer: The footer has no active content. It contains a link to the HEM home page, a copyright notice, and the HEM version number.

HEM uses a common set of icons. Each icon has tool tip labels which describe the function of the icon. The HEM icon set is listed in Table A.1.

A.2 Heuristic Sets

An overview of existing sets of heuristics can be obtained by choosing the *Heuristic Sets* item in the *Settings* menu. To add a new set the *Add Heuristic Set* link below the overview has to be clicked. To change an existing set, the Edit button is used.

Figure A.2 shows the form to change a set of heuristics. At the top of the form a title and a description can be entered in all system languages, which currently are German and English. By entering a number in the form field below the list of heuristics and clicking the more heuristic entries button a corresponding number of new rows can be added. Each row represents a heuristic. The first field is a number for ordering, the other fields are for the title and the description of the heuristic. A heuristic can be removed by clicking the Delete button after each row.










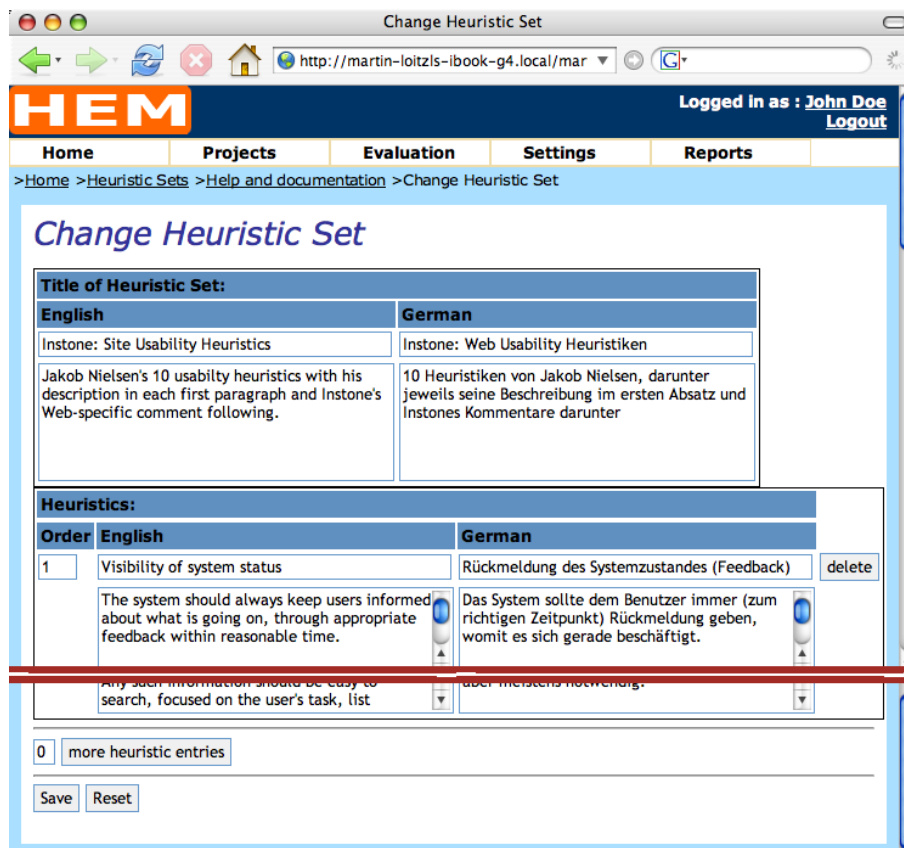
Label	Icon	Meaning
Edit		Edit the corresponding item in a list.
Delete		Delete the corresponding item in a list.
Up		Move the corresponding list item up one item.
Down		Move the corresponding list item down one item.
Append		Append the corresponding list item to another list.
Detach		Detach the corresponding list item from a list.
Use Text		Use the text of the corresponding list item in a form field.
Insert		Insert a list item at the specified position.
Export		Export the corresponding list item.

Table A.1: The HEM user interface uses this set of icons throughout the whole application.



Change Heuristic Set

Logged in as : John Doe
Logout

Home Projects Evaluation Settings Reports

>Home >Heuristic Sets >Help and documentation >Change Heuristic Set

Change Heuristic Set

Title of Heuristic Set:

English	German
Instone: Site Usability Heuristics	Instone: Web Usability Heuristiken
Jakob Nielsen's 10 usability heuristics with his description in each first paragraph and Instone's Web-specific comment following.	10 Heuristiken von Jakob Nielsen, darunter jeweils seine Beschreibung im ersten Absatz und Instones Kommentare darunter

Heuristics:

Order	English	German	
1	Visibility of system status The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.	Rückmeldung des Systemzustandes (Feedback) Das System sollte dem Benutzer immer (zum richtigen Zeitpunkt) Rückmeldung geben, womit es sich gerade beschäftigt.	delete
	Any such information should be easy to search, focused on the user's task, list	über meistens notwendig.	

0 more heuristic entries

Save Reset

Figure A.2: The upper part of the form contains the title and a description of this set of heuristics. The lower part contains a title and description. All data should be entered in all languages, since the list is shown in the user's language.

Figure A.3: The form to create a rating scheme. At the top, a title can be entered. The list below the title contains all rating scales already added to the current scheme. All available scales are shown in the list denoted by *Available Rating Scales*. The last drop-down field is used to define how the resulting priority of the rating scheme is calculated.

A.3 Rating Schemes

Using the Rating Scheme Manager an arbitrary number of rating scales can be combined into a rating scheme. There is already one scheme entered in HEM, a scheme for severity, which consists only of one scale, also called severity. It is also possible to combine two or more scales. For example, a severity scale and a frequency scale could be combined to a scale called *criticality* [Andrews, 2006]. A new rating scheme can be added by clicking the *Add Rating Scheme* link below the list of existing rating schemes.

Figure A.3 shows the form to create a rating scheme. At the top of the form the title of the rating scheme can be entered in all system languages. Below is the list of rating scales already associated with the current rating scheme. A rating scale can be removed from the scheme by clicking the Remove button. All available rating scales are listed below the *Associated Rating Scales* list and can be added by clicking the Add button.

The last form field is a drop-down list, where the method for result calculation can be selected. For example, there are two scales, severity and frequency, associated to a scheme called criticality. For each scale the mean value of all evaluators is calculated. If “addition” is set as the result operation for the criticality rating scheme, then the mean value of each rating scale is summed up to the final value. “Multiplication” and “mean” are also available as result operations. Multiplication could be used for schemes having an increasing (e.g. 1–10) and decreasing (e.g. 0–1) scale components.

Title of Rating Scale:	
English	German
Frequency	Häufigkeit

Rating Scale Entries:			
Value	English	German	
0	Very seldom (<1%)	sehr selten (<1%)	Delete
1	Seldom 1-10%	selten 1-10%	Delete
2	Frequently (11-50%)	häufig (11-50%)	Delete
3	Oftentimes (51-89%)	sehr häufig (51-89%)	Delete
4	Almost always (>90%)	fast immer (>90%)	Delete

0

Figure A.4: A rating scale consists of a title and several values. Each value has a numerical value and a caption in each available language. The caption is presented to the evaluators in a drop-down field. Values can be added and deleted by using the corresponding buttons.

A.4 Rating Scales

With the Rating Scale Manager several rating scales can be entered. Rating scales are used in the Rating Scheme Manager. An overview of existing scales can be obtained by choosing the *Rating Scales* item in the *Settings* menu. A new rating scale can be created by clicking the *Add Rating Scale* link below the list.

Figure A.4 shows the form to edit a rating scale. At the top of the form a title can be entered in all system languages. To add more value fields, a number has to be entered in the field below the list and the more value entries button has to be clicked. A value can be deleted by using the Delete button at the end of each value line.

A.5 Evaluation Environments

Different kinds of evaluation equipment is used depending on the interface to be evaluated. Different forms for entering descriptions of evaluation environments can be created using the Environment Manager. An evaluator uses different equipment for example when evaluating a web site than when evaluating a video recorder.

All available evaluation environment forms can be listed by choosing the *Environments* item in the *Settings* menu. To add a new environment the *Add Environment* link below the list has to be clicked. At the top of the form a title and description for the environment can be entered in all system languages. In order to add form fields a number has to be entered into the field at the lower end of the screen and the more environment entries button has to be clicked. To delete fields the Delete button at the end of each line can be used.

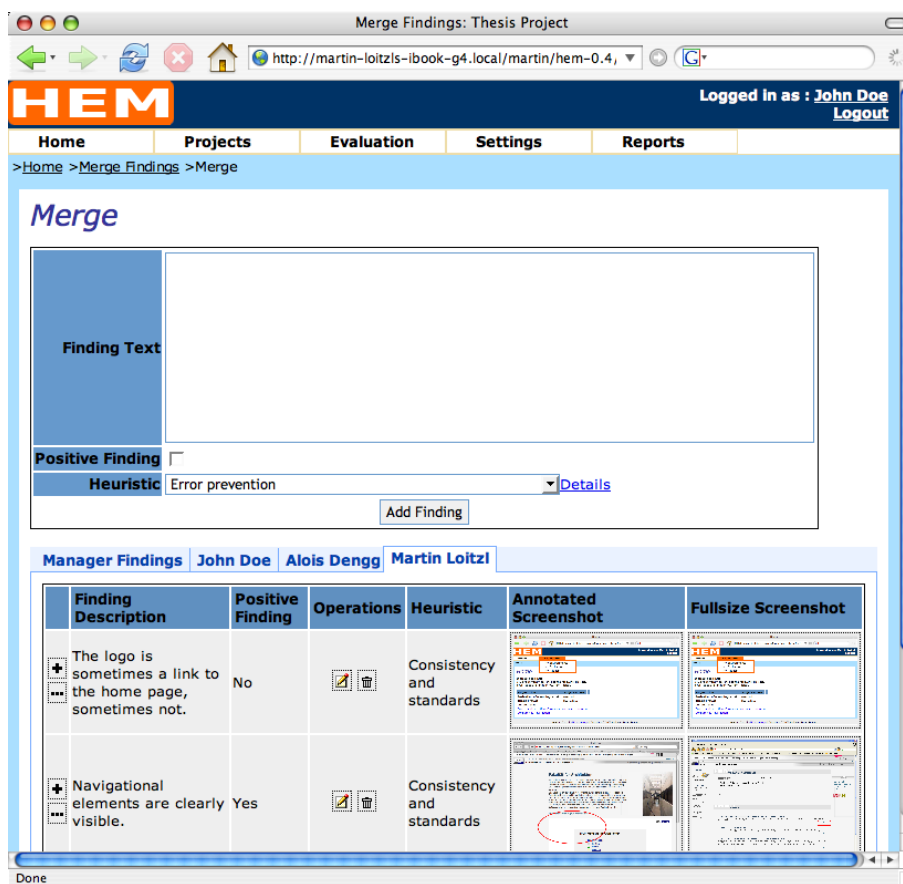


Figure A.5: The Finding Merger with an evaluator's tab selected.

A.6 Finding Merger

The Finding Merger allows the manager to merge the individual finding lists of the evaluators into one list of findings which the evaluators later rate. Starting with the evaluator who submitted the longest list, the manager looks for each finding if another evaluator discovered a similar problem. The manager then creates a “manager finding” and associates all matching “evaluator findings”.

An overview of projects in the “Merging” phase can be obtained by choosing the *Finding Merger* item in the *Evaluation* menu. If there are many screenshots or the manager has a low-bandwidth Internet connection, the manager can choose to merge without screenshots displayed by unchecking the “Merge with screenshots displayed” check box below the project overview list in the Finding Merger application.

Figure A.5 shows the “Merge View” of the Finding Merger. It is divided into two parts. The upper part contains the editing part, where the text of the manager's finding can be entered, and the finding can be marked as a positive impression. If heuristics are used in the project the corresponding heuristic can be selected from a drop-down list. If the manager wants to see a detailed description of the heuristics the *Details* link can be used.

The lower part shows the lists of findings as tabs. The first tab always contains the aggregated list of findings created by the manager. The remaining tabs show the evaluators' individual findings, one tab for each evaluator.

To create a new manager finding the insert button between the findings in the “Manager Findings”

Finding Description	Positive Finding	Operations
Home: Unklar was sind Links, was ist Werbung wie kommt man zu weiteren Seiten	No	[+]
Home: Login Button nicht auf der Homeseite, aber auf Folgeseiten	No	[+]

(a) One or both of the findings from the evaluator can be assigned to the manager finding currently being edited by clicking the Add button denoted by a plus sign. The description of the evaluator can be reused by clicking the Use Text button denoted by three dots.

(b) The first finding of the evaluator has been assigned to the manager finding. The Add button changed to a Remove button and the background colour turned red to reflect the action of the manager. The finding can be removed from the manager finding by clicking the remove button.

Figure A.6: The manager of an evaluation summarises a finding by relating all corresponding findings from the evaluators. The descriptions of the evaluators' findings can be changed using the Edit button. The Delete button can be used to delete an evaluator's finding.

tab can be used. To associate an evaluator finding to the manager finding the Append button which is shown at the beginning of every evaluator finding line is used. When an evaluator finding is associated the background colour changes to red, and the Append button is replaced by a Detach button to reflect the action of the manager (Figure A.6). The Remove button can be used to remove an evaluator finding from the manager finding. The text of an evaluator finding can be reused by clicking the Use Text button below the Append button to copy the text into the text field of the editing part.

In the evaluator tabs already associated findings are marked with a different background colour. Findings which are not already merged have a brighter background colour. This helps to follow the progress of the merging process.

To reorder the manager findings the up and down arrows in the Operations column can be used. To delete or edit a manager finding the corresponding buttons in the Operations column can be used. The evaluator findings can be changed by the manager by using the Edit or the Delete button.

In the best case each evaluator has submitted both a full size and an annotated screenshot. The manager can choose the best screenshot for the report by clicking the "choose screenshot" link in the screenshot column in the "Aggregated Findings" tab. In the appearing window, each evaluator is listed and the submitted screenshots are shown; by clicking the radio buttons a screenshot can be selected. If the manager wants to choose other screenshots for a finding, the screenshots can be deleted by editing the finding and clicking the "Detach Screenshots" check box in the edit part of the merge view.

A.7 User Manager

The User Manager is also available to evaluators, but with a simplified user interface. Evaluators can only change their own attributes. A manager or administrator can additionally choose one of three groups for the user: evaluators, managers and administrator. A manager or administrator can also enter a comment

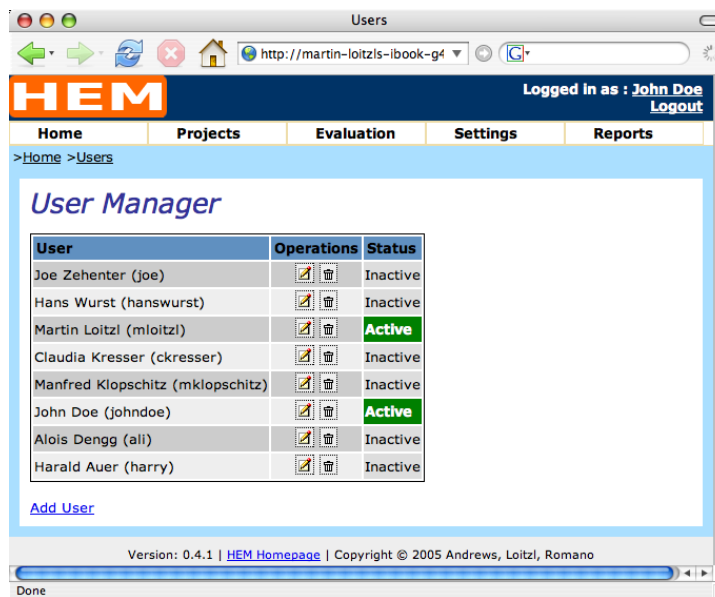


Figure A.7: The User Manager shows an overview list of users available in a HEM installation. A user account can be changed and deleted by using the corresponding buttons. The last column of the table shows the current state of the user, active or inactive.

which is only visible for managers or administrators.

An overview of users can be obtained by clicking the *Users* item in the *Settings* menu (see Figure A.7). In each line the user's first and last name, the user name, an Edit and a Delete button, and the current state of the user is shown. A user can have two states: active or inactive. If a user account is not active, the user cannot login. When a user is deleted, only the user attributes are deleted. All data collected in heuristic evaluations are kept. A new user account can be created by clicking the *Add User* link below the user account list.

A.8 Project Manager

With the Project Manager new projects can be created and existing projects can be changed. Ongoing projects can be exported as a single file, including all users and settings belonging to the project. An overview of all available projects can be obtained by clicking the *Overview* item in the *Projects* menu.

A project's settings can be changed by clicking on the Edit button after the projects title in the overview. The form is shown in Figure A.8. The first part of the form holds the project's title and description in all system languages. Below are two select boxes. The first shows all users already assigned to the project, the second shows available users. A user can be removed or assigned to the project by selecting the user's name and clicking the right or left arrow.

The next part is for choosing a set of heuristics. Evaluations can be conducted with or without heuristics. Sets of heuristics can be managed using the Heuristic Set Manager described in Section A.2.

Next, one of the available evaluation environment description forms has to be selected. Environments can be managed by using the Environments Manager described in Section A.5.

The next part provides the possibility for choosing the rating scheme for the project. Rating schemes consist of one or more rating scales which can be managed with the Rating Scale Manager described in Section A.4. A rating scheme can be assembled with the Rating Scheme Manager described in Section

Change Project

Logged in as : **John Doe**
[Logout](#)

Home **Projects** **Evaluation** **Settings** **Reports**

>Home >Projects >Evaluation of www.tugraz.at >Change Project

Change Project

Title of Project	
English:	German:
Evaluation of www.tugraz.at	Evaluierung von www.tugraz.at

Description of Project	
English:	German:
The evaluation of www.tugraz.at using the HEM web application.	Die Evaluierung von www.tugraz.at mit HEM.

Users in Project	Available Users
Joe Zehenter Martin Loitzl Harald Auer Alois Dengg	Manfred Löpschitz Claudia Kresser Hans Wurst John Doe

< >

Heuristics to use	Instone: Site Usability Heuristics
Environment	Website Evaluation
Ratingscheme	Severity
Project Phase	Merging
<input type="button" value="Save"/> <input type="button" value="Reset"/>	

Version: 0.4.1 | [HEM Homepage](#) | Copyright © 2005 Andrews, Loitzl, Romano

Done

Figure A.8: The form to change a project's settings.

A.3.

The last item contains the project's phase. A HEM project is in one of five phases:

- 1. Not Started:** Evaluators can log in and change their attributes.
- 2. Evaluation:** Evaluators can conduct their evaluation.
- 3. Merging:** The manager creates the merged list of all evaluator findings.
- 4. Rating:** Evaluators can submit their ratings for the combined list of findings.
- 5. Finished:** The project is finished, reports can be generated using the Report Generator described in A.9.

All users marked active can log in at any time. The tasks they have depend on the projects they are assigned to and the phases of the projects.

To export a project a manager can click the Export button in the corresponding line of the project overview. The project import form can be reached by clicking the Import Project item in the Projects menu. At the top of the form the maximum upload file size of your server is displayed. If the file to be imported is larger than that file size the PHP installation¹ has to be reconfigured. All imported user accounts are deactivated by default.

A.9 Report Generation

The *Reports* menu has two items: *Overview* gives a list of all projects and the *Exported* item leads to the directory containing the statically exported reports. Each project listed in the overview is followed by several functions. With the first one a preview can be obtained which is available in all project phases; this allows the projects progress to be tracked. In the preview screenshots are shown as thumbnails to speed up page loading.

To create the statically generated report with a static style sheet and correctly sized screenshots one of the language links in the "Generate Reports" column has to be used. After the generation process the browser is automatically redirected to the created report. Exported reports can also be downloaded as an archive (zip, or tgz) using the links in the "Download Report" column of the overview table.

¹see <http://www.php.net/manual/en/ini.core.php#ini.upload-max-filesize>

Appendix B

HEM Evaluator Guide

A heuristic evaluation is conducted in three steps:

Evaluation: The evaluators conduct their evaluation by entering discovered usability problems. The evaluator can also add screenshots to these findings.

Merging: The manager of the evaluation summarises findings using the individual finding lists of the evaluators and creates a single merged list of findings.

Rating: The summarised list generated by the manager is rated by the evaluators.

Considering the steps of an evaluation, the evaluator's interface consists of three parts: a form to enter the description of the evaluation equipment that has been used, a findings collector, where the individual findings are entered, and a ratings collector to rate the summarised list findings. Additionally, a Home application is available which lists the tasks of each evaluator during the different phases of an evaluation.

B.1 The HEM Window

As illustrated in Figure B.1 the HEM window is divided into five areas:

Title Bar: The topmost element of the HEM interface is the title bar. On the left side there is the HEM logo, the right side shows the name of the currently logged in user and a logout link. The user's attributes can be changed by clicking on the user's name.

Menu Bar: Below the title bar is the menu bar. The menu bar holds a drop-down menu. Figure B.1 shows the home application with the evaluation menu expanded.

Breadcrumb Navigation: Below the menu bar breadcrumb navigation is shown. It displays the current location of the user. If an item has a function, it is displayed as hyperlink leading to the corresponding part of the application.

Main Application Area: The content of the main area depends on the part of the application currently in use.

Footer: The footer has no active content. It contains a link to the HEM home page, a copyright notice, and the HEM version number.

HEM uses a common set of icons. Each icon has tool tip labels which describe the function of the icon. The HEM icon set is listed in Table B.1.

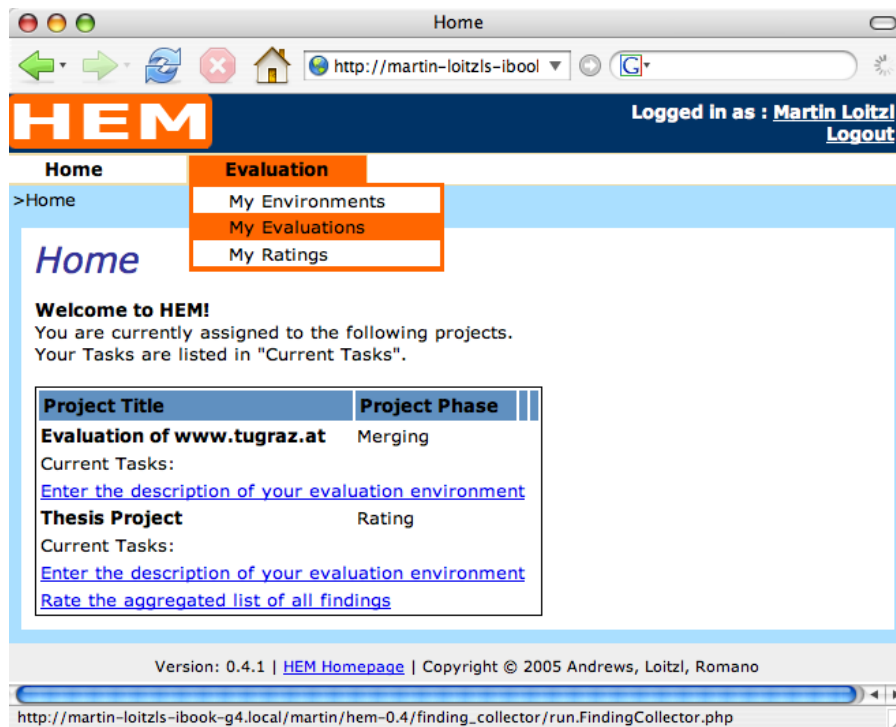


Figure B.1: This image shows the Home application of HEM with the evaluation menu expanded. The Home application contains the tasks that currently have to performed by the user.






Label	Icon	Meaning
Edit		Edit the corresponding item in a list.
Delete		Delete the corresponding item in a list.
Up		Move the corresponding list item up one item.
Down		Move the corresponding list item down one item.
Insert		Insert a list item at the specified position.

Table B.1: The HEM user interface uses this set of icons throughout the whole application.

B.2 The Home Application

Figure B.1 shows the Home application. The task list shows the tasks of the currently logged in user. The user is participating in one project, the “Thesis Project”, which is currently in the evaluation phase. The user can enter the evaluation environment or can conduct the evaluation. The task list of the home application adapts to the phase of the project; each evaluator can see what currently is to be done. If the user is assigned to more than one project, each project has its own block with the corresponding tasks listed.

B.3 Entering the Evaluation Environment

The form for entering the evaluation environment can be reached by clicking the *Enter the description of your evaluation environment* link in the task list or by choosing the corresponding project on the *My Environments* item of the *Evaluation* menu.

B.4 The Finding Collector

The Finding Collector can be started using the *My Evaluations* item in the *Evaluation* menu by clicking the project to evaluate, or directly by clicking on *Conduct your evaluation* link in the task list of the Home screen.

There are two views in the finding collector: The change or add form (Figure B.2) and the finding overview (Figure B.3).

Add/Change Form: With this form the evaluator can enter a description of a finding, mark it as a positive impression, and can choose a corresponding heuristic. An annotated and a full size screenshot can also be added. If heuristics are activated in the project the evaluator can obtain help for the configured set of heuristics by clicking the *Details* link after the drop-down field containing the heuristics. Already attached screenshots can be removed by clicking the Delete button, for example if the evaluator makes better ones later.

Finding Overview: Each finding is followed by a row that contains an icon with a horizontal arrow. By clicking this icon a new finding is inserted at the specified position. Within the Operations part of each finding there are icons for editing, deleting and reordering findings. Reordering findings can help evaluators organise their work.

B.5 The Rating Collector

After the manager has created the summarised list of findings, the list has to be rated by each evaluator individually. The Ratings Collector is used for this task and can be started by clicking the *Rate the aggregated list of all findings* link on the task list of the Home application, or by choosing the corresponding project in the *My Ratings* item of the *Evaluation* menu.

As shown in Figure B.4 the possible values are shown in a drop-down list. Each finding row consists of a finding description, the finding’s heuristic, and a thumbnail view of the screenshots. By clicking on the thumbnails a full size view can be obtained.

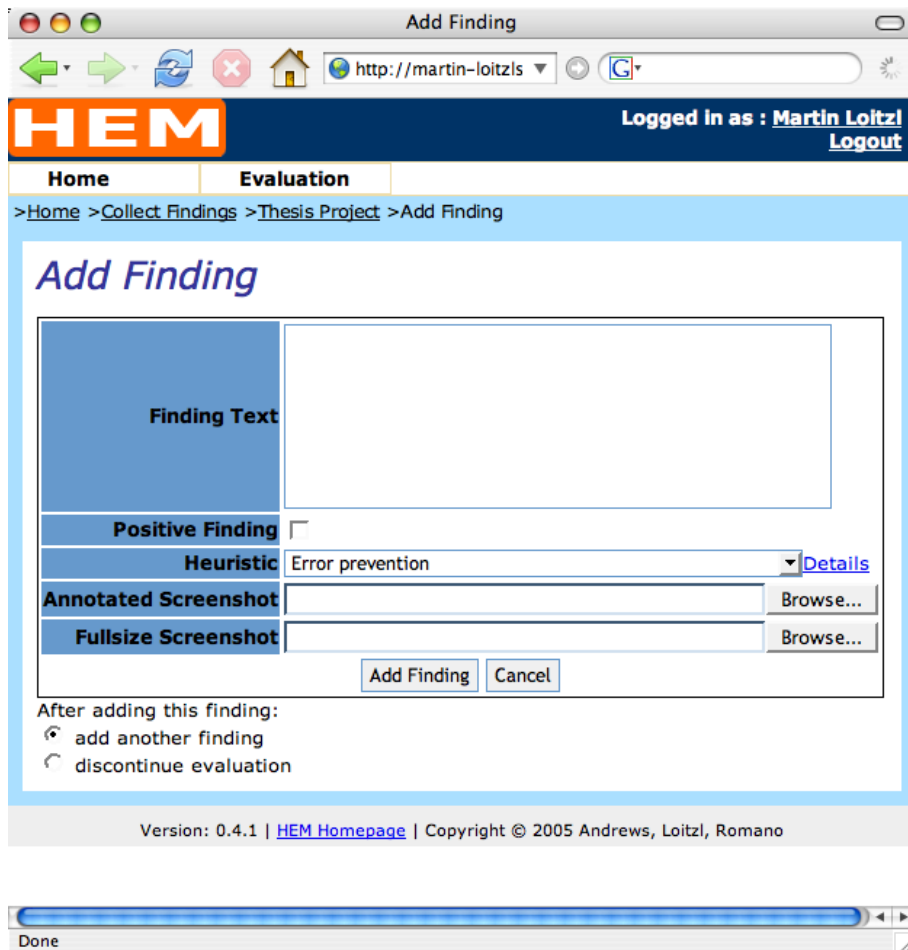


Figure B.2: The Add Finding screen. If the project is conducted using heuristics, a drop-down field containing the heuristics can be used by the evaluator to choose the violated heuristic. The *Details* link beside the heuristic drop-down field provides further details about the set of heuristics to be used. In addition, a full size and an annotated screenshot can be added to the finding description.

The screenshot displays a web browser window titled "List of Findings". The address bar shows the URL "http://martin-loitzls-ibook-g4.local/martin/hem-0.4/". The page header features the "HEM" logo and a user login status: "Logged in as : Martin Loitzl Logout". Below the header, there are navigation tabs for "Home" and "Evaluation", and a breadcrumb trail: ">Home >Collect Findings >Thesis Project".

The main content area is titled "Individual List of Findings" and contains a table with the following data:

Finding Text	Positive Finding	Operations	Heuristic	Annotated Screenshot	Fullsize Screenshot
The logo is sometimes a link to the home page, sometimes not.	No	[Icons: edit, delete, down arrow]	Consistency and standards	[Thumbnail screenshot]	[Thumbnail screenshot]
Navigational elements are clearly visible.	Yes	[Icons: edit, delete, up arrow, down arrow]	Consistency and standards	[Thumbnail screenshot]	[Thumbnail screenshot]
With every click inside the 'Job' part of the web site, a new window is opened. May be confusing for the user.	No	[Icons: edit, delete, up arrow]	Error prevention	[Thumbnail screenshot]	[Thumbnail screenshot]

At the bottom of the page, the footer text reads: "Version: 0.4.1 | [HEM Homepage](#) | Copyright © 2005 Andrews, Loitzl, Romano".

Figure B.3: The overview of findings in the Finding Collector containing three findings. A thumbnail of the screenshots is shown. The insert button (arrow) between the rows can be used to add a finding at the specified position. Findings can be reordered by using the arrow buttons in the *Operations* column in each finding row.

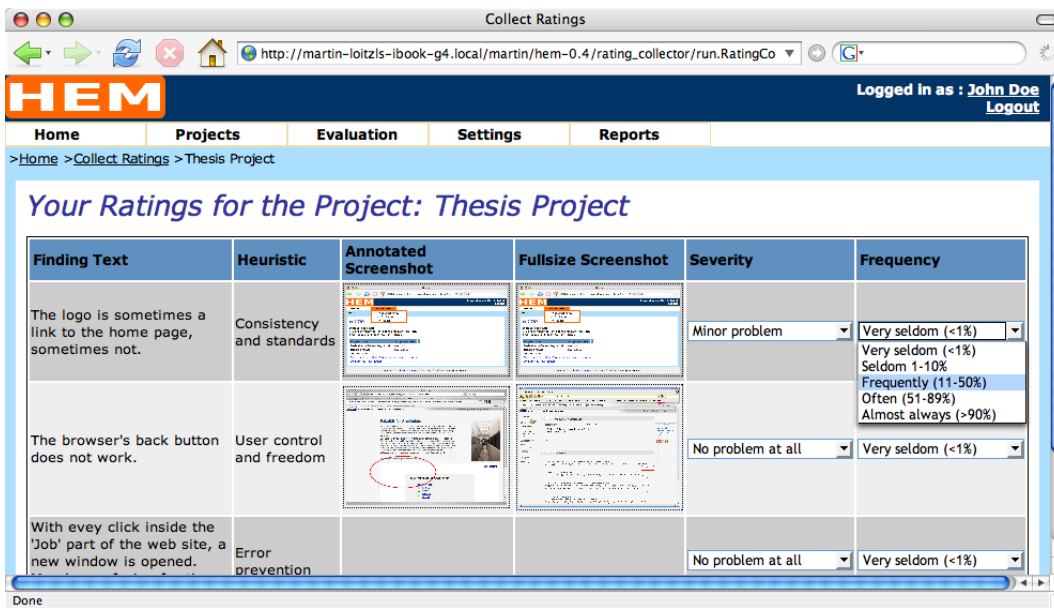


Figure B.4: The Ratings Collector is used to prioritise the summarised list of findings. Each row contains the description of the finding, screenshots, and drop-down fields containing the possible values for each rating scale.

Appendix C

HEM Developer Guide

HEM is a highly modular piece of software. It consists of many independent cooperating small applications. They all communicate with the a database over an object-oriented abstraction layer. These applications do not interact with each other directly which enables the easy exchange of modules and reduces unwanted side effects and dependencies.

HEM uses an object-oriented web application framework. The use of a framework emphasises the reuse of design over the reuse of code [Gamma et al. [1995]]. The difference between using a framework instead of a library is that the framework has an *inverted flow of control*. By using a framework a developer has to implement predefined functions and callbacks instead calling library functions - the framework runs the application.

Other advantages of frameworks are *modularity*, *re-usability* and *extensibility* [Fayad and Schmidt [1997]], see Chapter 7 for details.

C.1 Application Architecture

The HEM Application Framework is based on the PHP Application Framework described in Kabir [2003]. This framework provides access to a database interface, authentication and authorisation facilities, multilingual text handling, a template mechanism, and control functions.

C.1.1 The HEM Application Framework

To use the HEM framework the developer has to derive from the `PHPApplication` class and implement the `run` method. This enables the developer to use the whole functionality of the framework. This includes authentication and authorisation, multilingual functionality, and database connectivity. The framework API documentation is available at in the `doc` directory of the HEM distribution.

Every external library accessed by an adapter. This keeps the libraries exchangeable. All used libraries and their adapters are described in Section C.1.7. The framework, a class loader, all adapters and classes are located in the `framework/` directory.

To use the framework the developer writes a class inherited from the `PHPApplication` class. The constructor has an associative array as parameter. This array holds every possible configuration parameter. See Section C.4 for an example. The available parameters are listed and described in Table C.1.

A description of selected parameters:

app_type: The framework supports command line applications. Output functions are adapted to the kind of application. Possible values are: “WEB” or “COMMAND_LINE”.

Parameter	Data Type	Meaning
app_name	string	The name of the application.
app_version	string	The version number.
app_type	string	“web” or “command line” application.
app_db_url	string	Data Source Name (DSN) of the application database.
app_auth_dsn	string	Data Source Name (DSN) of the authentication database.
app_exit_point	string	Redirection address if user logs out.
app_authentication	boolean	Application requires authentication.
app_auto_authenticate	boolean	Users are automatically redirected to login application.
app_auto_check_session	boolean	not used .
app_auto_connect	boolean	Open database connection at application initialisation.
app_debugger	boolean	Activate the built-in debugger.
app_themes	boolean	Activate user selectable stylesheets.
app_check_browser_language	boolean	Application language is set to the browser’s language.
app_admin_auth	boolean	Authentication information is changed within the application.
app_session_name	string	The name of the session cookie.
app_label_file	string	File name of the multilingual label file.
app_error_file	string	File name of the multilingual error file.
app_message_file	string	File name of the multilingual message file.

Table C.1: List of all Configuration Parameters of the HEM Application Framework.

app_exit_point: If a user logs out, or a timeout occurs and the user is logged out by the system, the user is redirected to this address.

app_check_browser_language: The application’s language is set to the browser’s language. If a user has a custom setting, the browser language is overridden.

app_db_url: The Data Source Name of the application database.

app_auth_dsn: The authentication database is not the same as the application database. The authentication library can not only use databases for authentication, but also mail servers, file servers, etc.

C.1.2 Data Storage

HEM stores its textual data in a relational database and binary data in the file system of its host.

Textual Data

The database connection is maintained by a database adapter, whose adaptee is currently the PEAR::DB package. The adapter ensures that the library is easily exchangeable, for example if PEAR::DB maintenance is discontinued at some future point of time.

The Database itself is accessed by an object-oriented Application Programming Interface (API) rather than by direct SQL queries. This ensures a single point of control and thus more flexibility with security and control issues. The HEM API is described in Section C.1.3

Binary Data

Since MySQL lacks sufficient support for storing binary data, the file system holds the screenshots submitted by evaluators. Facilities for storing, deleting and displaying screenshots are provided by a class. The directory where screenshots are stored is configurable in the file `conf/conf.global.php`, see Section C.4.2 for a description of the configuration directory. By default this directory is inside HEM directory structure, which means that everyone can browse through the files. If this should be avoided, it is recommended to keep the directory outside the web server's document root.

Reports

Each generated report is stored in a separate directory. The name of the directory is the unique identifier of the project. A report consists of a HTML file, called *he.html*, a style sheet, called *report.css* and several screenshots stored in a subdirectory called *images*. The annotated screenshots are automatically resized to a width of 600 pixels. The original full size screenshots are not manipulated by the application. The reports are stored in a directory inside the HEM directory. The reports are thus visible to everyone with access to the HEM server. If this should be avoided the location of the reports directory can be changed in the `conf/conf.global.php` file to some location outside of the web server's document root.

C.1.3 The HEM Application Programming Interface (API)

HEM provides a full object-oriented Application Programming Interface (API). The class files of the API are stored in the *classes/* directory.

Content stored in the database is accessed by a class called `DBObject` which provides basic functionality for manipulating database content. Every entity in HEM has its own class, derived from `DBObject` class which has an insert, update and delete method.

In the constructor of each derived class, there must be a unique identifier to denote the desired entity, and a database handle, which in this case is the database handle of the HEM Application Framework. If the identifier already exists in the database, the *init_ok_* member variable is set to true and all member variables holding fields of the corresponding entity are set to the values stored in database. Any further functionality, specific to that particular entity, can be implemented in the derived classes.

All primary keys are generated by the HEM Application Framework. HEM is designed to be run in multiple installations. In order to be able to import and export projects independently, all primary keys are unique identifiers with a length of 32 bytes. If an identifier already exists, it can be assumed that it identifies the same dataset globally. For example, Nielsen's 10 heuristics are already stored in the HEM database. They do not have to be re-imported, because they are already there.

C.1.4 Multilingual Features

Multilinguality in HEM has two aspects: multilinguality of the user interface and of the content entered by the managers. The multilingual texts for the user interface are stored in language files. These hold all labels, errors, and messages in an associative array. The framework provides an interface for these three file types. The filenames are passed to the framework via the parameter array described in Section C.1.1.

Content entered by managers also has to be multilingual. For that purpose a translator class exists. All multilingual content is stored in all available languages in a single database table. All entities having multilingual attributes hold an identifier for the translation table rather than the content itself. The translator class provides a simple and transparent interface for dealing with multilinguality.

C.1.5 Configuration

All configuration files are located in the “conf/” directory of the HEM distribution. The HEM configuration consists of the following files:

conf.host.php: This file holds all configuration settings which depend on the host HEM runs on.

conf.global.php: All other settings are stored here.

constants.activitylog.php: HEM logs some important events to a database table. The events are defined by constants stored in this file.

constants.app.php: This is a file containing area constants required by PEAR::LiveUser, the authentication and authorisation library HEM utilises. Only one area is defined in HEM.

constants.groups.php: The LiveUser group constants are defined here. Currently there are 3 groups: evaluators, managers and administrators.

constants.project.php: A HEM project has five phases, the constants representing these phases, are stored in this file.

constants.rights.php: The HEM authorisation scheme is based on rights granted to groups. These rights are defined here.

C.1.6 Naming Conventions

HEM uses the following naming and coding conventions:

- Each file contains just one class. The name of the file is the same as the name of the class with the prefix “class.”.
- Files with configurations settings have the prefix “conf.”, files with constants start with “const.”.
- Error messages are stored in files starting with “error.”. The name of the file is the same as the class name that uses these error messages. This is equivalent for general messages and labels, except files start with “label.” and ““messages.” respectively.
- Each application is started by a file starting with “run.”. In that file the application class is instantiated and the associative array holding the parameters is passed to the application via the constructor. Finally, the run method is called, what starts the application.

C.1.7 External Libraries

HEM uses many external libraries. Every library used by the framework itself is abstracted by an adapter. This ensures that the libraries stay exchangeable. If, for example, a library is discontinued, another one can be used by simply rewriting the adapter, rather than changing all parts of the application which used the library. The HEM Application Framework uses the following PEAR¹ libraries:

¹PHP Extension and Application Repository <http://pear.php.net>

PEAR::DB

PEAR::DB is an object-oriented database abstraction library that layers standard existing PHP database functions. This library ensures full portability of all supported database management systems if all portability options are used. An application using PEAR::DB can use a broad variety of database systems and can also change the database management system after installation.

PEAR::LiveUser

HEM is a web application used by multiple users having different roles. Thus an authentication and authorisation facility is needed. This functionality is provided by the LiveUser package. The LiveUser library uses a container model to provide three different access rights models of increasing complexity and flexibility. HEM uses the *medium* model, which supports users, groups and rights. Every user is a member of one or more groups, and the groups are equipped with access rights.

PEAR::HTML_Template_IT

A template engine allows the separation of application logic from presentation. This makes concurrent development easier and reduces dependencies inside the application structure. There are a wide variety of template engines with different levels of complexity available. A rough distinction between different types can be made by looking at the template syntax. Libraries which have complex looping of conditional structures afford great control over the page rendering process but a HTML designer also needs to learn the syntax. The other sort of template engines are block-oriented ones without control structures. They have variables and blocks. Variables are substituted with data by the application logic and blocks can be used to iterate over the same kind of content multiple times, for example rows of tabular data. Template_IT uses the second and simpler approach.

C.2 Template Architecture

The HEM Application Framework utilises a template engine in order to use all benefits of template-based web application design. Furthermore, the HTML code is generated in XHTML 1.1 and uses CSS2 stylesheets. Application logic, content and design of content are thus fully separated and multiple persons are able to work in parallel without having full expertise of all concerns of web application development.

The HEM template architecture uses a *master* template incorporating all other template parts (see Figure C.1). The master template holds the basic structure of an XHTML file and a number of variables of the form “{*VARIABLE*}”. These variables are replaced with content by the template engine. The master template has two kinds of variables: the main content area and boxes.

C.2.1 Main Content

The main content area variable “{*CONTENT*}” represents the the output of the various applications. Each has its own template which ensures a fine-grained control over the template structure. The handling of template-related operations is completely handled by the HEM Application Framework. Section C.4 describes how this mechanism is used.

C.2.2 Boxes

A box is a piece of content which is the same over multiple applications. Consider the *userinfo* box shown in Figure C.1. The content does not depend on the currently used application part. As long as a user is logged in, the logout link and the user’s name has to be displayed.

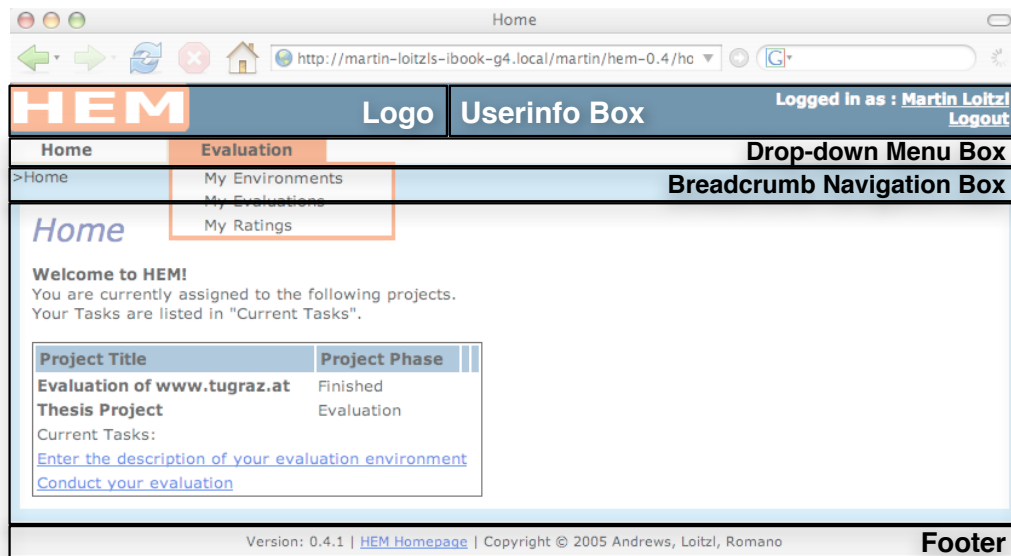


Figure C.1: The HEM template structure.

In order to fill the box variables of the master template with the corresponding contents the HEM Application Framework provides a simple box loading facility. The associative array “BOXES” in the file `conf/conf.global.php` holds the configuration settings for all boxes. The key of the array is the same string as the variable name of the box in the master template. Each key denotes a sub array of strings with the following keys:

class_file The file that holds the box class.

class_name The name of the class to instantiate.

get_function The name of the function returning the contents of the box.

Since the box classes are not derived from the HEM Application Framework, they are passed a reference of the calling class via the constructor to allow them to use the facilities of the framework and the configuration of the current application.

C.3 Directory Structure of HEM

The directory structure of HEM is listed in Table C.2. The HEM Application Framework, the external libraries, the HEM API and the configuration directory are in the first four directories. The next four directories store the parts of HEM covering the heuristic evaluation process. The next directory holds the report generator.

The next six directories contain applications for managing the heuristic evaluation. The login application provides a form for entering the username and password of the user. The directory also contains the userinfo box, which provides a logout button and shows the name of the user currently logged in. The next two directories contain a box for a drop-down menu and one for breadcrumb navigation.

The next directory contains an application for a web-based setup and finally the last directory contains an application to inform the user which tasks currently can be performed.

Directory	Contents
framework	The HEM Application Framework, its classes and a class loader.
classes	The HEM Application Programming Interface (API).
pear	The PEAR libraries used by HEM.
conf	All configuration settings for HEM.
environment_collector	Evaluators complete environment forms in the Environment Collector.
finding_collector	The evaluators enter their findings and upload their screenshots in the Finding Collector application.
finding_merger	The manager creates the merged list of the individual evaluator findings in the Finding Merger application.
rating_collector	The evaluators submit their ratings for the merged list of all findings in the Rating Collector application.
report_generator	The module to preview, generate and download reports.
proj_mgr	The Project Manager application.
user_mgr	User Manager and a module for forgotten password recovery.
environment_mgr	Forms for entering the several evaluation environments can be created with the Environment Manager application.
heuristicset_mgr	Application to manage sets of heuristics.
ratingscale_mgr	Application to manage rating scales .
ratingscheme_mgr	Application to compile rating schemes.
login	The login application and the user information box.
menu	The drop-down menu box.
breadcrumb_navigation	A box which holds the breadcrumb navigation.
setup	A web-based setup routine.
home	The home application which provides a task list for the current user.

Table C.2: The HEM directory structure.

```

1 class Stats extends PHPApplication {
2     function run() {
3         $this->debug("Run Method called");
4     }
5 }

```

Listing C.1: Deriving an application from the HEM application framework.

C.4 A HEM Example Application

This example shows how to implement a HEM application using all features of the HEM Application Framework. The example is a stub for a statistics application.

C.4.1 Getting Started

Each application in HEM uses the same directory structure. The application main class is stored in a file called `class.<ApplicationName>.php`, in this case `class.Stats.php`. The derived class has to implement the `run` method defined by the framework (see Listing C.1).

The class is instantiated in a file called `run.Stats.php` (see Listing C.2 - line 2). After instantiation the `run` method of the application class is called, as can be seen in Listing C.2 line 26. Before calling `run`, the application debugger is initialised, afterwards the debug information is dumped if debugging has been activated.

C.4.2 Configuration

Activation of debugging and all other configuration settings are controlled with a associative array passed to the application at the time of instantiation (Listing C.2 - line 3 to line 22). The variable `DEBUGGER` in line 14 is set in the global configuration file.

Configuration files exist for each application (Listing C.3). The particular configuration files are included in the `run` files (Listing C.2 - line 1). Each application configuration file includes the global configuration file and the global configuration file includes the host-dependent configuration file, which contains all variables which change depending on the host HEM runs on.

A very fine-grained control over configuration settings is thus ensured. Global settings can be used in each application's configuration file, but also specific settings can be used for each application.

C.4.3 Managing User Interaction

The user interacts with a web application via `GET` and `POST` requests. The HEM application framework provides methods for handling these requests.

An example of how to control `GET` requests is given in Listing C.4. In line 5 the `GET` variable `cmd` is evaluated by a method provided by the HEM application framework. The first argument is the name of the `GET` variable to look for and the second argument is the return value if the field was not set.

To control the return value with the second argument makes it easy to check if a command was given. The method also suppresses PHP notices when accessing variables that are not defined.

```

1  require_once "conf.Stats.php";
2  $thisApp =& new Stats(
3      array(
4          'app_name' => $APPLICATION_NAME,
5          'app_version' => '1.0.0',
6          'app_type' => 'WEB',
7          'app_db_url' => $AUTH_DB_URL,
8          'app_auth_dsn' => $AUTH_DB_URL,
9          'app_exit_point' => '',
10         'app_authentication' => TRUE,
11         'app_auto_authenticate' => TRUE,
12         'app_auto_check_session' => FALSE,
13         'app_auto_connect' => TRUE,
14         'app_debugger' => $DEBUGGER,
15         'app_themes' => TRUE,
16         'app_check_browser_language' => FALSE,
17         'app_admin_auth' => TRUE,
18         'app_session_name' => $SESSION_NAME,
19         'app_label_file' => $LABEL_FILE,
20         'app_error_file' => $ERROR_FILE,
21         'app_message_file' => $MESSAGE_FILE,
22     )
23 );
24
25 $thisApp->bufferDebugging();
26 $thisApp->run();
27 $thisApp->dumpDebugInfo();

```

Listing C.2: Running a HEM application

```

1  require_once('../conf/conf.global.php');
2
3  $PHP_SELF = $_SERVER['PHP_SELF'];
4
5  $APPLICATION_NAME = 'Stats';
6  $DEFAULT_LANGUAGE = 'US';
7
8  $APP_DIR = '/stats';
9  $REL_APP_PATH = $REL_APP_ROOT . $APP_DIR;
10 $TEMPLATE_DIR = $APP_ROOT . $APP_DIR;
11 $REL_TEMPLATE_DIR = $REL_APP_ROOT . $APP_DIR;
12
13 $STATS_TEMPLATE = 'stats.html';
14
15 require_once 'class.Stats.php';
16
17 $ERROR_FILE = 'errors.Stats.php';
18 $MESSAGE_FILE = 'messages.Stats.php';
19 $LABEL_FILE = 'labels.Stats.php';

```

Listing C.3: An application configuration file.

```
1 class Stats extends PHPApplication {
2     [..]
3     function getRequestExample() {
4         global $PHP_SELF;
5         $cmd = $this->getGetRequestField('cmd', null);
6         if (!is_null($cmd)) {
7             switch ($cmd) {
8                 case 'step1':
9                     $this->writeln("Step 1 selected: <a href=\""$PHP_SELF?cmd=
10                        =step2\"">proceed to Step 2</a>");
11                     break;
12                 case 'step2':
13                     $this->writeln("Step 1 selected: <a href=\""$PHP_SELF?cmd=
14                        =step1\"">proceed to Step 3</a>");
15                     break;
16                 default:
17                     $this->writeln("unknown step: <a href=\""$PHP_SELF?cmd=
18                        step1\"">go to Step 1</a> or <a href=\""$PHP_SELF?cmd=
19                        step2\"">go to Step 2</a>");
20                     break;
21             }
22         }
23     }
24 }
```

Listing C.4: Controlling user interaction with *GET* request fields.

```
1  function run() {
2      $project_object = & new Project(0, $this->dbi_);
3      $project_ids = $project_object->getProjectIds();
4
5      $this->dumpArray($project_ids);
6
7      $project_object = & new Project($project_ids[0], $this->dbi_);
8      $this->dumpArray($project_object->data_array_);
9  }
```

Listing C.5: Usage of the HEM API.

C.4.4 Using the HEM API

The source files of the HEM API are stored in the `classes/` directory. In order to use the files they have to be included either in the global configuration file or in the applications configuration file. Any part of the API that uses the database needs a reference of the database handle of the HEM application framework.

HEM API classes can either be instantiated with a unique identifier to initialise them with the corresponding database record, or if the identifier is `null` to initialise the object without a special identity.

The former case is used in Listing C.5 lines 2 and 3 for creating an anonymous object in order to retrieve all available project identifiers. The latter case is used in lines 7 and 8 to retrieve the data of a specific record. The retrieval of identifiers can also be filtered and ordered. The filter functionality can be used in two ways:

Simple: This filter works by passing an associative array with key/value pairs representing columns and values of the database table. The key/value pairs are combined with a boolean AND operator. Values are compared by the equal operator.

Complex: This filter also matches a number of columns, but also a boolean operation to combine fields and a comparison operator can be chosen.

The retrieved identifiers can also be ordered by passing a column name and order direction to the `getProjectIds` method.

An example output of Listing C.5 can be seen in Listing C.6. The project name is not stored directly in the database field. The identifier for the name refers to a translation identifier. The use of multilingual features is described in Section C.4.5.

C.4.5 Using Multilingual Features

As stated before, HEM uses two kinds of multilinguality. Strings regarding the user interface are stored in language files. These strings do not change during the use of HEM. The other kind of multilingual content is entered by HEM users.

User Entered Content

The content entered by users is accessed through a translator class. A usage example is shown in Listing C.7. In line 1 the `Translator` object is instantiated without an identifier. The translation is retrieved

```

1 Array
2 (
3     [pId] => ne79r8n3byted092sn7lx0bm64e17gy4
4     [pNameId] => n2526zrg5qj6z3477qv2detvplp9b9m3
5     [pDescriptionId] => lrv4cyh42xtpt27gqye2555irtzv9gib
6     [pPhase] => 2
7     [heurSetId] => io1cd6u3g7e6xp75hp6cpwvnd2ivvfk
8     [envId] => krhlwkcb40dvhhsaghulu8azgig8ouct
9     [schemeId] => 9inwqh48640qim60l2c67bz1lgscmhlz
10    [pAdded] => 2005-07-05 13:39:20
11 )

```

Listing C.6: A dump of an example HEM object data array.

```

1 $translator = & new Translation(0, $this->dbi_);
2 $translation = $translator->getTranslation($project_object->
    data_array_['pNameId'], $this->language_);

```

Listing C.7: Usage example of the HEM Translator.

in line 2 by calling the `getTranslation` method. This method has two parameters: the first one is the identifier for the desired translation, the second one is the current framework language. The framework language depends on the user's choice. If no user is logged in, the default language of the browser is set by the framework.

Interface Translation

The translations of user interface elements are stored in language files. There are three types of files: labels, errors, and messages. Each file has an corresponding prefix. The strings are stored in two dimensional associative arrays. An example of a `label` file is shown in Listing C.8. Lines 1 and 2 define the English translations 4 and 5 define the German ones. The first key of the array is the language, the second one is the name of the translation entry. The names of the language files have to be passed to the framework via the associative configuration array. These names are usually defined in the application configuration file.

The translations for labels can be used with the framework's `getLabelText` method. There are also `getMessageText` and `getErrorText` methods.

C.4.6 Using Templates

This section uses nearly all of the previously discussed features. In order to use templates, a template file has to be defined. This has already been done in Listing C.3 line 13. The template file is written in *HTML* code. It contains only the part of a *HTML* page used in the application. Everything else is defined by the master template. The Statistics template file is shown in Listing C.9. Variables are in curly brackets. A block is enclosed by a `BEGIN` and an `END` keyword followed by the name of the block in a *HTML* comment. Blocks can be used for iterations (see Listing C.9 lines 9 and 18).


```
1 $LABELS[ 'US' ][ 'LABEL_PROJECT' ] = 'Project Title';
2 $LABELS[ 'US' ][ 'LABEL_EDIT' ] = 'Edit';
3
4 $LABELS[ 'DE' ][ 'LABEL_PROJECT' ] = 'Projekt Bezeichnung';
5 $LABELS[ 'DE' ][ 'LABEL_EDIT' ] = '&Auml;ndern';
```

Listing C.8: An example HEM language file for the definition of user interface strings.

In order to use template variables in the PHP application the `showScreen` method of the framework can be used (see Listing C.10 line 3). The method has three parameters:

Template: The name of the *HTML* template to use.

Callback function: The function the framework has to call. This function fills all the template variables with values.

Application name: The name of the application. This can be used for setting the page title.

The callback method has one parameter, which is a reference to the template object. This reference has three important methods:

setVar: This method fills the template variables with data. The parameter can be either a pair of strings - variable name and its value, or an array. The array holds key/value pairs where the key is the name of the template variable. The value represents the data to be entered to the template variable (see Listing C.10 line 7).

setCurrentBlock: Set a block from the template as the current block. Blocks can be used for iteration.

parseCurrentBlock: Tells the framework that the current iteration is finished and that the block is ready to be put into the template buffer holding rendered template parts.

The callback function - `displayStatsOverview` - in Listing C.10 line 6 to 35 generates an overview of the projects currently available in HEM. An anonymous `Project` object and a translator instance is created. In line 10 all project identifiers are ascertained. In the following iteration all project objects are instantiated and used for the iteration over the project block. The `translator` object is used to translate the project's title. The last part of the function sets language dependent labels (lines 25 to 31). Both variants of the `setVar` method described above in this Section are used for demonstration. The return value of the callback function has to be `TRUE`, otherwise nothing is displayed. This can be used for error handling.

C.5 A HEM Example Box

A box is not a complete application derived from the HEM Application Framework. In order to ensure that a box is able to use the framework functionality, it receives a reference to the application object via the constructor. This also enables the box to access the current application's attributes and methods.

```

1 <h1>{STATS_TITLE}</h1>
2 <h2>{MESSAGES}</h2>
3 <table>
4   <tr>
5     <th>{LABEL_PROJECT_TITLE}</th>
6     <th>{LABEL_OPERATIONS}</th>
7     <th></th>
8   </tr>
9   <!-- BEGIN project_block -->
10  <tr class="{BG_CLASS}">
11    <td><a href="{URL}">{PROJECT_TITLE}</a>
12    </td>
13    <td class="operations">
14      <a href="{DISPLAYSTATS_URL}">{LABEL_DISPLAYSTATS}</a>
15    </td>
16    <td>{PROJECT_PHASE}</td>
17  </tr>
18  <!-- END project_block -->
19 </table>
20 <br />
21 <a href="{DISPLAYSTATS_SUMMARY_URL}">{LABEL_DISPLAYSTATS_SUMMARY}</a>

```

Listing C.9: The Stats HTML template. HEM variables are enclosed in curly brackets. An iterator block is enclosed in HTML comments.

C.5.1 Getting Started

The box has to be registered in the global configuration file `conf/conf.global.php`. The corresponding lines of code are shown in Listing C.11. The next step is to place a template variable in the master template `templates/index.html`. The name of the template variable has to be the same string as the key of the box in the `BOXES` configuration array.

The only two differences between a normal application and a box are:

- The calling application is accessed through the reference to the application object.
- The handlers for multilingual labels, errors, and messages of the application object are already used for the application. The box needs its own handlers.

A box can use the whole HEM API and all other framework functionality.

C.5.2 Configuration

Each box has a configuration file. The naming convention is the same as for applications. The Statistics box configuration file is called `conf.StatsBox.php`. A Box uses a different template than the application whose reference it receives. Lines 9 to 13 in Listing C.12 show how the template directory has to be redefined. A separate handler for multilingual labels has to be defined. The name of the labels file is set in line 15 of Listing C.12.

```

1  function statsOverview() {
2      global $STATS_TEMPLATE;
3      $this->showScreen($STATS_TEMPLATE, 'displayStatsOverview',
4          $this->getAppName());
5  }
6  function displayStatsOverview(& $tpl) {
7      $project_object=& new Project(0, $this->dbi_);
8      $translator=& new Translation(0, $this->dbi_);
9
10     $project_ids=$project_object->getProjectIds();
11     $i=0;
12
13     foreach($project_ids as $current_project_id) {
14         $current_project_object = & new Project($current_project_id,
15             $this->dbi_);
16
17         $tpl->setCurrentBlock('project_block');
18         $tpl->setVar(array(
19             'PROJECT_TITLE'=>$translator->getTranslation(
20                 $current_project_object->pNameId , $this->language_),
21             'PROJECT_DESCRIPTION'=>$translator->getTranslation(
22                 $current_project_object->pDescriptionId , $this->
23                 language_),
24             'BG_CLASS'=>($i%2?'odd':'even'),
25             ));
26         $tpl->parseCurrentBlock();
27     }
28
29     $tpl->setVar('STATS_TITLE', $this->getLabelText('
30         STATISTICS_TITLE'));
31     $tpl->setVar('LABEL_PROJECT_TITLE', $this->getLabelText('
32         LABEL_PROJECT_TITLE'));
33
34     $tpl->setVar(array(
35         'LABEL_OPERATIONS'=>$this->getLabelText('LABEL_OPERATIONS')
36         ,
37         'LABEL_INCLUDE'=>$this->getLabelText('LABEL_INCLUDE'),
38         ));
39
40     return TRUE;
41 }

```

Listing C.10: Usage of the showScreen method.

```
1 $BOXES = array(  
2     [..]  
3     'STATS_BOX' => array(  
4         'class_file' => 'stats/class.StatsBox.php',  
5         'class_name' => 'StatsBox',  
6         'get_function' => 'getStatsBox',  
7     );  
8 );
```

Listing C.11: Registering a box in the global configuration file.

```
1 require_once('../conf/conf.global.php');  
2  
3 $PHP_SELF = $_SERVER['PHP_SELF'];  
4  
5 $STATSBOX_TEMPLATE = 'stats_box.html';  
6 $APPLICATION_NAME = 'STATSBOX';  
7 $APP_DIR = '/stats';  
8  
9 global $REL_APP_ROOT, $APP_ROOT;  
10  
11 $REL_APP_PATH = $REL_APP_ROOT . $APP_DIR;  
12 $TEMPLATE_DIR = $APP_ROOT . $APP_DIR;  
13 $REL_TEMPLATE_DIR = $REL_APP_ROOT . $APP_DIR;  
14  
15 $LABEL_FILE = $APP_ROOT . $APP_DIR ."/". 'labels.StatsBox.php';
```

Listing C.12: Configuration file of a HEM box.

```
1 function StatsBox(& $app_object) {
2     require_once "conf.StatsBox.php";
3
4     $this->app_object_ = $app_object;
5     $this->app_dir_ = $APP_ROOT . $APP_DIR;
6
7     $this->lbl_handler_ = new LabelHandler(array(
8         'name' => '',
9         'language' => $this->app_object_->language_,
10        'file' => $LABEL_FILE,
11    ));
12
13    $this->template_ = $STATSBOX_TEMPLATE;
14    $this->template_handler_ = new TemplateHandler($this->app_dir_
15        );
16    $this->template_handler_->loadTemplatefile($this->template_,
17        true, true);
18 }
```

Listing C.13: The constructor of the `StatsBox` class.

C.5.3 Object Creation

Unlike applications, boxes have their own constructor to instantiate the template and label handler. Additionally, the constructor receives a reference to the calling framework object to enable the box to use the API of the HEM application framework, in particular the database handle and the current language.

In Line 2 of Listing C.13 the box's configuration file is included, line 3 assigns the application object reference to a member variable called `$this->app_object_`. In line 8 the `LabelHandler` object is created. The parameters are passed using an associative array with three fields: `name` can be used for the name of the calling application or box, `language` is the language to use, and `file` is the name of the file that holds the label array.

In line 12 the file name of the template of the box is defined. Lines 13 instantiates the template handler and line 14 loads the template file.

C.5.4 Using Templates

In order to use the template defined in the configuration file, a separate template handler was instantiated in Section C.5.3. A member variable called `$this->template_handler_` is available. Lines 6 to 11 of Listing C.14 give an example of how to use the template handler.

Labels are listed in the file `labels.StatsBox.php`, a label handler has already been instantiated in the constructor (see Section C.5.3). The usage of the label handler is shown in line 9.

```
1 function getStatsBox() {
2     $project_object = & new Project(0, $this->app_object_->dbi_);
3     $project_ids = $project_object->getProjectIds(array(
4         'pPhase' => array('op' => '<', 'value' => '3', 'cond
5             ' => '''),
6         ));
7     $this->template_handler_->setCurrentBlock('stat_item_block');
8     $this->template_handler_->setVar(array(
9         'LABEL' => $this->lbl_handler_->write('LABEL_PROJECTS'),
10        'VALUE' => sizeof($project_ids),
11        ));
12    $this->template_handler_->parseCurrentBlock();
13
14    $content = $this->template_handler_->get();
15    return $content;
16 }
```

Listing C.14: The `getStatsBox` method of the `StatsBox` class.

Appendix D

HEM Report Example

This Appendix presents the draft report generated using HEM during the TUGraz heuristic evaluation project discussed in Chapter 11. The report has to be finished using an editor or word processor. The report is in the German language, because the evaluation was conducted in the German language.

Bericht über Heuristische Evaluierung

Alois Dengg
Harald Auer
Martin Loitzl
Joe Zehenter

Evaluierung von www.tugraz.at

Die Evaluierung von www.tugraz.at mit HEM.

1. Zusammenfassung

Executive Summary für Managementebene (1 Seite).

Der Manager des Klienten hat keine Zeit den ganzen Bericht zu lesen, sondern liest nur die Zusammenfassung und möchte wissen wie die Evaluierung gemacht wurde und was die Hauptergebnisse sind.

2. Umgebung der Evaluierung

Hardware und Software Konfigurationen, die die Evaluatoren benutzt haben.

Tabelle 1: Umgebung der Evaluierung

Evaluator	Alois Dengg	Harald Auer	Martin Loitzl	Joe Zehenter
Alter	33	31	28	31
Geschlecht	männlich	männlich	männlich	male
Webbrowser	Firefox v1.0.4	Firefox 1.0.7	Safari 2.0.1	Opera 8.5 Build 7700
Betriebssystem	Windows XP V2002 SP2	Windows XP	Apple Mac OS X 10.4.2	Windows XP
Verbindung	hoch	LAN	ADSL 768kbit	LAN
Monitor Farben	32bit	32bit	24bit	32bit
Monitor Auflösung	1280x1024	1400x1050	1024x786	1024x768
Monitor Größe	17"	15"	12"	17"
Datum der Evaluierung	29.09.2005	29.09.2005	29.09.2005	26.9.2005
Zeitraum der Evaluierung	16:00-18:00	13:00-17:00	10:30-12:30	19.30

3. Positive Eindrücke

Beschreibung von Punkten, die positiv aufgefallen sind, auch mit Screenshots.

Screenshots sollten immer im PNG Format abgespeichert werden, da PNG verlustfrei ist und sehr gut komprimiert. Hier sollten herunterskalierte (etwa 600 x 400 Pixel) Versionen der Screenshots eingebunden werden, die auch mit Notizen versehen werden können. Allerdings verlinkt mit *lokalen Kopien* der *nicht-editierten*, originalen Screenshots in voller Auflösung.

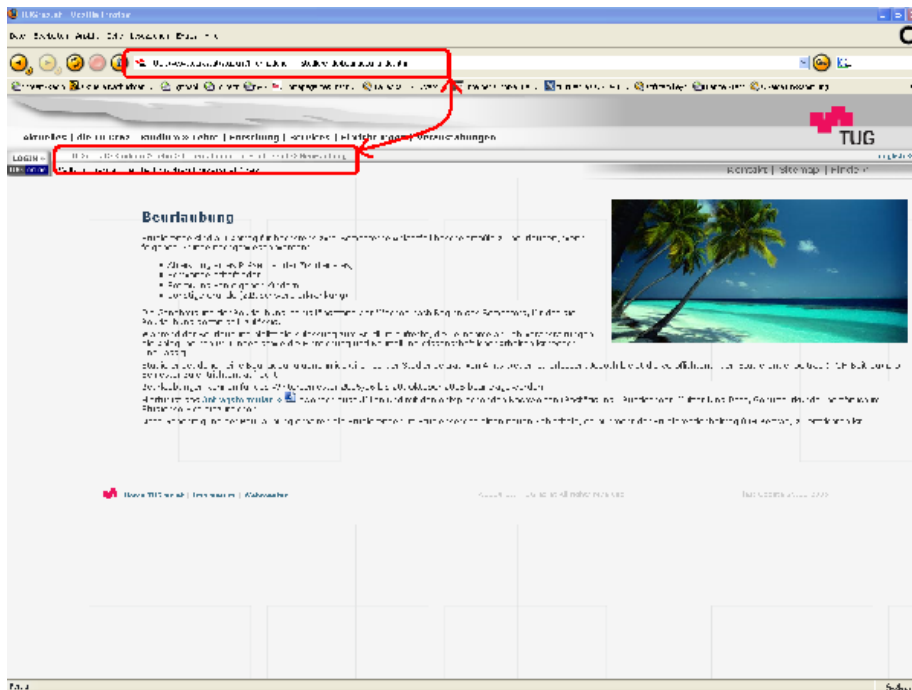
3.1. Sehr mächtiges Suchp...

Sehr mächtiges Suchportal, das auch TUG-Online einbezieht (leider aber nur in Deutsch).

3.2. Die URL sind permane...

Die URL sind permanent verfügbar und der Name gibt Rückmeldung über den Inhalt. Adresse der Seite in der Adressleiste des Browsers ist "hackable" und spiegelt die Struktur der Seite wieder.

Abbildung 1: marked.png



3.3. Die Webseite funktio...

Die Webseite funktioniert auch mit deaktiviertem Stylesheet, was sehr gut für Menschen mit körperlichen Einschränkungen ist, die alternative Ein/Ausgabe Geräte verwenden.

3.4. Die Website gibt rüc...

Die Website gibt Rückmeldung über die aktuelle Position innerhalb der Site. Aktuelle Position innerhalb der Site einfach festzustellen.

Abbildung 2: Position.png



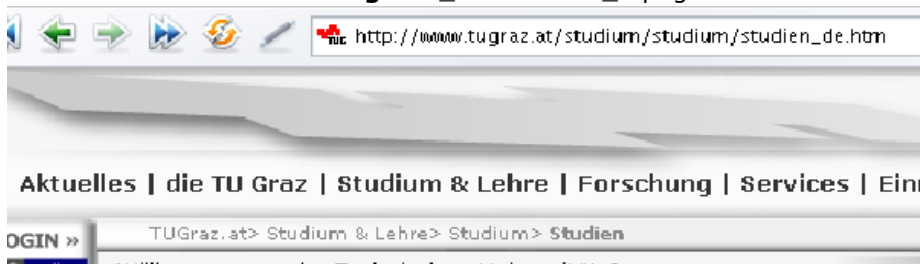
3.5. Ich weiß sofort ob e...

Ich weiß sofort ob es sich rentiert auf die Uni zu gehen und ob es die Witterungseinflüsse zulassen.

3.6. Übersichtlich und sch...

Übersichtlich und schön gestaltete Sitemap, Menüzeilen, Farbgebung, überhaupt Design sehr durchgängig, übersichtlich.

Abbildung 3: u_screenshot_4.png



4. Analyse der Hauptprobleme

Beschreibung der 5 Hauptprobleme nach absteigender Bewertung der Gesamtliste in Kapitel 5

Screenshots sollten immer im PNG Format abgespeichert werden, da PNG verlustfrei ist und sehr gut komprimiert. Hier sollten herunterskalierte (etwa 600 x 400 Pixel) Versionen der Screenshots eingebunden werden, die auch mit Notizen versehen werden können. Allerdings verlinkt mit *lokalen Kopien* der *nicht-editierten*, originalen Screenshots in voller Auflösung.

4.1. Kontaktinformationen...

Kontaktinformationen könne nicht über ein Mailformular verwertet werden, sonder nur über eine e-Mail adresse. Dies kann zu Problemen führen, falls am gerade verwendeten Gerät kein konfigurierter e-Mail Client zur Verfügung steht.

Abbildung 4: SucheErgebnis.png

map://www.sagn.../sagn.../sagn.../sagn.../sagn.../sagn.../sagn.../sagn.../sagn.../sagn.../sagn.../

Suchergebnisse für "huber"		
Huber, Christian	c.huber@TU Graz.at	+43 (0) 316 / 873-7474
Huber, Heidrun	huber@zv.tu-graz.ac.at	+43 (0) 316 / 873-0
Huber, Maximilian	maximilian.huber@tugraz.at	+43 (0) 316 / 873-0
Huber, Christoph	christoph.huber@TU Graz.at	+43 (0) 316 / 873-7905; 7908
Huber, Ralph-Peter	ralph.huber@lec.at	+43 (0) 316 / 873-9175; 9176
Huber, Gerhard	gerhard.huber@TU Graz.at	+43 (0) 316 / 873-8367
Huber, Michael Gerhard	huber@mel.tu-graz.ac.at	+43 (0) 316 / 873-0
Neue Suche / New search		

4.2. Es gibt keine Unters...

Es gibt keine Unterscheidung zwischen internen und externen Links. Es wird teilweise auch Information die an und für sich zur selben webseite gehört in neuen Fenstern geöffnet.

4.3. Weiterführende Infor...

Weiterführende Information auf anderen Webseiten wird teilweise in neuen Fenstern, teilweise auch im selben Fenster, oder in iFrames dargestellt.

4.4. Link führt nur zu Qu...

Link führt nur zu Quellcode.Keine Aussage ob falsch verlinkt oder ob Plugin fehlt.

4.5. Die Seite ist kein v...

Die Seite ist kein valides HTML 4.01. Einige Browser könnten bei der Darstellung der Seite Probleme mit der Darstellung haben.

4.6. Das Seitenmenü ist n...

Das Seitenmenü ist nicht einheitlich auf der ganzen Seite und liefert auch keine Rückmeldung über die aktuelle Position des Benutzers auf der Webseite.

5. Gesamtliste der gefundenen Probleme

Zusammengefasste Liste aller gefundenen Probleme in absteigender Reihenfolge der ermittelten Severity.

Legende: Bedeutung der verwendeten Kürzel

Kürzel	Bedeutung
AD	Alois Dengg
HA	Harald Auer
ML	Martin Loitzl
JZ	Joe Zehenter
j	Gefunden durch diesen Evaluator

Schwere	Bedeutung
0	Gar kein Problem
1	Kosmetisches Problem
2	Leichtes Problem
3	Schweres Problem
4	Katastrophales Problem
∅	Durchschnitt

Tabelle 2: Gesamtliste der gefundenen Probleme

Problembeschreibung	Heuristik	Gefunden von				Schwere				
		AD	HA	ML	JZ	AD	HA	ML	JZ	∅
Kontaktinformationen könne nicht über ein Mailformular verwertet werden, sonder nur über eine e-Mail Adresse. Dies kann zu Problemen führen, falls am gerade verwendeten Gerät kein konfigurierter e-Mail Client zur Verfügung steht.	Umkehrbare Aktionen			j	j	3	4	3	3	3.25
Es gibt keine Unterscheidung zwischen internen und externen Links. Es wird teilweise auch Information die an und für sich zur selben webseite gehört in neuen Fenstern geöffnet.	Rückmeldung des Systemzustandes (Feedback)	j	j	j		4	3	4	2	3.25
Weiterführende Information auf anderen Webseiten wird teilweise in neuen Fenstern, teilweise auch im selben Fenster, oder in iFrames dargestellt.	Konsistenz				j	3	2	4	3	3.00

manchen Seite gibt es zum Beispiel einen "Sidebar", auf anderen wird er jedoch nicht verwendet.										
Das Layout der Webseite ändert sich im Verlauf der unterschiedlichen Inhalte ständig.	Konsistenz		j	j	j	3	3	3	1	2.50
Die Suche ist aufgeteilt in die jeweiligen Datenhaltungssysteme und nicht einheitlich. Der Benutzer erwartet jedoch in allen Datenbeständen der TUGraz als Einheit zu suchen. Eine geteilte Suche entspricht nicht dem mentalen Modell des Benutzers.	Konsistenz			j	j	1	3	3	3	2.50
Die Darstellung von Hyperlinks ist inkonsistent. Darüberhinaus sind auch die Darstellung von Überschriften und Inhaltsgliederung inkonsistent. Hyperlinks und reiner Text ist oft nicht zu unterscheiden.	Konsistenz				j	1	3	3	3	2.50
Der Seitentitel gibt keine Rückmeldung über die aktuelle Position auf der Webseite. Die URL und die Navigation jedoch schon. Das macht die Seiten kompliziert zu bookmarken, was die Benutzerfreiheit einschränkt.	Umkehrbare Aktionen			j		3	2	2	2	2.25

Die Seite ist zwar mehrsprachig implementiert, jedoch werden Inhalte die nur in deutscher Sprache vorhanden sind ohne Rückmeldung bei englischer Sprachwahl trotzdem in deutsch angezeigt.	Konsistenz	j	j		j	3	2	1	3	2.25
Die Seite verwendet teilweise zum Anzeigen von externen Inhalte iFrames. Diese sind schwer zu bookmarken, was die Flexibilität des benutzers einschränkt.	Flexibilität und Effizienz			j		2	2	3	1	2.00
Dirket von Fremdanbietern eingebundene Information ist teilweise falsch. Ein Beipsiel ist, das das aktuelle Wetter in Graz angezeigt wird, die Werte jeoch falsch oder nicht aktuell sind.	Ästhetik und minimales Design				j	2	3	1	2	2.00
Diverse Seiten (z.b. Abschnitt Studium) bieten zu viel undstrukturierte Information. Diese Seiten stellen reine Linklisten dar und bieten keine relevante Information.	Ästhetik und minimales Design	j			j	3	1	2	2	2.00
Hilfe nicht vorhanden	Hilfe und Dokumentation	j				2	0	2	3	1.75
Zwar gut aufgebaut Sitemap, aber es gibt Widersprüche zum logischen Aufbau der Seite.	Konsistenz		j			1	2	2	1	1.50

Kein permanentes Aussehen der Startseite. Durch Einbinden von News sieht die Startseite jede Woche anders aus. Grund dafür ist auch, dass die Grafik alles überragt.	Erkennen ist besser als Erinnern		j			1	3	1	0	1.25
Aktuelles: zum Punkt "Aktuelle Themen":- die Überschriften sehen aus wie Links, sind allerdings keine Links.	Konsistenz				j	1	1	2	1	1.25

6. Anhang: Individuelle Problemlisten

Tabelle 3: Liste für Evaluator Alois Dengg

Problembeschreibung
Beim Umschalten der Sprache zwischen deutsch/englisch und englisch/deutsch, wird das Menü und die Titelzeile verändert der Inhalt bleibt immer in deutsch.
Unübersichtliche und uneinheitliche Aufteilung des Inhaltbereiches. zB: Aktuelles - Veranstaltungen
Unnötige Informationen: Wetterbericht Uneinheitliche und unübersichtliche verwendung der linken und rechten Spalte
Übersichtlich und schön gestaltete Sitemap
Sprach nicht konsistent Button schwer erkennbar
Kontakt: Für diese Information muß kein neues Fenster geöffnet werden. Weiters ist die Information nicht ausreichend.
Hilfe nicht vorhanden
Interne und Externe Links nicht markiert. Im gleichen Menü ein interner danache ein externer Link
Fehlermeldung ohne Informationen, ohne Design, andere Sprache

Menü sehr unübersichtlich: Der ausgewählte Hauptmenüpunkt wird nicht markiert
Untermenü in der rechten Spalte (unüblich und gewöhnungsbedürftig)
Untermenü unstrukturiert und inkonsistent auf den einzelnen Seiten. Nach klicken auf den markierten Menüpunkt siehe Bild 2

Feststellen der aktuellen Position nur über Breadcrumb Navigation möglich

Rückgängigmachen einer Aktion nur über Browsernavigation (vor - zurück) möglich

Es wurde versucht so viele Informationen wie möglich anzubieten, dies hat zur Folge dass viele Seiten nur eine lange unübersichtliche Linksammlung sind.

Tabelle 4: Liste für Evaluator Harald Auer

Problembeschreibung
In der englischen Version sollte wenigstens die Startseite komplett in Englisch gehalten sein.
Kein permanentes Aussehen der Startseite. Durch Einbinden von News sieht die Startseite jede Woche anders aus. Grund dafür ist auch, dass die Grafik alles überragt.
Will man auf der Startseite mehr von der aktuellen Information sehen, kommt man auf eine Seite mit stark geändertem Design: A: Header B: Seitenleiste
Über die Seite ist es schwierig wieder zurückzukommen. A: Über diese vermeintliche Möglichkeit kommt man nur zur Hauptseite der Pressestelle B: Es wird ein neues Fenster mit der Startseite geöffnet (passiert bei den anderen Seiten aber nicht)
Link führt nur zu Quellcode. Keine Aussage ob falsch verlinkt oder ob Plugin fehlt
Zwar gut aufgebaut Sitemap, aber es gibt Widersprüche zum logischen Aufbau der Seite
Ich weiß sofort ob es sich rentiert auf die Uni zu gehen und ob es die Witterungseinflüsse zulassen.
Sehr mächtiges Suchportal, das auch TUG-Online einbezieht (leider aber nur in Deutsch)
Wenn ich nach Telefonnummern suche, komme ich auf eine Seite bei der das Design komplett ändert. Zurück komme ich auch nur mehr über die Browsernavigation.
Weiterführende Links werden als solche nur schwer erkannt -> Verlust von Information. Broken links
Vermischung von externen Links mit internen und solchen, die ein neues Fenster öffnen, obwohl die Information zur gleichen Seite gehört.

Wenn schon Sidebars eingeführt werden dann sollen sie auch auf allen Hauptseiten verwendet werden. Manchmal reicht der Text bis an den Rand, dann fehlt wieder mal eine Beschriftung dafür, oder die Reihenfolge ändert sich....

Adresse der Seite in der Adressleiste des Browsers ist "hackable" und spiegelt die Struktur der Seite wieder

Tabelle 5: Liste für Evaluator Martin Loitzl

Problembeschreibung
Es werden externe Inhalte eingebunden, wie im Screenshot vom Büro des Rektors, in dem anderes Link Design verwendet wird.
Die Seite ist kein valides HTML 4.01. Einige Browser könnten bei der Darstellung der Seite Probleme mit der Darstellung haben.
Die Seite wechselt innerhalb der selben Domain www.tugraz.at oft das design. Zum Beispiel die Stadtplan seite.
Die Suche ist aufgeteilt in die jeweiligen Datenhaltungssysteme und nicht einheitlich. Der Benutzer erwartet jedoch in allen Datenbeständen der TUGraz als Einheit zu suchen. Eine geteilte Suche entspricht nicht dem mentalen Modell des Benutzers.
Die Webseite macht keinen Unterschied zwischen Links die zu anderen Seiten führen und links die auf die selbe seite zeigen.
Der Seitentitel gibt keine Rückmeldung über die aktuelle Position auf der Webseite. Die URL und die Navigation jedoch schon. Das macht die Seiten kompliziert zu bookmarken, was die Benutzerfreiheit einschränkt.
Die Kontaktseite bietet kein Mailformular an, sondern nur eine e-Mail adresse. Dies kann zu Problemen führen, falls am gerade verwendeten Gerät kein konfigurierter e-Mail Client zur Verfügung steht.
Die Seite verwendet teilweise zum Anzeigen von externen Inhalten iFrames. Diese sind schwer zu bookmarken, was die Flexibilität des Benutzers einschränkt.
Wenn eine Suche im ersten Suchmodul keine Ergebnisse liefert wird man darauf hingewiesen und bekommt einen Link auf eine neue Suche. Dieser Link führt aber nicht auf die ursprüngliche Suche, sondern auf ein anderes Suchmodul.
Die Website gibt Rückmeldung über die aktuelle Position innerhalb der Site.
Die Webseite funktioniert auch mit deaktiviertem Stylesheet, was sehr gut für Menschen mit körperlichen Einschränkungen ist, die alternative Eingabegeräte verwenden.
Die URL sind permanent verfügbar und der Name gibt Rückmeldung über den Inhalt.

Tabelle 6: Liste für Evaluator Joe Zehenter

Problembeschreibung
\Aktuelles, zum Punkt "Das Wetter": - gute Idee - allerdings wird das Wetter vom Nachmittag (?) angezeigt, es hat jetzt sicher keine 22 Grad... Gleich ned anzeigen, aber falsch schaut echt komisch aus.
\Aktuelles: zum Punkt "Aktuelle Themen": - die Überschriften sehen aus wie Links, sind allerdings keine Links.
Englische Sprache: - keine durchgängige Gestaltung auf Englisch - nicht einmal die Startseite (siehe Screenshots) - zieht sich durch gesammte Seite
\Suche, sehr unübersichtliches Layout durch Mischen von tugraz.at-Design mit TUGOnline, Newserver, Google und Gebäudesuche.
\Suche: Ergebnisse mit Emailadressen und direktem mailto-Link. Emailadressen können einfach ausgelesen werden, sollte Webmailinterface sein, wie bei TUGOnline.
\Suche: beim Suchfeld Nachname bei Tel.-Nr/Mailadressen kann kein Stern eingegeben werden (wie explizit angegeben), es kommt für "hub*"die Meldung, dass für "hub*" keine Einträge gefunden wurden. Bei Firefox funktioniert allerdings, wie gerade zufällig gemerkt.
\Suche "Tel.-Nr / Mailadressen"->Ergebnisse: Feld "Neue Suche / New search" führt auf <http://www.cis.tugraz.at/phonesearch.html>, nicht zurück auf die tugraz.at-Suchseite.
Überschriften sind nicht durchgängig, meistens in Groß-Kleinschreibung, manche nur in Großschreibung, siehe Screenshot.
\Services: anderes Design, nur Links, allerdings in größerer Schrift als sonst, Unterlinks haben wieder richtige Schrift.
\Services: praktisch alle Links Öffnen ein neues Fenster und Führen direkt in TUGOnline, warum wird hier nicht TUGOnline eingebunden, wie sonst so oft.
Aktuelle Position innerhalb der Site einfach festzustellen.
Plötzliches andere Schrift bei einigen Links (Kontakt Sitemap Finde »), passt überhaupt nicht zum Rest.
Menüzeilen, Farbgebung, überhaupt Design sehr durchgängig, übersichtlich.
Mit weiterführenden Links überladene Seiten, zB Rubrik "Studium".
\die TU Graz: Punkt "Links->Organigramm" (rechts unten) führt auf eine JPG-Grafik, warum wird die Grafik nicht in das normale Design eingebunden?

Appendix E

Wikipedia Entries

This Appendix contains Wikipedia entries “Comparison of web application frameworks” and “PHP”.

Comparison of web application frameworks

From Wikipedia, the free encyclopedia

Name	URL	Language	Features
Apache Cocoon	http://cocoon.apache.org/	Java	
Apache Struts	http://struts.apache.org/	Java	
ATK - Achievo Toolkit	http://www.achievo.org/atk/	PHP	
BinaryCloud			
Buxus	http://www.buxus.sk	PHP	LAMP based development framework
Catalyst	http://catalyst.perl.org	Perl	
Django	http://www.djangoproject.com/	Python	object-relational mapper, URL dispatcher, template system, automatic admin interface, database API, pluggable apps, middleware for additional features (e.g. caching, compression, session)
Helma	http://helma.org/	Java, JavaScript	
Horde	http://www.horde.org/	PHP	
JSF	http://java.sun.com/j2ee/javaserverfaces	Java	
Maypole	http://maypole.perl.org/	Perl	
Ozone	http://www.ozoneframework.org/	PHP	
Nexista	http://www.nexista.com/	PHP, XML, XSLT	Nexista is an Open Source php/xml/xslt based development framework for building secure and scalable web applications. It aims to provide a complete development environment that remains flexible and easy to learn.
PageKit			
Quixote	http://www.memsexchange.org/software/quixote/	Python	
RIFE	http://rifers.org/	Java	
Ruby on Rails	http://www.rubyonrails.org/	Ruby	MVC Framework: ActiveRecord for object relational database mapping, ActionController for Controllers and Views: URL dispatching, template languages, etc. It is meant to create web applications fast. AJAX support for Prototype.js/Script.aculo.us.
Spring Framework	http://www.springframework.org/	Java	
Tapestry	http://jakarta.apache.org/tapestry/	Java	
TurboGears	http://www.turbogears.org/	Python	

WebObjects	http://developer.apple.com/webobjects	Java	
WebSphere	http://www.ibm.com/websphere	Java	
Zope	http://www.zope.org/	Python	
PRADO	http://www.xisc.com/	PHP	
ZK	http://zk1.sourceforge.net Live demo (http://www.potix.com/zkdemo/userguide)	Java	an Ajax-based event-driven engine to automate interactivity, a rich set of XUL and XHTML components to enrich usability, and a markup language to simplify the UI design.

See also

- Web application framework
- List of web application frameworks

Retrieved from "http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks"

Categories: Web application frameworks | Software comparison

- This page was last modified 23:48, 6 February 2006.
- All text is available under the terms of the GNU Free Documentation License (see **Copyrights** for details).
- Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc.
- Privacy policy
- About Wikipedia
- Disclaimers

PHP

From Wikipedia, the free encyclopedia
(Redirected from Php)

PHP is a programming language that can be used to create websites. Short for "**PHP: Hypertext Preprocessor**", it is an open-source, reflective programming language used mainly for developing server-side applications and dynamic web content, and more recently, a broader range of software applications.

PHP allows interaction with a large number of relational database management systems, such as MySQL, Oracle, IBM DB2, Microsoft SQL Server, PostgreSQL, and SQLite. PHP runs on most major operating systems, including Unix, Linux, Windows, and Mac OS X, and can interact with many major web servers. The official PHP website (<http://www.php.net/>) contains very extensive documentation (<http://www.php.net/manual/>).

There is a command line interface (CLI), as well as GUI libraries such as the Gimp Tool Kit (GTK+) and text mode libraries like Neurses and Newt.

PHP is the result of the efforts of many contributors. It is licensed under the PHP License, a BSD-style license. Since version 4, it has been powered by the Zend engine.

Contents

- 1 History
 - 1.1 Support for objects
- 2 Usage
- 3 Syntax
 - 3.1 Sample code
 - 3.1.1 Hello world
 - 3.1.2 phpinfo
 - 3.1.3 99 Bottles of Beer
 - 3.2 Data structures
 - 3.2.1 Arrays
 - 3.2.2 Strings
 - 3.2.3 Boolean
 - 3.2.4 Integer
 - 3.3 Objects
 - 3.4 Whitespace
 - 3.5 Comments
- 4 Resources
 - 4.1 Libraries
 - 4.2 Source code encoders
- 5 Support
- 6 Criticism
 - 6.1 Syntax
 - 6.2 Built-in functions
 - 6.3 Security
 - 6.4 Miscellaneous
- 7 See also

- 8 References
- 9 External links
 - 9.1 PHP.net (official PHP website)
 - 9.2 Reference material
 - 9.3 Security
 - 9.4 Books
 - 9.5 Training
 - 9.6 Support
 - 9.7 Resources
 - 9.8 Advocacy

History

PHP was originally designed as a small set of Perl scripts, followed by a rewritten set of CGI binaries written in C by the Danish-Canadian programmer Rasmus Lerdorf in 1994 to display his résumé and to collect certain data, such as how much traffic his page was receiving. "Personal Home Page Tools" was publicly released (<http://groups.google.ch/group/comp.infosystems.www.authoring.cgi/msg/cc7d43454d64d133?oe=UTF-8&output=gplain>) on 8 June 1995 after Lerdorf combined it with his own Form Interpreter to create PHP/FI.

Zeev Suraski and Andi Gutmans, two Israeli developers of the Technion - Israel Institute of Technology, rewrote the parser in 1997 and formed the base of PHP 3, changing the language's name to the recursive acronym "**PHP: Hypertext Preprocessor**". The development team officially released PHP/FI 2 in November 1997 after months of beta testing. Public testing of PHP 3 began immediately and the official launch came in June 1998. Suraski and Gutmans then started a new rewrite of PHP's core, producing the Zend engine in 1999 (a page at www.zend.com (<http://www.zend.com/zend-engine-summary.php>) states that PHP 3 was powered by Zend Engine 0.5). They also founded Zend Technologies in Ramat Gan, Israel which has since overseen the PHP advances.

In May 2000, PHP 4, powered by the Zend Engine 1.0, was released.

On July 13, 2004, PHP 5 was released, powered by Zend Engine II. PHP 5 includes new features such as PHP Data Objects (PDO) and more performance enhancements taking advantage of the new engine.

Support for objects

Up until version 3, PHP had no object-oriented features. In version 3 basic object functionality was added. The same semantics were implemented in PHP 4 as well as pass-by-reference and return-by-reference for objects but the implementation still lacked the powerful and useful features of other object-oriented languages like C++ and Java.

In version 5, which was released in July 2004, PHP's object-oriented functionality has been very much enhanced and is more robust and complete.

Usage

The LAMP architecture (short for Linux, Apache, MySQL, PHP) has become popular in the Web industry as a way of deploying inexpensive, reliable, scalable, secure web applications (the 'P' in LAMP can also stand for Perl or Python).

Examples of PHP applications include phpBB and MediaWiki.

The PHP model can be seen as an alternative to Microsoft's ASP.NET/C#/VB.NET system, Macromedia's ColdFusion system, Sun Microsystems' JSP/Java system, the Zope/Python system, the Mod perl/Perl system, and more recently the Ruby on Rails framework.

Syntax

Sample code

Hello world

Here is a Hello World code example (See *Basic syntax* (<http://php.net/manual/en/language-basic-syntax.php>) in the PHP manual):

```
<?php
echo 'Hello World!';
?>

<?php
file_put_contents('php://stdout', 'Hello World!');
?>

<?php
$f=fopen('php://stdout');
fwrite($f, 'Hello World!');
fclose($f);
?>
```

The (arguably) simplest "Hello World" program in PHP is:

```
<?print 'Hello World!';?>
```

A slightly smaller example:

```
<?='Hello World!';?>
```

Or, arguably (since text not closed in `<? ?>` tags is echoed:

```
Hello World
```

phpinfo

This simple code snippet displays information about PHP's configuration settings, loaded modules, variables, and more:

```
<?php
phpinfo();
?>
```

99 Bottles of Beer

For an example of 99 Bottles of Beer, see [99_Bottles_of_Beer_computer_program#PHP](#).

Data structures

Arrays

```
$myArray = array('key 1' => 'string value 1', 2005, new stdClass());
```

[1] (<http://www.php.net/manual/en/language-types.array.php>)

Arrays are heterogeneous, meaning a single array can contain objects of more than one type. They can be used as ordered lists of only values and as hashes with keys and values, even simultaneously.

Strings

Variables are evaluated inside double quotation marks (" "), but not inside single quotation marks (' '). Functions and other expressions are not evaluated inside double quotes but can be added to strings using periods for concatenation.

```
$myString = 'string' . function() . 'rest of string'; [2]
(http://php.net/manual/en/language.variables.php)
```

A period (.) concatenates strings together. [3] (<http://php.net/manual/en/language-types.string.php>)

Boolean

Boolean variables have two possible values **True** and **False**

```
$myBoolean = True; $myBoolean = False;
```

Another form of boolean would be replacing true and false with 1 and 0 (respectively) but is not a standard of coding.

Integer

Integers are number variables. They are whole numbers that can be positive, negative, octal, and hexadecimal

```
$decimal = 1234; // decimal number
```

```
$negative = -123; // a negative number
```

```
$octal = 0123; // octal number (equivalent to 83 decimal)
```

```
$hexadecimal = 0x1A; // hexadecimal number (equivalent to 26 decimal)
```

Objects

Support for object-oriented programming in PHP 5 (from the PHP Manual):

New Object Model

PHP's handling of objects has been completely rewritten, allowing for better performance and more features. In previous versions of PHP, objects were handled like primitive types (for instance integers and strings). The drawback of this method was that semantically the whole object was copied when a variable was assigned, or passed as a parameter to a method. In the new approach, objects are referenced by handle, and not by value (one can think of a handle as an object's identifier).

Private and Protected Members

PHP 5 introduces private and protected member variables, they allow you to define the visibility of class properties.

Private and Protected Methods

Private and protected methods are also introduced.

Abstract Classes and Methods

PHP 5 also introduces abstract classes and abstract methods. An abstract method only declares the method's signature and does not provide an implementation. A class that contains abstract methods needs to be declared an abstract class.

Interfaces

A class may implement an arbitrary list of interfaces.

Object Cloning

If the developer asks to create a copy of an object by using the reserved word *clone*, the Zend engine will check if a `__clone()` method has been defined or not. If not, it will call a default `__clone()` which will copy all of the object's properties. If a `__clone()` method is defined, then it will be responsible to set the necessary properties in the created object. For convenience, the engine will supply a function that imports all of the properties from the source object, so that they can start with a by-value replica of the source object, and only override properties that need to be changed.

Unified Constructors

PHP 5 introduces a standard way of declaring constructor methods by calling them by the name `__construct()`.

Destructors

PHP 5 introduces a destructor concept similar to that of other object-oriented languages, such as C++: When the last reference to an object is destroyed, the object's destructor (a class method named `__destruct()` that receives no parameters) is called before the object is freed from memory.

Exceptions

PHP 4 had no exception handling. PHP 5 introduces an exception model similar to that of other programming languages.

More additions and examples of the additions mentioned above are available in the *Classes and Objects* chapter (<http://php.net/manual/en/language.oop5.php>) of the PHP 5 manual.

It should be noted that the static method and class variable features in Zend Engine 2 do not work the way some expect. There is no virtual table feature in the Engine, so the static variables are bound with a name at compile time instead of with a reference. This can lead to unexpected behavior; if you do not understand this.

Here is an example of creating an object in PHP5:

```
<?php
class Car {
    public $miles; // variable that can be accessed outside the class
    private $mpg; // variable that can only be accessed within the class
    protected $mph; // variable that can only be accessed from within the class, and
                    // from any inherited child classes

    public function __construct($param) { // is called when object "Car" is created
        dosomething($param);
    }

    public function startcar() { // methods can be public, private, or protected
        starttheacar();
    }

    public function stopcar() {
        stoptheacar();
    }

    public function getMpg() {
        return $this->mpg;
    }
}

$car = new Car($param);
echo $car->miles; // echos the value of the property "miles" of the class "Car"
?>
```

Whitespace

PHP treats new lines as whitespace, in the manner of a free-form language (except when inside string quotes). Statements are terminated only by a semicolon (;) except in a few special cases. [4] (<http://php.net/manual/en/language.basic-syntax.instruction-separation.php>)

Comments

```
// comment
/*
 * multi-line comment
 */
# comment -- its use is discouraged
```

Resources

Libraries

PHP includes a large number of free and open-source libraries with the core build. PHP is a fundamentally Internet-aware system with modules built in for accessing FTP servers, many database servers, embedded SQL libraries like embedded MySQL and SQLite, LDAP servers, and others. Many functions familiar to C programmers such as the `printf` family are available in the standard PHP build.

PHP extensions exist which, among other features, add support for the Windows API, process management on Unix-like operating systems, cURL, and the ZIP/gzip/bzip2/RAR/LZF compression formats. Some of the more unusual features are on-the-fly Macromedia Flash generation, integration with Internet relay chat, and generation of dynamic images (where the content of the image can be changed). Some additional extensions are available via the PHP Extension Community Library (PECL).

This is the present list of all officially documented libraries: [5] (<http://www.php.net/manual/en/>)

- Apache
- BCMath
- Bzip2
- Calendars
- CCVS
- COM
- ClibPDF
- cURL
- Cybercash
- DB2
- DBM
- dbx
- DB++
- DOM XML
- NET
- FrontBase
- filePro
- FriBiDi
- FTP
- Gettext
- GD Graphics Library
- GNU Multi-Precision Library
- Hyperwave
- iconv
- Informix
- Ingres II
- InterBase
- IRC
- LDAP
- Lotus Notes
- mailparse
- MCAL
- Mcrypt
- MCVE
- Mhash
- Mimetype Functions
- MS-SQL
- Ming
- mnoGoSearch
- mSQL
- MySQL
- Mowhawk
- muscat
- Ncurses
- ODBC
- Oracle
- GD Graphics Library
- OpenSSL
- Ovrimos SQL
- PDF
- PayFlow Pro
- PDO
- PostgreSQL
- Printer
- Pspell
- GNU Readline
- GNU Recode
- Regular expressions
- QT-Dom
- Semaphores
- SESAM
- Session Handling
- Shared memory
- SNMP
- Sockets
- SQLite
- Streams
- Sybase
- Token
- vpopmail
- WDDX
- Win32 API
- XML (XPath)
- XML-RPC
- XSLT
- YAZ
- Yellow Pages / NIS

- IMAP, POP3 and NNTP
- POSIX
- ZIP
- Zlib

Source code encoders

Encoders offer some protection for intellectual property by hindering source code reverse engineering.

These include SourceGuardian PHP Encoder (<http://www.sourceguardian.com>) , IonCube (<http://www.ioncube.com>) and Zend (<http://www.zend.com>) .

Support

PHP has a formal development manual (<http://www.php.net/docs.php>) that is maintained by the open source community. In addition, answers to most questions can often be found by doing a simple internet search. PHP users assist each other through various media such as chat, forums, newsgroups and PHP developer web sites. In turn, the PHP development team actively participates in such communities, garnering assistance from them in their own development effort (PHP itself) and providing assistance to them as well. There are many help resources available for the novice PHP programmer.

Criticism

Criticisms of PHP include those general criticisms ascribed to other scripting programming languages and dynamically typed languages. In addition, specific criticisms of PHP include:

Syntax

- PHP does not enforce the declaration of variables, and variables that have not been initialized can have operations (such as concatenation) performed on them; however, an operation on an uninitialized variable does raise an E_NOTICE level error, errors that are hidden by default. This leads to security holes with register_globals (not on by default), as mentioned below. See also error_reporting(<http://php.net/manual/en/function.error-reporting.php>) .
- Within sections of the built-in function selection there is little or no consistency regarding argument order (examples: order of subject array and other data for array handling functions, order of needle and haystack in various search functions).
- Variables in PHP are not limited to one type. It is possible to assign an integer value to the variable \$Q, then assign a string value, and then assign an array to it. Functions are also not allowed to (directly) force the types of their arguments, and overloading is not allowed. This can often lead to difficult-to-debug code.
- Type checking is not strict. In most languages "0" (the string "0") is not equivalent to 0 (the integer zero), which in turn is not equal to the value FALSE (which is the boolean value for 0). In PHP, this is not the case unless strict comparisons are used.

```

<code>
</code>

```

Built-in functions

- Built-in function names have no standard form, with some employing underscores (e.g. strip_tags (http://www.php.net/strip_tags) , html_entity_decode (http://www.php.net/html_entity_decode)) while

others do not (e.g. stripslashes (<http://www.php.net/stripslashes>) , htmlentities (<http://www.php.net/htmlentities>)). Furthermore, some functions are verb_noun() while others are noun_verb() and some are prefixed_byModuleName while others use a module_suffix_scheme. Although all new functions do follow a naming standard, old names remain for backward compatibility reasons.

- Some functions have inconsistent output. Statements like *This function may return Boolean FALSE, but may also return a non-Boolean value which evaluates to FALSE, such as 0 or ""* can be found in the documentation (<http://php.net/manual/en/function.stripslashes.php>) . This is related to PHP's dynamic typing. A workaround is using strict (===) type checking as opposed to loose (==) . See also the manual on type juggling (<http://php.net/manual/en/language.types.type-juggling.php>) .

- The number of built-in functions is said to be too numerous, with many functions performing the same actions, but with just slightly different data, results, etc. This is said to make it difficult to program in the language without the frequent consultation of a reference work.
- There are over 3,000 functions, sharing the same global namespace. Most functions are not enabled by default, but become available when PHP is linked against the required libraries. To mitigate this, function names are usually prefixed with their library name.
- There is a "magic quotes" feature that inserts backslashes into user input strings. The feature was introduced to reduce code written by beginners from being dangerous (such as in SQL injection attacks), but some criticize it as a frequent cause of improperly displayed text or encouraging beginners to write PHP which is vulnerable to SQL-injection when used on a system with it turned off. (Always be sure to check for "magic-quotes": `get_magic_quotes_gpc()` ; (http://www.php.net/get_magic_quotes_gpc) and to unset "magic-quotes-runtime": `set_magic_quotes_runtime(0)` ; (http://www.php.net/set_magic_quotes_runtime)) Magic Quotes Runtime defaults to Off. The default Magic Quotes GPC setting is determined by which of the PHP ini files that you use. It is On by default in php.ini-dist, and Off in php.ini-recommended. For more information, see the security section in the *Magic Quotes* chapter (<http://php.net/manual/en/security.magicquotes.php>) of the PHP manual.

Security

- If register_globals (http://www.php.net/register_globals) is enabled in PHP's configuration file, PHP automatically puts the values of Post, Get, Cookie and Session Parameters into standard variables, which can be a significant security risk for scripts that assume those variables are undefined. As of version 4.2.0 (http://php.net/release_4_2_0.php) register_globals defaults to off. For more information, see the security section in the *Using Register Globals* chapter (http://php.net/manual/en/security_globals.php) of the PHP manual.
- Other languages, such as ASP.NET, include functionality to detect and clean harmful cross-site scripting or other malicious code automatically, whereas PHP does not. See also strip_tags(<http://php.net/manual/en/function.strip-tags.php>) .
- In the majority of cases, Linux and Unix web servers with PHP installed (using mod_php) typically run PHP scripts as "nobody", which can make file security in a shared hosting environment difficult. PHP's **Safe Mode** can emulate the security behavior of the OS to partially overcome this problem.

Miscellaneous

- Error messages are said to be confusing, although this is a common criticism levelled at many programming languages. For further information, see the manual section on PHP parser tokens (<http://php.net/manual/en/tokens.php>) . (PHP's error reporting when set to report all has especially useful tips. The only one that might be confusing to a new programmer is the one where it expects a semi colon at the end of your page. This is normally caused by not closing a brace somewhere in your code.)
- The many settings in the PHP interpreter's configuration file (*php.ini*) mean that code that works with one installation of PHP might not work with another. For example, if code is written to work with register_globals turned on, it won't work on another system that has register_globals turned off. This makes it necessary to write code that is cross-platform compatible by assuming that register_globals will be off and therefore calling a global variable with its prefix in front of its name, such as \$_POST['variable'], \$_SERVER['variable'] and \$_COOKIE['variable']—not, simply, \$variable. For more information, see the manual page on using external variables (<http://php.net/manual/en/language.variables.external.php>) .

- Some PHP extensions use libraries that are not threadsafe, so rendering with Apache 2's Multithreaded MPM (<http://httpd.apache.org/docs-2.0/mpm.html>) (multi-processing module) may cause crashes.
- There is no native support for Unicode or multibyte strings (mbstring is provided as an extension). This is an improvement planned for the next major revision (5.5 or 6.0).
- A library has been developed called WAMP or Web Application Structure for PHP that delivers a three-tier framework for PHP web applications more similar to other formal enterprise products like WLS (Web Logic Server) instead of PHP's usual script-style, code-lacker-friendly approach.
- PHP5 is not fully backward compatible with PHP4.

See also

- Paamayim Nekudotayim
- Standard PHP Library

References

- Jason E. Sweat. *Guide to PHP Design Patterns*. PHIarchitecture. 2005. ISBN 0973589825.
- Ilia Alshanetsky. *Guide to PHP Security*. PHIarchitecture. 2005. ISBN 0973862106.
- Chris Shiflett. *Essential PHP Security*. O'Reilly Media. 2005. ISBN 059606565X.
- Larry Ullman. *PHP and MySQL for Dynamic Web Sites*. Peachpit Press. 1st edition, 2003. ISBN 0321186486.

External links

PHP.net (official PHP website)

- PHP: Hypertext Preprocessor (<http://www.php.net/>)
- Selected sub-pages of php.net:
 - PHP license information (<http://www.php.net/license/>)
 - PHP.net domain-based graph of PHP deployment (<http://www.php.net/usage.php>)
 - When using the PHP.net website, access the content you would like to see quickly (<http://www.php.net/urhowto.php>)
- Selected sub-domains of php.net (<http://www.php.net/sites.php>) :
 - PEAR: PHP Extension and Application Repository (<http://pear.php.net/>)
 - PECL: PHP Extension Community Library (<http://pecl.php.net/>)
 - Smarty: Template Engine (<http://smarty.php.net/>)
 - PHP-GTK extension providing an object-oriented interface to GTK+ classes and functions (<http://gtk.php.net/>)
 - PHP Presents (<http://talks.php.net/>) and PHP Conference Material Site (<http://conf.php.net/>) — Sites collecting slides of talks given by well known people from the PHP community. The former one is also known as *prea2* (version 2 of the latter).

Reference material

- PHP Cheat Sheet (<http://www.ilovejackdaniels.com/php/php-cheat-sheet/>) - A quick reference for PHP

Security

- Sitepoint's Top 7 Security Blunders (<http://www.sitepoint.com/article/php-security-blunders>) with PHP.
- The OWASP Top Ten Security Vulnerabilities (<http://www.sklar.com/page/article/owasp-top-ten>) as applied to PHP.
- PHP Security Consortium (<http://phpsec.org/>) — International group of PHP experts dedicated to promoting secure programming practices.
- WACT: PHP Application Security Wiki (http://www.phpwact.org/security/web_application_security) — The Web Application Component Toolkit's wiki page on PHP security resources.

- Hardened PHP Project (<http://www.hardened-php.net>) — Patchset that adds security hardening features to PHP.

Books

- PHP in a Nutshell (<http://hndzilla.org/phpbook>)
- Essential PHP Security (<http://phpsecurity.org/>)
- Computer-Books.us (<http://www.computer-books.us/php.php>) — A collection of PHP books available for free download
- PHP Book reviews (<http://www.phpclasses.org/reviews>) — Reviews of PHP books and others of related interests
- PHP Book Reviews (http://www.php-editors.com/php_book_reviews.php)

Training

- Official PHP Certification and training (<http://www.zend.com/education/>)

Support

- Wikis
 - PHP Wiki (http://www.wiki.cc/php/Main_Page)
 - Php Documentation Wiki (<http://info.phpnixus.net/>)
- Forums
 - DMOZ Open Directory
 - Webmaster Forums (http://dmoz.org/Computers/Internet/Web_Design_and_Development/Chats_and_Forums/) at the Open Directory Project
 - Dev Network (<http://www.devnetwork.net>)
 - Official PHP Support Page (<http://php.net/support.php>)
 - PHP Freaks (<http://www.phpfreaks.com>)
 - Dev Shed (<http://forums.devshed.com>)
 - PHPBuilder (<http://www.phpbuilder.com>)
 - Codewalkers (<http://codewalkers.com/forum/index.php>)
 - User groups
 - Directory of country specific PHP user groups (<http://www.phpclasses.org/browse/group/>), from the users of the PHP Classes site.
 - New York PHP (<http://www.nyphp.org/>), the largest user group in North America (they maintain several active mailing lists (<http://www.nyphp.org/content/maillinglist/mlist.php>)).
 - Blogs
 - Planet PHP (<http://planet-php.net/>)
 - Piku's PHP Blog (<http://www.madphp.net/piku/>)
 - Chris Shiflett: The PHP Blog (<http://shiflett.org/>)
 - whenpenguinsattack (<http://www.whenpenguinsattack.com>)
 - California
 - OCGPHP - Orange County PHP User Group (<http://ocphp.lucentminds.com/>) (with a smile)
 - Newsgroups / Usenet
 - comp.lang.php (<http://groups.google.com/group/comp.lang.php>)
 - Chat / IRC
 - IRC channels #php, #phphelp and #phpromania on EFNet, IRCNet, DALnet and other networks.
 - Other IRC channels include #php (<irc://irc.freenode.net/#php>) and #phpfreaks (<irc://irc.freenode.net/phpfreaks>) on freenode.
 - Mailing lists / Listservers
 - PHP mailing lists (<http://www.php.net/mailling-lists.php>)
 - PHP.net's news server (<news://news.php.net/>)

Resources

- Open Directory Project:
 - PHP (<http://www.dmoz.org/Computers/Programming/Languages/PHP/>)
 - Frameworks (<http://www.dmoz.org/Computers/Programming/Languages/PHP/Scripts/Frameworks/>)
 - Integrated development environments, debuggers and other tools (<http://www.dmoz.org/Computers/Programming/Languages/PHP/Tutorials/>)
 - Tutorials (<http://www.dmoz.org/Computers/Programming/Languages/PHP/Tutorials/>)
 - General resource sites (<http://www.dmoz.org/Computers/Programming/Languages/PHP/Resources/>)
- PHP for GUI Desktop Application Development
 - PHP-GTK (<http://gtk.php.net/>)
 - WinBinder (<http://www.hypervisual.com/wmbinder/>)
 - PHP-QT (<http://php-qt.berlios.de/>)
 - gambArt (<http://www.klorofil.org/>)
- PHP Classes (<http://www.phpclasses.org>) — Ready to use PHP components in the form of classes of objects (registration required) and Book Reviews
- PHP Developer's Network (<http://forums.devnetwork.net/>)
- PHP Editors and IDE Reviews (<http://www.php-editors.com/>) - Website dedicated to PHP Editors and IDE's
- PHP Freaks (<http://www.phpfreaks.com>)
- PHP Frequently Asked Questions (<http://php.faqs.com>)
- PHPXref.com: Use the Source (<http://phpxref.com/>) - A repository of cross referenced PHP applications
- BioPHP.org (<http://biophp.org/>) - PHP for bioinformatics
- Codewalkers (<http://www.codewalkers.com>)

Advocacy

- Experiences of Using PHP in Large Websites (<http://www.ukuug.org/events/linux2002/papers/html/php/>)
- Making the Case for PHP at Yahoo! (<http://public.yahoo.com/~radwin/talks/yahoo-phpcom2002.htm>) — Michael J. Radwin, Yahoo! Engineer explains why Yahoo! has chosen PHP over Apache mod_include (http://httpd.apache.org/docs/mod/mod_include.html) , ASP, ColdFusion, Perl, JSP, Servlets, J2EE, XSLT and ClearSilver:
 - Follow-up: One Year of PHP at Yahoo! (<http://public.yahoo.com/~radwin/talks/one-year-of-php-oscon2003.htm>)
- Newsforge report (<http://newsforge.com/newsforge/02/06/11/011243.shtml?tid=5>) of Netcraft web host survey that says PHP is widely used
- The PHP Scalability Myth (http://www.onjava.com/pub/a/onjava/2003/10/15/php_scalability.html) — Jack Herrington
- PHP Seales (<http://shifflett.org/archive/46>) — Chris Shifflett

<p>Major programming languages (more) (edit (http://en.wikipedia.org/w/index.php?title=Template:Major_programming_languages&action=edit))</p> <p><i>Industrial:</i> ABAP Ada AWK Assembly C C++ C# COBOL Common Lisp Delphi Eiffel Fortran Java JavaScript Lisp Objective-C Pascal Perl PHP SQL PowerBuilder Python RPG Ruby SAS Smalltalk T-SQL Tcl Visual Basic VB.NET Visual FoxPro</p> <p><i>Academic:</i> APL J Haskell Logo ML Prolog Scheme</p>	<p><i>Other:</i> ALGOL BASIC Clipper Modula-2 Modula-3 MUMPS PL/I Simula</p>
---	--

Retrieved from "http://en.wikipedia.org/wiki/PHP"

Categories: Wikipedia spam cleanup | Curly bracket programming languages | Free software | Imperative programming languages | Object-oriented programming languages | PHP programming language | Procedural programming languages | Programming languages | Scripting languages

This page was last modified 20:59, 20 February 2006.

- All text is available under the terms of the GNU Free Documentation License (see **Copyrights** for details).
- Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc.
- Privacy policy
- About Wikipedia
- Disclaimers

Bibliography

- Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, and Shlomo Angel [1977]. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, New York. ISBN 0195019199. (Cited on page 57.)
- Keith Andrews [2004]. *Writing a Thesis: Guidelines for Writing a Master's Thesis in Computer Science*. Graz University of Technology, Austria. <ftp://ftp2.iicm.edu/pub/keith/thesis/thesis.zip>. (Cited on page iii.)
- Keith Andrews [2005]. *Skeleton Heuristic Evaluation Report*. <http://courses.iicm.edu/hci/practicals/materials/en/he/he.html>. (Cited on pages 33 and 34.)
- Keith Andrews [2006]. *Lecture Notes: Human-Computer Interaction*. Graz University of Technology, Austria. <http://courses.iicm.edu/hci/>. (Cited on pages 6, 7, 9, 11, 33 and 142.)
- Apple Computer Inc. [1992]. *MacIntosh Human Interface Guidelines*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN 0201622165. (Cited on pages 21 and 44.)
- Kent Beck [2000]. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN 0201616416. (Cited on pages 22 and 84.)
- Kent Beck and Ward Cunningham [1987]. *Using Pattern Languages for Object-Oriented Programs*. OOPSLA 87. <http://c2.com/doc/oopsla87.html>. (Cited on page 57.)
- Kent Beck, Ron Crocker, Gerard Meszaros, John Vlissides, James O. Coplien, Lutz Dominick, and Frances Paulisch [1996]. *Industrial Experience with Design Patterns*. In *ICSE '96: Proceedings of the 18th International Conference on Software Engineering*, pages 103–114. IEEE Computer Society, Washington, DC, USA. ISBN 0818672463. (Cited on page 57.)
- Randolph G. Bias [1994]. *The Pluralistic Usability Walkthrough: Coordinated Empathies*, pages 63–76 in Nielsen and Mack [1994]. John Wiley and Sons, Inc., New York, NY, USA. ISBN 0471018775. (Cited on page 21.)
- Randolph G. Bias and Deborah J. Mayhew (Editors) [1994]. *Cost-Justifying Usability*. Academic Press, Inc., Orlando, FL, USA. ISBN 0120958104. (Cited on page 200.)
- Randolph G. Bias and Deborah J. Mayhew [2005]. *Cost-Justifying Usability, Second Edition: An Update for the Internet Age*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 0120958112. (Cited on pages 9 and 12.)
- Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal [1996]. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley and Sons, Inc., New York, NY, USA. ISBN 0471958697. (Cited on pages 58, 59, 60, 64, 66 and 68.)

- Gilbert Cockton, Darryn Lavery, and Alan Woolrych [2003]. *Inspection-Based Evaluations*. The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications, pages 1118–1138. (Cited on page 21.)
- Alan Cooper [2004]. *The Inmates Are Running the Asylum : Why High Tech Products Drive Us Crazy and How to Restore the Sanity*. Sams Publishing, Indianapolis, IN, USA, second edition. ISBN 0672326140. (Cited on pages 7 and 9.)
- James O. Coplien [1992]. *Advanced C++ Programming Styles and Idioms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN 0201548550. (Cited on page 57.)
- Donald Allan Cox [1998]. *Supporting Results Synthesis in Heuristic Evaluation*. Master's thesis, University of Calgary, Department of Computer Science. (Cited on pages 32, 36 and 37.)
- Joseph S. Dumas and Janice C. Redish [1993]. *A Practical Guide to Usability Testing*. Intellect, Norwood, NJ. ISBN 1841500208. (Cited on page 10.)
- Al Fama [2005]. *PHP Security Guide*. PHP Security Consortium Website. <http://phpsec.org/projects/guide/>. (Cited on page 75.)
- Mohamed Fayad and Douglas C. Schmidt [1997]. *Object-Oriented Application Frameworks*. Communications of the ACM, 40(10), pages 32–38. ISSN 0001-0782. doi:10.1145/262793.262798. (Cited on page 155.)
- Andy Field and Graham J. Hole [2002]. *How to Design and Report Experiments*. Sage Publications Ltd., London, UK. ISBN 0761973834. (Cited on page 11.)
- Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides [1995]. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN 0201633612. (Cited on pages iii, ix, xi, 57, 58, 59, 60, 61, 62, 64, 70 and 155.)
- Jesse James Garrett [2005]. *Ajax: A New Approach to Web Applications*. <http://adaptivepath.com/publications/essays/archives/000385.php>. (Cited on page 136.)
- JoAnn T. Hackos and Janice C. Redish [1998]. *User and Task Analysis for Interface Design*. John Wiley and Sons, Inc., New York, NY, USA. ISBN 0471178314. (Cited on page 7.)
- James Hom [1998]. *The Usability Methods Toolbox*. <http://jthom.best.vwh.net/usability/index.htm>. (Cited on pages 10, 11 and 27.)
- IETF [1981]. *RFC 793: Transmission Control Protocol DARPA Internet Program Protocol Specification*. <http://www.faqs.org/rfcs/rfc793.html>. (Cited on page 67.)
- Keith Instone [1997]. *Site Usability Heuristics for the Web*. WebReview.com. <http://user-experience.org/uefiles/writings/heuristics.html>. (Cited on page 43.)
- ISO 13407 [1999]. *Human-Centred Design Processes for Interactive Systems*. <http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=21197>. (Cited on pages 3 and 4.)
- ISO 9241-11 [1998]. *Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs) – Part 11: Guidance on Usability*. <http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=16883&ICS1=13&ICS2=180&ICS3=>. (Cited on pages 3, 5 and 21.)
- IUSR [1999]. *Industry Usability Reporting Project: Common Industry Format for Usability Test Reports*. <http://zing.ncsl.nist.gov/iusr/documents/cifv1.1b.htm>. (Cited on pages 33 and 34.)

- Mohammed J. Kabir [2003]. *Secure PHP Development: Building 50 Practical Applications*. John Wiley and Sons, Inc., New York, NY, USA. ISBN 0764549669. (Cited on pages 73, 82, 107 and 155.)
- Michael J. Kahn and Amanda Prail [1994]. *Formal Usability Inspections*, pages 141–171 in Nielsen and Mack [1994]. John Wiley and Sons, Inc., New York, NY, USA. ISBN 0471018775. (Cited on pages 27 and 28.)
- Marja-Riitta Koivunen and Charles McCathieNevile [2001]. *Accessible Graphics and Multimedia on the Web*. Website of the World Wide Web Consortium (W3C). <http://www.w3.org/2001/05/hfweb/heuristics.htm>. (Cited on page 45.)
- Mike Kuniavsky [2003]. *Observing the User Experience: A Practitioner's Guide to User Research*. Morgan Kaufmann Series in Interactive Technologies, San Francisco, CA, USA. ISBN 1558609237. (Cited on page 7.)
- James Landay [2005]. *Denim: An Informal Tool For Early Stage Web Site and UI Design*. <http://dub.washington.edu/denim/>. (Cited on page 9.)
- Clayton Lewis, Peter G. Polson, John Rieman, and Cathleen Wharton [1992]. *Cognitive Walkthroughs: A Method for Theory-Based Evaluation of User Interfaces*. International Journal of Man-Machine Studies, 36(5), pages 741–773. ISSN 0020-7373. doi:10.1016/0020-7373(92)90039-N. (Cited on page 16.)
- Macromedia [2004]. *Macromedia Director MX 2004*. <http://www.macromedia.com/software/director/>. (Cited on page 9.)
- Reenal Mahajan [2003]. *A Usability Problem Diagnosis Tool: Development and Formative Evaluation*. Master's thesis, Virginia Polytechnic Institute and State University. (Cited on page 37.)
- Marilyn M. Mantei and Toby J. Teorey [1988]. *Cost/Benefit Analysis for Incorporating Human Factors in the Software Lifecycle*. Communications of the ACM, 31(4), pages 428–439. ISSN 0001-0782. doi:10.1145/42404.42408. (Cited on page 12.)
- Aaron Marcus [2002]. *Return on Investment for Usable User-Interface Design: Examples and Statistics*. Aaron Marcus and Associates, Inc. http://www.amanda.com/resources/ROI/AMA_ROIWhitePaper_28Feb02.pdf. (Cited on page 9.)
- Roger E. Masse, G. V. Suresh, and Jim Zahniser [1998]. *A Guide to Usability Engineering Web Resources*. <http://www.otal.umd.edu/guse/>. (Cited on page 49.)
- Microsoft Corporation [1999]. *Microsoft Windows User Experience (Microsoft Professional Edition)*. Microsoft Press, Redmond, WA, USA. ISBN 0735605661. Foreword By- Windows User Experience Team. (Cited on pages 21 and 44.)
- Günter Modes [2005]. *A Usage Study of Desktop Users*. Master's thesis, Graz University of Technology, Institute for Engineering- and Business Informatics. (Cited on page 12.)
- Rolf Molich and Jakob Nielsen [1990]. *Improving a Human-Computer Dialogue*. Communications of the ACM, 33(3), pages 338–348. ISSN 0001-0782. doi:10.1145/77481.77486. (Cited on pages 34 and 39.)
- Michael J. Muller and Anne McClard [1995]. *Validating an Extension to Participatory Heuristic Evaluation: Quality of Work and Quality of Work Life*. In *CHI '95: Conference Companion on Human Factors in Computing Systems*, pages 115–116. ACM Press, New York, NY, USA. ISBN 0897917553. doi:10.1145/223355.223457. (Cited on pages 16 and 41.)

- Michael J. Muller, Lisa Matheson, Colleen Page, and Robert Gallup [1998]. *Methods & Tools: Participatory Heuristic Evaluation*. Interactions, 5(5), pages 13–18. ISSN 1072-5520. doi:10.1145/285213.285219. (Cited on pages 20, 21 and 41.)
- Jakob Nielsen [1992]. *Finding Usability Problems through Heuristic Evaluation*. In *CHI '92: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 373–380. ACM Press, New York, NY, USA. ISBN 0897915135. doi:10.1145/142750.142834. (Cited on pages 16 and 35.)
- Jakob Nielsen [1993]. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 0125184050. (Cited on pages 3, 6, 7, 8, 11, 15, 23, 39, 49 and 139.)
- Jakob Nielsen [1994a]. *Guerrilla HCI: Using Discount Usability Engineering to Penetrate the Intimidation Barrier*, pages 245–272 in Bias and Mayhew [1994]. Academic Press, Inc., Orlando, FL, USA. ISBN 0120958104. (Cited on pages 12 and 13.)
- Jakob Nielsen [1994b]. *Heuristic Evaluation*, pages 25–62 in Nielsen and Mack [1994]. John Wiley and Sons, Inc., New York, NY, USA. ISBN 0471018775. (Cited on pages 33, 34, 35, 36, 40, 92 and 135.)
- Jakob Nielsen (Editor) [1995]. *Advances in Human-Computer Interaction*, volume 5. Ablex Publishing Corporation, Norwood, NJ, USA. ISBN 0567501966. (Cited on page 200.)
- Jakob Nielsen [1999]. *Designing Web Usability: The Practice of Simplicity*. New Riders Publishing, Thousand Oaks, CA, USA. ISBN 156205810X. (Cited on page 74.)
- Jakob Nielsen [2003]. *Return on Investment for Usability*. Jakob Nielsen's Alertbox. <http://www.useit.com/alertbox/20030107.html>. (Cited on page 9.)
- Jakob Nielsen and Marco G. P. Bergman [1995]. *Independent Iterative Design: A Method that Didn't Work*, volume 5, pages 235–248 in Nielsen [1995]. Ablex Publishing Corp., Norwood, NJ, USA. ISBN 0567501966. (Cited on page 12.)
- Jakob Nielsen and Robert L. Mack (Editors) [1994]. *Usability Inspection Methods*. John Wiley and Sons, Inc., New York, NY, USA. ISBN 0471018775. (Cited on pages 15, 16, 21, 35, 197, 199, 200 and 201.)
- Jakob Nielsen and Rolf Molich [1990]. *Heuristic Evaluation of User Interfaces*. In *CHI '90: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 249–256. ACM Press, New York, NY, USA. ISBN 0201509326. doi:10.1145/97243.97281. (Cited on pages 15, 31, 34 and 35.)
- Donald A. Norman [1988]. *The Psychology of Everyday Things*. Basic Books. ISBN 0465067093. (Cited on pages ix, 24 and 25.)
- OCLC [2005]. *Fourteen Heuristics Used in OCLC Heuristic Evaluations*. <http://www.oclc.org/policies/usability/heuristic/set.htm>. (Cited on page 42.)
- PEAR [2005]. *The PHP Extension and Application Repository*. <http://pear.php.net/>. (Cited on page 109.)
- Peter G. Polson and Clayton Lewis [1990]. *Theory-Based Design for Easily Learned Interfaces*. Human Computer Interaction, 5(2&3), pages 191–220. doi:10.1207/s15327051hci0502&3_3. (Cited on page 17.)
- Vesa Purho [2000]. *Heuristic Inspections for Documentation: 10 Recommended Documentation Heuristics*. STC Usability SIG. <http://www.stcsig.org/usability/newsletter/0004-docsheuristics.html>. (Cited on page 46.)

- Jeffrey Rubin and Theresa Hudson [1994]. *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*. John Wiley and Sons, Inc., New York, NY, USA. ISBN 0471594032. (Cited on page 10.)
- Douglas Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann [2000]. *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects*, volume 2. John Wiley and Sons, Inc., New York, NY, USA. ISBN 0471606952. (Cited on page 58.)
- Andrew Sears [1997]. *Heuristic Walkthroughs: Finding the Problems Without the Noise*. International Journal of Human-Computer Interaction, 9(3), pages 213–234. ISSN 0001-0782. doi:10.1207/s15327590ijhc0903_2. (Cited on pages 18 and 19.)
- Ben Shneiderman and Catherine Plaisant [2004]. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, fourth edition. ISBN 0321197860. (Cited on page 45.)
- Grant Skinner and Jess McMullin [2003]. *Usability Heuristics for Rich Internet Applications*. Boxes and Arrows Website. http://www.boxesandarrows.com/archives/usability_heuristics%_for_rich_internet_applications.php. (Cited on page 44.)
- Sidney L. Smith and Jane N. Mosier [1986]. *Guidelines for Designing User Interface Software*. Technical Report ESD-TR-86-278, MITRE Corporation, Bedford, MA, USA. (Cited on pages 21 and 22.)
- Sun Microsystems Inc. [1999]. *Java Look & Feel Design Guidelines*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN 0201615851. (Cited on pages 21 and 44.)
- Usability Glossary [2005a]. *Usability Glossary: Co-Discovery Method*. http://www.usabilityfirst.com/glossary/term_181.txt. (Cited on page 11.)
- Usability Glossary [2005b]. *Usability Glossary: Usage Study*. http://www.usabilityfirst.com/glossary/term_725.txt. (Cited on page 12.)
- uzReview [2003]. *Uzilla Review (uzReview)*. <http://uzilla.mozdev.org/heuristicreview.html>. (Cited on page 50.)
- Markus Völter, Alexander Schmid, and Eberhard Wolff [2002]. *Server Component Patterns: Component Infrastructures Illustrated with EJB*. John Wiley and Sons, Inc., New York, NY, USA. ISBN 0470843195. (Cited on page 58.)
- W3C [1998]. *Cascading Style Sheets, level 2*. <http://www.w3.org/TR/REC-CSS2/>. (Cited on page 74.)
- W3C [2000]. *XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)*. <http://www.w3.org/TR/xhtml1/>. (Cited on page 74.)
- W3C [2003]. *SOAP Version 1.2 Specification Assertions and Test Collection*. <http://www.w3.org/TR/2003/REC-soap12-testcollection-20030624/>. (Cited on page 136.)
- Wikipedia [2005a]. *Wikipedia: Web Application Frameworks*. http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks. (Cited on page 70.)
- Wikipedia [2005b]. *Wikipedia: PHP*. <http://en.wikipedia.org/wiki/Php>. (Cited on page 73.)
- Dennis Wixon, Sandra Jones, Linda Tse, and George Casaday [1994]. *Inspections and Design Reviews: Framework, History and Reflection*, pages 77–103 in Nielsen and Mack [1994]. John Wiley and Sons, Inc., New York, NY, USA. ISBN 0471018775. (Cited on page 26.)

Zhijun Zhang, Victor Basili, and Ben Schneidernan [1998]. *Perspective-Based Usability Inspection*. In *Proceedings of the Usability Professionals' Association Conference*, pages 281–282. The Usability Professionals' Association, Bloomington, IL, USA. (Cited on pages ix, 23, 24, 25 and 26.)