Martin Umgeher

Automated Usability Evaluation in Agile Projects

Dissertation

vorgelegt an der

Technischen Universität Graz

zur Erlangung des akademischen Grades
Doktor der Technischen Wissenschaften
(Dr.techn.)

durchgeführt am Institut für Softwaretechnologie
Technische Universität Graz
Vorstand: O. Univ.-Prof. Dipl.-Ing. Dr. techn. Wolfgang Slany

Graz, im Dezember 2010

Deutsche Kurzfassung der Dissertation

Die kontinuierliche, automatische Verifikation von Qualitätskriterien ist eine zentrale Praktik moderner Software-Entwicklungsprozesse. Die automatisch evaluierbaren Anforderungen reichen von statischer Code-Analyse und funktionalen Tests bis zu automatisierten Builds und Deployments. Durch die Automatisierung wird der manuelle Aufwand der Verifikationen minimiert.

Agile Software-Entwicklungsprozesse hängen im Speziellen von automatisierter Verifikation ab. Die häufigen, inkrementellen Code-Änderungen, auf denen agile Entwicklung basiert, bergen stets die Gefahr in sich, Fehler einzuschleusen. Daher ist die ständige Verifikation von Integrität und Korrektheit von entscheidender Bedeutung für den Erfolg agiler Projekte.

Automatisierte Verifikation beschränkt sich großteils auf funktionale Anforderungen, währen nicht-funktionale Belange, wie z.B. Usability, nur selten evaluiert werden. In dieser Dissertation wird eine Erweiterung der bestehenden, etablierten Software-Verifikationsschritte vorgeschlagen. Automatisierte Usability-Evaluierungen (AUE) erlauben es, Usability auf dem selben Automatisierungs-Level zu prüfen, wie dies für funktionale Korrektheit geschieht. Die Anforderungen an solche AUE-Werkzeuge sowie allgemeine Eigenschaften von Prozessautomatisierungssystemen werden diskutiert. Darauf aufbauend wird das Konzept eines AUE-Werkzeuges für die Evaluierung von Web-Software entwickelt, das die Voraussetzungen für die Integration in bestehende Automatisierungssysteme erfüllt. Durch automatisches, exploratives Durchwandern von Web-Anwendungen kann jede Seite einzeln untersucht werden; zusätzlich wird die Navigationsstruktur analysiert. Die Anwendung des Konzeptes wird anhand der Integration in den agilen Entwicklungsprozess Extreme Programming und dessen Automatisierungssystem, Continuous Integration, veranschaulicht. Implementation und Anwendbarkeit im Kontext kommerzieller Software-Entwicklung werden besprochen.

Das präsentierte Konzept basiert auf den Erfahrungen und Ergebnissen, die während eines dreijährigen angewandten Forschungsprojektes gesammelt wurden. Im Rahmen dieses Projektes wurden verschiedene Aspekte agiler Entwicklungsmethoden, mit besonderem Fokus auf der Integration von Usability-Praktiken (z.B. Einbeziehen von Usability-Experten, Anwendung von AUE-Methoden), untersucht. Diese Experimente wurden durch extern durchgeführte Untersuchungen (Experten-Evaluationen und Benutzer-Studien) evaluiert. Die Forschungsergebnisse werden in dieser Dissertation präsentiert.

Abstract

The continuous, automated verification of software requirements has become a core practice in modern development processes. The automatically verifiable requirements are manifold, ranging from static code analysis and functional testing to automated building and deployment. The automated nature of these verifications minimizes the required manual effort, allowing their continuous application throughout the development life cycle. This practice helps to spot errors as soon as possible, thereby reducing the cost of repair.

In particular, agile software development processes depend on automated verification. Agile development is based on frequent, incremental changes, a practice which perpetually bears the danger of introducing errors. Thus, the constant verification of the developed software's integrity and correctness is crucial for the success of agile projects.

In current software development practice, automatically and continuously applied verifications are mostly limited to evaluating functional requirements, while non-functional concerns of software quality, for example usability, are rarely employed. In this thesis, an extension of the established set of verification methods is proposed: using automated usability evaluation (AUE) techniques, usability aspects can be analyzed and verified on the same level of automation as functional correctness. The requirements for the integration of AUE methods into existing automation systems, as well as characteristics of these systems, are examined. Then, an AUE tool for Web software development meeting the requirements is presented. By automatically exploring Web applications, the tool accesses and inspects all pages of the Web application; additionally, the navigation graph structure is analyzed. The tool's application is exemplified by describing its integration into the Extreme Programming (XP) development method and XP's automation framework, Continuous Integration. Implementation issues are highlighted, and applicability for commercial software development is discussed.

The presented approach is based on the experiences and results gathered during an applied research project. In the course of this project, different aspects of agile development were studied over a period of three years. A special focus was put on the integration of usability practices, probing usability expert involvement and application of AUE methods. These experiments were evaluated by additional, externally conducted examinations, namely expert evaluations and user studies. The results of this research are presented in this thesis.

Contents

Kurzfassung (abstract in German) **1**

Abstract **2**

Table of Contents **3**

List of Tables **7**

List of Figures **8**

1 Introduction **10**

1.1 Agile Development 10

1.1.1 Software Development Processes 10

1.1.2 Concepts of Agile Development 12

1.1.3 Automation in Agile Development 15

1.2 Usability 20

1.2.1 What is Usability? 20

1.2.2 Usability Engineering 21

1.2.3 Automated Usability Evaluation 25

1.3 Research Context 28

1.4 Results of this Thesis 29

1.4.1 Agile Development Process Description 30

1.4.2 Application Development 30

1.4.3 AUE Tool and Process Integration 31

1.5 Structure of this Thesis 32

2 User Interface Design for a Mobile Multimedia Application: An Iterative Approach **34**

2.1 Introduction 34

2.2 Related Work 36
2.2.1 Applications 36
2.2.2 Iterative User Interface Design 37
2.3 Usage Scenarios 37
2.3.1 TV Archive for Subway Riders 37
2.3.2 Radio Archive for Car Drivers 37
2.3.3 Media Recommendations for Users 38
2.4 Usability 38
2.4.1 Iterative User Interface Design 39
2.4.2 An Iterative Design Example 40
2.4.3 User-Centered Application Design 42
2.5 User-Based Recommendations 42
2.5.1 Interactive Model 43
2.5.2 Behavior-Based Model 43
2.5.3 Model Combination 43
2.5.4 Implications 44
2.6 Conclusion 44

3 Optimizing Extreme Programming 46
3.1 Introduction 47
3.2 Project Environment 47
3.3 Process 49
3.3.1 Fully Implemented Practices 49
3.3.2 Partially Implemented/Modified Practices 52
3.4 Reflection 53
3.5 Conclusion 55

4 Integrating Extreme Programming and User-Centered Design 56
4.1 Introduction 56
4.2 Common Values of XP and UCD 57
4.2.1 End-User Involvement 58
4.2.2 Continuous Testing 58
4.2.3 Iterative Development 58
4.3 Project and Team Setup 59
4.4 The Design Process 59
4.4.1 Approach to UCD 59
4.4.2 Choosing the Type of Mock-Up 61
4.4.3 Frequency of End-User Tests 61

4.4.4 The Testing Workflow 62
4.4.5 Feedback from a Usability Test 62
4.4.6 Testing Issues 63
4.5 Conclusion 64

5 Agile User-Centered Design Applied to a Mobile Multimedia Streaming Application 66
5.1 Introduction 66
5.2 Similarities between XP and UCD 68
5.2.1 End-User Involvement 68
5.2.2 Continuous Testing 68
5.2.3 Iterative Development 69
5.3 Project and Team Setup 69
5.4 Application Features 70
5.5 Agile Usability Process 71
5.5.1 Approach to User-Centered Design 71
5.5.2 Choosing the Type of Mock-Up 72
5.5.3 Frequency of End-User Tests 73
5.5.4 Integration of HCI Instruments 73
5.5.5 Testing Issues 74
5.5.6 Communication and Collaboration 75
5.5.7 The Planning Game 75
5.5.8 Pair Programming 76
5.5.9 On-Site Customer 76
5.6 Usability Study 76
5.6.1 Improvements of Layout and Design 77
5.6.2 Improvements of the Usability of the Prototype 77
5.6.3 A Task Example 79
5.7 Conclusion 82

6 Concept and Design of a Contextual Mobile Multimedia Content Usability Study 84
6.1 Introduction 84
6.2 Related Work 86
6.3 Application 86
6.3.1 Features 87
6.4 Selection of Respondents 89
6.5 Study Setup 89

6.5.1 Media Content 90
6.5.2 Diary Study 91
6.5.3 Contextual Interview 92
6.6 Expected Results 93
6.7 Conclusion 94

7 Continuous Usability Evaluation 96
7.1 Introduction 96
7.2 Related Work 100
7.2.1 Automated Usability Evaluation 100
7.2.2 Development Process Integration 103
7.3 Automated Usability Evaluation in Continuous Integration Environments 103
7.3.1 Capture 104
7.3.2 Analysis 107
7.3.3 Critique 109
7.3.4 Integration in the Automated Build Process (CI) 111
7.4 Graph-Related Usability Problems 113
7.4.1 Self link 113
7.4.2 Inconsistently Used Link Labels 113
7.4.3 Inconsistently Labeled Link Target URLs 114
7.5 Prototype Implementation 114
7.5.1 Implementation Notes 115
7.5.2 Walkthrough 116
7.5.3 Limitations 128
7.6 Conclusion and Future Work 130

8 Epilogue 131
8.1 Conclusions 131
8.2 Challenges and Perspectives 133

Bibliography 134

Acknowledgments 147

List of Tables

3.1 Subjective Metric (Shodan 2.0 Input Metric Survey). 54

List of Figures

1.1 The steps performed during Continuous Integration. 18

2.1 Iterative User Interface Design Workflow. 39
2.2 From Paper Mock-Up to Mobile: The first Search-Results Screen. . . . 40
2.3 An additional HTML Mock-Up: A refactored Search-Results Screen. . 41

3.1 Application, Research and Business Aspects in a Release. 48
3.2 Selected Story Cards on the Release-Board (Release Planning). . . . 51
3.3 Selected Story Cards on the Iteration-Board (Iteration Planning). . . 51
3.4 Executable Code versus Test Code and Test Coverage. 53

4.1 Iterative User Interface Design Workflow [52]. 60
4.2 The Integration of HCI Instruments into XP [122]. 63
4.3 The Prototype of the Home Page. 64

5.1 Iterative User Interface Design Workflow [52]. 71
5.2 The Integration of HCI Instruments into XP [122]. 74
5.3 The Prototype of the Home Page [54]. 78
5.4 The Prototype of the Channel Page showing the Calender. 79
5.5 The Menu Entries without any visual Separation. 79
5.6 Improvements of Menu Layout and Arrangement. 81
5.7 Use the Space on top of the Clip Detail Page more efficiently. 82

6.1 Home Page. 87
6.2 Categories Page. 88

7.1 Simple page graph. 105
7.2 The steps performed during Continuous Integration, extended by an AUE step. 112
7.3 Self link. 113

7.4 Inconsistently used link labels. 114
7.5 Inconsistently labeled link targets. 115
7.6 Dialogs for creating and loading projects. 117
7.7 Project overview. 118
7.8 Crawl job summary. 119
7.9 Dialog for configuring and running a crawl job. 119
7.10 Extended crawl dialog. 121
7.11 List of all pages in the graph. 122
7.12 List of all links in the graph. 122
7.13 Page details. 123
7.14 Graph comparison. 124
7.15 Problem list. 125
7.16 Inconsistent link labels list. 126
7.17 Inconsistent link targets list. 127

Chapter 1

Introduction

This chapter introduces the two core aspects of the research presented in this thesis, agile development methods and usability evaluation methods. Then, it presents the subject of this thesis, automated usability evaluation in agile projects, and gives an introduction on the context of the presented research. Finally, the overall structure of this thesis is outlined.

1.1 Agile Development

Agile development was created as a response to the many shortcomings of traditional development processes in software engineering. In the remainder of this section, the roots of agile development are explored in more detail. Then, agile development is introduced, followed by a description of the role of process automation in agile development, which is an essential part of this thesis.

1.1.1 Software Development Processes

When software development started to become a major business in the 1960s and 1970s, the need to structure, manage, and plan software projects arose. Initially, existing and well-established development processes from construction and manufacturing industries were applied to software projects. These processes implemented a sequential paradigm: product development was broken down into steps or “phases”, requiring each phase to be completed before continuing with the next. In the domain of software development, this approach was later described as “Waterfall Model”.

The Waterfall Model is based on the sequential execution of well-defined life cycle phases (e.g.: requirements specification, design, implementation, testing, and maintenance) which build upon each other. A phase is started only when the outcome of the previous phase is considered to be complete and correct.

The approach strictly separates planning steps from construction steps: when initiating the implementation phase of the project, the project plan created in the previous phase is assumed to contain all relevant information needed to implement the project. If in a later stage of the project this plan turns out to be inconsistent with the actual requirements and needs to be changed, this is considered to be a mistake of a previous phase. However, in many software projects the final requirements are not initially known, hence a comprehensive plan can often not be provided. There are various valid reasons why requirements can change during a commercial software project:

- The customer might find out that an important aspect was not considered in the actual requirements analysis, and that the developed software product would not be useful without introducing this aspect. This is a frequent mistake when "translating" the customer's needs into software requirements.
- Both developers and users of the software might get new insights during development or by test-using prototypes, which, when applied to the project implementation, could substantially increase the quality of the final product [80].
- General technical advances during the course of the project might make parts of the original plan obsolete.
- Better problem solutions than originally planned might be found during the implementation, making development more efficient or the produced application more effective.
- The legal context of the intended usage of the application might change, requiring the application to be adapted accordingly.

The sequential development paradigm assumes that any project can be planned in such detail that the actual implementation becomes the mere mechanic process of following a plan. Again, this does not hold for many software projects: due to the inherent complexity of software systems, unforeseen issues can arise during the implementation, forcing the developers to diverge from the plan. This is especially true if the project team has no or little experience with the used technologies or in the project's business domain; without the proper pre-knowledge, it is almost impossible to plan all steps in detail.

Another implication of the Waterfall Model is the emphasis placed on documentation artifacts. Comprehensive documentation of all relevant outcomes is the only means of communication between adjacent phases, as different phases often are conducted by different teams, preventing implicit knowledge transfer. Due to this reason, Waterfall Model-oriented development methods are often critizised as "heavy-weight processes". A benefit of this approach is that work results only have to be verified at phase borders: when a phase is completed, its results are verified, and assumed to be correct in later phases.

Many less strict variations of the original Waterfall Model have been developed (e.g., the "Sashimi Model" which allows the overlapping of adjacent phases). However, especially for small and medium-sized projects, the strict sequential development process has proven to be impractical for many software development contexts.

To address these problems of Waterfall Model-oriented methods, new development processes following an iterative paradigm were created, which allow to adapt the initial plan in case of changed requirements.

1.1.2 Concepts of Agile Development

The term "agile development" was coined by a group of software professionals in 2001. This group met in order to discuss the shortcomings of the established and commonly applied "heavy-weight" development processes and aimed at finding more "light-weight" processes. The results of the meeting were written down and published as the "agile manifesto" [16], an abstract description of the agreed-on values and practices. Since then, the term "agile" has been used to describe a set of software development methods which share the ideas of the agile manifesto. Some of the most popular methods are Extreme Programming (XP) [15] and Scrum [101].

In the following, some core concepts of agile development are described.

Iterations

Agile development is based on the iterative development paradigm. In contrast to the separate, coarse-grained phases of sequential development methods, agile development focuses on short-term iterations, each of which contains some or all development phases:

- At the start of the iteration, requirements are defined in detail and the iteration is planned.
- During the iteration, the planned features are implemented and tested.

- At the end of the iteration, a working version of the developed software is available.

The duration of an iteration varies depending on the concrete development method; typically, iterations last between one and four weeks.

Detailed planning is only performed on iteration base: an iteration plan consists of a list of implementation task descriptions; developers sign up for tasks and thus take responsibility for implementation during the iteration. This keeps the planning overhead low and minimizes the impact of requirement changes on project planning. Each iteration aims at adding features which immediately increase the customer benefit when using the software. Therefore, tasks are planned with a scope that is both valuable for the customer and completable in a single iteration.

At the end of each iteration, the developed software must be in a working, usable state. This allows to frequently present incremental updates of the software to the customer and get feedback, which can be incorporated into the next iteration's planning. Furthermore, these periodic results document the project progress most accurately, so little emphasis is put on written-down documentation describing the project.

Customer Involvement

Agile development puts much emphasis on the project "customer". In general, the customer represents the stakeholders of the project. Depending on the project context, this could be a company commissioning a contractor to develop a software, or the company employing the agile team (in case of in-house development), or even the general public (in case of open-source projects).

Collaboration with the project customer is not restricted to initial requirements analysis and final approval. Development aims at frequently providing new releases (ideally every few iterations) which are deployed and used in a productive environment as early as possible. This serves two reasons: first, each released and deployed version adds some value for the customer; and second, problems can be spotted early, and the customer can provide feedback on the new version which helps to validate the performed work.

The customer is required to take an active role in agile development. Usually, an agile team contains a customer representative. This person is assigned to the team by the project customer and mediates between development activities and stakeholder interests:

- The customer representative supports iteration planning by prioritizing the upcoming development activities: features are prioritized according to their expected value for the customer, and planned accordingly.
- During iterations, the representative acts as a problem-domain expert for the developers, in case any unexpected issues arise.
- At the end of an iteration, the current version of the software is demonstrated to the representative, who in turn provides feedback on the implemented features.

Incremental Development

In contrast to “traditional”, plan-driven development approaches, agile development accepts requirement changes as an integral part of software projects. Therefore, agile development does not attempt to avoid change, but rather welcomes it as an opportunity to improve the quality of the developed product. This acceptance for change is reflected in the way agile development teams work:

- The customer, not the development team, decides the order of implemented features. This means that implementation tasks are not grouped by technical aspects; it is well possible that an iteration contains tasks which require working on different parts of the software. As a result, subsystems of the developed software are rarely “complete” but rather constantly subject to change.
- Since the software is required to work at each iteration’s end, all performed changes need to be integrated with the entire software system immediately (or, at least, in every iteration). Developers cannot implement subsystems independently from the overall system and integrate them at a later point in time; instead, the system is constantly kept in an “integrated” state.
- Instead of building comprehensive solutions that also cover assumed future demands on the software, only the immediately necessary functionality is implemented—work beyond that point is seen as a potential waste of time, as requirements might change until the next iteration, rendering the additional work obsolete.

Team Management

The focus of agile development lies on individual persons, as opposed to viewing team members as replaceable positions in the process organigram. Personal meetings are preferred over document-based correspondence; frequent communication between the team members is the preferred way of knowledge transfer. Consequently, agile development proposes to locate the entire project team in one place to ease direct communication. Furthermore, the whole team participates in iteration planning, and everyone is involved in decision-making. Also, developers are encouraged to work on all parts of the system instead of specializing on specific parts in order to distribute knowledge in the entire team.

Agile teams are considered "self-organizing", so team structure and role distribution are subject to team-internal decisions and are allowed to evolve during the course of the project.

"Agile Values"

For the sake of completeness, the four "values" stated in the agile menifesto are listed below:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contracted negotiation
4. Responding to change over following a plan

1.1.3 Automation in Agile Development

The incremental development style of agile development increases the danger of introducing bugs or other unexpected effects to the software system, as even small code changes can have unintended side effects on seemingly unrelated parts of the system. To always ensure that the software is working, and working as expected—which is a core value of agile development—, frequent verification of the software's structural and functional integrity is mandatory.

Structural integrity describes the (static) correctness of source files and the source code. The integrity of source files depends on the correct naming, locating, and referencing of files: when changing a file's name or moving it to a different location, other parts of the system which reference that file need to be updated accordingly. The integrity of source code depends on the syntactical correctness: when existing code is changed or deleted, all references to that code need to be updated.

Functional integrity describes the correctness of the software's dynamic behavior. This subsumes both the interaction between subsystems and modules of the software, as well as the interaction between user and software.

Agile development promotes a range of practices to support preservation of software integrity, for example:

Refactoring During implementation, structural code changes (i.e., changes which do not alter software features, but only affect the program structure) are performed in a controlled way by means of so-called "refactorings", i.e., patterns of simple programming steps which, when applied correctly, ensure that the software's functionality is unchanged [96]. Refactoring tools can automate this process to a large extent. Most modern software IDEs (integrated development environments; e.g., eclipse, Visual Studio) have built-in support for refactoring.

Automated Build Structural integrity can largely be verified by building the software, i.e., by transforming the project's source code to a runnable and deployable software. Any changes applied to the source code are verified by building; if a change violates the structural integrity, the build fails, requiring the responsible changes to be identified and fixed. To allow frequent building, agile development aims at making the build process fast and simple, which is achieved by automating the build.

Unit tests On code level, functional integrity is controlled by unit tests [88]. These tests test separate modules and functions, by invoking the implemented functionality and comparing expected and actually retrieved results. Unit tests are implemented as test programs; while their creation requires work from developers, their execution and evaluation is fully automated. Therefore, unit tests are run as regression tests: each new test is added to the project's test suite; when evaluating a code change, *all* unit tests are run, not just the tests directly related to the changed code. This allows to spot unintended side-effects of the performed code changes.

System tests On system level, functional integrity is controlled by system tests which evaluate the software in its entireness. Test users interact with the deployed system and perform representative work tasks. System tests involve the project customer who, both as user of the software and as domain expert, can give feedback and point out problems. This type of test requires the software to be built successfully, thus relying on the build system.

All these practices have in common that they (one way or another) apply automation in order to support the fast-paced development style of agile methods. While agile development does not specify any specific automation systems, the implementation of agile practices often relies on automation. For example, one of the core practices of XP is "continuous integration" (CI), which in the following shall serve as an example of how automation drives agile projects.

CI denotes both a toolset and a work process. Its main aim is to support an incremental development style: as development tasks are broken down into increments, developers are required to frequently perform small yet consistent and meaningful changes. All changes are applied to a shared code base, thereby directly integrating them with the base system. If this integration fails due to integrity violations (i.e., the build or some automated tests failed), the CI system notifies the development team which in turn is responsible for immediately fixing the problem.

CI applies automation on two levels. First, the steps required to build an application, to deploy it, to test it, and to notify about any occurring integrity violations are implemented as an automated task. Second, the invocation of this task itself can be automated as well. Common triggers for invoking CI are:

- The CI system discovers when the project was changed and a re-integration is required in order to verify the project's integrity (*implicit trigger*).
- The CI system is executed at fixed times, e.g., as a "nightly build" (*scheduled trigger*).
- Programmers can also trigger CI manually (*explicit trigger*).

Figure 1.1 shows a diagram depicting the steps which are run during a CI execution. Any errors occurring during the steps are collected and passed on to the Result Notification step. The diagram also includes the context of CI execution: CI is triggered by an external event, and, when finished, notifies the development team about its success or about the occurred errors.

Technically, CI environments are based on three main aspects: a central source code repository, an automated build system, and automated testing[1].

[1]None of these practices is specific to CI or XP; notably, code repositories and automated builds have been successfully used in software engineering for decades. However, their combination and usage as specified by CI yields a powerful tool for development and quality control, which also can be applied outside the scope of agile development methodologies.

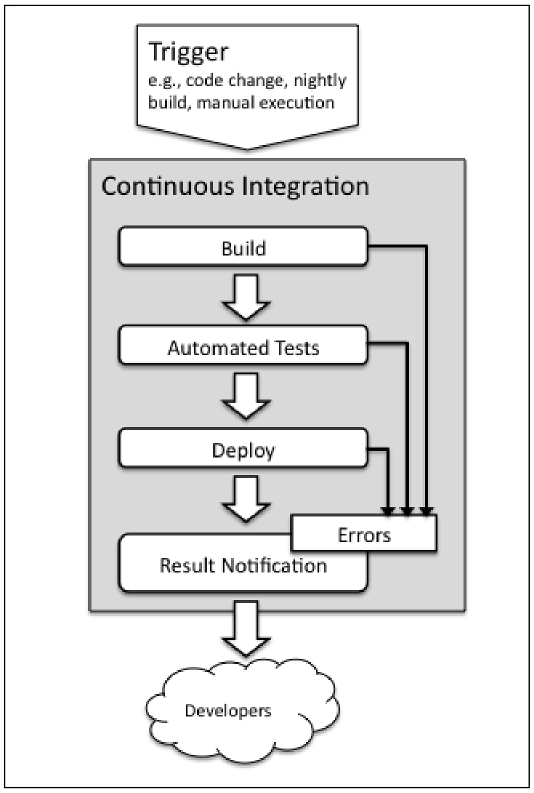


Figure 1.1: The steps performed during Continuous Integration.

Shared Code Repository

All developers concurrently send (“commit”) their code changes to a shared repository and retrieve (“update”) the latest code version from it. This serves different purposes:

- By frequently committing incremental changes and retrieving updates, the common code base gets distributed among the team. Therefore, all developers work with the most current code version, which helps to spot inconsistencies early.
- All changes get instantly integrated with the base system. This avoids the (often problematic) explicit integration of separately developed subsystems.
- CI is informed when changes are committed by a developer and can trigger verification steps.

- Single developers do not "own" parts of the code base, so inconsistencies in the code can be spotted early.

The most common issue of concurrent development is a "commit conflict" which occurs if two (or more) developers perform changes to the same resource at the same time. Version control systems (VCS; e.g., CVS, Subversion) which serve as code repositories can detect such conflicts. In some cases, conflicts can be resolved automatically by the VCS, otherwise the affected developer is informed and requested to solve the conflict, or to revert the change.

Automated Build

Depending on the size and complexity of a project, building an application from source code can be a complicated, time-consuming, and error-prone task. Building, amongst other things, comprises compiling source files, resolving dependencies between subsystems, configuring components, and creating installable binaries; in Web projects, it is also common to deploy the built application on a test server. An automated build system provides means to automate all required build steps in order to perform the whole build process by invoking a single command.

In a CI environment, the automated build is triggered automatically whenever a developer commits a new version to the central code repository, or (if the build takes too long for frequent executions) at least once a day at a fixed time, usually at midnight ("nightly build"). The frequent automatic building helps to identify "broken builds": a developer might commit a change which, when integrated with the entire application, causes the build to fail (or, in less serious cases, yield a warning message); for example, a resource referenced by other files was deleted, or a function used by other parts of the application was renamed without also changing the dependent code. The sooner such a broken build is disclosed, the easier it is to identify the cause of the error and subsequently to fix the problem; therefore, the CI system can send alert messages (e.g., via e-mail) to the originator of the responsible change or to the whole development team.

Automated Tests

Every software project requires testing in some way in order to validate the developed application regarding implicit and explicit requirements. Testing can be conducted on different levels: for example, system tests ensure that the whole application behaves as expected by executing realistic use cases, whereas unit tests test a single unit of source code (i.e., a function or a class) in great detail. In general, a software test is performed by executing a series of interactions with (part of) the tested software system and comparing the actual results with expected values. Most software tests can be automated by means of a test program which automatically performs the required test steps, validates the retrieved results, and yields the success or failure of the test. Test automation has two main benefits over manual testing: the tests run much faster, and they are reproducible without the danger of human errors slipping in.

XP advocates that tests are written before the code they should test, and no code is written without test (Test-Driven Development [14]). The tests have to be maintained: when existing code is changed, the respective tests have to be changed accordingly. This leads to a large number of up-to-date test cases which cover a big part of the implemented functionality.

The CI environment executes the project's entire test suite after every automatically triggered (and successfully performed) build. This is done to verify that a committed change also functionally integrates with the application: changes can have unintended side-effects on other parts of the application, so all tests are run to ensure that previously working code is not broken. The CI system's notification mechanism used to inform about build failures is also used to notify about failed tests (in fact, failing tests are also considered to "break the build"). This immediate feedback helps to fix the introduced errors as soon as possible, which in turn minimizes the impact on other developers' work.

1.2 Usability

1.2.1 What is Usability?

Usability is an abstract, qualitative attribute which describes a device's ease-of-use. The usability of a device reflects the quality of interaction between human users and the device. Therefore, it not only depends on device characteristics, but also on the specific users and usage contexts.

The "devices" usability is concerned with are any objects persons can interact with, ranging from simple things like books and door knobs, to complex machines like mobile phones and cars. For the domain of computer programs ("virtual devices"), and more specifically Web applications (which are the main focus of this work), a common definition[2] of usability [91] comprises five demands for usable software:

[2] There exist many defintions of usability, which, although differing in their details, overlap with the presented five points [103].

learnability, efficiency, memorability, error avoidance, and satisfaction.

Learnability of a software describes how easy it is for a user to perform tasks without prior knowledge of the software. Depending on the usage context, a learning phase can be acceptable. Professional programs usually are too complex to be used instantly without explicitly learning the user interface. However, a simple application like an informational Web site should be usable intuitively, without requiring the user to explicitly learn how to use it (assuming that the user has pre-knowledge with other, similar Web sites).

Efficiency of Use describes how well a user who is already familiar with a software can use this software. Usage efficiency can be measured (and hence compared) as successfully performed tasks per time unit.

Memorability describes how well previous knowledge of the software's usage can be reapplied after not using it for a period of time.

Few and Noncatastrophic Errors The error aspect of usability expresses how easy it is for users to commit a usage error, and how hard to recover from such an error. On the one hand, software should prevent the user from committing errors in the first place; on the other, once an error has occurred, the software should support the user in fixing the problem. In general, the software should be designed so that "catastrophic" (i.e., unrecoverable) errors cannot occur.

Subjective Satisfaction describes how "pleasant" it is for users to use the software. This is especially important for public Web sites which aim at attracting new users: a low user satisfaction can cause users to turn away from the site and instead use alternative offers on the Web.

Another aspect of software usability which is gaining more and more importance is *accessibility*, i.e., the degree to which users with disabilities (e.g. visually impaired people) can use a software product.

1.2.2 Usability Engineering

The process of applying usability principles and improvement practices to software development is called *usability engineering*. The aim of usability engineering is to ensure a certain level of usability, taking into account the targeted user group and usage context. This makes it necessary to measure usability. The techniques for measuring usability are commonly referred to as *usability evaluation* methods. Hence, the role of usability engineering in the software development process is to use the results of usability evaluations in order to improve the usability of the developed software.

Most usability evaluation methods can be broken down into three main activities [59]:

Capture Initially, the relevant data for the usability evaluation has to be gathered. In general, any kind of data related to the interaction between human and computer can be of interest. The specific kind of data which is collected depends on the applied evaluation method; it can range from qualitative feedback from a user to a detailed record of every action a user performs on a user interface.

Analysis The collected usability data is inspected in order to identify potential problems of the evaluated user interface.

Critique The found usability problems have to be put into context concerning their severeness and relevance. Also, possible improvements or changes of the user interface which would solve (or at least mitigate) the problems can be suggested.

Usability is a non-functional requirement of engineering. As such, it cannot be deduced directly from the software under development. Instead, different metrics which indicate “good” or “bad” usability have to be collected; these metrics are quantified and put into context with each other. For example, one measure would be the number of committed user errors during a test usage session; another would be the number of successfully performed tasks during a given time period; yet another would be the number of identified deviations from a usability design guideline.

Quantitative usability measures allow to compare software interface designs. This serves different purposes:

Evaluate designs When deciding between design alternatives, measurements help to choose the variant with the best usability.

Validate usability improvements When trying to improve usability, measurements allow to validate the performed steps.

Deduce best practices By measuring different design approaches, best practices for usable design can be identified, which can be reapplied in future designs.

There exist different approaches to usability evaluation; the most important evaluation classes are listed below:

- testing the software with users ("usability testing")
- applying usability expertise to judge the software's usability ("usability inspection")
- questioning users about the software to retrieve qualitative feedback, or presenting general questions to potential users in order to collect information about the users' needs ("usability inquiry")

It is not always necessary to implement a design in order to measure its usability. Another approach is to use so-called "paper prototypes", i.e., manually or virtually drawn visualizations of the user interface which are used to simulate the intended interactions [107]. Prototyping can be used with any of the mentioned usability evaluation approaches.

In the following, the evaluation classes "usability testing" and "usability inspection" are discussed in more detail.

Usability Testing

In *usability testing*, test users are observed in a controlled environment while using the software under test to perform a list of tasks [99]. Usability information is gathered by observing in detail how the users manage to fulfill the requested tasks.

The data collected during usability tests can be of different kinds [33] [99]. To analyze the steps users perform when using the software, all relevant user actions (e.g., menu item selection, navigation, ...) have to be recorded. Also, the user's general behavior can be observed in order to deduce further aspects, e.g., whether the user seems satisfied, frustrated, or lost in the user interface. To analyze the usage efficiency of a software, the start and end times of tasks are recorded. With eye tracking, the eye movements of the test user can be recorded [94]. Finally, pre- and post-test questionnaires allow to collect additional qualitative feedback [39].

Furthermore, there exist some more specialized techniques for usability testing. To get a deeper understanding of what users think when using the software, a "think aloud" test can be performed [74]. Here, test users are instructed to verbalize their thoughts while performing the test tasks. Another method is "co-discovery" testing where a pair of test users uses the software together [33]. The conversation between the users is recorded in order to gain insights into the users' understanding of the software.

The overall number of test users participating in a usability test depends on the goal of the test: in order to find as many usability problems as possible, large test groups are preferable, while frequent tests with small test groups help to incrementally improve a design. The test can either take place in the "test department" of a company testing a product (either with in-house testers or external test persons brought in), or in the workplace of actual users [91]. Additionally, tests can also be carried out remotely: test users use the software in their own environment (e.g., home or office) instead of a test laboratory [5]. Usage and timing data is collected programmatically and later analyzed and interpreted by the test conductors.

Usability Inspection

Usability inspection subsumes a list of methods where usability experts evaluate a software design [92].

To measure the usability of a software, the evaluators inspect different properties of the user interface and check them against predefined principles. There exist different so-called "usability heuristics", lists of generic usability principles which are applied to evaluate the general usability ("heuristic evaluation" [93]). To test for more specific properties of the software, the evaluators use (or create) detailed guidelines which follow the specific usability requirements of the respective software design ("guideline checking" [78]). In both cases, the test results in a list of deviations from the predefined principles.

During a so-called "cognitive walkthrough" [73], the evaluators perform tasks, trying to act and think like novice users of the software. The focus of this test lies on finding out how easily the interface can be used from scratch, without any previous knowledge (e.g., gained from reading a manual).

To measure efficiency of a software, an "action analysis" can be performed: the test tasks are further broken down into smaller steps or "actions", for each of which a time estimate is provided. The actions can either be atomic (e.g., moving the mouse pointer, or reading and recognizing a word on the screen; "keystroke level" [24]) or abstract (e.g., selecting a menu item, or pressing the OK button; "back-of-the-envelope" [74]). The expected time of task completion is then calculated by summing up the action estimates.

1.2.3 Automated Usability Evaluation

Automated usability evaluation (AUE) tools[3] automate different aspects of usability evaluations. The degree of automation varies significantly among available tools, ranging from support tools for "manual" evaluation methods which automate single evaluation steps, to comprehensive fully-automated evaluation techniques.

In the following, some potential benefits of AUE tools are described [59]:

Reduced costs A major benefit of automation is the reduction of time spent on evaluations. AUE tools can reduce (or even supersede) the effort of human evaluation conductors, thereby significantly decreasing the overall cost of usability evaluations. For example, instead of employing an observer who manually notes down all actions performed by a user, an AUE tool can automatically capture and record all user inputs; this approach has the additional benefit that the gathered action log is already in a computer-readable format, easing its use in further analysis steps.

Increased consistency In contrast to human usability evaluators, automated tools work deterministically. Repeated evaluations of the same user interface will always yield the same results, as they do not depend on the subjective judgement and interpretation of humans. Also, a usability problem occurring multiple times will be reported consistently.

Increased coverage Manual usability evaluations usually only cover a relatively small part of the user interface due to time and cost constraints: evaluation conductors choose a representative subset of the interface and then extrapolate the found usability problems to the entire application. For example, the participants in a usability test are instructed to achieve specific goals during the test session; this limits the effective exploration space of the test users. In contrast, AUE tools are hardly restricted concerning time and other resources, so they can assess an interface in its entirety. Also, the repeated application of AUE tools is more cost-effective than a repeated evaluation with human participants, and thus can be applied more frequently.

Less expertise required AUE tools which cover the *analysis* and *critique* activities of usability evaluations (see Section 1.2.2) encode expert usability knowledge, reducing the need for this kind of expertise among the evaluators. This

[3]In the context of AUE, the terms "tool" and "method" are used synonymously, as every (partially or entirely) automated evaluation method requires some kind of software tool which implements the automation aspect.

enables development teams which do not comprise usability experts to incorporate usability principles in their development process.

It is important to note that automated evaluations can not substitute manually conducted evaluations. AUE tools are applied in order to increase the efficiency and/or quality of usability evaluations; they serve as a complement to standard methods. A major drawback of AUE tools is that they can only identify expected usability problems: if an AUE tool is not instructed to search for a certain kind of problem, this problem cannot be found. Human testers, however, have the ability to also identify unexpected usability issues.

There is no concise definition for AUE; on the contrary, most software tools created or tailored for usability evaluations are labeled as AUE tools. Therefore, the term is often used ambiguously, complicating the discussion and comparison of new AUE methods due to the heterogeneous context.

A taxonomy for usability evaluation methods with a strong focus on AUE (published by Ivory and Hearst [59]) shall serve as a display for the complexity of the field of AUE tools. The taxonomy comprises three dimensions[4] which are used to categorize usability evaluation tools: the applied method, the provided automation type, and the additionally required non-automated effort.

Method class and type The evaluation method of the AUE tool is categorized on two levels. The taxonomy comprises five classes (*testing*, *inspection*, *inquiry*, *analytical modeling*, *simulation*), analogously to the usability evaluation classes presented in Section 1.2.2. Each class again comprises approximately ten more specific method types (e.g., for the *testing* class: *think-aloud test*, *log file analysis*, ...).

Automation type The automation type describes the evaluation activity which is automated by the AUE tool. The types are defined analogously to the equally named basic evaluation activities. Four levels of automation are listed: *none* (no automation; the described method is not an AUE method), *capture* (automated recording of usability data), *analysis* (automated detection of usability problems in the data), and *critique* (automated solution proposal).

Effort level The effort level describes the amount of non-automated work required for applying the tool. Four levels of effort are listed: *minimal effort* (almost

[4]In the original publication, the authors list *method class* and *method type* as separate dimensions, resulting in four dimensions altogether; however, *method type* is a sub-categorization of *method class*, hence the two are non-orthogonal and cannot be seen as independent dimensions.

no additional workload), *model development* (requires the creation of a model depicting the user interface), *informal use* (requires the evaluated software to be used with freely chosen tasks), and *formal use* (requires the evaluated software to be used with predefined tasks).

To some degree, the applied "method class" also determines the possible automation. Obviously, evaluations incorporating test users (i.e., *testing* and *inquiry* method classes) cannot be entirely automated since the relevant aspect of such evaluations is the nondeterministic behavior and subjective opinion of human testers; still, AUE tools can aid as a valuable enhancement to increase the efficiency of such methods. In contrast, evaluations which aim at objectively judging a user interface based on predefined rules (i.e., *inspection* method class) can be largely automated, given that the *capture* activity of the evaluation can be performed automatically: if gathering of the relevant usability data does not require human involvement, then also the remaining activities (*analysis* and *critique*) can be automated.

In the context of this document, the focus lies on AUE tools of the latter kind, i.e., tools which are fully or largely automated. In the following, some commonly used types of this evaluation method class (and examples thereof) are listed[5]:

Property checks There exists a wide range of tools for checking usability-relevant properties of user interfaces. These tools usually are used by user interface designers to certify that basic usability criteria are met.

A common example for this type of AUE are tools for checking the color contrast, e.g. according to the W3C's Web Content Accessibility Guidelines [125]. This comprises simple static contrast checks which only requires the input of foreground and background colors[6], as well as dynamic analysis of an entire Web page, analyzing the contrast of all elements of a Web page[7]. Another common example are screen resolution simulators which allow to view a Web page's appearance in different screen resolutions[8].

Guideline checks This type of AUE tool analyzes a user interface in its entirety, compares its elements to a predefined rule set, and highlights rule deviations.

Technical guideline are applied to verify that a user interface's implementation is standard-compliant and thus can be assumed to be rendered correctly (i.e.,

[5]The presented AUE tools are mainly focused on usability of Web-based software; however, there exist many similar tools for desktop-based software as well.

[6]E.g., `http://snook.ca/technical/colour_contrast/colour.html`

[7]E.g., `http://www.checkmycolours.com/`

[8]E.g., `http://www.screen-resolution.com/`

without display errors) on the users' computers. For example, the World Wide Web Consortium (W3C) [123] has published a set of online tools which can be used free of charge to verify that a Web page's source code (e.g., HTML[9], CSS[10], RDF[11]) is well-formed.

Other guideline checks are concerned with qualitative usability properties; these tools are similar to the "heuristic evaluation" method in that they test for high-level usability principles. For example, a range of tools[12] covers accessibility requirements (e.g., alternative texts for images on Web pages; appropriate color contrasts; all functionality is controllable via keyboard) of Web pages. More general usability aspects (also including aesthetic properties) can be checked as well [90].

Model analysis Based on existing or especially created models of the application structure, model analysis tools allow to verify usability-related aspects, not merely perceiving user interface screens as independent units but also taking into account the navigational context.

Many software development methods implicitly or explicitly comprise the creation of application models. For example, many Web development frameworks are based on models of the navigation structure which can be accessed for usability evaluations [6]; more generically, model-driven development (MDD) by definition structures development around a comprehensive model which is suitable for AUE [36].

1.3 Research Context

The research presented in this thesis to a large extent has taken place in the context of the project "Mobile Multi-Media Replayer" (m3). This project was conducted within the framework of SOFTNET Austria, and was partially funded by the Federal Ministery of Economics and Labour of Styria, Austria.

The research goal of m3 was to evaluate agile development methods (particularly XP) regarding their suitability for small and medium size enterprises. A strong focus was put on integrating usability engineering methods with agile development. The project topic was to plan, implement and evaluate a Web-based video portal for

[9]E.g., `http://validator.w3.org/`
[10]E.g., `http://jigsaw.w3.org/css-validator/`
[11]E.g., `http://www.w3.org/RDF/Validator/`
[12]E.g., `http://wave.webaim.org/`

mobile devices. The aim was to exemplify the agile development method on a "real-world" example, in contrast to the "toy applications" often implemented in scientific projects. Therefore, the developed software was also presented to market experts and potential customers. The retrieved feedback was integrated in the development cycle to further simulate a business-driven development project.

The project's core team was located at the Institute for Software Technology of the Technical University Graz. The project consortium additionally comprised four Austrian companies, each of which brought in important knowhow: network and communication technology consulting was covered by Kapsch CarrierCom; video content analysis was facilitated by Sail Labs Technology; user interface design was supported by E-NOVATION Gmbh; and usability expertise was provided by CURE - Center for Usability Research & Engineering.

During the course of project, the benefits of continuous and automated testing—a key practice of XP—became apparent, thus the practice was extended to the field of usability. In a feasibility study conducted in tight cooperation with CURE, usability-relevant tests were added to the project's suite of automated tests. Then, as this first attempt turned out to be successful, a more comprehensive testing scheme was devised to verify further usability aspects.

The concept of an automated usability evaluation module as part of the Continuous Integration system was derived from the positive experience gained in the project. This concept is presented in this thesis.

1.4 Results of this Thesis

The research conducted in the m3 project and resumed for this thesis yielded results on different levels:

- an extensively documented and evaluated agile development process integrating usability engineering as a core practice
- a software product which was evaluated concerning usability and market potential
- concept and implementation of a fully automated usability evaluation tool, ready for integration in the agile development process

1.4.1 Agile Development Process Description

The XP development method is optimized for small team sizes. It incorporates a strong integration of the project customer who is involved in all design decisions throughout the project. This practice is founded in the observation that the initial project conception, formulated as a requirement specification document, rarely is kept unchanged during the course of the project. XP promotes the often inevitable requirement changes to a first-class right of the project customer, thereby increasing the customer's control, but also granting additional responsibility.

One major drawback of this approach, however, is that the "big picture" of the developed software is not necessarily kept consistent during the project. This can cause problems especially in the user interface of the software, where inconsistencies are particularly visible. As a result, the developed software suffers from usability problems.

The extended XP process described in this thesis aims at solving this issue by incorporating User-Centered Design in the process workflow. This is implemented by three main factors:

- Development and user interface design iterations are synchronized so that functionality and usability can be planned on the same level.

- A usability expert is co-located with the development team or, if not possible, at least available for requests (e.g., via phone or instant messaging). As a minimum requirement, the expert must participate in planning meetings (both at iteration and release scope). This ensures that no design decision is taken without considering its usability implications; furthermore, it helps to increase the awareness for usability among the development team.

- A range of usability engineering instruments is used at different stages of the project: User Studies, Extreme Personas, Usability Expert Evaluations, Usability Tests, and Automated Usability Evaluations.

In the remainder of this thesis, the tailoring of the standard XP method, the integration of usability engineering, and the applied usability instruments are described in detail.

1.4.2 Application Development

During the m3 project, a Web-based video portal for mobile devices was developed. The application facilitates video streaming to avoid the necessity for storing video files on end-user devices. Furthermore, the video portal is implemented as a Web application instead of a native rich client application and thus can be accessed via the Web browser available on any modern handheld device. The application allows users to browse the available video material by various categories as well as to search in the entire metadata, including the spoken text of the videos.

The user interface was optimized for the use on handheld devices, taking into account the various limitations of such gadgets (e.g., small screen resolution; high color contrast due to unfavorable lighting conditions when used outdoors; limited input capabilities). The specific usability needs of this kind of application served as a testbed for the usability-aware XP development process described in this thesis. The success of the development approach was evaluated by means of a usability expert evaluation and a user study.

1.4.3 AUE Tool and Process Integration

Driven by the experiences with AUE gathered during the m3 project, a comprehensive, fully automated and integrated usability evaluation tool for agile Web application development was designed. The tool automates the "capture" and "analysis" activities, and to some degree also the "critique" activity of usability evaluations.

Capturing of usability data is implemented by means of a Web crawler, i.e., a tool component which automatically explores the Web application in a methodical manner by following navigational links and downloading pages. The crawler collects two kinds of usability-relevant data: page content (both as HTML content and as a screenshot of the rendered page) and navigation structure information (in the form of a graph data structure). The content of downloaded pages can be analyzed to identify in-page usability problems, e.g., by performing guideline checks (see Section 1.2.3). The graph structure of the application, on the other hand, can be used to analyze various navigation-related properties of the application, including navigation consistency, page reachability, and click path lengths.

A central design goal of the developed AUE tool was to minimize the required "effort level", that is, to keep the configuration and maintenance workload for software developers using the tool as low as possible. While additional, optional configuration effort can increase the benefit (i.e., the number and quality of correctly identified usability problems), the most basic tool setup requires no more effort than to provide the Web application's start page. Furthermore, the tool can be integrated with the project's CI system. This yields two benefits: the execution of the usability evaluation is triggered automatically at defined points in time (e.g., after every change applied to the code base, or at every nightly build), and usability problems are reported to the developers with the same level of importance as functional errors identified by broken functional tests.

The described tool concept has been implemented as a comprehensive proof-of-conecept and is currently being evaluated by usability experts at CURE.

1.5 Structure of this Thesis

The body of this thesis mainly consists of pre-published texts, except Chapters 1, 7 and 8, which are previously unpublished. The pre-published texts have been edited and adapted to the context of this thesis where appropriate.

The following list gives a short description of each chapter's contents:

1. In this initial chapter, the two central fields of research discussed in this thesis are introduced: agile software development and usability engineering. Additionally, the context of research of this thesis is presented, and a preliminary overview of the thesis' results is given.

2. In Chapter 2, the project used as a testbed for the research of agile development methods is presented. The developed application, a Web-based platform for video playback on mobile devices allowing users to access a large database of content, is described. Additionally, the agile development process applied for conducting the project is outlined, with special focus on the integration of usability engineering practices.

   This chapter is a refined version of the texts "User interface design for a content-aware mobile multimedia application: An iterative approach" [51], published in *Frontiers in Mobile and Web Computing: Proceedings of MoMM2007 & iiWAS2007 Workshops*, 2007, and "User Interface Design for a Mobile Multimedia Application: An Iterative Approach" [52], published in *ACHI 08: Proceedings of the International Conference on Advances in Computer-Human Interactions*, 2008.

3. Chapter 3 describes the applied development process in detail, examining the threefold alignment of project activities (application development, business development, scientific work). The actually performed development activities are compared to the practices originally stipulated by XP.

   This chapter is a refined version of the text "Optimizing Extreme Programming" [50] published in *ICCCE 2008: Proceedings of the International Conference on Computer and Communication Engineering*, 2008.

4. Chapter 4 describes usability-related adaptations of the XP development process. The integration of five human-computer interaction instruments (User Studies, Extreme Personas, Usability Expert Evaluations, Usability Tests, and Automated Usability Evaluations) into the development process is explained.

   This chapter is a refined version of the text “Integrating Extreme Programming and User-Centered Design” [54] published in *PPIG 2008: Proceedings of the 20th Annual Psychology of Programming Interest Group Conference*, 2008.

5. In Chapter 5, the previously discussed aspects (application design, adapted XP process, applied usability practices) are recapitulated. Then, the results of a usability study (comprising usability tests and questionnaires) are summarized.

   This chapter is a refined version of the text “Agile User-Centered Design Applied to a Mobile Multimedia Streaming Application” [53], published in *USAB 2008: Proceedings of the 4th Symposium of the Workgroup Human-Computer Interaction and Usability Engineering of the Austrian Computer Society on HCI and Usability for Education and Work*, 2008.

6. Chapter 6 presents the concept and design of a further usability study conducted in the context of the m3 project. The selection of study participants and the study setup are described.

   This chapter is a refined version of the text “Concept and Design of a Contextual Mobile Multimedia Content Usability Study” [55], published in *ACHI: Proceedings of the Second International Conference on Advances in Computer-Human Interactions*, 2009.

7. In Chapter 7, an AUE tool which was designed and developed based on experiences in the m3 project is presented. Initially, the general architecture of the tool is outlined. Then, the tool’s AUE features (i.e., the discoverable usability problems) and potential extensions are summarized. The applicability for usability experts as well as software developers is discussed. Finally, the facilities for integrating the tool in CI environments are shown.

8. Chapter 8 presents conclusions drawn from the research presented in this thesis, including both successfully achieved results and identified problem areas. Finally, topics for future research in the field of AUE, with special focus on the context of agile development, are presented.

Chapter 2

User Interface Design for a Mobile Multimedia Application: An Iterative Approach

Mobile phones have become full-featured mobile computers. Applications providing good user experience and taking full advantage of the increasing capabilities of mobile phones are still rare. One such application is audio and video on mobile phones, which is expected to become a killer application in the near future. A lot of valuable audio and video content is hidden in archives of content providers. We are developing an application that enables a user to perform content-based search for audio and video content in large databases and play it on a mobile phone virtually anywhere, at any time. Our approach to application development focuses on the adoption of agile software development methodologies and user-centered design, emphasizing iterative user interface development involving usability engineers and non-technical users. Thus, the application evolves according to the needs of the end user, providing maximized usability and customer satisfaction.

2.1 Introduction

Mobile computing is leading a revolution. Our lives are changing at a pace never experienced before in human history. A wide variety of applications for mobile phones is available at the moment. Still, there are not so many full-featured applications which utilize the available bandwidth and are accepted by the users.

Studies show that multimedia – Audio and Video (AV) – consumption is on the edge to become one of the next killer applications for mobile devices [32]. User behavior in consuming AV is changing. Traditional broadcasting is losing more and more audience because online and mobile AV intrudes heavily into this area. A recent report states that 43 % of Britons, who watch video regularly from the Internet or on a mobile device, are now watching less TV than before [11]. Clearly, it is in the interest of broadcasting companies to adapt to these changes in user behavior and invest in these new technologies. For these companies, one of the major advantages of mobile phones, compared to other devices, is that they can charge for their services easily and directly, as the existing infrastructure can be reused. Additionally, the possibility to place advertisements for specific user groups is a huge benefit. At the same time, customers are given the flexibility to access rich multimedia content from anywhere, at any time.

The major problem for an average user is the combination of the overwhelming amount of multimedia content available and unsatisfactory user interfaces for accessing it. Usability is the key success factor for such applications.

For this reason, we are developing an application that enables a user to perform content-based search for AV content and play it on a mobile phone. This content includes radio and TV archive material, such as documentaries or other recordings of historical, political and cultural importance, discussion programs, movies, music videos, audio books, and music. The AV content will be stored in a database containing transcribed speech from the clips, as well as additional metadata, such as titles or a summary, where available. The media delivery will be based on standard web technology. This will enable people to use this service with almost any modern mobile phone. Furthermore, the application addresses not only the emerging functional and cognitive needs of the users, but also the objectives of the content providers. The application is designed keeping in mind the social interaction aspects of users. The system provides different community-building features to encourage interaction amongst them. The aim is to build a community platform for mobile phone users, where they can share their views and interests about AV content provided by the system. The feedback from the community will reflect the current trend of multimedia consumption as well.

One of the research goals of this project is to apply usability test procedures for mass-market applications on mobile phones. At present, usability testing for mobile phones is cumbersome and too expensive for small and medium sized enterprises. Another objective of this project is to automate certain parts of the usability testing procedures and provide a testbed for effective and efficient mobile usability testing. Special emphasis is placed on the adoption of agile software development methodologies, in particular Extreme Programming (XP), for mobile phones and their user interfaces.

This chapter presents an overview about related work. The following section presents different usage scenarios. Afterwards, the usability engineering process, which is applied to the iterative UI development cycle, is described. The next section defines various user-based recommendation approaches. Finally, a conclusion is given.

2.2 Related Work

This section provides an overview of related work. Basically, existing work can be divided into the categories applications and iterative user interface design. Related work in each of the categories is examined in the following subsections.

2.2.1 Applications

This section provides an overview of different AV search and streaming applications. They can be categorized by means of features they offer, particularly speech recognition, which is used to enhance the meta data for advanced search and streaming on mobiles.

Mobile YouTube [127] and MobiTV [25] allow to search for content and stream it on mobile phones. Mobile YouTube has the big advantage to have Google as parent organization, but the content is very limited and consists only of user-contributed material. In comparison to that, MobiTV offers different realtime worldwide TV channels streaming without search functionality, but for streaming on a mobile phone, client-side software is required.

In contrast to this, Blinkx [21], TVEyes [117], and EveryZing [35] perform speech recognition in order to enhance the search with additional meta data but do not provide mobile phone streaming. Blinkx provides user-contributed content as well as content from broadcasting companies. The content offered by TVEyes is restricted to news material, and the content of EveryZing is restricted to web based content.

Other applications offering online video search and streaming are JumpCut [63] and Joost [61]. Neither has the feature of speech recognition or streaming on mobiles. Jumpcut has the big community of Yahoo users but is limited to user contributed content. Joost provides more content, but is based on peer2peer technology requiring its own client-side software still in its beta phase.

The comparison shows that there is no application offering both features, speech recognition and streaming on mobiles, at the same time, which our application does.

2.2.2 Iterative User Interface Design

There already exist approaches of combining agile methodologies and Usability Engineering [48][81][29]. However, to the best of our knowledge none applies this specific composition of practices: we combine XP [15], user-centered design, and an iterative user interface design approach utilizing different HCI instruments such as user studies, extreme personas (a variation of the personas approach), usability expert evaluations, usability tests, and automated usability evaluations.

2.3 Usage Scenarios

In daily life, many people are at work or school at day time. They have no time to watch TV or listen to radio programs, because of the schedules set by the broadcasting companies. Time-delayed, played-back, and individually-delivered AV on mobile phones provides a new platform taking radio and TV into the street, car, public transport, waiting room, park, and virtually any other location. This type of application creates additional audiences, eager to access multimedia content during new prime times set by themselves, that is, in their commuting periods or other idle times. Also, a market survey shows that consumers are interested in using this technology and are ready to pay a realistic price for these services [25]. The basic idea of such a system can be illustrated by the following sample usage scenarios.

2.3.1 TV Archive for Subway Riders

- A commuter in the subway searches for “Fernando Alonso”.
- The system matches each word of the textual search query with the positions it occurs in each AV clip.
- The system presents a list of clips in which the name “Fernando Alonso” has been mentioned, sorted by temporal occurrence and relevance based on content, e.g., how often the name was mentioned.
- The user selects one of the presented entries.
- The system’s media server streams the selected clip to the user’s mobile phone.

2.3.2 Radio Archive for Car Drivers

- A user listens to the last sentences of a radio broadcast about the “European Constitution”. The user still has to travel with the car for some time and therefore searches for the keyword “European Constitution”.

- The system lists a number of related news items, interviews, and documentaries.
- The user selects the desired topic.
- The handsfree set of the mobile phone plays back the selected material through the car’s stereo.

2.3.3 Media Recommendations for Users

- A user wants to consume AV content but has nothing particular in mind.
- The user asks the system for recommendations.
- The system generates recommendations based on the user’s stored preferences and on the recent behavior of other users. A short description of each item is provided as well.
- The user selects an item and plays it on the mobile phone.

If the user interrupts the media stream, in all scenarios it is possible to resume at the previous location at any time, even weeks later. This feature is unavailable with regular broadcasting or streaming systems. The user of this system has more flexibility for consuming the AV content. This is particularly important because of the short continuous viewing or listening periods. For example, while commuting, interruptions and (possibly much) later resumptions will be the regular case.

Such behavior is rather uncommon for AV consumption so far, especially for viewing video. But it is not so much different from the way a book is read, having breaks between reading periods. Thus, it seems plausible that users will be willing to switch to this new way of listening and viewing AV content with interruptions, as it brings them the convenience of being able to decide what to consume in a just-in-time way, independent of place and time.

2.4 Usability

User interface design determines the success or failure of almost any application. Massive AV consumption on mobile phones will be accepted only, if users can easily find what they are searching for. But search on a mobile phone presents unique challenges as compared to a PC [82]. The inherent interface limitations of mobile phones strongly constrain the choices of user interface and interaction design. Special attention has to be paid to the constraints of small screens [67], possibly unfavorable lighting conditions, and limited text input capabilities.

We propose an iterative and user-centered approach to user interface design and system development in order to solve the stated problems.

2.4.1 Iterative User Interface Design

Usability is evaluated in small iterative steps to gain insight into whether the users' functional and cognitive requirements are met. User interface prototypes of the system are developed and tested throughout the development process. As a result the fidelity of the prototypes increases and evolves.

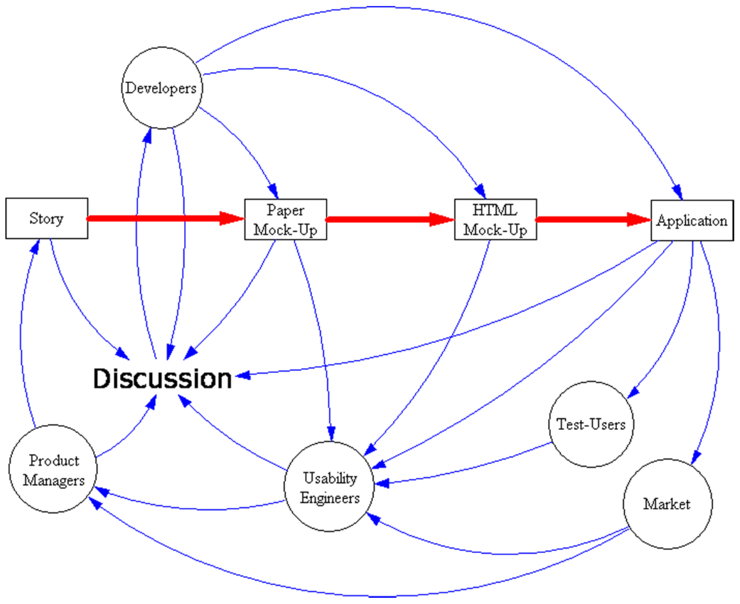
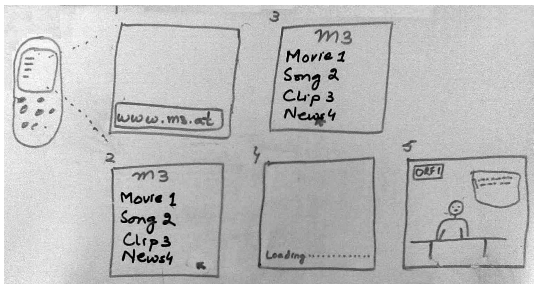
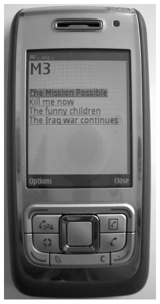


Figure 2.1: Iterative User Interface Design Workflow.

The workflow presented in Figure 2.1 illustrates the iterative design approach. The process starts with the creation of user stories by the customer or the product manager who acts as a representative of the customer. Developers create different paper mock-ups to collect and present ideas. A final mock-up is derived, serving as the basis for further development. The benefit of using paper mock-ups for the interaction design is that they can be designed and modified quickly. Because of that, the feedback given by the usability engineers and the users can be incorporated easily. An additional advantage is that it is easier for users to criticize simple and



(a) Paper Mock-Up.



(b) Application on Mobile.

Figure 2.2: From Paper Mock-Up to Mobile: The first Search-Results Screen.

rough mock-ups compared to ones which look neat and perfect from the graphic design perspective [102]. For simple interaction designs, a paper mock-up suffices as a basis for further discussions and the implementation. In more complex cases, an additional HTML mock-up is created based on the final paper mock-up.

The approach combines the quick feedback-and-change cycle of hand-drawn paper mock-ups with the more time-consuming process of computer-based prototypes. Paper mock-ups are used to get the basic concepts right, while HTML mock-ups are used for a more detailed view.

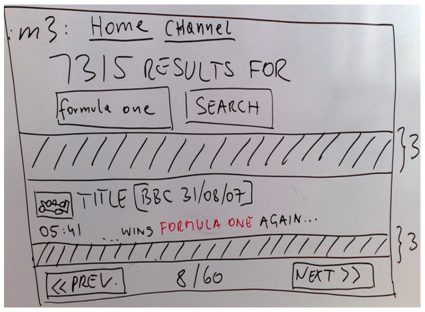
The designs are examined by usability engineers and tested by non-technical users. The feedback from the usability engineers, as well as from the users, is taken as input for further refinements of the design. Also, the results are incorporated into automated tests which are used, by employing test driven development, as an executable specification for the actual implementation. This feedback-and-change cycle provides insights into whether the user interface design is meeting different usability criteria.

For the actual user tests it is important to choose representative test users from different age groups, bearing in mind the targeted customers for the proposed service. These tests are conducted only after incorporating the feedback from the usability engineers on the paper as well as the HMTL mock-up. Therefore, the expensive part of involving real users can be done more effectively.

2.4.2 An Iterative Design Example

For the paper mock-up in Figure 2.2(a) the usability engineer raised the following issues:

- Missing strategy for displaying larger result sets (balance between pagination and scrolling).
- Missing feedback mechanism to highlight the pointed-to item (especially needed in unfavorable lighting conditions).
- Undefined application behavior after playback of the clip ends (no return option specified).



(a) Paper Mock-Up.

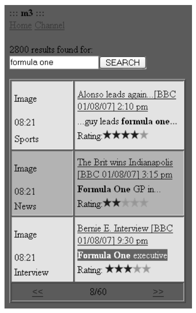
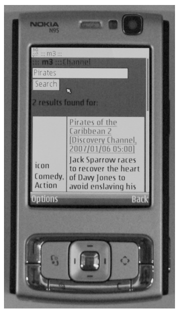

::: m3 :::
Home Channel
2800 results found for:
formula one SEARCH

Image 08:21 Sports	Alonso leads again. [BBC 01/08/07] 2:10 pm ...guy leads **formula one**... Rating ★★★★★
Image 08:21 News	The Brit wins Indianapolis [BBC 01/08/07] 3:15 pm **Formula One** GP in... Rating ★★★★★
Image 08:21 Interview	Bernie E. Interview [BBC 01/08/07] 9:30 pm **Formula One** executive Rating ★★★★★

<< 8/60 >>

(b) HTML Mock-Up.


::: m3 :::
m3 ::: Channel
Pirates
Search
2 results found for:
icon Comedy, Action | Pirates of the Caribbean 2 [Discovery Channel, 2007/01/06 05:00] Jack Sparrow races to recover the heart of Davy Jones to avoid enslaving his
Options Back

(c) Final Application.

Figure 2.3: An additional HTML Mock-Up: A refactored Search-Results Screen.

Figure 2.3 shows the mock-ups of an improved version of the same feature. Here, an HTML mock-up was created after the paper mock-up. The design was derived from the following user scenario (a so-called user story in XP [15]):

Search results presented to the user should contain clip-related information which can aid the user in choosing the clip. Also, the context in which the keyword was found, as well as the number of search results, should be visible. Furthermore, it should be possible to start a new search immediately.

It can be seen that two issues from the previous feedback, namely pagination/scrolling and item highlighting, have been addressed in the refactored design. On the refactored design depicted in Figure 2.3 the usability engineer provided the following feedback:

- Forms: It is common to leave some white space between text-input-field and the button. For graphical user interface solutions there are distances fixed in guidelines for the operating system - for mobiles we recommend to put the button in a second line (this is preferred to putting the button close to the input field).
- Background Colors (Table): The alternating rows should vary decently, and should preferably be coloured in slightly different shades - the selected colors are "eye bending".

2.4.3 User-Centered Application Design

User interface development cannot be separated from the development of the underlying application. Intended user interactions strongly influence the internal structure and functionality of the system [28]. A big issue in mobile user interface practice is that current approaches are not sufficient for mobile phones [108]. Therefore, another focus is placed on the combination of iterative user interface design and user-centered application design.

The design process and user tests provide feedback about the user interface which will be used for the system's functional requirements. It reveals the mental model of the users, how they expect the system to work. The assessment of each feature from the perspectives of the users influences the whole development process of the application and addresses the problem that conversation only with the stakeholders is not enough [60]. As the application development is done in short iterations, the developers are able to refactor the system continuously according to the feedback derived from the parallel, iterative, user interface design process. Hence, the system evolves according to the needs of the end user and the specifications derived from actual usage.

2.5 User-Based Recommendations

Web-based companies already use recommendation systems with great success. Amazon, for example, has millions of customers. Seeing the benefits of recommendations, Amazon has developed its own technique called “item-to-item collaborative filtering”. Their customers regularly take advantage of these recommendation facilities when making their purchases [75].

The personalized approach of our system makes it possible to implement user-based recommendations. The unique identification of a user is necessary for accounting purposes, implying that a user profile has to be managed by the system. This profile will be augmented with additional data, which is used for recommendations to the same and to other users. The data is collected by means of two information acquiring models, the interactive model and the behavior-based model.

2.5.1 Interactive Model

The interactive model is based on user ratings. After users finish consuming an item, they are able to rate it according to their liking and preferences. Information about which clips a user consumed is stored in his individual profile. In addition, the corresponding clip ratings are stored as well. The rating of a specific clip in each user profile affects the overall rating of the clip in the database. The individual ratings are still traceable. For more personalized recommendations, ratings of similar user groups can be combined.

2.5.2 Behavior-Based Model

The behavior-based model is applied by collecting usage data. Information about the clips consumed, and the duration of the consumption, is stored in the user profiles. This is used for user-specific recommendations. If many users stop the same clip after a short time, this clip is most likely not very interesting. Of course, this equally depends on the overall playing time. Therefore, a ratio measure is used for clip rating. The system will take into account that users are allowed to stop and resume clips at any time, which can influence the measurements. Alternatively, it is possible to consider ratings of a specific user group only, as described in 2.5.1.

2.5.3 Model Combination

When generating recommendations, the two models already described are combined. For the system, user ratings are more important than usage data. However, ratings may not be available for every item. In this case, only behavior data is used. Furthermore, the changing preferences of users are taken into account by adding a time-descending weighting factor.

There are different scopes for the rating mechanism. On the one hand, all users are considered, and on the other hand, just a specific user group is considered. This results in different recommendations. The default recommendation setting can be overridden by user preferences.

2.5.4 Implications

An attractive feature of the system is the possibility to target advertisements more precisely. This feature is useful for companies wanting to address specific user groups. Additionally, users benefit, because they receive only advertisements related to their interests. For example, Google's Gmail is using this technique for advertising purposes on its popular mail accounts. The large user base of Gmail is a valuable target for business. The advertising is tailored to users' mail contents. Gmail also offers the possibility to use mobile devices [41]. As Google has purchased YouTube, it is expected that this trend will continue. The advertising and search capabilities are, or will be soon, extended to video content as well [82].

The feature of collecting additional user data provides continuous feedback, enabling constant improvement of the system. By recording this information, valuable data about how the user is interacting with the system is obtained. This allows to react quickly to new usage patterns and needs as they arise.

2.6 Conclusion

The emerging technologies of delivering rich AV content on mobile phones will result in reducing the number of users for traditional TV and radio broadcasting services. This might compel traditional TV and radio broadcasting companies to become partners in this technology by opening their huge collections of AV content to the public. Today's consumers are willing to pay a reasonable price for this service [25]. According to current trends, the community of mobile phone application users will grow rapidly. The standards concerning codecs, formats, and technical infrastructure, required for AV content delivery on mobile phones are already well established. These general trends are in favor of the development of this type of application.

The critical factor for this kind of applications will be user acceptance, which heavily depends on the fact that the system suits the user's needs. To address this issue, in our software engineering process, usability engineers are accompanying the system development team during the whole project life cycle. The engineers provide suggestions that are incorporated continuously into the system. This process is facilitated by the short development iterations and has proven to provide early and valuable feedback. The test-driven development approach allows to convert these findings into a set of automated tests. These tests define the functionality of the system, serve as specifications for the development, and prevent previously discovered usability problems from reappearing. Furthermore, the inclusion of test-users provides additional benefits. This continuous input allows to adjust the system effectively according to the end-user's needs.

Chapter 3

Optimizing Extreme Programming

The vast amount of published literature explaining the "right way" of doing Extreme Programming shows, that in practice there simply is no single right way. Even though Extreme Programming is a simple and slim process, it has to be tailored to the nature of each team and project in order to provide the benefits it promises.

Our team has been working on a project employing the Extreme Programming methodology, experiencing unique issues arising from the distinct project setup and team composition, as well as the additional academic interests in the project. Initially, we aimed at applying "pure Extreme Programming", but it became more and more obvious that for our project some of the practices just cannot be applied in their "pure" form. The concrete interpretation of these practices determines if Extreme Programming can be applied successfully in the context of a team and a project.

In order to reach an optimized process for our project, we continuously evaluate different approaches of applying Extreme Programming practices on short release basis. We have noticed that some practices can be adopted directly, while others need to be tailored according to the unique environment. In this chapter, we reflect on our process based on the data collected through code analysis and process evaluation tools, as well as notes of process retrospective review meetings. The lessons we have learned can also help other teams to lead them to an optimized Extreme Programming process for the success of their projects.

3.1 Introduction

The number of software projects constantly increases, but the overall success rate is still rather low [77]. Many projects fail because of their inability to cope with the changing user requirements. Heavy up-front design without continuous feedback from the customer is another factor. To have a greater probability of success, the developers need a software development process which should be flexible enough to cope with the constantly changing requirements and which is also people-oriented. Agile software development methodologies have emerged in response to these needs, as agile methods give more value to individuals, working software, and change [16].

The intention of large scale research into software engineering techniques has been a formulation of an ideal methodology that can consistently and predictably lead to software development success [86]. A recent survey shows that agile software development has seen far better success rates in comparison to other methodologies [4]. Being an emerging agile methodology, Extreme Programming (XP) offers a number of practices, values, and principles, which are advised to be adopted in order to run a software development project [15]. XP is being experimented in different ways to make it fit to the specific needs of the projects as well as the development teams [111].

This is of interest for many academic, research and development organizations, as there is a room for more explorations in the area of agile development methodologies. Many experience reports in the field of agile research have been presented, helping other teams passing through the same process [111]. However, there is still a need for more experience reports of teams already using XP, giving valuable information for those, who are planning to adopt the XP methodology. In addition, this data also serves the purpose of defining the agility level of software development teams. In this chapter, we present our own experience about the XP methodology which we have gained by applying it to a software development project.

The next section describes our project environment. In Section 3.3 our XP process, focusing on the practices applied in our project, is presented. Section 3.4 describes our reflections. Finally, a conclusion is presented.

3.2 Project Environment

We are developing a multimedia streaming application for mobile devices as a testbed to analyze the XP methodology. XP is being applied in a progressive manner: each practice is consciously applied and constantly evaluated in order to yield process improvement. Hence, each team member is not only taking part in the software development process, but is also making a critical analysis of the way the process is being used. The objective of the project is twofold: on the one hand, having a software product fulfilling the user requirements, and on the other hand, XP process optimization as profound academic research.

We have been working on the project since summer 2007. The project's duration is three years, which is quite appropriate for the development of the product, as well as for the team members to carry out the research for their doctoral studies.

We are a team of six regular members: five developers and a product manager. The product manager plays the role of the "On-Site Customer", enabling the implementation of this XP core practice. Furthermore, he communicates with the partners who come from various domains, including telecommunication, content providing, and hardware infrastructure. Also, developers communicate directly with the engineers of a partner usability research company regarding usability issues.

This project's main scientific and academic goal is the analysis of agile software development methodologies. Another goal is research in the field of mobile application usability. Additionally, the business partners are interested in commercial aspects of the project. Figure 3.1 shows the allocation of the application, research, and business aspects after the first one-month release.

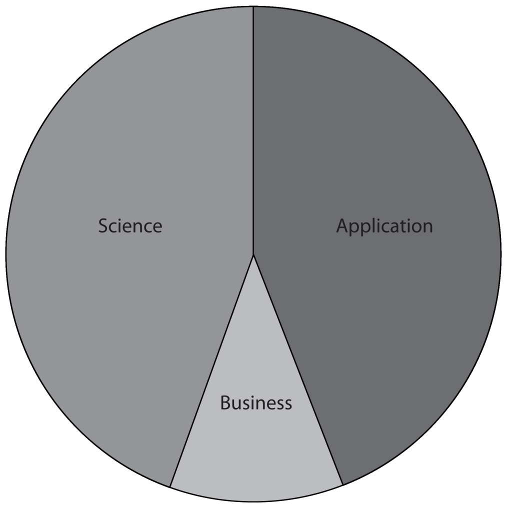


Figure 3.1: Application, Research and Business Aspects in a Release.

3.3 Process

It was pre-decided that XP will be used as a methodology. Therefore, effort was made to establish a basis for its implementation. None of the team members had worked in an agile development environment before, so available literature, especially [12][15], was used for initial guidance and reference. The team tried to apply all those main practices which could be applied at the initial stage of the project. In this way, some of the basic practices were adopted fully, while others were implemented partially, or in modified form. The following subsections outline how the practices have been implemented, as well as the current status.

3.3.1 Fully Implemented Practices

Small Releases

We aim to release a working version of our application to the project partners on a regular basis. In the early stages of the project, the duration of one release cycle was set to one month. This enabled us to quickly get feedback on our work from the partners in order to sharpen our vision of the project goal. As the project took shape, the release size was gradually increased to two and finally to three months. For now, we are satisfied with three-month release cycles, which complies with the quarterly planned business targets of the partners.

For tracking short-term progress, releases are further divided into iterations. Initially, we used a one-week iteration duration, but later we shifted to two week iterations in order to reduce the administrative overhead added by the iteration planning meeting.

The Planning Game

The planning meetings are held on iteration and release basis. Release meetings are attended by all project partners who, as stakeholders of the project, identify and define user requirements. These requirements are then are formulated as XP user stories.

In our project, we distinguish three main types of user stories: application, business, and science.

- Application stories are “traditional” XP user stories, representing features of the application.

- Business stories describe diverse business activities, e.g., collaboration with partners, meetings, presentations, etc.
- Science stories are only relevant for our team, hence they are not specified during the release planning meeting. They depict the academic and research activities of the team, e.g., paper-writing, performing process analysis, etc.

The stories generated during the release planning are written down on story cards and are prioritized by the participating partners. To visually represent the different story types, we use a simple color encoding for the different story types: application story cards are white, business cards are green, and science cards are yellow. Then, the developers estimate the time required for implementing the stories. Also, stories created in former release meetings that have not yet been implemented are re-estimated, if required. The final step of the release planning is to select a subset of the available stories. This is done by prioritizing the available stories and selecting as many top-priority stories as "fit in" the available velocity. The amount of user stories to be scheduled is determined by following calculation: the sum of their estimates is lower than or equal to the sum of the estimates of the user stories finished in the previous release.

The iteration meeting is held at the beginning of each iteration and is attended only by team members including the product manager. The product manager takes the role of the on-site customer and selects and prioritizes stories for the current release. The stories are then broken down into detailed tasks, which are again estimated by the developers. The intended results of the tasks are explicitly defined by writing acceptance criteria.

Figure 3.2 and Figure 3.3 show user story cards of release and iteration plannings stuck on whiteboards.

Pair programming

All production code is written by two developers working together on the same machine with one screen, one mouse and one keyboard [12][15]. This practice has been implemented from the first day. It helped us in sharing the project-specific knowledge and improving the technical skills of the developers.

We also applied working in pairs to non-technical stories. For example, the product manager pairs with a developer when writing customer-acceptance tests and when creating business assignments requiring technical knowledge. Working in pairs on research stories was not successful. For this kind of stories everyone works solo.

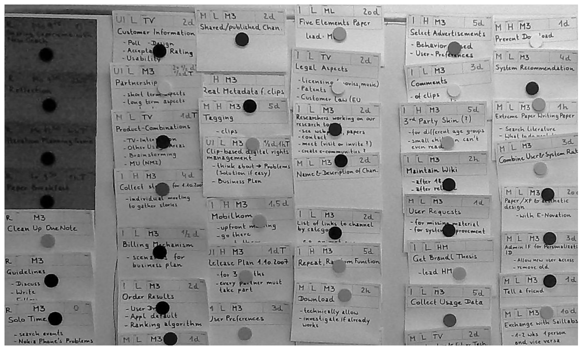


Figure 3.2: Selected Story Cards on the Release-Board (Release Planning).

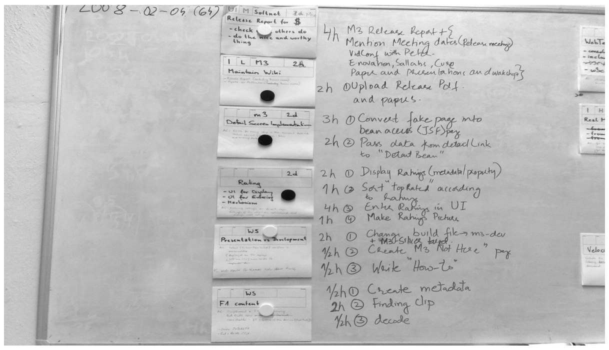


Figure 3.3: Selected Story Cards on the Iteration-Board (Iteration Planning).

In daily stand–up meetings, the developers sign up for tasks according to their interest. The pair partners are chosen voluntarily [106].

Sit Together

The team members including the product manager sit in one large room at their private workspaces. There are three separate pairing stations in the same room. Due to sitting in the same room, the face–to–face communication has resolved many difficulties which arose within the project, the team, and the process.

Collective Ownership

A Subversion repository is used for managing the code base. The code is shared by all developers. Whenever a chance for code improvement is identified and there is enough time at hand, the required actions are performed on the spot. The changes are communicated in the stand-up meetings, during pair programming, and sometimes through a short ad-hoc discussion involving all developers. One basis for a successful application of collective ownership is the strict adherence to coding standards.

40-hour Week

The purpose of the "40-hour week" practice is that developers should not work overtime, because tired developers make more mistakes during coding [46]. We strictly follow this practice.

3.3.2 Partially Implemented/Modified Practices

On-Site Customer

As the target group of the product being developed within in project is manifold, we cannot directly implement the on-site customer practice. Therefore, initially the product manager, as well as the developers, played the role of the customer. But soon, many shortcomings of this approach became apparent. The absence of common acceptance criteria for the stories resulted in long discussion cycles in the planning meetings and the implementation. We also felt difficulties in the prioritization of the stories. To overcome those problems, we decided that the product manager, who also communicates with our project partners, will play the role of the customer.

Metaphor

As everyone was new to the process and the project, there was no common understanding of the metaphor – the shared terminology about the project and the process [12][15]. This resulted in misunderstandings about the features to be implemented, which eventually led to the delaying of their delivery. The release planning meetings with partners, our internal iteration planning meetings, stand-up meetings, retrospective meetings, and pair programming have contributed to evolvement our metaphor.

Simple Design and Refactoring

From the very beginning the team aimed at keeping the design as simple as possible. The design started with creating paper prototypes to visualize customer requirements illustrating the customer how the requirements will be put into reality. An important factor of being able to keep the design simple is refactoring. For our team, simple design has been beneficial, because it facilitated the incorporation of changes demanded by the partners. Refactoring of code is not a routine practice in our team, but it is done on demand basis, that is, whenever any developer sees the opportunity to improve the code, or when we need to substantially change the application fundamentals, e.g., the usage of a new framework.

Test-first Programming

As all the team members were new to XP, it was difficult to follow the XP style of writing the failing automated test before any code [12][15].

Figure 4 shows a graph comparing lines of executable code, lines of test code and test coverage. For this, the data was collected using Emma, a Java code coverage tool [34], and LinesOfCodeWichtel [76], and is based on the work performed during the second release of the project. The low amount of test code and test coverage shows that the practice of testing is not well exercised by the team. For the subsequent releases, the implementation of this practice has improved. If not being impossible due to framework restrictions, tests are being written beforehand.

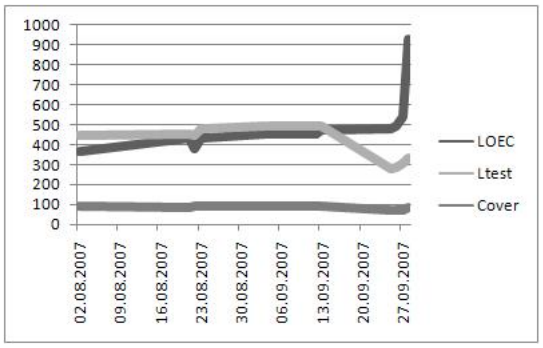


Figure 3.4: Executable Code versus Test Code and Test Coverage.

3.4 Reflection

In order to measure the performance of the team and to resolve human issues, a retrospective review meeting on the process is held after every iteration and release. This retrospective meeting is called "reflection meeting". It has helped us a lot to find out the reasons for difficulties faced during the process and their remedies. The common decisions that we take after these meetings are noted down and followed by all team members. Almost all XP values – communication, simplicity, feedback, courage, and respect – and XP practices adhere to human aspects [12][15]. The benefits of sitting together, face-to-face communication, feedback, stand-up meetings, the planning meetings, pair programming, and reflection meetings have contributed to improve not only our process, but also increased the overall morale of the team.

To review the development process, we collect empirical data from various sources describing our performance of each applied XP practice.

For a qualitative analysis, we perform the Shodan 2.0 survey [68] on a regular basis (e.g., at the end of each release). Additionally, we use quantitative data generated by the XP tracker tool "XPlanner" [126], and different code analysis tools [34][76].

Table 3.1: Subjective Metric (Shodan 2.0 Input Metric Survey).

Testing metrics	%
Test First Design	44
Automated Unit Tests	68
Customer Acceptance Tests	22
Planning metrics	%
Stand-up meetings	92
Short Releases	86
Customer Access / On-Site Customer	48
Planning Game	96
Coding metrics	%
Pair Programming	98
Refactoring	66
Simple Design	76
Collective Ownership	86
Continuous Integration	100
Coding Standards	84
Sustainable Pace	82
Metaphor	46

The data gathered using Shodan 2.0 Input Metric Survey shown in Table 1 gives an overview about the methodology and the extent to which a given XP practice is applied. As there was an explicit effort to apply these practices, low percentages indicate that either the team was not fully content with the practice, or the practice needs to be tailored for our project. For example, the team members perceived that pair programming was practiced for almost every development task, while metaphor was given the lowest rating because of the unfamiliarity of the team members with the project. These conclusions are also supported through iteration and release reflection notes and from key discussion points raised in stand-up meetings.

3.5 Conclusion

After working with XP practices for almost one year, our experience shows that most of the practices are helpful for a project with multiple objectives (in our case, research, application development and business). Pair programming helps in spreading knowledge. The benefits of co-location, face-to-face communication, stand-up and planning meetings, and retrospective review meetings have contributed to improve not only our process, but also to increase the overall morale of the team. The low ratings of some practices indicate that our team still needs more experience to apply them in a proper way.

We continuously try to optimize our approach to XP. Future results will help software development teams working under similar environments to improve their development process for the success of their projects.

Chapter 4

Integrating Extreme Programming and User-Centered Design

The success of a software development project is associated not only with tools and technologies, but it also depends on how much the development process helps to be user-centered and developer-oriented. Involving customers in the process and being people-oriented, Extreme Programming – one of the popular agile methods – can be a choice for developing a usable system. The project under study is a multimedia streaming application for mobile phones. The application allows to perform content-based search for audio and video content in large databases and play it on a mobile phone virtually anywhere, at any time. Our approach to application development focuses on the adoption of Extreme Programming and User-Centered Design, emphasizing iterative user-interface development involving usability engineers and end-users.

This chapter describes the process of integrating Extreme Programming with User-Centered Design and shows how an agile development technique facilitates to be user-oriented and at the same time preserves the social values of the development team.

4.1 Introduction

An inherently usable and technically elegant application cannot be considered a success, if it does not satisfy the end-users' needs. End-users are often left out of the development process [83]. A usable software application should focus on its end-users, their goals, and their satisfaction. Agile development processes - especially Extreme Programming (XP) - involve a customer as business representative, who is responsible to specify the business value of user requirements and prioritize them accordingly in the development. Along with this, XP possesses all the advantages of: on-time delivery, optimized resource investments, short release cycles, working high quality software, tight customer integration, incremental design and test driven development [12][15]. All these characteristics are in favor of the customer and ultimately benefit the end-user. XP, being people oriented, defines the whole social structure which is needed to run a development process in a productive way.

User-Centered Design (UCD) is a design approach focused on the information about the people who are the actual users of the product. This user focus is maintained by considering this information during planning, design, and development of a product [120].

Although XP and UCD are two different methodologies, both focus on the user. Due to the same main focus, both methodologies can be integrated very easily [42]. The integration will obviously result in a complementing process, which allows to gain the advantages of both worlds, and at the same time minimizes the deficiencies of both methodologies. The disadvantages of bot approaches are eliminated: XP lacks in knowing their true users and UCD lacks in a flexible and adaptive development methodology that lasts throughout the entire project [115].

In the context of this project, we integrate XP and UCD [44], utilizing different Human Computer Interaction (HCI) instruments such as user studies, personas, usability expert evaluations, usability tests, and automated usability evaluations [52].

The following section outlines the integration points of both methodologies by comparing the values of the two methodologies. Then, our project and team setting described in order to show in which context and by whom the process is used. We proceed by going into the details of our UCD process and afterwards examine the results of a usability study recently conducted by the usability engineers. Finally, a conclusion is given.

4.2 Common Values of XP and UCD

The core values of XP and UCD are applied to solve different issues: In XP, a simple implementation, fulfilling the minimum requirements of the application, is created and iteratively extended, while UCD tries to continuously improve the usability of the user interfaces. However, when comparing some of the core values, it seems obvious that the two development processes can benefit from each other's practices.

4.2.1 End-User Involvement

One of the core practices of XP is to have an *On-Site Customer*, a real user of the application under development, who is co-located with the programmers in order to answer domain-specific questions and give feedback on the system. This practice matches well with the testing of prototypes with actual users as proposed by UCD.

4.2.2 Continuous Testing

Continuous and extensive testing is at the heart of XP. It is mainly embodied by two practices: *Continuous Integration* runs all existing automated tests whenever the code base is changed or extended in order to check if the changes caused any undesired side effects. Most of these tests emerge from *Test-Driven Development*. First, automated tests checking the desired behavior are created. Then, the actual behavior is implemented and can be evaluated right away with the tests. This usually is done only for pure behavioral code, but can be extended to user interfaces. Tests can check the expected behavior of an interface, and these tests can be run whenever the code is changed.

The end-user tests of UCD are a valuable source for test targets. An unexpected user action that caused a problem in the application can be replicated as an automated test. By executing this test in the *Continuous Integration* process it is ensured that the problem, after solving it once, does not reappear.

4.2.3 Iterative Development

Both processes, XP [12] and UCD [44], propagate an iterative procedure of design and development [120]. An XP project yields *Small Releases* (another core XP practice) on a regular and frequent basis (usually a few months). Each release version is based on the previous one, incorporating new features and fixing bugs of the predecessor. Inside a release time frame, work is organized in “iterations” (usually taking one to four weeks). On an even smaller scope, many feedback-and-change iterations take place, especially in conjunction with *Test-Driven Development* and *Refactoring* (the practice of changing source code in order to improve its quality without changing its functionality).

UCD also proposes a design–test–modify circle for developing user interfaces. The scope of iterative development in XP and UCD differs. Releases and iterations in XP are mainly organizational units and *Refactoring* is considered to be a development tool. In contrast to this, UCD’s iterative user interface refinement is a more explicit process, as its involvement of external persons (the test users) makes it more complex. Nonetheless, iterative interface development of UCD fits well into the iteration principle of XP, because both approaches are aware of the value (and necessity) of evolutionary development.

4.3 Project and Team Setup

We are developing an application that enables a user to perform content-based search for audio and video content and play it on a mobile phone. This content includes radio and TV archive material, such as documentaries or other recordings of historical, political and cultural importance, discussion programs, movies, music videos, audio books, and music. The application is being designed keeping in mind the social interaction of users. The system provides different community-building features to encourage interaction amongst them [52].

In addition, one goal of the project is the analysis of agile software development methods, particularly XP, and to devise a usability test procedure for mass applications on mobile devices with emphasis on UCD and iterative user-interface design.

The team consists of six full-time regular members, having different social and cultural background; five developers (two of them are from South Asia and the others are from Europe)and a product manager who plays the role of the *On-Site Customer*.

The customer communicates with the project partners, who come from various domains, including user interface design, usability research, telecommunication, content providing, and hardware infrastructure. Also, developers communicate directly with the engineers of a partner usability research center regarding usability issues. The usability engineers working for our project are active in UCD research with the team.

4.4 The Design Process

The following subsections describe our adapted UCD process which is followed in application development.

4.4.1 Approach to UCD

User interface design plays a very important role in the acceptance of a web based application. The overall process of our approach to UCD is based on evaluating the usability of the application in small iterative steps. This helps us to gain insight into the real users' functional and cognitive requirements. We design prototypes of the user interface of the system and test them throughout the development process. As a result the fidelity of the prototypes increases and evolves.

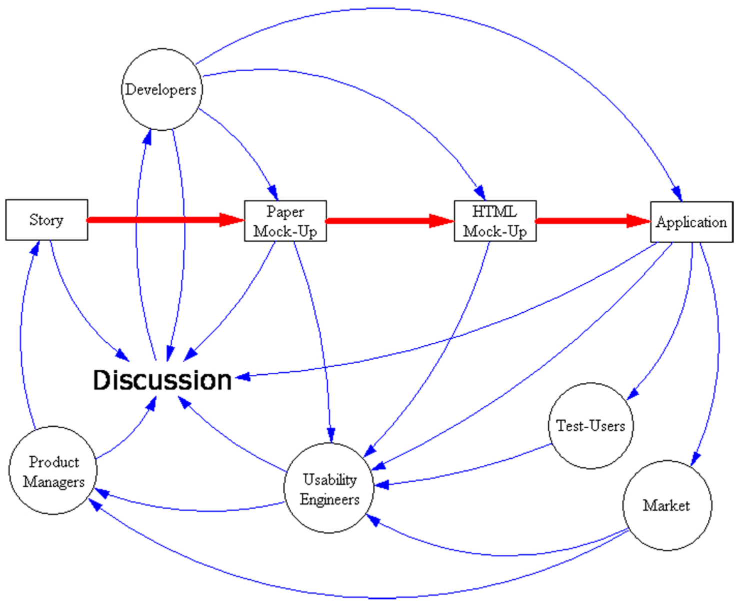


Figure 4.1: Iterative User Interface Design Workflow [52].

The work flow presented in Figure 4.1 illustrates our iterative design approach incorporating UCD into our XP process. From a broad perspective, the application development cycle starts with defining user stories (user-required application features), then comes to mock-up designing and at the end the actual implementation is performed. The process steps are implemented as follows:

- Different feature-related user stories of the application are created by the customer in coordination with all the stakeholders.
- Developers create different paper mock-ups for each of the required features to collect ideas and to present them to the customer.
- The customer decides which one of the mock-ups best suits his needs, or he suggests modifications to the mock-ups.
- A final mock-up is derived according to customer's likeness which then serves as the basis for the actual implementation.

- Once the implementation mock-up of a feature (or group of related features) is finished, the usability engineers are asked to give the feedback on it.
- After incorporating the feedback given by the usability engineers into the application, the end-user tests are conducted by the usability engineering team.
- The feedback on the application from the usability engineers, as well as from the test-users, is taken as input for further refinements in the user interface design of the application.
- The results are then incorporated into automated tests, which serve as an executable specification for the actual implementation.

This feedback-and-change cycle provides insights into whether the user interface design is meeting different usability criteria. As the application development is done in short iterations, the developers are able to refactor the system continuously according to the feedback derived from the parallel, as well as iterative, user interface design process. Hence, the system evolves according to the needs of the end-users and the specifications derived from actual usage.

4.4.2 Choosing the Type of Mock-Up

We make use of two different types of mock-ups; low fidelity paper mock-ups and high fidelity implementation mock-ups. The benefit of using paper mock-ups for the interaction design is that they can be designed and modified quickly. For simple interaction designs, a low fidelity paper mock-up suffices as a basis for further discussions and the implementation. An additional advantage is that it is easier to criticize simple and rough mock-ups compared to ones, which look neat and perfect from the graphic design perspective [102]. But for some features a high fidelity mock-up is required to clearly visualize the interface. As we have the benefit of an on-site customer co-located with the development team all the time, for those tasks a quick implementation mock-up is designed and presented to the customer. This implementation mock-up is then modified accordingly, if required by the customer. If our customer would not have been co-located with us all the time, it would have been difficult to have the benefit of this quick feedback-and-change cycle.

4.4.3 Frequency of End-User Tests

The end-user tests are made on an on-demand basis. That is, when the customer says that now is the appropriate time, from the business point of view, to run a usability test with test-users. Also, when there is enough of new functionality added to the application, it becomes effective to perform the usability tests and then continue with development. It would have been good, if user tests could have been made on regular basis, e.g., at the end of each release, but considering the expenses and resources required for it, we have kept it only on an on-demand basis. So, the expensive part of involving real users is done more effectively.

4.4.4 The Testing Workflow

Figure 4.2 describes our model of integrating HCI instruments (user studies, personas, extended unit tests, usability tests and usability expert evaluations) into the XP process [122]. It shows the interplay of the HCI instruments into the XP process. Applied correctly in different phases of the project the instruments are designed to reach the goal of improved usability of the application. It can be seen that end-users are integrated in two different ways. First, user studies are taken into account to develop personas [64]. The personas specify the direction of the development (by guiding the customer in identifying user stories) and are extended at the end of the iteration, when the vision about the user has broadened. This serves as an indirect end-user input for the development process. Second, feedback from usability tests performed by test-users (as part of the usability evaluations) serves as a direct input for further enhancement and development of the application [122].

4.4.5 Feedback from a Usability Test

The following points were highlighted as the result of a recent usability study conducted by the usability engineers on the latest version of the application available at that time. Figure 4.3 shows a screen shot of the application. The application was evaluated with ten test-users on a specific mobile phone. Results of the evaluation were (example issues):

- Improvements of layout and design. Some major changes in the layout and design of the user-interface were suggested by the usability engineers according to the feedback given by the test users.
- Improvements of the prototype's usability. Some usability issues became apparent for some navigation and data controls on the interface
- Improvements of the colour schemes. The users were also presented three different color schemes of the application. They graded the color schemes as best average and worst.

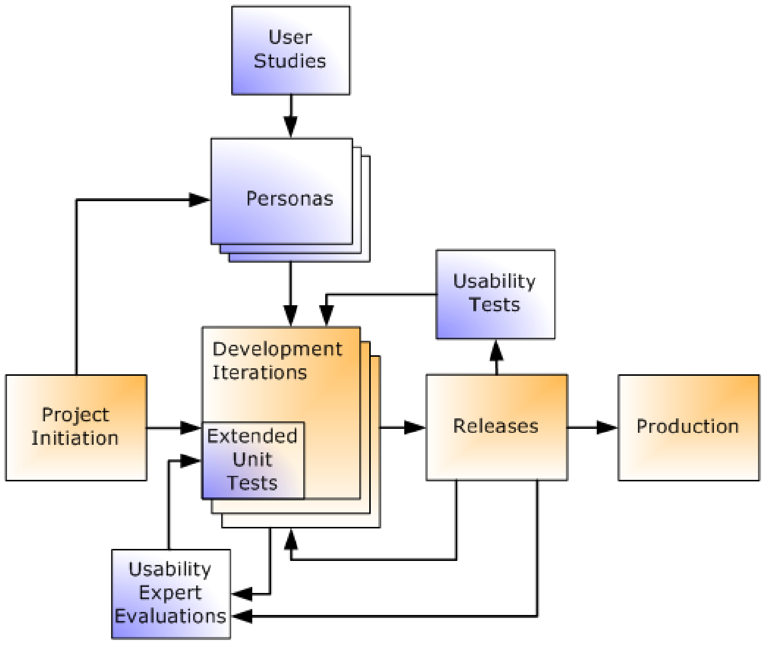


Figure 4.2: The Integration of HCI Instruments into XP [122].

During this study, two of the developers were participating as observers. This provided a great opportunity for the developers to see how users actually reacted to simple items and controls used in the interface. At the end of the study, many new stories were generated from the observations and interviews with the end-users. This approach of testing involves the end-users directly in eliciting their requirements: how they perceive the application, and what they want from the application.

4.4.6 Testing Issues

A big issue in mobile user interface practice is that current approaches are not sufficient for mobile phones [108]. For designing any software, use of UCD practices ensures that the product works [65]. This further supports the use of UCD the approach for user interface design. To enhance it further, we provide high fidelity implementation prototypes to our usability engineers for user testing. As paper prototypes are good and sufficient for verifying non mobile-based product requirements, in case of applications for mobile phones they are not sufficient for finding out and solving usability issues related to detailed interaction [65]. Also, it is very important that the application is tested on mobile phones and not on some web based simulator

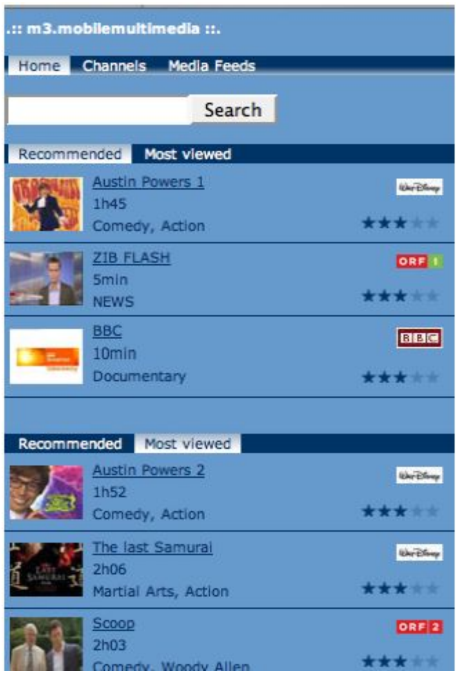


Figure 4.3: The Prototype of the Home Page.

for understanding the interaction issues concerning the use of mobile phone interfaces [65].

4.5 Conclusion

XP is a lightweight process that puts very little administrative overhead on the developers. Therefore, extending XP with additional practices is much easier than extending other, more restrictive, methodologies. The integration of usability engineering methods works especially well because of the many overlapping principles (e.g. iterative development, end-user incorporation) of XP and UCD.

The user interface design process of UCD is highly beneficial, as it provides feedback which is used for the system's functional requirements [42]. The assessment of each feature from the users' perspectives influences the whole development process of the application and addresses the problems which arise when the system requirements are gathered only by discussions with stakeholders [60].

When deciding about usability issues in our project, we try to involve not only the development team and the product manager, but also the usability engineers and all project stakeholders, especially end-users. This practice led to an application that, from the beginning, was lacking many of the teething troubles common to technician-dominated development teams and can be seen as a big success factor for our project.

Chapter 5

Agile User-Centered Design Applied to a Mobile Multimedia Streaming Application

Mobile computing is leading a revolution. Multimedia consumption on mobile devices is increasing day by day. The most important factor for the success of such applications is user acceptance. Additionally, the success of a software development project is associated not only with tools and technologies, but also depends on how much the development process is user-centered and developer-oriented. We are working on a project to develop a multimedia streaming application for mobile phones. This chapter describes our adopted development process: the integration of Extreme Programming – one of the popular agile methods – with User-Centered Design. Furthermore, it is shown how the integrated process facilitates user-orientation and at the same time preserves the social values of the development team. This chapter also presents a summary of a recently carried out usability study.

5.1 Introduction

The most important factor for the success of a software application is user acceptance. An inherently usable and technically elegant application cannot be considered a success if it does not satisfy the end-users' needs. End-users are often left out of the development process [83]. Agile development processes involve a customer as a business representative who is responsible to specify the business value of user requirements, but this customer needs not necessarily to be a real end-user.

Agile methods are becoming popular nowadays. Being a lightweight agile method, Extreme Programming (XP) has the advantages of: on-time delivery, co-located team, relying on the team members' knowledge rather than documentation, optimized resource investments, short release cycles, working high quality software, tight customer integration, incremental design, constant communication and coordination, rapid feedback, continuous refactoring, pair programming, and test driven development [12][15][2]. XP is a collection of well-known software engineering practices. XP aims at enabling successful software development despite vague or constantly changing software requirements. The novelty of XP is based on the way the individual practices are collected and lined up to function with each other [2]. It is also a people-oriented process with many social core practices.

Usability measures the quality of a user's experience when interacting with a product or system. User-Centered Design (UCD) is an approach for employing usability [118]. UCD, also called human-centered design, is an approach to user interface design which is based on information about the people who will use the product. UCD processes focus on users throughout planning, design, and development of a product [120]. Holzinger emphasizes that every software practitioner should be aware of different usability methods and apply them according to specific situation of a project [47].

There already exist approaches of integrating agile methodologies and Usability Engineering (UE) / UCD [48][37][81][29][85]. Memmel et al. point out that when UE becomes part of agile software engineering, it helps to reduce the risk of running into wrong design decisions by asking real end users about their needs and activities [85].

The focus of both methodologies, XP and UCD, on users makes it possible to integrate them [42]. The integrated process allows to combine benefits of both methodologies and makes it possible to reduce the shortcomings of each. XP needs to know its true end-users and UCD benefits from a flexible and adaptive development methodology which runs throughout the project life-cycle [115]. We integrate XP and UCD in our project, where we are developing an application that enables a user to perform content-based search for audio and video content and play the streamed content on a mobile phone [54]. The end-users are indirectly involved in the process by our use of different Human-Computer Interaction (HCI) instruments such as user studies, personas, usability expert evaluations, usability tests, and automated usability evaluations [122]. Usability of a mobile application is an important ongoing research issue. Numerous studies address UE / UCD issues for mobile applications [19][49][66]. We conduct various usability studies and in this chapter a summary of one of the studies is presented.

Section 5.2 outlines the similarities between XP and UCD. Section 5.3 examines the project environment. Section 5.5 describes the adopted process. Section 5.6 provides the details of a usability study. Section 5.7 concludes the chapter.

5.2 Similarities between XP and UCD

The core values of XP [15] and UCD [44] are applied to solve different issues. In XP, a simple implementation fulfilling the minimum requirements of the application is created and iteratively extended, while UCD tries to continuously improve the usability of the user interface. However, when comparing some of the core values it seems obvious that the two development processes can benefit from each other's practices.

5.2.1 End-User Involvement

One of the core practices of XP is to have a *Customer on Site* who is co-located with the programmers in order to answer domain-specific questions and give feedback on the system. This practice can be matched well with the testing of prototypes with actual users as proposed by UCD. Especially, if the customer is also the real end-user or if developers have direct access to end-users.

5.2.2 Continuous Testing

Continuous and extensive testing is at the heart of XP. It is mainly embodied by two practices: *Continuous Integration* runs all existing automated tests whenever the code base is changed or extended in order to check if the changes caused any undesired side effects. Most of these tests emerge from *Test-Driven Development.* First, automated tests checking the desired behavior are created. Then, the actual behavior is implemented and can be evaluated right away with the tests. This is usually done only for pure behavioral code, but can be extended to user interfaces. Tests can check the expected behavior of an interface, and these tests can be run whenever the code is changed.

The end-user tests of UCD are a valuable source for test targets. An unexpected user action that caused a problem in the application can be replicated as an automated test. By executing this test in the *Continuous Integration* process it is ensured that the problem, after solving it once, does not reappear.

5.2.3 Iterative Development

Both, XP and UCD, propagate an iterative procedure of design and development [44][12][120]. An XP project yields *Small Releases* (another core XP practice) on a regular and frequent basis (usually a few months). Each release version is based on the previous one, incorporating new features and fixing bugs of the predecessor. Inside a release time frame, work is organized in "iterations" (usually taking one to four weeks). On an even smaller scope, many feedback-and-change cycles take place, especially in conjunction with *Test-Driven Development* and *Refactoring* (the practice of changing source code in order to improve its quality without changing its functionality).

UCD also proposes a design–test–modify circle for developing user interfaces. The scope of iterative development in XP and UCD differs. Releases and iterations in XP are mainly organizational units and *Refactoring* is considered to be a development tool. In contrast to this, UCD's iterative user interface refinement is a more explicit process as its involvement of external persons (the test users) makes it more complex. Nonetheless, iterative interface development of UCD fits well into the iteration principle of XP, because both approaches are aware of the value (and necessity) of evolutionary development.

5.3 Project and Team Setup

We are working in a project where we develop an application that enables a user to perform content-based search for audio and video content and play it on a mobile phone. The project started in summer 2007 and will end in 2010. The application enables a user to search not only in the metadata but also in the spoken words of the AV clips. This content includes radio and TV archive material, such as documentaries or other recordings of historical, political and cultural importance, discussion programs, movies, music videos, audio books, etc. The application is being designed keeping in mind the social interaction of users and provides some Web 2.0 features [52].

In addition, one goal of the project is the analysis of agile software development methods, particularly XP, and to devise a usability test procedure for mass applications on mobile devices with emphasis on UCD and iterative user-interface design.

The team consists of six full-time regular members having different social and cultural backgrounds, five developers and a product manager who plays the role of the *On-Site Customer* of XP.

The customer communicates with the project partners who come from various domains, including user interface design, usability research, telecommunication, content providing, and software-hardware infrastructure. Also, developers communicate directly with the engineers of a partner usability research center regarding usability issues. The usability engineers working for our project are also active in UCD research with the team.

5.4 Application Features

The user interface of the current prototype comprises the following main features. The application's main screen provides the features: "Search", "Top rated", and "Most recent" clips. It provides links to the "Channel" and "Media Feeds" pages, as well as a link to the "Clip Detail" page when one clicks on the title of a clip. The application also implements a few other Web 2.0 features like "Recommended" and "Most viewed" clips.

Search:

"Search" allows to search the whole AV content by entering keywords. It displays the search results ordered by broadcast date (if any). For each result item, the clip's title, link, description, duration, originating channel (if any), and a representative thumbnail image are shown. The user can play a clip by clicking on the respective link.

Channel:

"Channel " allows to browse the schedule of TV and radio channels. A channel lists the original program schedule of the current day, but users can browse the schedule of previous days or can search within the channels. Users can select the date and the time (either Morning, Afternoon, Prime time, or Night), where the system then displays the list of clips in the selected time period, ordered by broadcast time. The resulting items are shown in a similar format as in the "Simple Search" result list with rating stars and a channel icon for each clip in the schedule. The top of the page also provides a dropdown list for selecting channels.

Media Feeds:

The "Media Feeds" feature is intended to provide the users a facility to create and consume a constantly updated stream of clips based on the users' search criteria.

This media feed can be sent to a friend by SMS or email.

Clip Detail:

The system shows the "Clip Detail" page when users click on the title of the clip. Users can rate a clip, add a comment, or view all comments. With the "Tell a friend" feature, users can send a clip to their friend by SMS or by email.

5.5 Agile Usability Process

The following subsections describe the process which is followed in application development.

5.5.1 Approach to User-Centered Design

User interface design plays an important role in the acceptance of a web based application. The overall process of our approach to UCD is based on evaluating the usability of the application in small iterative steps. This helps us to gain insights into the functional and cognitive requirements of real users. We design prototypes of the user interface of the system and test them throughout the development process. As a result the fidelity of the prototypes increases and evolves.

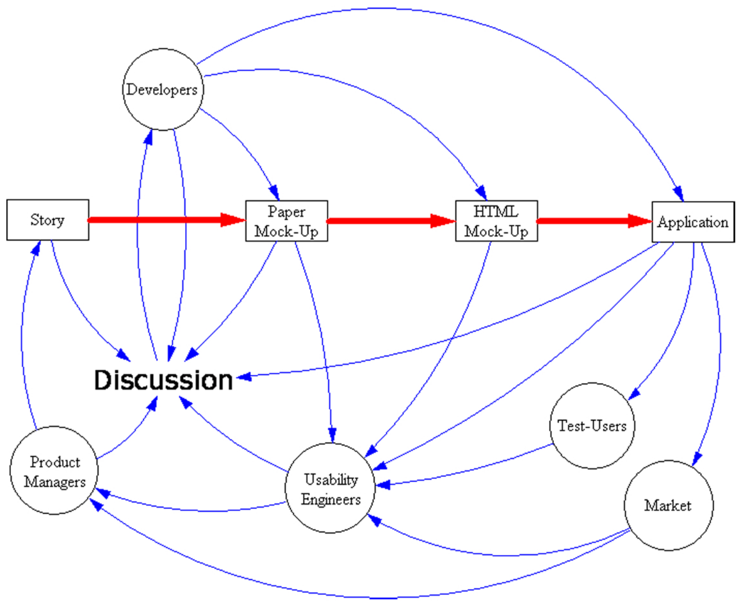


Figure 5.1: Iterative User Interface Design Workflow [52].

The work flow presented in Figure 5.1 illustrates the iterative design approach incorporating UCD into our XP process. From a broad perspective, the application development cycle starts with defining user stories, then comes to mock-up designing, and at the end to the actual implementation. The process is executed as follows:

- Different feature-related user stories of the application are created by the customer along with partners.
- Developers create different paper mock-ups for each of the required features to collect ideas and to present them to the customer.
- The customer decides which of the mock-ups best suits his needs or suggests modifications to the mock-ups.
- A final mock-up is derived according to customer's wishes, which then serves as the basis for the actual implementation.
- Once the implementation mock-up of a feature or a group of related features is finished, the usability engineers are asked to give feedback on it.
- After incorporating the feedback given by the usability engineers into the application, end-user tests are conducted by the usability engineering team.
- The feedback on the application from the usability engineers, as well as from the test-users, is taken as input for further refinements of the user interface design of the application.
- The results are then incorporated into automated tests which serve as an executable specification for the actual implementation.

This feedback-and-change cycle provides insights into whether the user-interface design is meeting different usability criteria. As the application development is done in short iterations, the developers are able to refactor the system continuously according to the feedback derived from the parallel, as well as iterative, UI design process. Hence, the system evolves according to the needs of the end user and the specifications derived from actual usage.

5.5.2 Choosing the Type of Mock-Up

We make use of two different types of mock-ups; low fidelity paper mock-ups and high fidelity implementation mock-ups. The benefit of using paper mock-ups for the interaction design is that they can be designed and modified quickly. For simple interaction designs, a low fidelity paper mock-up suffices as a basis for further discussions and the implementation. An additional advantage is that it is easier to criticize simple and rough mock-ups compared to ones, which look neat and perfect from the graphic design perspective [102]. But for some features a high fidelity mock-up is required to clearly visualize the interface. As we have the benefit of an on-site customer co-located with the development team all the time, for those tasks a quick implementation mock-up is designed and immediately presented to the customer. This implementation mock-up is then modified based on the immediate feedback of the customer. If our customer would not have been co-located with us all the time, it would have been difficult to have the benefit of this quick feedback-and-change cycle.

5.5.3 Frequency of End-User Tests

The end-user tests are performed on an on-demand basis, that is, when the customer says that now is the appropriate time, from the business point of view, to run a usability test with test-users. Also, when there is enough amount of new functionality added to the application, it becomes effective to perform usability tests and then proceed with development. It would have been good if user tests could have been made on regular basis, e.g., at the end of each release, but considering the expenses and resources required for, it we have kept it only on an on-demand basis. So, the expensive part of involving real users is done more effectively.

5.5.4 Integration of HCI Instruments

Figure 5.2 describes our model of integrating HCI instruments (user studies, personas, extended unit tests, usability tests, and usability expert evaluations) into the XP process [122]. It shows the interplay of the HCI instruments with the XP process. When applied correctly in various phases of the project, the instruments are designed to reach the goal of improved software quality not only in terms of technical quality, but also in terms of usability. End-users are integrated in two different ways. On the one hand, user studies are taken into account to develop personas [64]. The personas specify the direction of development by guiding the customer in identifying user stories and are extended at the end of the iteration when the vision about the user has broadened. This serves as an indirect end-user input for the development process. On the other hand, feedback from usability tests performed by test-users as part of the usability evaluations serves as a direct input for further enhancement and development of the application [122].

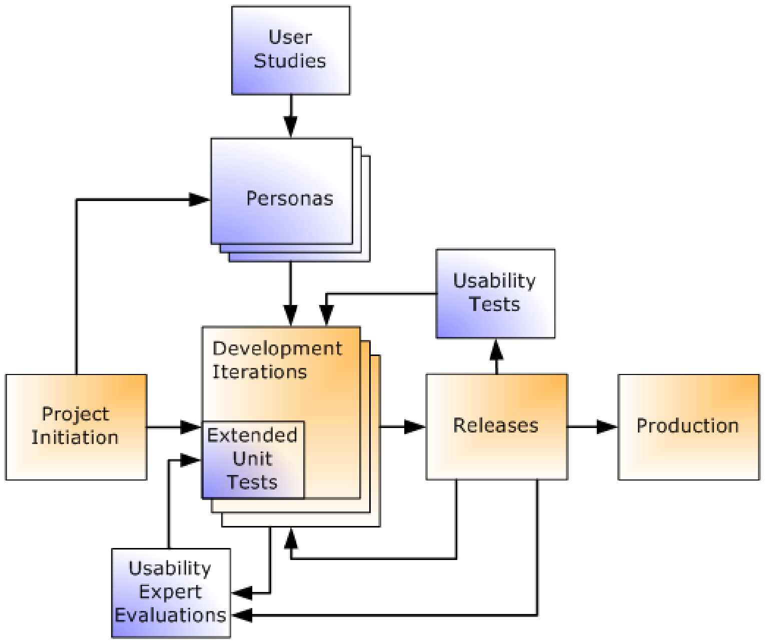


Figure 5.2: The Integration of HCI Instruments into XP [122].

5.5.5 Testing Issues

A big issue in mobile user-interface design practice is that current approaches are not sufficient for mobile phones [108]. For designing any software, use of UCD practices ensures that the product is accepted by the users [65]. This further supports the use of the UCD approach for user interface design. To enhance it further, we provide high fidelity implementation prototypes to our usability engineers for user testing. Paper prototypes are good and sufficient for verifying non mobile-based product requirements. But in case of applications for mobile phones, they are not sufficient for finding and solving usability issues related to detailed interaction on the small device with its limited user input capabilities [65]. Therefore, in our case the application is tested on mobile phones and not on any web based simulator in order to understand issues concerning the use of mobile phone interface [65].

5.5.6 Communication and Collaboration

Communication between stakeholders is an important characteristic of software development. Communication and collaboration between customers, business partners, developers, and other stakeholders enhance the overall team efficiency [83]. The value of communication is expressed by the XP practices of pair programming, metaphor, informative workspace, simple design, on-site customer, the planning game, and coding standards [46]. Other factors in communication are the use of whiteboards, positioning and sharing of desk facilities to ease pair programming, stand–up meetings, developers buying-in to the concepts of the rules and practices of XP, and collective code ownership [40]. We sit side by side in a spacious room having enough space for private workplaces, as well as for three separate pairing stations. This seating arrangement has promoted effective interaction in the team and has helped in resolving technical issues on the spot [70]. The teams' XP room is equipped with six whiteboards which are used to record the XP stories agreed at release and iteration planning meetings. Story cards are physically stuck to the whiteboards in prioritized order with adjacent notes written on the board. Various graphs showing architecture and velocity of the project are also drawn on the whiteboards. By looking at the whiteboards, anyone can see the current status of the project.

Email, phone calls, and video conferencing are the tools used in routine communication with the usability engineers and other partners. Personal visits to and by project partners are also made by the product manager and by other team members whenever necessary.

5.5.7 The Planning Game

We hold two types of planning meetings: release based meetings and iteration based meetings. A release lasts for three months, where within a release, an iteration lasts for two weeks. Project partners attend release meetings where through discussions user requirements are identified and defined in the form of so-called XP user stories [50]. The parallel with the UCD approach is visible in the understanding and appreciation of the users and their requirements [83]. The user stories are written down on story cards and are prioritized by the project partners. Developers then estimate the time required for implementing the stories.

At the beginning of each iteration, an iteration meeting is held which is attended only by the team members including the product manager. The product manager selects and prioritizes stories which fit in the current iteration depending on the available velocity. Then, developers break down the stories into detailed tasks and estimate them. Finally, the product manager defines the acceptance criteria for each story and task.

Before and after implementing the user stories, continuous feedback is obtained from the usability engineers. Then, these stories are modified according to the feedback of end-users and the usability engineers. Once again, this is a common step with UCD approaches; an understanding of the user goal and the tasks to achieve that goal. Addressing a requirement in terms of the user and their goals focuses development upon what is needed [83].

5.5.8 Pair Programming

This practice has helped us in spreading and sharing the project-specific knowledge and improving the technical skills of the developers. We also applied the practice of working in pairs with the product manager [50]. The product manager pairs with a developer when writing customer-acceptance tests, thus exposing the customer to the process and the internal status of the application, which helps in better understanding and implementing the end-users' requirements. This also has enhanced the enthusiasm of the team members to work in a collective and collaborative team environment.

5.5.9 On-Site Customer

In UCD, all activities are focused on providing business value through ensuring a useful, usable and engaging product. The customer is not only defined as the project stakeholder, but the end user as well [83]. The Manifesto for Agile Software Development [16] does not clearly demand end-users as customers. In our process, the product manager plays the role of an "on-site customer" and communicates with the various stakeholders. The end-users are indirectly involved by the usability engineers.

5.6 Usability Study

Usability tests are carried out to evaluate the running prototype. One of the usability studies was executed in January 2008 with 10 respondents using a mobile phone. The classical task-based usability test method was used [99]. Each respondent was asked to execute 5 different tasks. Tasks were carried out on a Nokia N95 mobile phone. To gather general feedback and general opinions, two interviews were carried out: One before and one after the task session (pre- and post-interview). Each task was accompanied by task specific post-questionnaires. Interview sessions lasted about 1 hour. For the tests the device's standard browser, as well as opera-mini 3.0, were used (the first is incorporating a web-like mouse pointer, the latter a link marker to navigate through the interface).

After the test, respondents had to judge three different visual design paper-prototypes. We used the AttrakDiff questionnaire [45] to capture the attitudes of the users towards the application in terms of graphical design, enjoyment, and aesthetics. The AttrakDiff questionnaire was filled-out after the task.

The following two subsections outline the results of the test in the form of improvement suggestions.

5.6.1 Improvements of Layout and Design

Main improvements should be made concerning the visual appearance of the site:

- The actual site, menu, and navigation layout is not ideal. Through the use of the color blue as text color and background color at the same time, equal text sizes throughout the interface and different alignments, the site's hierarchy is not visible for users.
- The current layout does not incorporate visually attractive design elements and is rated as pragmatic and monotone with a lack of stimulating elements (Attrakdiff questionnaire).

Figure 5.3 shows the prototype of the home page presented in the usability study.

5.6.2 Improvements of the Usability of the Prototype

On the "Channel" web page, a web-like calendar function to select dates should be integrated (the current function will not be usable for greater amounts of data). All navigation menu elements should be separated from content menu elements ("Home" vs. "Watch"). Furthermore, interactive elements ("Rate", "Comment" etc.) should be placed on a separate page and not on the bottom of a description page. Figure 5.4 shows the recommended prototype of the "Channel" page showing the calender. Figure 5.5 shows the menu entries without any visual separation.

For the further development of the prototype the sub-site "Media Feeds" should be separated into two categories introducing the sites "create Media Feed" and "watch Media Feed". Special attention should be given to feedback mechanisms, which at the moment do not support the user (feedback of search queries, display of media feed search results).

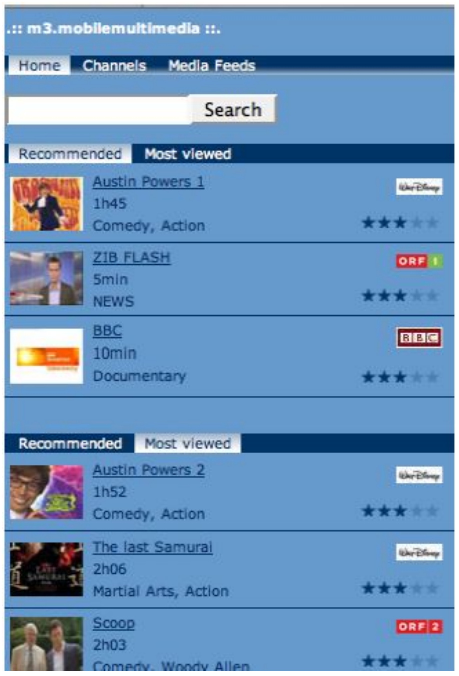


Figure 5.3: The Prototype of the Home Page [54].

From the mock-ups of three different designs, the AttrakDiff results suggest that a yellow design was most liked by the respondents. It was also suggested that the blue design may be used in the future, but the following improvements should be made:

- Accentuate contrast on whole site.
- Avoid light blue text on darker blue backgrounds.
- Introduce visually attractive design elements that increase the attractiveness of the site.
- Eliminate monotony by introducing more colors.

Two of the developers also observed the usability study session which gave them a chance to realize the impressions of actual end-users and their feelings. This helped in guiding the development according to the wishes of end-users.

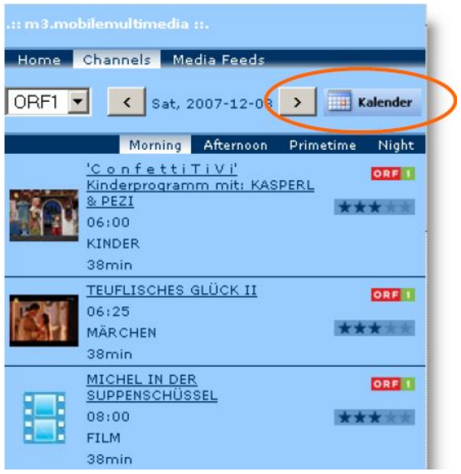
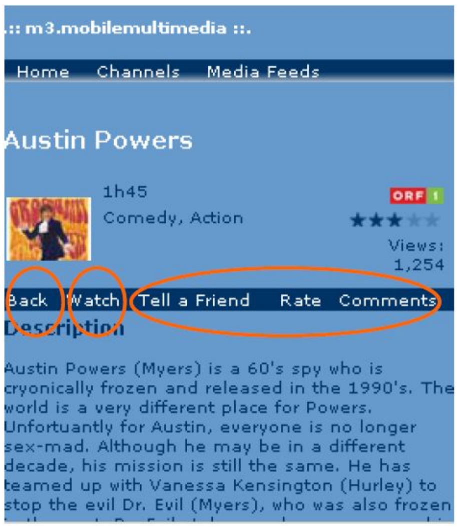


Figure 5.4: The Prototype of the Channel Page showing the Calender.



Austin Powers (Myers) is a 60's spy who is cryonically frozen and released in the 1990's. The world is a very different place for Powers. Unfortuantly for Austin, everyone is no longer sex-mad. Although he may be in a different decade, his mission is still the same. He has teamed up with Vanessa Kensington (Hurley) to stop the evil Dr. Evil (Myers), who was also frozen

Figure 5.5: The Menu Entries without any visual Separation.

5.6.3 A Task Example

In this subsection, a task example is presented. The task is: "Find the detailed description of a given movie, write a comment and rate it".

Facts on Task:

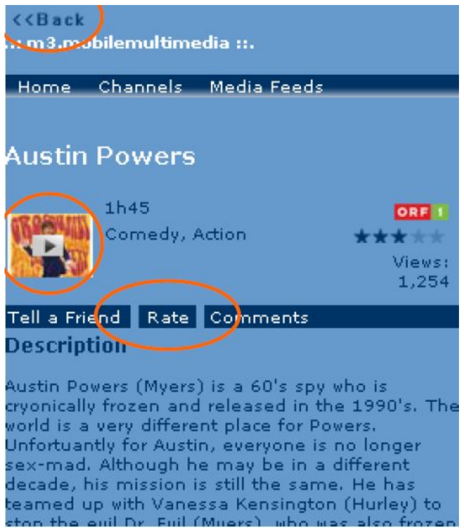
The task was completed without any greater difficulties by all respondents. On the "Home" page and on the "Channel" page respondents used the heading to find the detailed description and the video's thumbnail to watch the video. Respondents did not encounter much problems on the "Clip Detail" page. The prominent position of the links "Comments", "Rate" and "Tell a friend" – Figure 5.5 – on top of the description page helped respondents to understand which possibilities are offered. On the "Clip Detail" page there are two interaction paradigms that were both understood: Clicking on the link "Tell a fried" opens a new page. This did not cause any problems for users. The functions "Rate" and "Comment" are placed at the bottom of a "Clip Detail" page and users had to scroll down or use a link to jump down. In reference to both described paradigms, user comments indicate that the longer the list of comments is, the more uncomfortable the site is to browse. Further, the task uncovered that on the mobile interface respondents did not recognize that they were scrolling down the page when using the anchor-links "Comment" and "Rate". To get back to the top of the site they pushed the "back" button. This did confuse some of the respondents as they jumped back to "Home" although their intention was to get to the top of the "Clip Detail" page. Of course, this depends on how the browser implements the "back" functionality.

A solution that incorporates interactive functions ("Rate", "Comments", "Tell a Friend") on a separate page is recommended.

Suggested improvements resulting fro this concrete tasks:

- Back Button: A dedicated back button should be integrated on top of the page. This is the place, where basic navigation elements are expected.
- Watch Button: A watch button should be designed and integrated consistently. An additional watch button – if necessary – should be placed on a particular spot on the site and not be integrated in the navigation menu. The watch button should be visually highlighted.
- Tell a friend, Rate and Comments: These elements describe interactive functions on the site and therefore should be kept together and aligned to the left side of the page.

Figure 5.6 shows the recommended menu layout and arrangement.



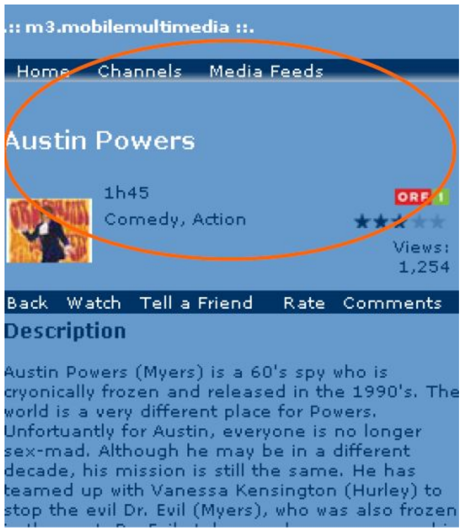
Austin Powers (Myers) is a 60's spy who is cryonically frozen and released in the 1990's. The world is a very different place for Powers. Unfortuantly for Austin, everyone is no longer sex-mad. Although he may be in a different decade, his mission is still the same. He has teamed up with Vanessa Kensington (Hurley) to stop the evil Dr. Evil (Myers), who was also frozen

Figure 5.6: Improvements of Menu Layout and Arrangement.

Respondents' Feedback/Comments:

- All respondents indicated that in their opinion the "Clip Detail" page provides a good overview.
- The design of the "Comments" and "Rating" section is good and intuitive. Too many comments on one page should be avoided as the page would get too long (1 respondent).
- Comments should be ordered in chronological sequence, beginning with the most recent entry (1 respondent).
- It should be possible to select which information is sent to another person via the "Tell a friend" function (the video's description, the video itself, etc.). Radio buttons should be used to specify one out of different possibilities (1 respondent).
- The space on top of the "Clip Detail" page (heading) should be used in a better way. This would provide more space for description texts (1 respondent).

Figure 5.7 shows the space on top of the "Clip Detail" page which should be used more efficiently.



Austin Powers (Myers) is a 60's spy who is cryonically frozen and released in the 1990's. The world is a very different place for Powers. Unfortuantly for Austin, everyone is no longer sex-mad. Although he may be in a different decade, his mission is still the same. He has teamed up with Vanessa Kensington (Hurley) to stop the evil Dr. Evil (Myers), who was also frozen

Figure 5.7: Use the Space on top of the Clip Detail Page more efficiently.

5.7 Conclusion

Agility is an invitation to adapt, to mold, and to reshape the software development methodology according to the requirements of a project. Being a lightweight agile process, it is easy to extend the XP process with additional practices. Our XP process fits well into the UCD approach, because of the many overlapping principles (iterative development, end-user incorporation, testing) of both methodologies. The usability engineers in the project are well integrated into the development team. It would have been even better if one of the usability engineers had been present physically with the team all the time, as face-to-face communication is more helpful in quickly resolving design issues. We could not conduct usability tests frequently with end-users due to time and budget constraints, but mitigated it with usability expert evaluations. The whole development process is influenced when each feature of the application is assessed from users' perspective. This addresses the problems which arise when the system requirements are gathered only by discussions with stakeholders [60]. The involvement of all stakeholders, particularly end-users, in the process, can increase the chance of the success of a project.

We continuously try to optimize our process as long as the project lasts and will provide further insights whether the process has been able to enhance the usability of the application. In October 2008, we will conduct a contextual mobile multimedia content usability study which will give insights into mobile HCI concerning the coherence of content types, consumption times, and consumption contexts. Integrating of end-users indirectly in the form of HCI instruments, co-location, communication, and planning meetings has not only contributed to improving our process, but also has led to increasing the overall morale of the team.

Chapter 6

Concept and Design of a Contextual Mobile Multimedia Content Usability Study

The popularity of consuming multimedia content on mobile phones is increasing more and more, not only because of the availability of the technical infrastructure, but also because of the mobility in modern society. We are developing a mobile multimedia streaming application. The crucial factor for such applications in order to be adopted and successful is user acceptance. This chapter presents the preliminary concept and design of a contextual mobile multimedia content usability study. The study is conducted within a research project on agile software development methodologies with special emphasis on Extreme Programming and continuous usability evaluation. Past work included satisfaction of the needs of end users by means of focusing on user-experience in all steps of the development process. To gain scientific relevant data, the careful design of a study is considered most important. The study which will be conducted in October 2008 will give insights into mobile Human-Computer Interaction concerning the coherence of content types, consumption times, and consumption contexts.

6.1 Introduction

We are working on a research project where we are developing an application that enables a user to perform content-based search for audio and video content and play it via streaming on a mobile phone. The basic research goal is to examine agile software development methodologies, in particular Extreme Programming, with special emphasis on User-Centered Design. This is obtained by two means. On the one hand, we have established a development process where the quality focus is not only placed on technical excellence, but also on delivering a usability-tested high-quality end-product. On the other hand, we have created a testbed for effective and efficient mobile usability testing automating certain parts of the usability testing procedures.

Within this research project, a lot of topics already have been covered. Our adopted development process, the integration of Extreme Programming with User-Centered Design, examined in [54], facilitates user-orientation and at the same time preserves the social values of the development team. The techniques of enhancing Extreme Programming by integrating Human-Computer Interaction (HCI) instruments (user studies, personas, extended unit tests, usability tests, and usability expert evaluations) are treated in [122]. An iterative and user-centered approach to user interface design, where usability is evaluated in small iterative steps to gain insight into whether the users' functional and cognitive requirements are met, is published in [52]. Also, the results of a usability study applying a classical task-based usability test method [99], executed in January 2008, was made public in [53]. The outcome of a study on the relation of basic human values to behavior patterns of the usage and production of mobile multimedia content, conducted with a technique referring to the means-end theory, can be found in [72].

For us, the next logical step in the realm of mobile HCI is the evaluation of the coherence of consumption behavior of different content type categories and the consumption behavior in different contexts. This chapter presents the preliminary study setup of a usability evaluation study to be conducted in October 2008. The goal of the study is to examine mass-market capability of a mobile multimedia streaming application. More precisely, the aim is to analyze adoption and consumption behavior of such applications by means of a field trial, where diary studies and contextual interviews will be used. The resulting data is intended to give insights into what needs to be in place from the content type setup perspective in order to make such applications successful.

This chapter presents an overview about related work. The following section briefly outlines the application being developed. Then, the selection of the respondents is described. The next section examines the study setup concerning the media content, the diary study, and the contextual interview. Finally, a conclusion is given.

6.2 Related Work

This section provides an overview of related work in the field of usability evaluation of contextual mobile multimedia content. Basically, existing work in this area can be divided into the categories contextual studies, studies on mobile video consumption, and studies on technical issues in mobile multimedia consumption.

In [1] the importance of context in interactive mobile applications is stated, and definitions and categories of context, in order to create a framework for the development of context-aware applications, are presented. The work in [110] aims at understanding different mobile contexts and provides design implications for mobile and context-aware human-computer interaction. The necessity of considering the mobile user's context in conjunction with the user's cultural context is shown in [22].

In the realm of mobile video consumption [95] presents a study identifying the social motivations and values of people when using mobile video technologies. Also, [100] focuses on human behavior, human needs, and interaction design concerning the creation, management, and consumption of moving images using mobile devices.

In [62] the effects of codecs and combinations of audio and video streams with low bitrates and different contents on the perceived video quality of mobile devices are described. A methodology to evaluate the perceived quality of mobile video with variable physical quality is introduced in [79]. Interestingly, the outcome of the studies conducted with this methodology was that when watching high motion videos users prefer high-resolution images to high frame rate.

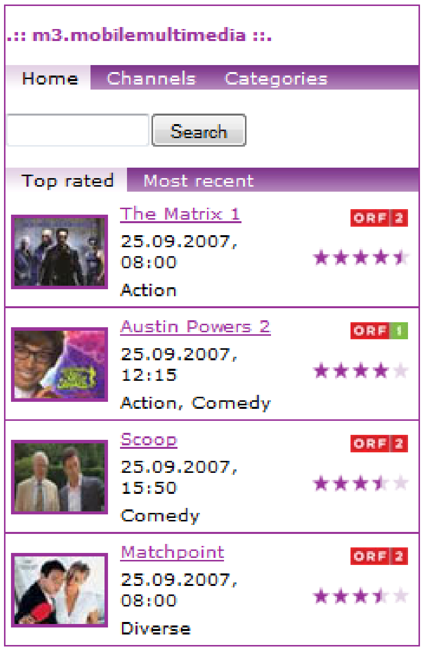
However, to the best of our knowledge, none of the existing work examines a contextual mobile multimedia study aiming at relating user context, consumption behavior, and content type categories.

6.3 Application

The application being developed enables a user to perform content-based search for audio and video content and play it on a mobile phone via streaming. The user is able to search not only in the meta-data, but also in the spoken words of the audio and video clips. The content includes radio and TV archive material, such as documentaries or other recordings of historical, political and cultural importance, discussion programs, movies, music videos, audio books, etc. The content setup for the study is especially tailored and described in Section 6.5.1. Moreover, the application is being designed keeping in mind the social interaction of users and provides several Web 2.0 features [52].

6.3.1 Features

The user interface of the application comprises several main features. The main screen (see Figure 6.1) provides the features “Search”, “Top rated”, and “Most recent” clips. Links to the “Channels” and “Categories” pages are presented. When one clicks on the title of a clip the “Clip Detail ” page is shown. The application also implements other Web 2.0 features like “Rate” and “Comment” clips respectively “Others also watched” and “Tell a Friend” the link to a clip [53].



The Matrix 1
25.09.2007, 08:00
Action

Austin Powers 2
25.09.2007, 12:15
Action, Comedy

Scoop
25.09.2007, 15:50
Comedy

Matchpoint
25.09.2007, 08:00
Diverse

Figure 6.1: Home Page.

Search

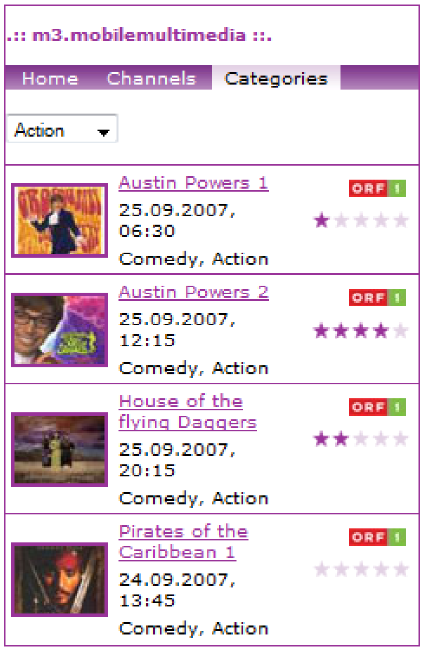
“Search” allows to search the whole content by entering keywords (see Figure 6.1). The returned search results are ordered by broadcast date (if any). For each result item, the clip’s title, link, rating, duration, originating channel (if any), content type, and a representative thumbnail image is shown. The user can play a clip by clicking on the image respectively go the “Clip Detail” page when clicking on the title.

Channels

“Channels” allows to browse the schedule of TV and radio channels. One channel lists the program schedule of a particular day. Different channels can be selected by using a drop-down box, listing all available channels, at the top of the page. The user can browse to previous or future days and is able to search within a channel. The search result items are presented in the same format as the search result items from the “Search” feature of the main screen.

Categories

The “Categories” page (see Figure 6.2) displays the whole content filtered by content type. By means of a drop-down box, containing all content type categories, the user can select to have only clips of a specific content type presented.



Austin Powers 1
25.09.2007, 06:30
Comedy, Action

Austin Powers 2
25.09.2007, 12:15
Comedy, Action

House of the flying Daggers
25.09.2007, 20:15
Comedy, Action

Pirates of the Caribbean 1
24.09.2007, 13:45
Comedy, Action

Figure 6.2: Categories Page.

Clip Detail

The system responds with the “Clip Detail” page when a user clicks on the title of a clip. On this page the user is given the possibility to “Rate” a clip, add a “Comment”, or view comments. In addition to that, the “Others also watched” feature displays clips which have been watched by users with similar interests. The “Tell a friend” functionality enables a user to send the link to a clip to a friend by SMS or by email.

6.4 Selection of Respondents

For the study 16 respondents in the age group between 18 and 35 are chosen. From the chosen respondents the percentage of men and woman is balanced. Ideally, the respondents are interested in the following topics:

- They are interested in politics, economics, and other classical news themes. Respondents have to indicate to watch news on television on a regular basis (at least three times a week).
- The respondents are interested in technical content, especially in the field of computer science and information technologies. They are interested in products and news from Internet and telecommunication branches.
- They are interested in television series from the entertainment sector. Respondents have to indicate to watch pre-evening series on television on a regular basis (at least three times a week).
- The respondents are interested in music and music television. They are interested in particular bands in the pop and rock scene.

All respondents need to have at least some experience in using mobile multimedia, no matter if it is listening to music on an mp3-player or using the photo-gallery of their mobile.

6.5 Study Setup

The following section describes the study setup. First, the choice of the media content is examined. Then, the setup and execution of the individual methods of the study is outlined.

Fundamentally, the study is composed of two methods which are not depending on each other and therefore may be executed independently:

- Diary Study.
- Contextual Interview.

Both methods will be executed with the same respondents. Although there are no dependencies between the two, ideally the diary study is executed shortly before the contextual interview. This allows to examine user-experience issues when they are still in the minds of the respondents. As a prerequisite, each respondent fills out a questionnaire covering basic and demographic data as well as data concerning current mobile multimedia consumption.

6.5.1 Media Content

In order to be able to gather relevant data, the basic parameter for the study is the choice of specific content type categories. Four types of video content, each representing different information and entertainment levels, are chosen. Moreover, in the four media content types, the importance of the audio component (spoken text, audio) and the video component (visual information, graphics and pictures) is different. On the one hand, the video sequences for the diary study have a length between 15 and 20 minutes. On the other hand, the video sequences for the contextual interview have a shorter duration of approximately 10 minutes for the purpose of minimizing the overall interview time.

1. Music Video Content: When watching a music video, the audio as well as the video information is important. In practice, the audio content can exist without the additional video content. Consequently, the user does not have to pay primarily attention to the screen.

2. News Content: News content is a mixture of audio and video material. In addition to that, text is an important factor of news content as well. Nevertheless, video material strongly supports the given information. Often, news content is separated in different themes and topics.

3. Documentaries/Scientific and Technical Content: This type of content is comparable to news content, hence both, visual and auditive, information is important. In contrast to news content, scientific content forces the viewer to keep up with the video, since all information in the video sequence is important.

4. Television Series/Entertainment Content: Series have a continuous plot over the whole video sequence. In comparison to scientific and news content, the visual and auditive information is equally important to keep up with the story. But, even if missing seconds of both information, one is still able to follow the story.

6.5.2 Diary Study

In the diary study, each respondent is given a mobile device with which she or he is told to use the application for one week. The web interface, as well as the content presented to the respondents, is especially tailored to the diary study. The setup fulfills the following characteristics:

- The web interface for the diary study has limited possibilities in comparison to the actual application developed.
- The content type categories are limited to four different types (see Section 6.5.1).
- In each content type category approximately 40 to 50 clips are presented (except the news category, where there are less but only current clips).

For the sake of gathering relevant data for the diary study, two different groups of respondents are created:

- One group is told that they should watch at least X videos per day.
- The other group is told that they should watch less than X videos per day. This group is created in order to provide more realistic data during the one week study.

All respondents are told to choose time and place of consumption, as well as content types, on their own. After each video session, the respondents fill out a questionnaire and take a photo of the current context with the mobile device.

For the purpose of recording activities of the respondents, different log files (web server logs and streaming server logs) of the application are used. The user tracking of the application, respectively the activities recorded in the log files, give information about:

- When are videos watched.
- How long is each video watched.
- How many and which videos are watched during one video session.

With the overall setup of the diary study it is possible to get the following results:

- The average length of video sessions.

- Information on the content selection according to the day time and context.
- Overall information on the content selection (mix up of content types during sessions).
- Information on the context (when and where videos are watched and correlations to content types).

6.5.3 Contextual Interview

For the contextual interview, each respondent is given a mobile device. She or he is supposed to use the application in four controlled video sessions, each in a different context. The web interface presented is tailored as described in Section 6.5.2. In each video session, the respondent watches four different videos from four different content type categories. The content setup for the contextual interview is described in Section 6.5.1. In the contextual interview, sixteen clips are shown to each respondent on the whole.

During one contextual interview (one video session in a specific context) the respondent is accompanied but not interrupted. The procedure is as follows:

- The interview starts with a pre-questionnaire covering basic and demographic data as well as data concerning mobile multimedia consumption.
- The respondent is explained the device.
- The respondent watches four videos from different content type categories.
- After each video, the respondent gives qualitative feedback to the interviewer and fills out post-video questionnaires (see Section 6.5.3).

After each contextual interview the respondent gives qualitative feedback to the interviewer and fills out post-session questionnaires (see Section 6.5.3).

For the purpose of getting significant results from the individual contextual interviews, different locations are necessary. The interviews are conducted in two indoor locations and two outdoor locations. The locations themselves, as well the order of the locations the interviews are conducted in, are as follows:

1. Indoor Location: Usability Lab.
2. Indoor Location: Cafe.

3. Outdoor Location: Tube and Bus: Respondents watch a video traveling a predefined distance in the public transport network. Respondents will have to change three times the means of transport having to cross train and bus stations and moving between people.

4. Outdoor Location: Public Spaces: Respondents watch a video while standing on and walking across a public space.

After all four sessions the respondent fills out an overall questionnaire.

Post-Video Questionnaires

The post-video questionnaires the respondent fills out after each video cover three different types of feedback:

- Emotion after Watching the Video: To capture the post-video-watching emotion the "SAM - Self Assessment Manikin" will be used [23].
- User Experience and Sensation of Usability: Similar to relevant concepts out of the Appeal Measurement questionnaire by Hoonhout [10].
- Qualitative Feedback concerning the Video.

Post-Session Questionnaires

The post-session questionnaires the respondent fills out after each contextual interview cover three different types of feedback:

- Context specific Feedback.
- User Experience and Sensation of Usability: Similar to relevant concepts out of the Appeal Measurement questionnaire by Hoonhout [10].
- Qualitative Feedback concerning the Session.

6.6 Expected Results

The basic concept of the study is that we use a method-mix of diary studies and contextual interviews. Therefore, from each individual method different results are expected. We also expect that the use of these two methods will give insight into the two approaches, hence backing up and assisting each other.

The expected findings from the diary study are:

- Coherence between daytime of consumption and content type.
- Coherence between context of consumption and content type.
- Average number of watched clips per session.
- Average time how long a clip is watched.

From the contextual interview, the following results are expected:

- Availability of different contexts: Ability to define "Contextas" like "Personas" [64] for mobile multimedia consumption.
- Availability of qualitative user feedback on content types in different contexts.
- Availability of Manikin [23] ratings according to content type and context.
- Availability of user experience data concerning context variables like light, noise, people, and others.

All findings, from the diary study, as well as from the contextual interview, are correlated to sex and age of the respondents.

6.7 Conclusion

The current trend in mobile multimedia consumption is more than obvious: it is becoming more and more popular. For this reason, we are developing a mobile multimedia streaming application with special focus on agile software development methodologies and usability. The most critical factor for this kind of application will be user acceptance. While developing this application we took all efforts to satisfy the needs of the end user. We combined Extreme Programming with User-Centered Design, integrated HCI instruments in our development process, developed an iterative UI development process, conducted usability studies and did studies on the relation of basic human values to behavior patterns of the usage and production of mobile multimedia content.

With the study whose setup is described in this chapter and which will be conducted in October 2008, we expect new insights in the field of mobile user-experience. In order to gain scientific relevant data, we consider the careful design of the study as being crucial. The data to be gathered promises the ability to draw conclusions on coherence of content types, consumption times, and consumption contexts. Furthermore, deriving different contexts, as well as the availability of qualitative and experience data, is an important goal. Future work includes publishing the results of this study.

Chapter 7

Continuous Usability Evaluation

Usability is largely accepted as a core success factor of software projects. Still, usability evaluation techniques are not as widely adopted as, for example, the evaluation of functional correctness. While modern development processes advocate a constant verification of functionality, usability often is only monitored sporadically, if at all. As a result, usability issues often are identified too late in the development life cycle and thus cannot be corrected comprehensively.

In projects employing agile development methodologies, Continuous Integration (CI) systems are often used to automate the steps necessary to ensure functional correctness. Automated tests which assert functional and implementation requirements are created for all relevant parts of the code base. CI systems automatically build and test the application whenever the project's code base is changed, and immediately notify the development team if the build or any tests failed.

In this chapter, concept and prototypical implementation of a fully automated Web usability evaluation tool are presented, which can be easily integrated into CI systems. With the presented approach, usability aspects can be evaluated concurrently during development, allowing for early detection of usability problems.

7.1 Introduction

Usability is a core success factor of software development. This is especially true for Web applications which—in contrast to many desktop applications—often do not have a homogenous target user group but are rather intended to be used by as many people as possible. The users of Web applications can differ vastly in their knowledge of computer interaction idioms and often are not willing to spend time learning how to use a new user interface, so a smooth usage experience is crucial in order to increase user acceptance.

The importance of usability for Web applications is widely accepted in the scientific community, as can be deduced from the constantly growing number of published usability methods and tools. A study by Ivory and Hearst [59] describes and classifies 58 Web usability evaluation methods; a literature review by Insfran and Fernandez [58] lists 51 recent publications concerned with such methods. However, the acceptance and adoption of usability evaluation in practical software development lags behind other development aspects, especially functional correctness. While functional testing on various levels (from unit tests to system tests) has virtually become a mandatory practice in software development, explicit usability evaluation is far less common, especially in small projects.

This imbalance partially results from the different degrees of development process integration of the two aspects. Functional correctness can in large part be evaluated by the development team, and automated testing has become an integral part of development (especially in agile development methods). Furthermore, test conduction can be fully automated, for example by means of a CI environment: once the test system is set up, both test execution and result reporting are performed automatically; developers are only required to react on reported test failures.

In contrast, usability evaluation (for both Web and desktop applications) is more complicated to integrate in the project workflow due to the following reasons:

- The members of the software development team usually lack the required expertise to conduct usability evaluations. Therefore, it is a common practice to involve external usability experts for this function. Communication between developers and usability experts, who usually have a human factors-related background, can be problematic due to differences in domain knowledge and profession-specific vocabulary [104].

- Some evaluation methods are resource-intensive in terms of time and/or money [9] and thus cannot be conducted on demand (e.g., when required for design decisions).

- Some methods are based on observing the behavior of "real" users ("usability testing", see Section 1.2.2) using the application, requiring the application to be deployed and running. These methods are reasonable only at advanced stages of the project life cycle, when the application is mature and comprehensive enough to be presented to users. However, at this point in development, the application design is also harder to change than at the early stages, since a lot of usability problems have manifested deeply in the application's architecture; therefore, fundamental usability problems often can only be corrected superficially. This issue is aggravated when using a sequential development method (i.e., Waterfall Model-oriented methods) which produces a runnable—and thus testable—program only at the end of the project life cycle.

- Some methods require additional effort which can complicate the development workflow. For example, certain methods (see Section 7.2) require the creation of models depicting the interaction structure, or to extend the sources with specific code used by the evaluation.
- Some methods only work with specific development methods or application frameworks; thus the decision for such a method has to be made at the project start since a later integration is not feasible. For example, some methods (see Section 7.2) build on top of model-driven development (MDD) methodologies, and thus can only be applied when using MDD for project development.

Part of the available usability evaluation methods falls in the category of Automated Usability Evaluation (AUE). AUE tools are software programs which are intended to aid in the evaluation of usability aspects. The concrete nature of automation in the respective tools is manifold, ranging from support in conducting questionnaires to automatic checking of usability guidelines. Some AUE tools are intended to assist usability experts in their work, while others implement expert knowledge and thus allow non-experts to conduct usability evaluations. While AUE tools of the latter kind cannot replace user tests and expert evaluations, they can be considered a valuable complement to these measures, and a starting point for implementing further usability practices.

AUE tools can alleviate the integration of usability evaluations into the development workflow. To be accepted by project managers and developers, an AUE tool is required to cause as little cost as possible, and subsequently should minimize the additionally introduced workload (including both the initial setup and the recurring execution overhead). Furthermore, broad adoption of an AUE tool is also determined by the degree of technical independence. Tools which are bound to a specific software framework, programming language, or development process, are unlikely to be adopted due to the restricted scope of applicability.

This chapter presents the concept and the prototypical implementation of an AUE tool for evaluating the usability of Web applications. The presented tool concept fulfills the above stated requirements of effort minimization and technological independence:

- The tool processes the deployed Web application (e.g., hosted on a test server) and thus is largely independent of the technology used to implement and host the application[1].
- The tool automatically crawls the whole application, thereby building a graph representation of the application's navigation structure as perceived by the user. All visited pages as well as the navigation model are analyzed regarding several usability metrics and properties. These steps do not require any effort of developers besides initial configuration of the analysis; thus, evaluation is cheap in terms of consumed developer time.
- The tool can be integrated into Web projects at any stage of development, only requiring the application to be runnable on a Web server. In iterative development methods, features are added by and by to the application. This allows to continuously evaluate the usability, from start to end of the project.
- The tool can be integrated into the automation script of a CI system; thus the usability evaluation can be executed whenever the application is changed. The notification mechanism of the CI system is used to notify developers about found usability violations. Furthermore, evaluation results from different builds can be compared in order to track development of usability compliance and application complexity.

The aim was to design a tool which is simple enough to be integrated and used by developers without specific usability knowledge, yet powerful and comprehensive enough to support usability experts. The latter is achieved by providing a presentation and analysis module which can be used to manually "drill down" into the usability data. Also, the raw data (i.e., graph structure and page data) of each executed evaluation is stored, allowing to track the development of usability aspects over time, as well as to compare different user interface designs.

The tool promotes an incremental integration into the project workflow: initially, it can be used as-is, without forcing any additional workload (e.g., configuration) on the project team; when the acceptance of developers rises, the tool allows for additional, more in-depth analysis which cannot be easily automated. This approach is well-suited for (but not limited to) agile development methods where development is fast-paced due to short iterations and almost immediate feedback (many agile meth-

[1]While the prototype is limited to plain HTML pages, the approach can be adapted for more complex—and modern—Web technologies, like AJAX (see Section 7.5.3).

ods require a representative of the "customer" to be co-located with the development team in order to answer project-specific questions and to give feedback).

Similar approaches employing the same technical key elements (i.e., crawling, graph representation, graph analysis) have been discussed in scientific literature (most significantly, [114] and [20]). The main difference to the tool presented in this chapter is the emphasis put on *when* and by *whom* usability is evaluated: existing approaches are meant to be used as standalone tools, mostly operated by a usability expert. The presented approach, on the other hand, proposes to automatically evaluate usability aspects on a regular basis.

The remainder of this chapter is structured as follows: Section 2 summarizes related work in the fields of AUE and process integration of usability engineering; Section 3 describes a system for automated usability evaluation of Web applications and its integration into CI environments; Section 4 explains some usability problems which are related to (or derivable from) the navigation graph of the application, and which are used as examples in the following section; Section 5 presents the current state of the prototypical implementation of the presented tool concept; finally, Section 6 summarizes the work presented in this chapter, draws conclusions and gives an outlook on future work.

7.2 Related Work

This section summarizes existing work (both scientific and practical) related to the work presented in this chapter. Two fields are examined: automated usability evaluation techniques, and integration of usability engineering practices into the software development workflow.

7.2.1 Automated Usability Evaluation

The term "Automated Usability Evaluation" is not used consistently, neither in the scientific literature nor in the vocabulary of practitioners. The degree of human involvement in the evaluation process varies significantly between different approaches labeled AUE, ranging from tools which perform a static guideline-driven analysis of a user interface without involving any human user, to tools recording and analyzing usage sessions of real users, to tools optimizing the workflow of usability experts, to integration tests (run against deployed applications) asserting usability aspects.

In the following, the focus is put on model-checking and guideline-checking techniques for Web application, both of which form the base of the work presented in this chapter. In general, these techniques only require configuration effort but no human actions during the actual analysis process, which is the main requirement for fully automated usability analysis.

Model-Checking

Various types of usability evaluations without involvement of real users have been discussed in the scientific literature. Similar to the technical approach presented in this chapter, some described AUE tools perform their evaluation by crawling a deployed Web application. Tonella et al. [113] describe a tool which reverse-engineers a Web application and constructs a graph meta-model, depicting pages as vertices and links as edges, and additionally models the server-side state of the application. The graph data is used to analyze graph characteristics (e.g., average link path lengths) and to generate paths through the application for manually conducted tests. In a later publication [114], the same authors discuss the applicability of this approach for modern Web technologies. Similarly, Alfaro et al. [31] describe MCWEB, a "model-checking tool for Web site debugging", which uses a graph to calculate various usability-related metrics.

Benedikt et al. [20] describe VeriWeb, a tool which systematically explores the paths through a Web application. Each encountered page is validated against a configurable set of rules and guidelines. Additionally, graph-related properties like the length of traversed paths can be checked.

Other tools use development artifacts (i.e., models describing the application structure) to perform usability evaluations. The use of models created in Model-Driven Development approaches for usability evaluation has been discussed by Fernandez et al. [36], highlighting the benefits of early conduction of usability evaluation in the project life cycle. Evaluated usability aspects include graph properties (e.g., breadth and depth of navigation paths) as well as UI properties (e.g., color contrast). Other approaches (Feuerstack et al. [38] and Atterer [6]) are also based on model-oriented development methods but require to extend the models with additional, usability-specific metadata.

Ahmad et al. [3] describe the prediction of a Web application's "navigational burden"—that is, the amount of "navigational efforts" (e.g., mouse clicks to follow Web links) required to complete intended tasks—based on the application model graph.

Thimbleby [112] highlights a range of usability properties and metrics that can be calculated from graph-like models. The models are represented as finite state machines where transitions describe user interactions, changing the internal and external device state. In the described approach, these models not only serve as a usability tool but rather form the central documentation and design tool for the entire development process.

Beer et al. [18] describe IDATG (Integrating Design and Automated Test Case Generation), a tool for generating integration test cases and test paths; the model has to be explicitly constructed using the IDATG application.

Miao et al. [89] describe a Web-centric approach to derive test paths from a Web application relation graph.

Guideline-Checking

Besides model analysis, another important aspect of AUE is guideline-checking. Beyond scientific literature, a lot of—mostly Web-hosted—guideline-checking tools are available, the majority of which performs guideline checks on single pages or whole Web sites. Guideline-checking tools perform a static analysis of HTML documents, extracting various properties and validating them against well-established guidelines. Existing tools cover many areas of different relevance for usability. Verifications range from checking purely technical aspects, to accessibility concerns, to page layout and visual aesthetics (e.g., whitespace between elements, color contrast, count of used fonts).

The World Wide Web Consortium (W3C) [123] provides a set of freely usable validators for checking Web pages for well-formed HTML, XML, and CSS source code. Furthermore, the W3C's Web Accessibility Initiative (WAI) [124] offers a comprehensive list of accessibility checkers. An example of such an accessibility validation service is AChecker [7] which checks single pages for conformance with a variety of accessibility guidelines (BITV 1.0, Section 508, Stanca Act, WCAG 1.0 and 2.0). The NIST Web Metrics tool suite [90] offers a range of guideline-based usability checks for Web pages. Au et al. [8] describe a guideline-checking-based approach for evaluating usability of software on mobile devices. Vanderdonckt et al. [119] propose a separation of guidelines evaluation logic and the evaluation engine: first, rules are defined in terms of the Guideline Definition Language (GDL); then, a Web page is parsed and checked against the GDL rules by the evaluation engine. This approach allows to independently develop rules and engine, and enables flexible configuration of the evaluated guidelines.

A different approach for extracting usability-relevant metrics from Web pages is presented by Tuch et al. [116]: the file size of JPEG-compressed screenshots can be used as a simple, yet significant measure for the visual complexity of Web pages.

7.2.2 Development Process Integration

The integration of usability methods into software development processes, especially into agile development methods, has been discussed extensively in recent years. However, most existing work in this field focuses on the process integration of manually conducted usability methods; this is reflected by a recent survey by Hussain et al. [57], showing that less than 8% of the 92 software professionals participating in the survey use automated usability tools.

Hussain et al. [56] discuss the integration of usability experts into the process workflow of XP. Meszaros and Aston [87] describe the integration of UI paper prototyping into agile projects. Patton [97] describes incorporating Interaction Design methods in an agile development process. Propp et al. [98] propose the use of software models to optimize the conduction of user tests performed by usability experts.

Other approaches are more comprehensive, comprising extended process frameworks. Memmel et al. [84] describe CRUISER, an “agile cross-discipline user interface and software engineering lifecycle”. Chamberlain et al. [26] describe a framework for integrating agile development and user-centred design.

7.3 Automated Usability Evaluation in Continuous Integration Environments

In recent years, iterative software development approaches have gained more and more attention. These development methodologies (e.g., the family of agile development [27] processes like Extreme Programming (XP) [13] or Scrum [17]; the Rational Unified Process [69]) have emerged as a reaction to the problems and drawbacks of Waterfall Model-oriented software development. A major issue of the Waterfall Model is its inflexibility to cope with changing project requirements, which are an inherent aspect of most non-trivial business-oriented software projects. Iterative methodologies work with short-term detail plans instead of comprehensive project plans, so the cost of adapting a plan to changed requirements is substantially lowered.

To cope with changing requirements and their possible side-effects on seemingly unrelated project aspects, constant verification of the project’s integrity and validity is mandatory. XP, amongst other iterative methods, advocates automation for continuously testing structural and functional validity. The technical foundation for this automation is *Continuous Integration* (CI), which describes both a work process and a technical framework. CI is described in detail in Section 1.1.3.

With the CI infrastructure in place, AUE can be integrated into the development process seamlessly, on the same level as automated functional testing. The requirement on any integrated AUE method is that the execution—except initial setup and configuration—must not require any manual action or intervention from developers; all three main activities of an evaluation—capture, analysis, critique [59]—must be fully automated:

- Capture: The data for the subsequent usability analysis must be gathered automatically, without the involvement of real users or developers.
- Analysis: Usability-relevant properties and metrics must be deduced from the collected data automatically in order to identify usability defects.
- Critique: A report summarizing all relevant findings must be produced, possibly including suggestions for improvements or solutions.

In this section, the concept for a tool fulfilling the above requirements is presented, and the integration into a CI environment is discussed. Then, the current development state of the prototypical implementation is presented. The prototype serves as a proof-of-concept and covers the tool concept to a large extend.

The design for the presented AUE tool[2] was led by the following principles:

- Web UI technology constantly changes, so AUE tools for Web applications have to be adapted to new technologies constantly. Therefore, no such tool can be considered to be "finished" if it claims general applicability.
- An AUE tool can be useful even if it does not provide full coverage (e.g., analyze all pages in a Web application; find all usability problems of a certain kind). Quality and overhead have to be balanced: while exhaustive usability tests are likely to yield results of higher accuracy and significance, they also cause higher costs in terms of time and money. A fully automated AUE tool, on the other hand, causes little to no overhead, and can establish a ground level of usability quality.

7.3.1 Capture

The *capture* activity of the AUE tool is implemented in terms of a Web crawler, thus requiring the evaluated application to be deployed and running. The crawler systematically visits all pages of the application and builds a model of the application's navigational structure. This approach ensures that the resulting graph model depicts

[2]In this section, the term "tool" refers to the tool concept, not its prototypical implementation.

the navigational structure as perceived by a human user. The model is implemented in terms of a directed graph; vertices represent pages, edges represent links. In this context, a "page" is considered the application's entire response to a URL request as rendered by a browser, ignoring any implementation details transparent to users; a "link" is an HTML anchor element. Figure 7.1 shows a simple graph comprising three pages and five links. In the example graph, all pages contain links to all other pages, only the link from C to B is missing.

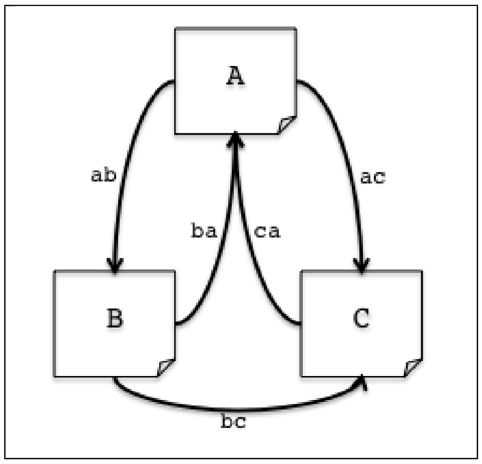


Figure 7.1: Simple page graph.

The crawler attaches metadata to the graph vertices and edges: each vertex stores the page's URL and title, the connection information (connection success, response code, content type), the full HTML source code, and a screenshot image; each edge stores the link's label (text label or image URL), the target URL (which can differ from the target page's URL if the target URL redirected to another URL[3]) and the link's position and size on the page.

The crawler implementation used in the prototype requires to be configured with:

- Start URL: The URL where crawling should start; typically the home page.
- URL filter: A filter describing which link URLs should be followed; allows separate filter rules for protocol, host, port, path, and query parts of URLs.[4] For example, filtering can be used to limit the crawler to application-internal

[3]Web applications can use URL redirection to forward the browser (and thus the user) to a different URL due to various reasons (e.g., moved pages, short aliases for long URLs).

[4]The de-facto standard mechanism for controlling crawler-like applications, robots.txt (Standard for Robots Exclusion, `http://www.robotstxt.org`), is not used due to its lack of flexibility.

links, and to block certain parts of the application. Also, it is used to exclude access to non-HTML content (via file extension filtering) which might be costly to provide (e.g., dynamically generated images) and thus might slow down the crawling process.

- Maximum crawl depth: The maximum allowed number of crawl iterations. Since Web applications can dynamically generate content and thus might provide an infinite virtual URL space, the maximum crawl depth is used to limit the crawler's scope.

Using this configuration data, the crawler implements the following algorithm (in the following, the terms "page" and "link" denote meta-objects describing the actual entities):

1. The start URL is parsed and the resulting page is added to the graph as a vertex and to the $sourcePages$ list.

2. The target URLs of all links of all pages in $sourcePages$ are collected and stored in $targetUrls$.

3. Duplicate and already visited URLs are removed from $targetUrls$; URLs not matching the crawl filter criteria are removed from $targetUrls$ and stored in $blockedUrls$.

4. All remaining URLs in $targetUrls$ are parsed, and the resulting pages are stored in the $newPages$ list and added to the graph as vertices. Multiple parse tasks are performed in parallel in order to speed up the process.

5. For each URL in $blockedUrls$, a dummy vertex (without any attached metadata except the URL) is added to the graph.

6. For each page p_s in $sourcePages$, the corresponding vertex v_s is looked up in the graph. For each link of p_s, the vertex v_t corresponding to the link's target URL is looked up in the graph; then, an edge from v_s to v_t is added to the graph.

7. If $newPages$ is empty or the maximum crawl depth has been reached, the algorithm stops; else, $sourcePages$ is set to $newPages$, then $newPages$, $targetUrls$ and $blockedUrls$ are cleared, and the algorithm continues with step (2).

The parser component used in this algorithm takes a URL as input and builds a meta-object describing the page as rendered in a browser. The parser performs the following steps:

1. The parser opens an HTTP connection to the URL.
2. If the connection cannot be established, the parser stops and yields an "error" page object.
3. If the connection indicates non-HTML content (via the `Content-Type` property of the HTTP header), the parser stops and yields a "content" page object.
4. The HTML page is loaded; if a URL redirection is encountered, the steps (1) to (3) are repeated for the redirect URL.
5. The page title and link information is extracted; a screenshot image is created; then, the parser yields a page object built from this data.

The link URLs retrieved from parsed pages are normalized [71]. This helps to identify semantically identical but syntactically different URLs, ensuring the integrity of the created graph structure: if equivalent URLs are treated as different, the same page will be parsed multiple times, and multiple unrelated nodes will be added to the graph. In the following, some of the applied normalization steps are listed:

- Relative URLs are made absolute (e.g., the URL `content/home.html` in expanded to `http://foo.com/content/home.html`).
- The default HTTP port (80) is removed if present in the URL.
- URLs pointing to directories are ensured to end with a slash.
- Default directory index files (e.g. `index.html`) are removed.

7.3.2 Analysis

Based on the automatically gathered data, the *analysis* activity is performed on two different levels: usability evaluation of single pages, and of the overall application structure. For both levels, there exists a host of techniques and tools which can be retrofitted in order to be used by a fully automated usability evaluator. A detailed discussion of the possible evaluation methods is out of the scope of this chapter. Where appropriate, existing work is referenced.

Page evaluation is based on analyzing the DOM tree and the rendered representation of a singe page. Possible evaluations include (but are not limited to):

- Accessibility guideline checks [124].

- Layout guideline checks, e.g., sufficient whitespace between form elements; appropriate color contrast, horizontal scrolling detection (Fighting Layout Bugs library [109]).
- Visual complexity measures, e.g., comparing the file size of JPEG-compressed screenshots [116]; semantic or phonetic similarity of neighboring link labels; percentage of the "clickable" area of a page.
- Error detection, i.e., checking for the presence of error messages by matching the page's text content against regular expression patterns [20].
- Validating well-formedness of HTML and CSS source code [123].

Structure evaluation uses the generated graph model of the application's navigational structure in order to calculate graph properties and metrics, as well as to evaluate the application-wide consistency of the UI. For example, graph analysis can be used to evaluate the following aspects:

- Overall complexity, e.g., page and link count; average and maximum number of links per page.
- Navigational burden, e.g., average and worst-case click path lengths [3] [112].
- Semantic consistency, e.g., a link label should always be used to link to the same page; a page should always be linked with the same label (see Section 7.4).
- Structural consistency, e.g., every page should link to the Home page; a page should not contain multiple links to another page; a page should not link to itself (see Section 7.4).
- Error recovery, i.e., after clicking a link by mistake, what are the average and worst-case costs (in terms of click path length) of returning to the source page [112].
- Reachability constraints, e.g., the application should not contain "dead ends" (pages which do not provide any links to the application).

Some of the listed evaluations require specific configuration in order to be performed. For example, in-page error detection requires a list of regular expressions which indicate errors; evaluation of consistent Home page linking requires to know the Home page URL. Some quantitative properties require threshold values in order to be interpreted and evaluated, e.g., the longest tolerable length of click paths.

For some evaluations, rule exceptions have to be defined. For example, the consistent usage of link labels can in general be evaluated automatically. However, in some contexts the re-use of labels is meaningful, hence certain labels (or labels on certain pages) should not be flagged: e.g., in a page sequence implementing some application task, each page might be connected to the following page via a link labeled "Next Page".

Other evaluations do not require any specific configuration due to their "objective" nature: e.g., HTML and CSS are defined standards, hence standard adherence can be evaluated without further configuration; the same applies for accessibility guidelines. Also, some graph properties (e.g., "dead ends" in the navigation structure) can be calculated without configuration.

7.3.3 Critique

Based on the automatically generated analysis data, the *critique* activity can be provided by means of a report listing all found defects. Each defect item consists of three elements:

- The defect type provides a name and a general description of the defect.
- The defect reason specifies why a defect was identified (omitted if this information is obvious from the defect type).
- The defect context represents the application entities which are affected by the defect.

The context information is intended to locate the defect as specifically as possible in order to help solving the problem. Depending on the applied evaluation, the context's scope can be one of the following:

- Page element: a specific HTML element on a specific page; the outcome of evaluations which statically analyze the HTML code. For example, the context of an accessibility defect due to a missing `alt` tag for an image is the affected `img` element, including the respective page and location in the HTML source code.
- Link: similar to page element, but with added information about the target page; usually the outcome of structural analysis. For example, if the link label

foo is used to link to different pages, the defect's reason is "Inconsistent usage of label foo", and the context is a list of all links with that label.

- Page region: a graphical region (position and size in pixels) on a specific page; the outcome of evaluations which are implemented via visual or DOM analysis of the rendered page. For example, a color contrast defect's context is the area enclosing the affected text.
- Entire page: identifies defects affecting whole pages. For example, a page contains horizontal scrolling.
- Entire application: identifies defects affecting the whole application. For example, the average link count exceeds a threshold value.

The defect list can be arranged by different views: by defect instance, by affected page, or by defect type.

- Defect instance view lists all defects as separate items.
- Page-wise view lists all pages containing any defects. Subsequently, for each page a list of defects found on the page or involving the page is shown. Each item comprises a textual description and the context (i.e., the affected entities) of the defect.
- View by defect type lists all identified defect types. For each defect type, a list of affected entities is presented. Here, entities represent the "cause" of the defect.

Using the collected visual data (i.e., page screenshot and element position), defects can also be presented graphically by showing the screenshot with an overly indicating and describing the defect position. This approach has been demonstrated by the Fighting Layout Bugs library which automatically highlights affected page regions (e.g., texts overlapping borders) on a screenshot image.

The gathered usability data can also be inspected manually in order to investigate usability aspects which cannot be analyzed automatically. For this purpose, an interactive reporting and analysis tool is required. The following list gives an overview of some evaluations possible with this kind of data:

- Structure inspection: the application structure can be queried to calculate custom properties (e.g., minimum path length between two specific pages).

- Graph comparison: graphs produced at different points in time in the project life cycle can be compared (e.g., with regard to added/removed pages and links).
- Simulation: a graph can be modified virtually (e.g., by adding links not present in the actual application); then, the resulting experimental graph can be analyzed and compared to the actual graph.
- Project progress: if every generated graph is preserved, the progress of the developed application can be tracked along the whole development process.

7.3.4 Integration in the Automated Build Process (CI)

An AUE tool as described in the Sections 7.3.1 to 7.3.3 can be used as a standalone tool similar to existing AUE software. However, a bigger benefit can be achieved by integrating the tool into a CI system:

- Usability evaluation is not triggered explicitly by developers, but implicitly by the build script, and hence cannot be "forgotten" by developers.
- CI "knows" when the application was changed, so the usability effects of any performed change can be evaluated immediately.
- The project team is periodically notified about the state of the developed application's usability.
- Serious usability defects can break the build (and thereby force immediate fixing).
- The tool creates periodic "snapshots" of the application's status, thus tracking the project progess. This is not only useful for usability concerns but also for project management.
- The acceptance of usability as a core software quality can be increased in the project team. Also, developers gain usability expertise.

Different project contexts value different usability aspects, so the effect of found defects on the build has to be specified individually. Constructs which are identified as errors might be accepted or even intended in some contexts. In order to avoid a potentially large number of false-positive usability error notifications from the build system, an error filter is required which can assign an "effect" to each found error:

- Omit: The defect is logged, but developers are not notified about it.
- Notify: Developers are notified about the defect (usability warning).
- Break build: Serious defects can break the build, causing an alert notification for the development team (usability error).

The concrete implementation of the integration depends on the used build automation framework. Figure 7.2 shows a diagram depicting the steps which are run during a CI execution, extended by a step running an AUE analysis. The AUE step is performed after deploying the application because the AUE approach described in this chapter depends on a deployed and running application.

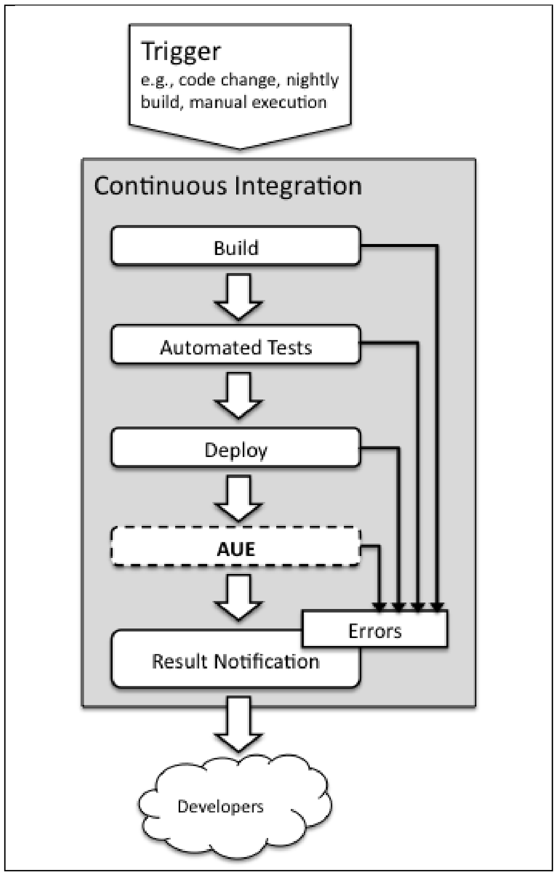


Figure 7.2: The steps performed during Continuous Integration, extended by an AUE step.

7.4 Graph-Related Usability Problems

In this section, some usability problems which can be found by analyzing the page graph of a Web application are described.

7.4.1 Self link

A page should not contain links to itself. Such links add unnecessary complexity to a site, which increases the cognitive load on the users.

Figure 7.3 shows a graph containing two pages, A and B. The link labeled x links from A to A and is a "self link"; this link should be avoided. The link labeled y also links from A to A, but targets an in-page anchor in A (`A#foo`) and thus is not considered a "self link". The link labeled z is a normal link between two different pages.

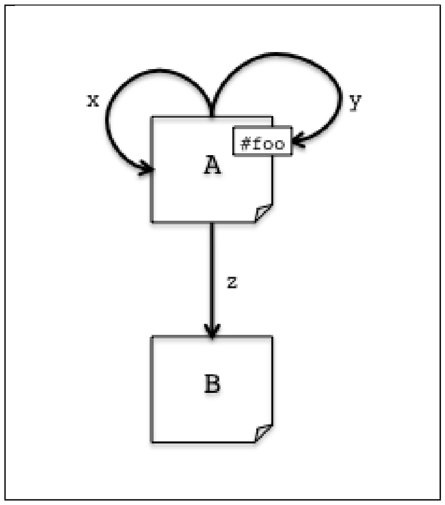


Figure 7.3: Self link.

7.4.2 Inconsistently Used Link Labels

A link label should always be used to link to the same page. If a link label is used to link to different pages, this increases the navigational complexity for the user. There are, however, exceptions to this rule. A common exception are links used to access lists of similar items, e.g., "Play" links next to each song in a playlist; "Next" links on each page of a multi-step input sequence.

Figure 7.4 shows a graph containing four pages, A, B, C and D. Both links are labeled x; however, one link targets page B, the other targets page D. This is generally considered inconsistent; the label x should only be used to target one distinct page.

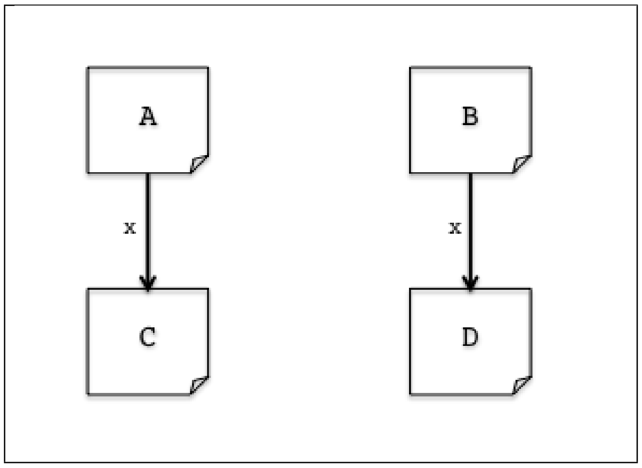


Figure 7.4: Inconsistently used link labels.

7.4.3 Inconsistently Labeled Link Target URLs

All links to a specific target URL should have the same label. If a target URL is linked with different labels, this increases the navigational complexity for the user. Each in-page anchor in a page represents a different link target URL.

Figure 7.5 shows a graph containing three pages, A, B and C. Page C is targeted by three links, each bearing a different label. The links labeled x and y are considered inconsistent, as they both target the same page but use different labels. The link z, however, targets an in-page anchor in C (`C#foo`), which is not considered inconsistent.

7.5 Prototype Implementation

A proof-of-concept of the tool concept presented in this chapter has been developed to large extents. All relevant aspects (crawler, page and structure evaluations, reporting front-end, build integration) have been implemented. The prototype was named "Fawoo" (for "Fully Automated Web Usability").

From the user's point of view, the application model is structured as follows: a Fawoo deployment allows to create and store "projects". When crawling a Web application, the results (i.e., graph model data, analysis data, and crawl metadata) are stored as a "crawl job" inside a project. A Fawoo project typically reflects a

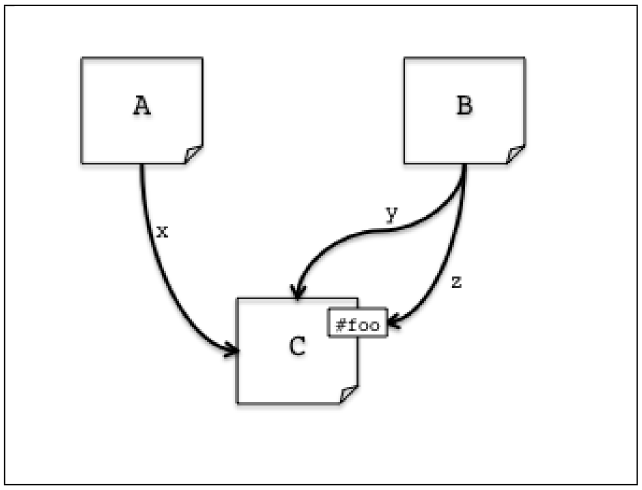


Figure 7.5: Inconsistently labeled link targets.

development project and thus contains crawl jobs for one specific Web application. The different crawl jobs inside a project represent snapshots of the Web application, taken at different times during development. Crawl jobs can be viewed and analyzed separately or compared with each other.

In the remainder of this section, some implementation details of Fawoo will be presented. This is followed by a walkthrough of the most important functions of Fawoo. Then, limitations of the prototypical implementation are discussed.

7.5.1 Implementation Notes

Fawoo is implemented as a Web application using the Java programming language[5], an is hosted by a Tomcat[6] server. For the user interface, the WingS library [7] is used.

The crawler implementation uses the Selenium/WebDriver library [105]. WebDriver allows to programmatically load and render Web pages, read Document Object Model (DOM) information, and take screenshots. This “virtual browser” approach allows to extract usability data not available by merely parsing HTML code. For example, HTML element positions and sizes cannot be deduced from the HTML markup; this information is only accessible via the page’s DOM, which in turn is a result of rendering the page. Furthermore, WebDriver provides different native rendering engines (currently Internet Explorer, Firefox, and Chrome). This allows

[5]`http://www.oracle.com/technetwork/java/index.html`
[6]`http://tomcat.apache.org/`
[7]`http://www.wingsframework.org/`

to perform page rendering in different browsers, helping to detect browser-specific layout bugs.

Fawoo is intended to be used by the development team to analyze a Web application hosted on an internal test server, so download bandwidth was not considered when implementing the prototype. While "traditional" crawlers try to limit the workload they put on the crawled servers (e.g., by pausing after each downloaded page), the Fawoo crawler tries to maximize the data throughput (by downloading and parsing multiple pages in parallel) in order to minimize crawl duration. Thus, crawling a public Web server with the Fawoo crawler could be considered a denial-of-service attack and should be avoided unless the server operator is notified and allows the experiment.

7.5.2 Walkthrough

The views (or "input masks") available in Fawoo can be grouped into four categories:

- Project views are used to manage and display projects.
- Job views are used to manage jobs (e.g., to create a new job by crawling).
- Graph views are used to analyze the page graph structure of a crawled Web application.
- Problem views are used to view usability problems identified in the analyzed Web application.

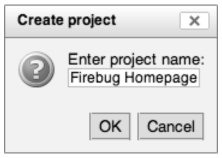
In the following, sample screenshots of all relevant views are presented. On some screenshots displaying data tables, columns unimportant for the displayed context have been hidden in order to reduce the screenshot size.

The screenshot sequence forms a scenario: a project is created, a page is crawled, the page graph structure is explored, and the found usability problems are examined. Throughout this section, the Web site of Firebug is used as a continuous example. Firebug[8] is a popular browser plug-in for analyzing technical aspects of Web pages from within the browser, which often is used to support usability evaluations.

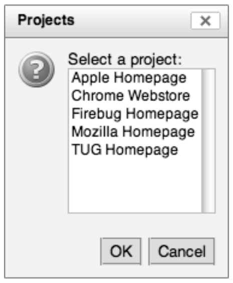
Project Views

Figure 7.6 shows the dialogs used to create new projects, and to load existing projects. In the first screenshot, a new project named "Firebug Homepage" is created. In the

[8] `http://getfirebug.com/`



(a) Project creation dialog.



(b) Project selection dialog.

Figure 7.6: Dialogs for creating and loading projects.

second screenshot, the dialog lists the projects stored by Fawoo, including the project "Firebug Homepage".

Once a project is loaded, an overview of the project's contents is displayed (Figure 7.7). The list in the top left corner displays all jobs currently stored in the project. When selecting a job (by clicking its name), the user interface changes to "job mode" and displays the contents of the selected job; when marking a job (by clicking the checkbox next to the job), it is added to the comparison table displayed in the main part of this view. The comparison table allows to get a quick overview of the marked jobs' contents by listing each job's crawl parameters, graph metrics, and usability metrics.

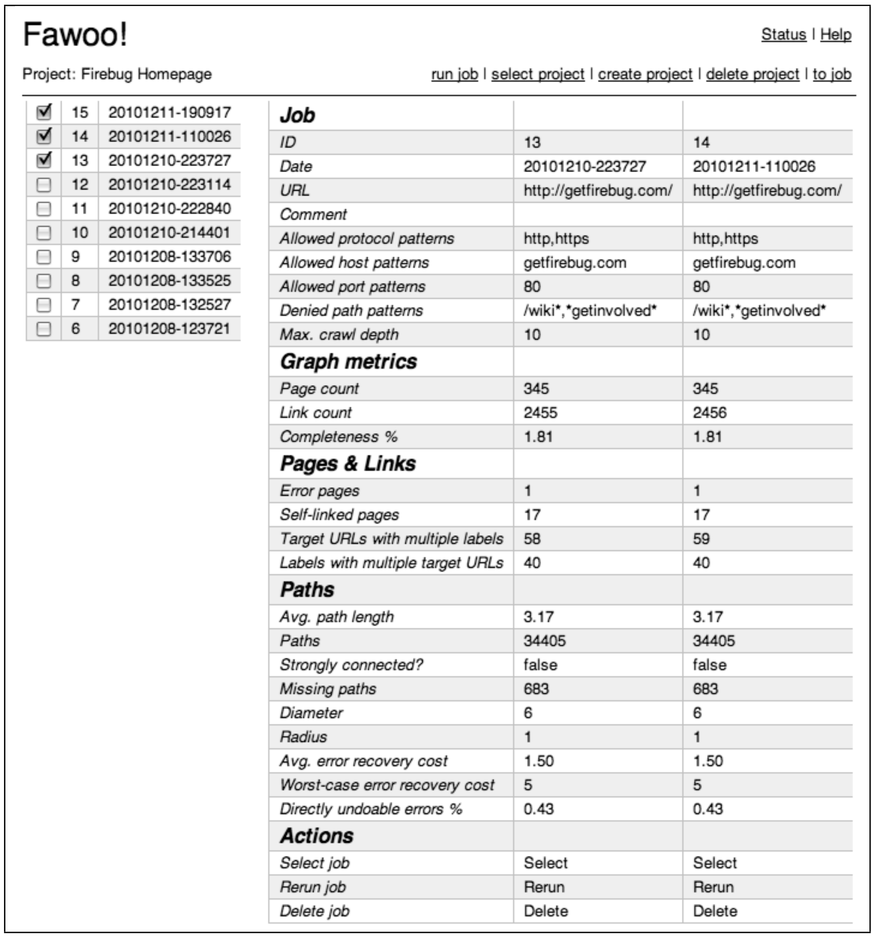
In the example screenshot, two jobs are displayed in the comparison table. The crawl dates indicate that the jobs have been created on two consecutive days; hence, the page graph properties are almost identical. The only notable difference is the increased number of links in the second graph (one link was added).

Further actions available on this view (via the menu in the top right corner) are: running a new crawl job, selecting and creating projects, deleting the current project, and switching to "job mode" (opens the last selected job).

Job Views

When a crawl job is selected in the project overview, the user interface changes to "job mode" (Figure 7.8). In this mode, different views on the selected job are provided. Initially, the job summary view is displayed. This shows the same information as the comparison table on the project overview, for the selected job only.

Other actions available on all job views (via the menu in the top right corner) are: running a new crawl job using the same crawl parameters, deleting the job, and



Fawoo!

Status | Help

Project: Firebug Homepage

run job | select project | create project | delete project | to job

✓	15	20101211-190917
✓	14	20101211-110026
✓	13	20101210-223727
☐	12	20101210-223114
☐	11	20101210-222840
☐	10	20101210-214401
☐	9	20101208-133706
☐	8	20101208-133525
☐	7	20101208-132527
☐	6	20101208-123721

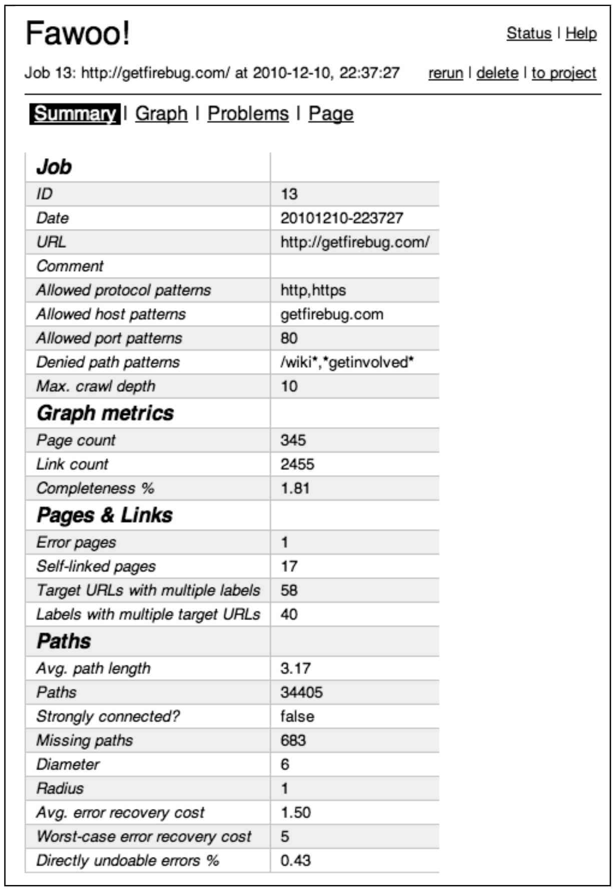
Job		
ID	13	14
Date	20101210-223727	20101211-110026
URL	http://getfirebug.com/	http://getfirebug.com/
Comment		
Allowed protocol patterns	http,https	http,https
Allowed host patterns	getfirebug.com	getfirebug.com
Allowed port patterns	80	80
Denied path patterns	/wiki*,*getinvolved*	/wiki*,*getinvolved*
Max. crawl depth	10	10
Graph metrics		
Page count	345	345
Link count	2455	2456
Completeness %	1.81	1.81
Pages & Links		
Error pages	1	1
Self-linked pages	17	17
Target URLs with multiple labels	58	59
Labels with multiple target URLs	40	40
Paths		
Avg. path length	3.17	3.17
Paths	34405	34405
Strongly connected?	false	false
Missing paths	683	683
Diameter	6	6
Radius	1	1
Avg. error recovery cost	1.50	1.50
Worst-case error recovery cost	5	5
Directly undoable errors %	0.43	0.43
Actions		
Select job	Select	Select
Rerun job	Rerun	Rerun
Delete job	Delete	Delete

Figure 7.7: Project overview.

switching back to "project mode".

When the creation of a new crawl job is initiated (either from the project overview or from a job view), a dialog for entering the crawl parameters is displayed. This dialog offers two display modes: simple mode just allows to enter the start URL for crawling; for all other crawl parameters, default value are used. Figure 7.9 shows the crawl dialog in simple mode, with the URL `http://getfirebug.com/` entered as start URL. When the crawl process is started, the crawl dialog displays progress information (elapsed time so far, number of parsed pages, number of followed links) and allows to cancel crawling (also shown in Figure 7.9).

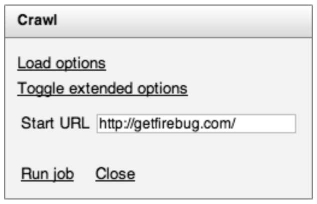
In extended mode, the crawl parameters can be set explicitly (Figure 7.10). These include URL filter settings (for defining which URLs get visited by the crawler), performance settings (the number of parallel parser threads), and HTTP connection


Fawoo!
Status | Help
rerun | delete | to project
Summary | Graph | Problems | Page

Job 13: http://getfirebug.com/ at 2010-12-10, 22:37:27

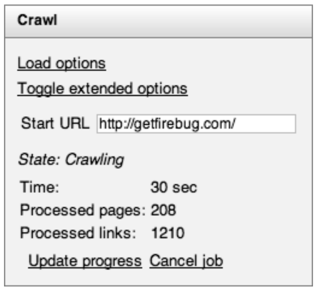
Job	
ID	13
Date	20101210-223727
URL	http://getfirebug.com/
Comment	
Allowed protocol patterns	http,https
Allowed host patterns	getfirebug.com
Allowed port patterns	80
Denied path patterns	/wiki*,*getinvolved*
Max. crawl depth	10
Graph metrics	
Page count	345
Link count	2455
Completeness %	1.81
Pages & Links	
Error pages	1
Self-linked pages	17
Target URLs with multiple labels	58
Labels with multiple target URLs	40
Paths	
Avg. path length	3.17
Paths	34405
Strongly connected?	false
Missing paths	683
Diameter	6
Radius	1
Avg. error recovery cost	1.50
Worst-case error recovery cost	5
Directly undoable errors %	0.43

Figure 7.8: Crawl job summary.



Start URL http://getfirebug.com/

(a) Simple crawl dialog.



Start URL http://getfirebug.com/

State: Crawling

Time: 30 sec
Processed pages: 208
Processed links: 1210

(b) Crawl progress dialog.

Figure 7.9: Dialog for configuring and running a crawl job.

settings. Some additional parameters (e.g., the used browser rendering engine) are currently hard-coded and cannot be changed in the user interface.

In the example screenshot, the URL filter is set to block URLs with paths matching `/wiki*` (denoting the wiki part of Firebug) or `*getinvolved*` (denoting the developer community part of Firebug); the host filter is set to allow `getfirebug.com`, which implicitly excludes sub-domains like `blog.getfirebug.com` (in order to crawl sub-domains as well, the host filter would have to be changed to `*getfirebug.com`)[9]. These settings prevent the crawler from visiting parts of the application which contain *ad-hoc* structures, as found in wiki-style and other user-generated sites. Such sites (or, as in this case, sub-sites) can grow very large and thus cause a lot of workload for the crawler. Furthermore, changes can be performed at unforeseen times and uncontrolled by the owner, causing Fawoo to identify structural changes even though the actual Web application has not been changed. In other contexts, however, crawling wikis might be useful, for example if the structural development of a wiki is to be analyzed.

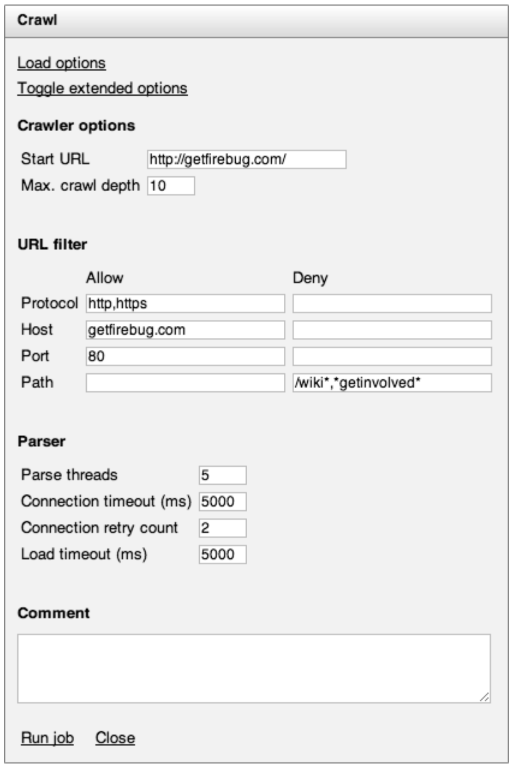
Graph Views

The different graph views are used to manually explore the job's graph, e.g., to find pages or links matching certain search criteria.

Page list view displays a filterable list of all pages contained in the job's graph (Figure 7.11). For each page, the URL, the page title, the number of links to this page, the number of links on this page, and the page type are displayed. The link labeled "show/hide controls" in the lower right corner of the page list toggles display of filter controls. The page type is used to encode different kinds of errors which can be encountered during crawling (e.g., malformed URL, unknown host, invalid HTML) and marks special leaf pages (e.g., non-HTML content type, blocked by the URL filter); "normal" pages are indicated by the page type `Ok`. When clicking on a page's URL, the detail view for the selected page is displayed.

Link list view displays a filterable list of all links contained in the job's graph (Figure 7.12). As on the page list view, the link labeled "show/hide controls" in the lower right corner of the link list toggles display of filter controls. For each link, the source and target pages, the link fragment (the URL part targeting an in-page anchor, e.g., `http://foo.com/bar#foo`), and the link's text label or label image URL are displayed (alternatively to separately displaying target page and link fragment,

[9]Pages blocked by the URL filter still are added to the page graph, with their type property set to `BlockedPage`. This ensures that links targeting blocked pages are preserved.



Crawler options

Start URL	http://getfirebug.com/
Max. crawl depth	10

URL filter

	Allow	Deny
Protocol	http,https	
Host	getfirebug.com	
Port	80	
Path		/wiki*,*getinvolved*

Parser

Parse threads	5
Connection timeout (ms)	5000
Connection retry count	2
Load timeout (ms)	5000

Comment

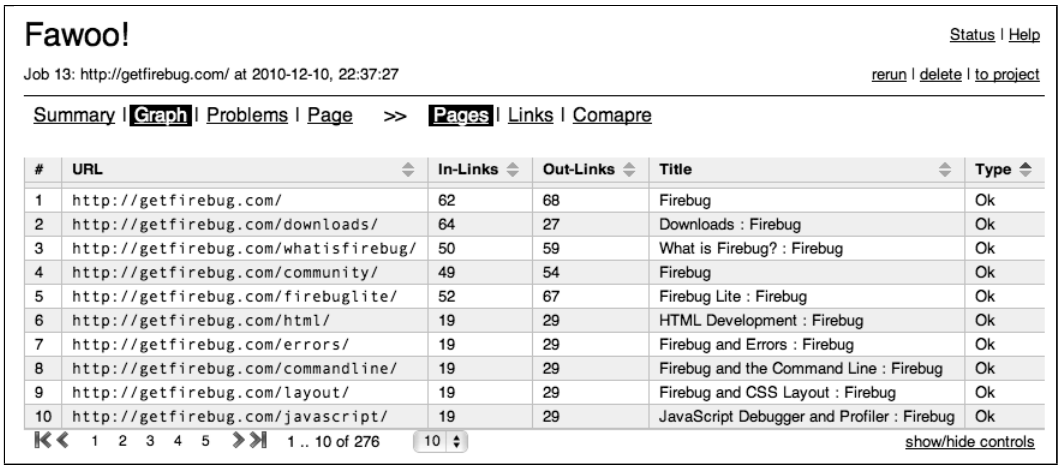
Figure 7.10: Extended crawl dialog.

the target URL combining these two values can be displayed) When clicking on a source or target page's URL, the detail view for the selected page is displayed.

Page detail view shows a summary of a specific page in the job's graph (Figure 7.13). The displayed information comprises a summary of the page's attributes, a list of all links to this page, and a list of all links from this page. The page detail view can be invoked from various other views which display lists of pages or links.

Graph comparison view allows to compare two jobs (Figure 7.14). The view shows a table summarizing the selected jobs, and lists all pages which where added to and removed from the second job, as well as all links which where added to and removed from the second job.

In the example screenshot, the only difference between the two jobs' graphs is one link which was added to the second graph.


Fawoo!

Status | Help

Job 13: http://getfirebug.com/ at 2010-12-10, 22:37:27

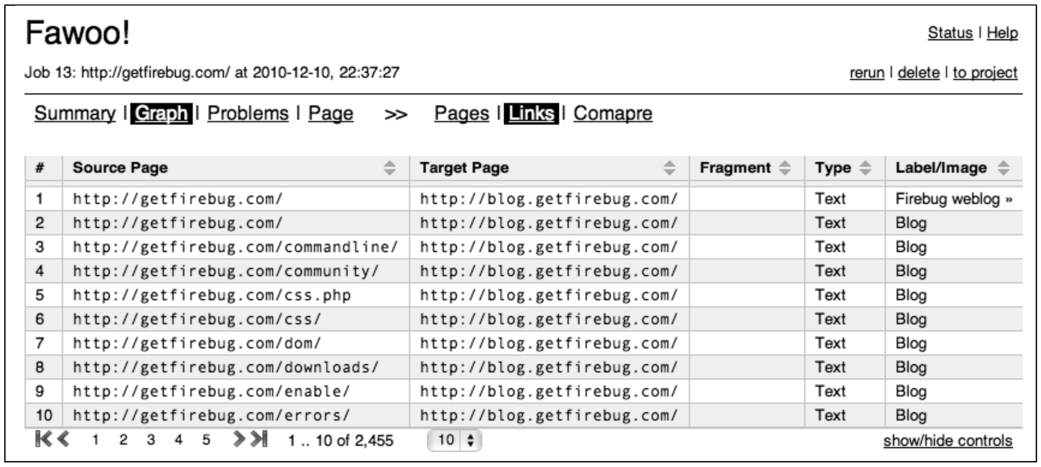
rerun | delete | to project

Summary | Graph | Problems | Page >> Pages | Links | Comapre

#	URL	In-Links	Out-Links	Title	Type
1	http://getfirebug.com/	62	68	Firebug	Ok
2	http://getfirebug.com/downloads/	64	27	Downloads : Firebug	Ok
3	http://getfirebug.com/whatisfirebug/	50	59	What is Firebug? : Firebug	Ok
4	http://getfirebug.com/community/	49	54	Firebug	Ok
5	http://getfirebug.com/fireburglite/	52	67	Firebug Lite : Firebug	Ok
6	http://getfirebug.com/html/	19	29	HTML Development : Firebug	Ok
7	http://getfirebug.com/errors/	19	29	Firebug and Errors : Firebug	Ok
8	http://getfirebug.com/commandline/	19	29	Firebug and the Command Line : Firebug	Ok
9	http://getfirebug.com/layout/	19	29	Firebug and CSS Layout : Firebug	Ok
10	http://getfirebug.com/javascript/	19	29	JavaScript Debugger and Profiler : Firebug	Ok

1 2 3 4 5 1 .. 10 of 276 10 show/hide controls

Figure 7.11: List of all pages in the graph.


Fawoo!

Status | Help

Job 13: http://getfirebug.com/ at 2010-12-10, 22:37:27

rerun | delete | to project

Summary | Graph | Problems | Page >> Pages | Links | Comapre

#	Source Page	Target Page	Fragment	Type	Label/Image
1	http://getfirebug.com/	http://blog.getfirebug.com/		Text	Firebug weblog »
2	http://getfirebug.com/	http://blog.getfirebug.com/		Text	Blog
3	http://getfirebug.com/commandline/	http://blog.getfirebug.com/		Text	Blog
4	http://getfirebug.com/community/	http://blog.getfirebug.com/		Text	Blog
5	http://getfirebug.com/css.php	http://blog.getfirebug.com/		Text	Blog
6	http://getfirebug.com/css/	http://blog.getfirebug.com/		Text	Blog
7	http://getfirebug.com/dom/	http://blog.getfirebug.com/		Text	Blog
8	http://getfirebug.com/downloads/	http://blog.getfirebug.com/		Text	Blog
9	http://getfirebug.com/enable/	http://blog.getfirebug.com/		Text	Blog
10	http://getfirebug.com/errors/	http://blog.getfirebug.com/		Text	Blog

1 2 3 4 5 1 .. 10 of 2,455 10 show/hide controls

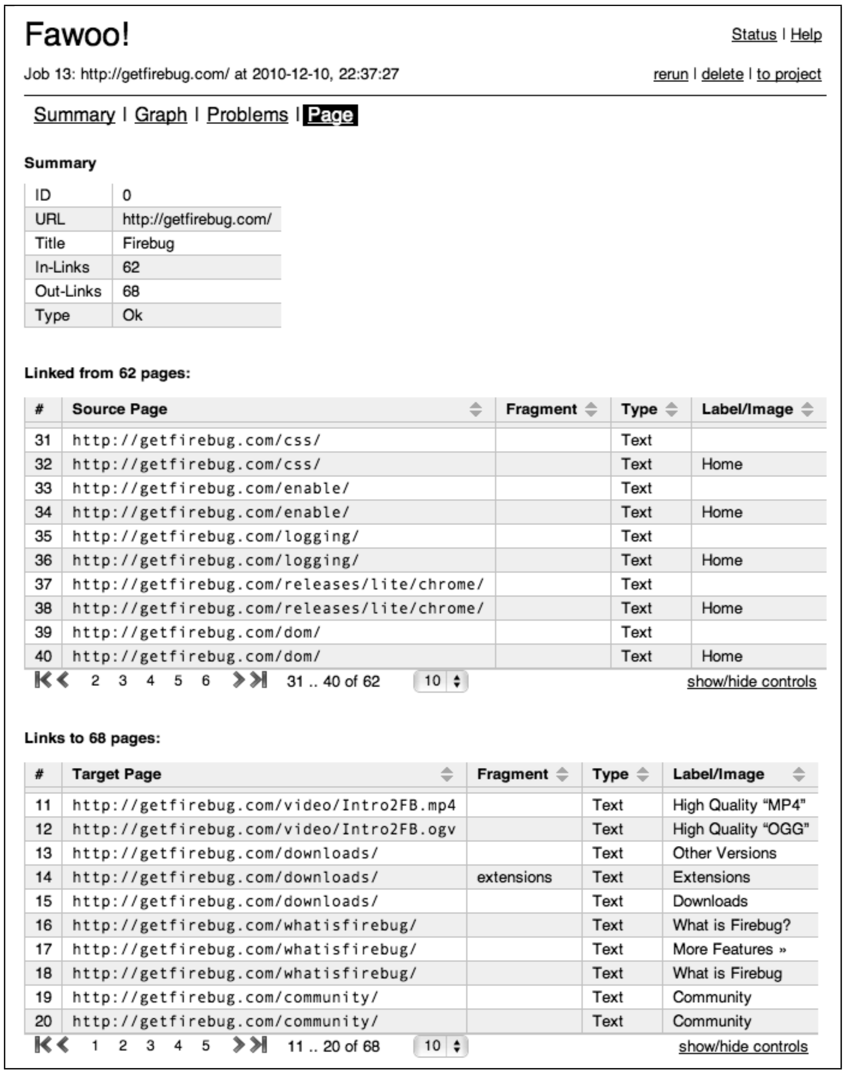
Figure 7.12: List of all links in the graph.

Problem Views

Problem views show various usability problems which where found when analyzing a job's crawl results.

Problem list view is the main problem view (Figure 7.15). It shows a filterable list of all distinct found usability problems. For each problem, the problem type, the problem context (i.e., the location or context of the problem; the affected element), and a textual problem description are displayed. When a problem is selected, additional context information is displayed in a second view below the list.

In the example screenshot, the list displays ten out of 124 identified usability



Fawoo!

Status | Help

Job 13: http://getfirebug.com/ at 2010-12-10, 22:37:27

rerun | delete | to project

Summary | Graph | Problems | Page

Summary

ID	0
URL	http://getfirebug.com/
Title	Firebug
In-Links	62
Out-Links	68
Type	Ok

Linked from 62 pages:

#	Source Page	Fragment	Type	Label/Image
31	http://getfirebug.com/css/		Text	
32	http://getfirebug.com/css/		Text	Home
33	http://getfirebug.com/enable/		Text	
34	http://getfirebug.com/enable/		Text	Home
35	http://getfirebug.com/logging/		Text	
36	http://getfirebug.com/logging/		Text	Home
37	http://getfirebug.com/releases/lite/chrome/		Text	
38	http://getfirebug.com/releases/lite/chrome/		Text	Home
39	http://getfirebug.com/dom/		Text	
40	http://getfirebug.com/dom/		Text	Home

2 3 4 5 6 31 .. 40 of 62 10 show/hide controls

Links to 68 pages:

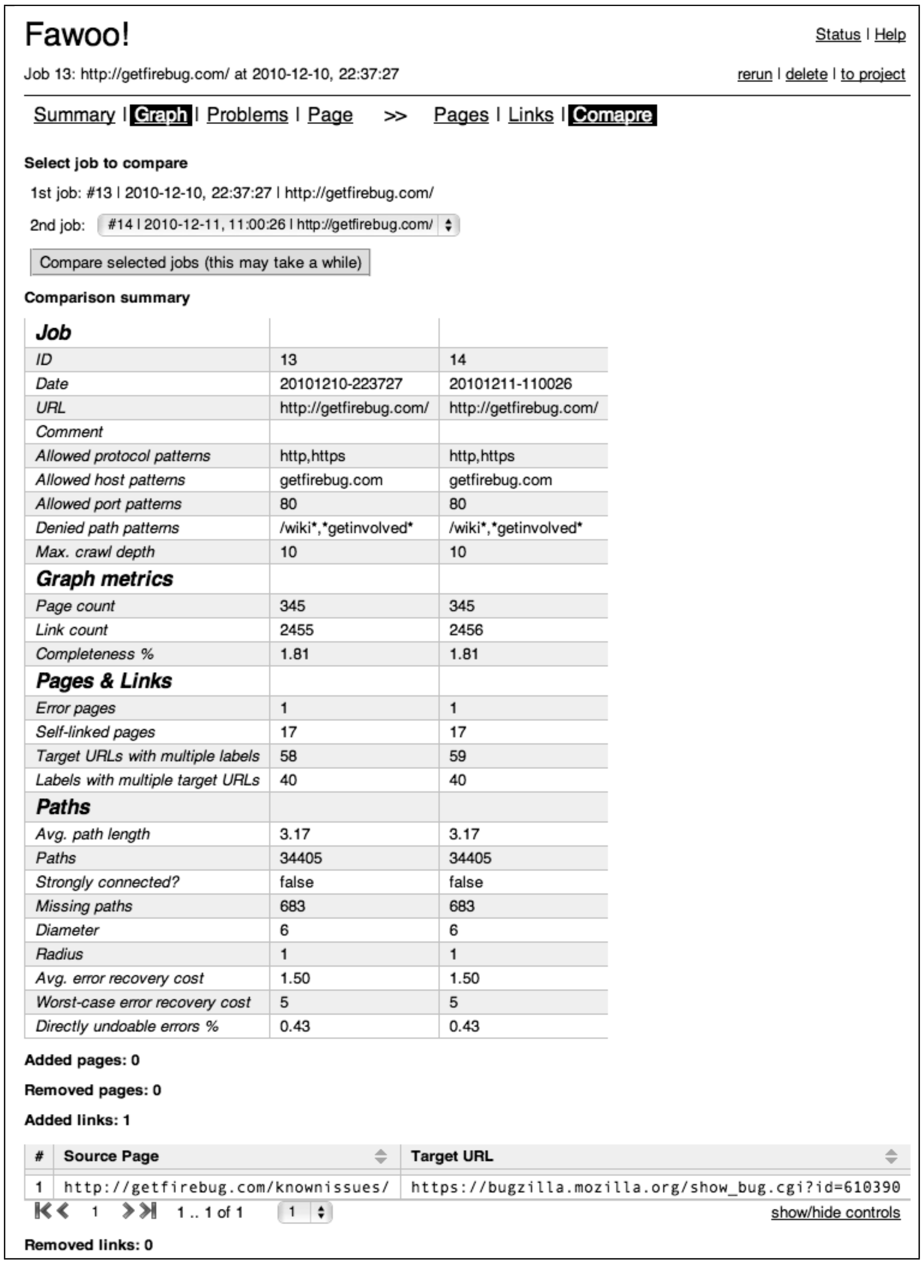
#	Target Page	Fragment	Type	Label/Image
11	http://getfirebug.com/video/Intro2FB.mp4		Text	High Quality "MP4"
12	http://getfirebug.com/video/Intro2FB.ogv		Text	High Quality "OGG"
13	http://getfirebug.com/downloads/		Text	Other Versions
14	http://getfirebug.com/downloads/	extensions	Text	Extensions
15	http://getfirebug.com/downloads/		Text	Downloads
16	http://getfirebug.com/whatisfirebug/		Text	What is Firebug?
17	http://getfirebug.com/whatisfirebug/		Text	More Features »
18	http://getfirebug.com/whatisfirebug/		Text	What is Firebug
19	http://getfirebug.com/community/		Text	Community
20	http://getfirebug.com/community/		Text	Community

1 2 3 4 5 11 .. 20 of 68 10 show/hide controls

Figure 7.13: Page details.

problems. The displayed problems comprise two problem types: self links (see Section 7.4) and uppercase link labels (use of labels only containing uppercase letters is discouraged due to reduced readability). For the selected problem (an uppercase link label problem), the context data is displayed (for this problem type, a list of all links affected by the problem; in this case, only one link is affected).

Inconsistent link labels view (Figure 7.16) is a custom view for a specific usability error type, "inconsistently used link labels" (see Section 7.4; this error type is also displayed in the problem list). The view shows a list of all inconsistently used link


Fawoo!
Status | Help
rerun | delete | to project
Summary | Graph | Problems | Page >> Pages | Links | Comapre
Compare selected jobs (this may take a while)
show/hide controls

Job 13: http://getfirebug.com/ at 2010-12-10, 22:37:27

Select job to compare

1st job: #13 | 2010-12-10, 22:37:27 | http://getfirebug.com/

2nd job: #14 | 2010-12-11, 11:00:26 | http://getfirebug.com/

Comparison summary

Job		
ID	13	14
Date	20101210-223727	20101211-110026
URL	http://getfirebug.com/	http://getfirebug.com/
Comment		
Allowed protocol patterns	http,https	http,https
Allowed host patterns	getfirebug.com	getfirebug.com
Allowed port patterns	80	80
Denied path patterns	/wiki*,*getinvolved*	/wiki*,*getinvolved*
Max. crawl depth	10	10
Graph metrics		
Page count	345	345
Link count	2455	2456
Completeness %	1.81	1.81
Pages & Links		
Error pages	1	1
Self-linked pages	17	17
Target URLs with multiple labels	58	59
Labels with multiple target URLs	40	40
Paths		
Avg. path length	3.17	3.17
Paths	34405	34405
Strongly connected?	false	false
Missing paths	683	683
Diameter	6	6
Radius	1	1
Avg. error recovery cost	1.50	1.50
Worst-case error recovery cost	5	5
Directly undoable errors %	0.43	0.43

Added pages: 0

Removed pages: 0

Added links: 1

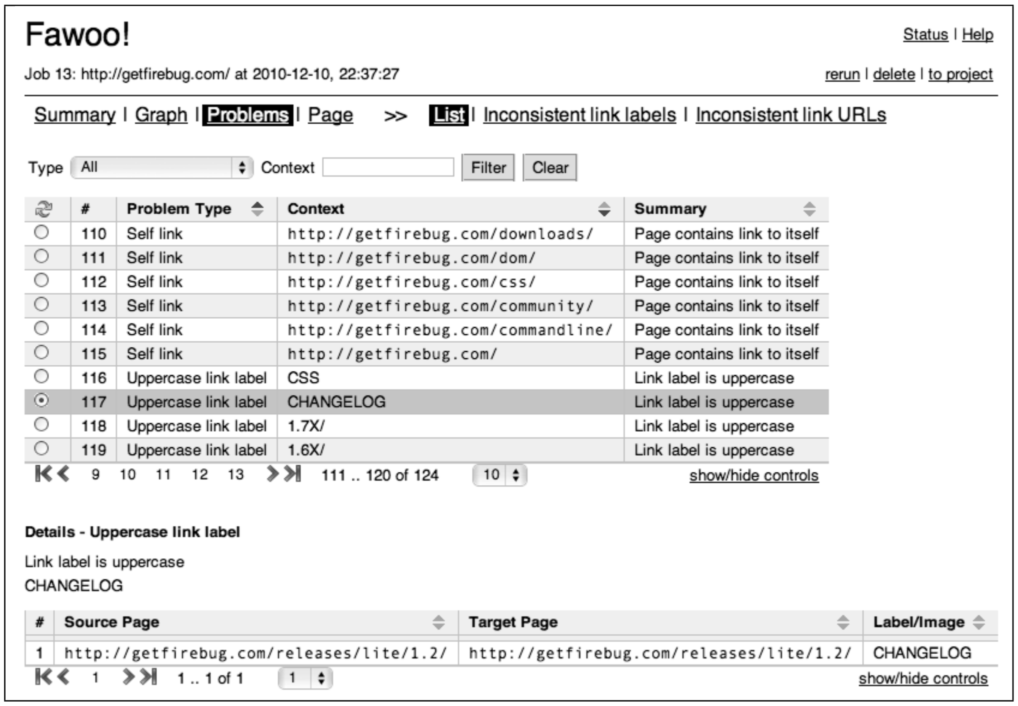
#	Source Page	Target URL
1	http://getfirebug.com/knownissues/	https://bugzilla.mozilla.org/show_bug.cgi?id=610390

1 .. 1 of 1

Removed links: 0

Figure 7.14: Graph comparison.

labels in the graph. When a label is selected, the target URLs which are linked with the selected label are displayed in a second list. When one of the URLs is selected, the links targeting the selected URL with the selected label are displayed in a third list.


Fawoo!
Status | Help
rerun | delete | to project
Summary | Graph | Problems | Page >> List | Inconsistent link labels | Inconsistent link URLs
Type All Context Filter Clear
show/hide controls

Job 13: http://getfirebug.com/ at 2010-12-10, 22:37:27

	#	Problem Type	Context	Summary
○	110	Self link	http://getfirebug.com/downloads/	Page contains link to itself
○	111	Self link	http://getfirebug.com/dom/	Page contains link to itself
○	112	Self link	http://getfirebug.com/css/	Page contains link to itself
○	113	Self link	http://getfirebug.com/community/	Page contains link to itself
○	114	Self link	http://getfirebug.com/commandline/	Page contains link to itself
○	115	Self link	http://getfirebug.com/	Page contains link to itself
○	116	Uppercase link label	CSS	Link label is uppercase
●	117	Uppercase link label	CHANGELOG	Link label is uppercase
○	118	Uppercase link label	1.7X/	Link label is uppercase
○	119	Uppercase link label	1.6X/	Link label is uppercase

9 10 11 12 13 111 .. 120 of 124 10

Details - Uppercase link label

Link label is uppercase
CHANGELOG

#	Source Page	Target Page	Label/Image
1	http://getfirebug.com/releases/lite/1.2/	http://getfirebug.com/releases/lite/1.2/	CHANGELOG

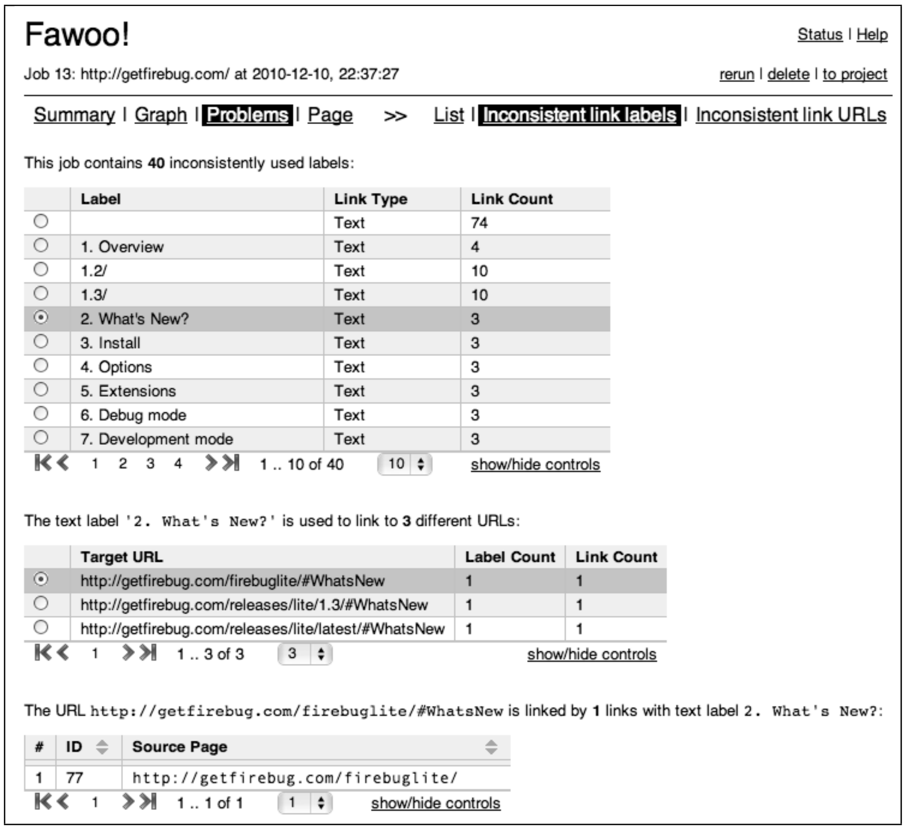
1 1 .. 1 of 1 1

Figure 7.15: Problem list.

In the example screenshot, the label `What's New?` is selected in the first list. This label is used for links to three different URLs; the URL `http://getfirebug.com/firebuglite/#WhatsNew` is selected in the second list. In the third list, the sole link with the label selected label (`What's New?`) linking to the selected URL (`http://getfirebug.com/firebuglite/#WhatsNew`) is displayed.

Inconsistent link target URL view (Figure 7.17) is a custom view for a specific usability error type, "inconsistently labeled link target URLs" (see Section 7.4; this error type is also displayed in the problem list). The view shows a list of all inconsistently linked URLs in the graph. When a URL is selected, the different labels which are used to link to the selected URL are displayed in a second list. When one of the labels is selected, the links targeting the selected URL with the selected label are displayed in a third list.

In the example screenshot, the URL `http://blog.getfirebug.com/` is selected in the first list. Four different link labels are used to link to this URL; one of them, `Blog`, is selected in the second list. In the third list, all 24 links with the label `Blog` linking to `http://blog.getfirebug.com/` are displayed.



Fawoo!

Status | Help

Job 13: http://getfirebug.com/ at 2010-12-10, 22:37:27

rerun | delete | to project

Summary | Graph | Problems | Page >> List | Inconsistent link labels | Inconsistent link URLs

This job contains **40** inconsistently used labels:

	Label	Link Type	Link Count
○		Text	74
○	1. Overview	Text	4
○	1.2/	Text	10
○	1.3/	Text	10
◉	2. What's New?	Text	3
○	3. Install	Text	3
○	4. Options	Text	3
○	5. Extensions	Text	3
○	6. Debug mode	Text	3
○	7. Development mode	Text	3

1 2 3 4 1 .. 10 of 40 10 show/hide controls

The text label `'2. What's New?'` is used to link to **3** different URLs:

	Target URL	Label Count	Link Count
◉	http://getfirebug.com/firebuglite/#WhatsNew	1	1
○	http://getfirebug.com/releases/lite/1.3/#WhatsNew	1	1
○	http://getfirebug.com/releases/lite/latest/#WhatsNew	1	1

1 1 .. 3 of 3 3 show/hide controls

The URL `http://getfirebug.com/firebuglite/#WhatsNew` is linked by **1** links with text label `2. What's New?`:

#	ID	Source Page
1	77	http://getfirebug.com/firebuglite/

1 1 .. 1 of 1 1 show/hide controls

Figure 7.16: Inconsistent link labels list.

Build Integration

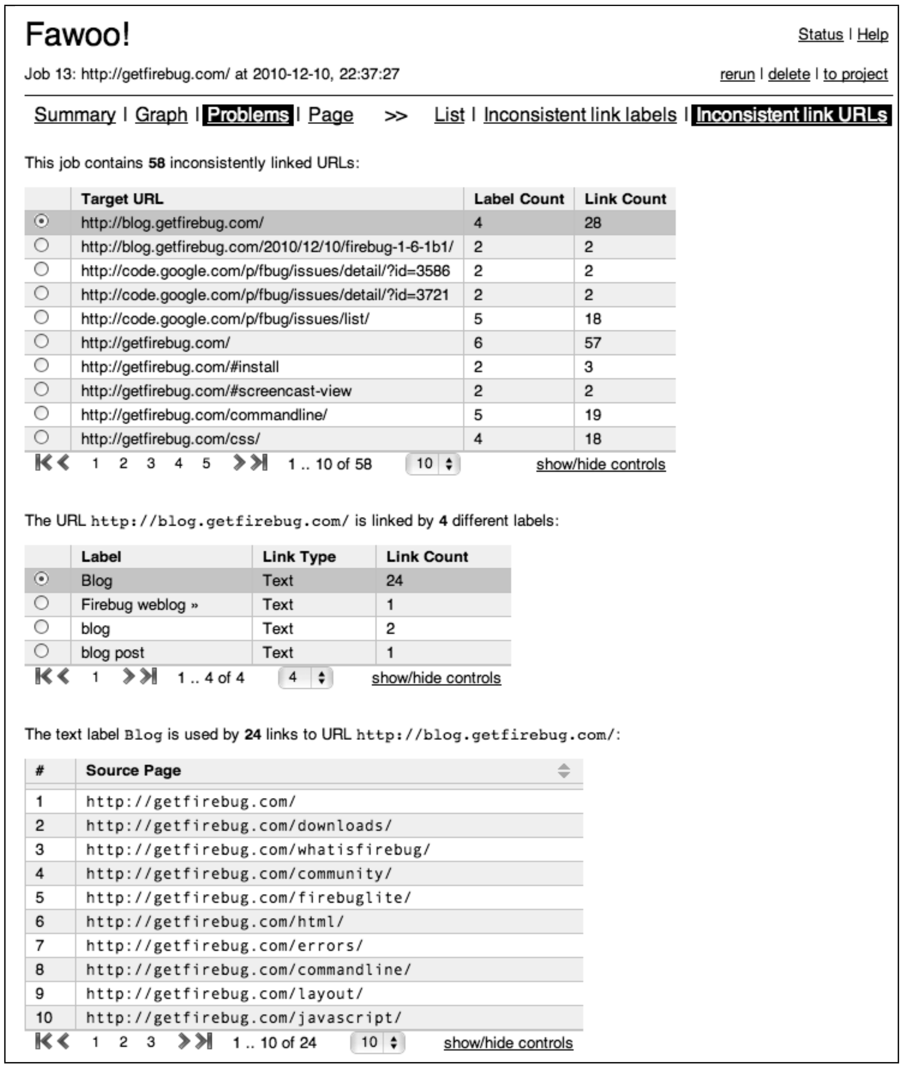
Besides being invoked manually from the user interface, the crawler can also be run automatically. In the context of a Continuous Integration environment, a call to the crawler can be added to the `ant` build script. First, the `ant` task implementing the usability evaluation has to be imported:

```
<taskdef name="fawoo" classname="org.fawoo.FawooTask"/>
```

Then, a target invoking the task `fawoo` for a specific project is defined:

```
<target name="run-fawoo">
  <fawoo home-dir="/opt/fawoo-home/" project-name="foo"/>
</target>
```

This target can be invoked anywhere in the build script; prior to its execution, the application has to be built and deployed successfully.



Fawoo!

Status | Help

Job 13: http://getfirebug.com/ at 2010-12-10, 22:37:27

rerun | delete | to project

Summary | Graph | Problems | Page >> List | Inconsistent link labels | Inconsistent link URLs

This job contains **58** inconsistently linked URLs:

	Target URL	Label Count	Link Count
●	http://blog.getfirebug.com/	4	28
○	http://blog.getfirebug.com/2010/12/10/firebug-1-6-1b1/	2	2
○	http://code.google.com/p/fbug/issues/detail/?id=3586	2	2
○	http://code.google.com/p/fbug/issues/detail/?id=3721	2	2
○	http://code.google.com/p/fbug/issues/list/	5	18
○	http://getfirebug.com/	6	57
○	http://getfirebug.com/#install	2	3
○	http://getfirebug.com/#screencast-view	2	2
○	http://getfirebug.com/commandline/	5	19
○	http://getfirebug.com/css/	4	18

1 2 3 4 5 1 .. 10 of 58 10 show/hide controls

The URL `http://blog.getfirebug.com/` is linked by **4** different labels:

	Label	Link Type	Link Count
●	Blog	Text	24
○	Firebug weblog »	Text	1
○	blog	Text	2
○	blog post	Text	1

1 1 .. 4 of 4 4 show/hide controls

The text label `Blog` is used by **24** links to URL `http://blog.getfirebug.com/`:

#	Source Page
1	http://getfirebug.com/
2	http://getfirebug.com/downloads/
3	http://getfirebug.com/whatisfirebug/
4	http://getfirebug.com/community/
5	http://getfirebug.com/firebuglite/
6	http://getfirebug.com/html/
7	http://getfirebug.com/errors/
8	http://getfirebug.com/commandline/
9	http://getfirebug.com/layout/
10	http://getfirebug.com/javascript/

1 2 3 1 .. 10 of 24 10 show/hide controls

Figure 7.17: Inconsistent link targets list.

In this example, only two parameters are passed to the `fawoo` task: the tool's home directory and the name of the project. The home directory contains the master database storing references to all projects and their associated configurations; the project name identifies one of the projects and allows the task to load the required configuration data (e.g., start URL, URL filter, evaluation parameters, etc.) from the database. This approach requires a project to be created manually using the graphical front-end before running an automated evaluation.

7.5.3 Limitations

The presented tool still lacks support for some important technologies used in Web applications, namely HTML forms, JavaScript and AJAX.

HTML Forms

The crawler can only follow static HTML anchor links but does not support navigation based on HTML forms. Form navigation requires to fill input values into form elements (e.g., text input fields, drop-down lists) before submitting the form to the server (usually by clicking a button). Furthermore, a form often triggers a server-side program which dynamically generates a result page. Each distinct set of form input values might yield a different page.

In order to crawl forms, the crawler would have to be configured with input parameter sets for all forms in the Web application. This poses three problems:

1. Input parameters for all forms have to be provided, which is complex and work-intensive.
2. A representative subset of all possible input parameter combinations has to be chosen for each form in order to limit the crawler's workload (thereby also reducing its coverage).
3. All forms have to be known before crawling, which requires a-prior knowledge of the application's navigation model.

The first two problems can, for example, be solved, or at least alleviated, by using "SmartProfiles", a technique described by Benedikt et al. [20] (this approach still requires a lot of non-automatable work).

The third mentioned problem can be solved by explorative crawling: The crawler is equipped with a lookup L which provides input parameters sets for forms. In crawl session s_i, the crawler reports all unknown forms F_i it encounters. Appropriate input values for F_i have to be provided and are added to L. In the next crawl session s_{i+1}, the crawler can navigate past the forms F_i using L, and will eventually stop crawling with a new set F_{i+1} of unknown forms. After n iterations, this process can be stopped if either F_n is empty, or n exceeds a threshold level (this limits the exploration of graphs which contain a virtually infinite number of pages due to dynamic page generation).

Even though technically possible, form navigation is neither included in the tool concept nor implemented by the tool prototype as the above mentioned requirements contradict the philosophy of striving for full automation of usability evaluation.

This limitation causes the following drawbacks: If a page p_t accessed via a form interface on page p_s is also reachable by a static link on page p_u, only the form-based "link" from p_s to p_t is not added to the graph model; p_t will eventually be reached by another link path via p_u. If, however, p_t is only accessible via the form, neither p_t nor any page only accessible by link paths via p_t are parsed and thus can neither be analyzed nor added to the graph model. For example, a search result page is usually only accessible via a search form. Neither the link from the page hosting the form to the result page, nor the result page itself will be added to the graph; however, the pages accessible from the result page (e.g., product descriptions) usually are also accessible by other means (e.g. a link-based browsing interface) and thus are covered by the crawler. Another example with more serious limitations would be a login form providing access to the secured part of an application. As every page in the secured part requires to pass the login form, no such page will be crawled.

JavaScript and AJAX

JavaScript-driven navigation is not supported by Fawoo. Web applications can attach JavaScript functions to links (or form elements). So instead of following the URL in the `href` attribute of the link tag, the browser runs a client-side script which might redirect to a new page. While the used "virtual browser" software component (WebDriver) supports execution of JavaScript code, JavaScript navigation is currently not implemented in Fawoo.

Asynchronous page updates are not supported by Fawoo. The crawler and the graph model are based on the assumption that each visible "state" of the application is represented by a unique URL. The current trend towards asynchronous technologies (e.g., AJAX) leads to applications where servers send data instead of documents to clients (browsers), and rendering of that data is performed on the client-side (e.g., by dynamically manipulating the DOM). The application state no longer is reflected by a URL, but by the page's contents. The structure of such an application still can be represented as a graph [112]; however, a new hashing function for deriving application state identifiers from pages has to be found. Tonella and Ricco [114] propose to utilize the client-side DOM state for such a function.

Features

Besides technical limitations, some of the described features are missing in the current prototype implementation.

Fawoo contains a fixed set of evaluation types. In order to increase flexibility of development and usage, a plug-in system for dynamically attaching evaluation modules would be feasible.

The used “virtual browser” is not exploited to the full extent. HTML element positions and sizes are not evaluated for usability aspects, and the Fighting Layout Bugs library (which is based on the same “virtual browser”, Selenium/WebDriver) is not integrated. Thus, some important usability metrics are not yet covered by Fawoo. However, all metrics have been implemented in standalone proof-of-concept studies and thus can be considered to also work in the context of an automated usability evaluation tool like Fawoo.

7.6 Conclusion and Future Work

In this chapter, the concept and the prototypical implementation of a tool for fully automated usability evaluation of Web applications was presented. The tool is capable of evaluating single pages as well as the entire application structure regarding usability aspects. Furthermore, the tool can be integrated into a Continuous Integration system in order to automate is execution.

Fully automated usability evaluations cannot replace evaluations incorporating humans, be it analysis by usability experts or feedback from users. Still, AUE methods can point out some common usability errors of Web applications, and give clues for further investigations targeted at potential problems. Automation allows to apply usability evaluation comprehensively (i.e., checking the whole application, not just some representative pages) and repeatedly (i.e., the usability impact of every change to the UI can be evaluated). Furthermore, integration of AUE methods can increase awareness of usability concerns amongst developers.

Future work on the presented tool—and on Web AUE in general—incorporates support for HTML forms and client-side technologies like AJAX. Furthermore, the tool’s applicability for real-world projects will be evaluated by testing it in the context of a commercial project.

Chapter 8

Epilogue

In this concluding chapter, the research results presented in this thesis are reviewed, and topics for future research in the field of automated usability analysis are presented.

8.1 Conclusions

In this thesis, different concepts of the integration of usability engineering practices into agile software development have been examined. These concepts are mainly based on two aspects: the agile development method Extreme Programming (XP) has been adapted in order to better support usability concerns; and a novel toolset supporting the specific demands of usability evaluation in the context of agile development has been created.

Adapted Development Process

The presented adaptions to the standard XP development process comprise three main factors:

- By synchronizing development and user interface design iterations, feature requirements and usability demands no longer have to be planned separately but are part of the same iteration plan.
- Usability experts are considered an integral part of the development team, similar to the "customer on site" required by XP. Instead of consulting external experts at defined time points in the project life cycle (e.g., before release plannings), a dedicated expert is available for inquiries and feedback. This ensures that no design decision is taken without considering its usability implications.

- Existing usability engineering instruments have been integrated into the development workflow. The *Personas* approach has been adjusted to the special demands of XP, resulting in *Extreme Personas*; different automated usability evaluation methods have been integrated.

The effects of these adaptions have been evaluated using different measurements:

- The development process has been monitored using the Extreme Programming Evaluation Framework (XP:EF, [68] [121]). The results of XP:EF showed that the applied process adaptions do not interfere with the development workflow of XP. On the contrary, XP:EF illustrated that the adapted process allows to apply XP practices to other development aspects, namely the user-interface design and the human-computer interaction model.
- The usability of the developed software product has been evaluated with user studies and usability expert evaluations. In both cases, the feedback on the software's usability was confirming, indicating positive effects of the usability adaptions.
- A general rise in the awareness and acceptance of usability concerns among the development team could be observed.

Automated Usability Evaluation Toolset

An AUE tool for Web applications, capable of integration in Continuous Integration (CI) systems, was developed. The tool monitors the usability effects of code changes, thereby allowing for continuous, automated evaluation of usability concerns. Revisiting the design goals stated in Section 1.1:

- The tool automates the collection of usability data by crawling the deployed Web application. Each visited page is analyzed in order to identify usability problems. In addition to that, the navigation graph of the crawled Web application is extracted, allowing for structural analysis.
- The manual effort required from the members of the project team is minimal as the tool can be used without any additional configuration overhead. However, for optimal results some configuration steps are recommended.
- By integration into a CI system, the tool's evaluations can be executed implicitly (e.g., at every committed code change, or during nightly builds). The

results of the analysis steps are reported to the development team, causing the software build to fail in case of severe usability defects.

The tool has been implemented as a proof-of-concept and was successfully integrated into the popular CI system CruiseControl [30]. The AUE quality of the tool is currently being examined in an expert-driven evaluation conducted by CURE [43]. HCI experts participating in the evaluation perform heuristic evaluations of a Web site's usability with and without the AUE tool and fill out questionnaires. The preliminary results of this analysis indicate that the tool provides a solid coverage of common usability issues, and additionally helps to identify some otherwise easily overlooked issues.

8.2 Challenges and Perspectives

In the field of automated usability evaluation for Web applications, the biggest challenge for future development lies in the support of current and upcoming technologies used for building Web applications. The concept presented in this thesis requires a relatively simple Web architecture in order to be applied; many state-of-the-art technologies (e.g., client-side scripting) are not supported due to the lack of standardized programming interfaces. The supported navigation model is based on anchor tags in plain HTML documents, which are relatively easy to parse; every visible and relevant application state is reflected by a unique URL. However, more advanced technologies allow programmers to embed arbitrary logic in their Web applications. For example, navigation actions are not limited to links but can be attached to virtually any page element; state transitions are not necessarily reflected by changed URLs; Web pages can change the displayed content dynamically. There exist promising approaches for accessing pages employing such technologies; however, currently no such approach is sufficiently developed for general applicability.

Another challenge is the porting of the presented approach to desktop-based software technologies. In theory, the concept can be applied to any software technology, provided that there is a uniform way of parsing the user interface, and a function for deriving a unique identifier representing the application state is available. Many popular user interface frameworks either incorporate explicit model creation for input masks and navigation (e.g., by means of markup languages), or support deriving of such models from the executed application (e.g., by means of reflection), and hence are potential platforms for fully automated usability evaluation as described in this thesis. However, the diversity of existing frameworks prevents the creation of a generally applicable approach.

Bibliography

[1] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *HUC '99: Proceedings of the 1st international Symposium on Handheld and Ubiquitous Computing*, pages 304–307, London, UK, 1999. Springer-Verlag.

[2] Pekka Abrahamsson, Juhani Warsta, Mikko T. Siponen, and Jussi Ronkainen. New directions on agile methods: A comparative analysis. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 244–254, Washington, DC, USA, 2003. IEEE Computer Society.

[3] Rashid Ahmad, Zhang Li, and Farooque Azam. Measuring navigational burden. In *SERA '06: Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications*, pages 307–314, Washington, DC, USA, 2006. IEEE Computer Society.

[4] Ambysoft. IT project success rates: Survey results. http://www.ambysoft.com/surveys/success2007.html. Last visit: 15.01.2008.

[5] Morten Sieker Andreasen, Henrik Villemann Nielsen, Simon Ormholt Schrøder, and Jan Stage. What happened to remote usability testing?: an empirical study of three methods. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1405–1414, New York, NY, USA, 2007. ACM.

[6] Richard Atterer. Model-based automatic usability validation: a tool concept for improving web-based UIs. In *NordiCHI '08: Proceedings of the 5th Nordic conference on Human-computer interaction*, pages 13–22, New York, NY, USA, 2008. ACM.

[7] ATutor. ATRC web accessibility checker. http://www.achecker.ca. Last visit: 30.04.2010.

[8] Fiora T. W. Au, Simon Baker, Ian Warren, and Gillian Dobbie. Automated usability testing framework. In *AUIC '08: Proceedings of the ninth conference on Australasian user interface*, pages 55–64, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.

[9] Jakob Otkjaer Bak, Kim Nguyen, Peter Risgaard, and Jan Stage. Obstacles to usability evaluation in practice: a survey of software development organizations. In *NordiCHI '08: Proceedings of the 5th Nordic conference on Human-computer interaction*, pages 23–32, New York, NY, USA, 2008. ACM.

[10] Christoph Bartneck. Interacting with an embodied emotional character. In *DPPI '03: Proceedings of the 2003 International Conference on Designing Pleasurable Products and Interfaces*, pages 55–60, New York, NY, USA, 2003. ACM.

[11] BBC. Online video eroding tv viewing. http://news.bbc.co.uk/2/hi/entertainment/6168950.stm. Last visit: 31.05.2007.

[12] Kent Beck. *Extreme Programming Explained: Embrace Change (1st Edition).* Addison-Wesley Professional, 1999.

[13] Kent Beck. *Extreme Programming Explained: Embrace Change (1st Edition).* Addison-Wesley Professional, 1999.

[14] Kent Beck. *Test-Driven Development: By Example.* Addison-Wesley, 2002.

[15] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change (2nd Edition).* Addison-Wesley Professional, November 2004.

[16] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development. http://www.agilemanifesto.org/, 2001. Last visit: 26.03.2008.

[17] M. Beedle and K. Schwaber. *Agile Software Development with Scrum, 1st edition.* Prentice Hall PTR, 2001.

[18] Armin Beer, Stefan Mohacsi, and Christian Stary. Idatg: An open tool for automated testing of interactive software. In *COMPSAC '98: Proceedings of the 22nd International Computer Software and Applications Conference*, pages 470–475, Washington, DC, USA, 1998. IEEE Computer Society.

[19] Raquel Benbunan-Fich and Alberto Benbunan. Understanding user behavior with new mobile applications. *Journal of Strategic Information Systems*, 16(4):393–412, 2007.

[20] Michael Benedikt, Juliana Freire, and Patrice Godefroid. Veriweb: Automatically testing dynamic web sites. In *In Proceedings of 11th International World Wide Web Conference (WWW2002*, 2002.

[21] Blinkx. Video search engine. http://www.blinkx.com. Last visit: 01.11.2007.

[22] Jan Blom, Jan Chipchase, and Jaakko Lehikoinen. Contextual and cultural challenges for user mobility research. *Communications of the ACM*, 48(7):37–41, 2005.

[23] M.M. Bradley and P.J. Lang. Measuring emotion: the self-assessment manikin and the semantic differential. *Journal of Behavioral Therapy and Experimental Psychiatry*, 25(1):49–59, March 1994.

[24] Stuart K. Card, Thomas P. Moran, , and Allen Newell. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum, 1983.

[25] Christer Carlsson and Pirkko Walden. Mobile tv - to live or die by content. In *HICSS '07: Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, page 51, Washington, DC, USA, 2007. IEEE Computer Society.

[26] Stephanie Chamberlain, Helen Sharp, and Neil Maiden. Towards a framework for integrating agile development and user-centred design. In *7th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2006*, volume 4044 of *LNCS*, pages 143–153, Heidelberg, Germany, 2006. Springer.

[27] D. Cohen, M. Lindvall, and P. Costa. *An introduction to agile methods.* New York: Elsevier Science, 2004.

[28] Larry L. Constantine and Lucy A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design.* ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.

[29] Larry L. Constantine and Lucy A. D. Lockwood. Usage-centered software engineering: An agile approach to integrating users, user interfaces, and usability into software engineering practice. In *ICSE '03*, pages 746–747. IEEE Computer Society, 2003.

[30] Cruise Control. Cruise control. http://cruisecontrol.sourceforge.net/. Last visit: 8.3.2010.

[31] Luca de Alfaro, Thomas A. Henzinger, and Freddy Y. C. Mang. MCWEB: A model-checking tool for web site debugging. In *In World Wide Web Conference (WWW) – Poster Proceedings*, pages 86–87, 2001.

[32] Jen-Wen Ding, Chin-Tsai Lin, and Kai-Hsiang Huang. ARS: An adaptive reception scheme for handheld devices supporting mobile video streaming services. In *International Conference on Consumer Electronics. ICCE '06*, volume 1, pages 141– 142, 2006.

[33] Joseph Dumas and Janice Redish. *A Practical Guide to Usability Testing, Revised Edition*. Intellect, 1999.

[34] EMMA. Emma: Java code coverage tool. http://emma.sourceforge.net/. Last visit: 26.03.2008.

[35] Everyzing. Everyzing. http://www.everyzing.com. Last visit: 01.11.2007.

[36] Adrian Fernandez, Emilio Insfran, and Silvia Abrahao. Integrating a usability model into model-driven web development processes. In *WISE '09: Proceedings of the 10th International Conference on Web Information Systems Engineering*, pages 497–510, Berlin, Heidelberg, 2009. Springer-Verlag.

[37] Jennifer Ferreira, James Noble, and Robert Biddle. Agile development iterations and UI design. In *Agile 2007*, pages 50–58. IEEE Computer Society, 2007.

[38] Sebastian Feuerstack, Marco Blumendorf, Maximilian Kern, Michael Kruppa, Michael Quade, Mathias Runge, and Sahin Albayrak. Automated usability evaluation during model-based interactive system development. In *HCSE-TAMODIA '08: Proceedings of the 2nd Conference on Human-Centered Software Engineering and 7th International Workshop on Task Models and Diagrams*, pages 134–141, Berlin, Heidelberg, 2008. Springer-Verlag.

[39] William Foddy. *Constructing Questions for Interviews and Questionnaires: Theory and Practice in Social Research.* Cambridge University Press, 1994.

[40] Robert Gittins and Sian Hope. A study of human solutions in extreme programming. In *PPIG 2001, The 13th Annual Psychology of Programming Interest Group Conference, Bournemouth, UK. 10th - 12th September 2008*, pages 41–51, 2001.

[41] Google. About Gmail. http://mail.google.com/mail/help/intl/en/about.html. Last visit: 25.05.2007.

[42] Bengt Göransson, Jan Gulliksen, and Inger Boivie. The usability design process – integrating user-centered systems design in the software development process. *Software Process: Improvement and Practice*, 8(2):111–131, 2003.

[43] Cornelia Graf, Peter Wolkerstorfer, and Manfred Tscheligi. FAWOO - evaluation of an automated usability evaluation tool through HCI experts. Unpublished manuscript, 2011.

[44] Jan Gulliksen, Bengt Göransson, Inger Boivie, Stefan Blomkvist, Jenny Persson, and Äsa Cajander. Key principles for user-centred systems design. *Behaviour & Information Technology, Special Section on Designing IT for Healthy Work*, Vol. 22 No. 6:397–409, 2003.

[45] Marc Hassenzahl, Michael Burmester, and Franz Koller. AttrakDiff: Ein Fragebogen zur Messung wahrgenommener hedonischer und pragmatischer Qualität. In G. Szwillus and J. Ziegler, editors, *Mensch and Computer 2003: Interaktion in Bewegung*, pages 187–196, Stuttgart, 2003. B. G. Teubner.

[46] Orit Hazzan and James E. Tomayko. Human aspects of software engineering: The case of extreme programming. In *XP*, pages 303–311, 2004.

[47] Andreas Holzinger. Usability engineering for software developers. *Communications of the ACM*, 48:71–74, 2005.

[48] Andreas Holzinger, Maximilian Errath, Gig Searle, Bettina Thurnher, and Wolfgang Slany. From Extreme Programming and Usability Engineering to Extreme Usability in software engineering education (XP+UE→XU). In *COMPSAC '05: Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05) Volume 2*, pages 169–172, Washington, DC, USA, 2005. IEEE Computer Society.

[49] Andreas Holzinger, Gig Searle, and Alexander K. Nischelwitzer. On some aspects of improving mobile applications for the elderly. In Constantine Stephanidis, editor, *Universal Access in HCI*, volume 4554 of *Lecture Notes in Computer Science*, pages 923–932. Springer, 2007.

[50] Zahid Hussain, Martin Lechner, Harald Milchrahm, Sara Shahzad, Wolfgang Slany, Martin Umgeher, and Thomas Vlk. Optimizing extreme programming. In *ICCCE 2008: Proceedings of the International Conference on Computer and Communication Engineering, Kuala Lumpur, Malaysia*, pages 1052–1056. IEEE, 2008.

[51] Zahid Hussain, Martin Lechner, Harald Milchrahm, Sara Shahzad, Wolfgang Slany, Martin Umgeher, Thomas Vlk, and Peter Wolkerstorfer. User interface design for a content-aware mobile multimedia application: An iterative approach. In *Frontiers in Mobile and Web Computing: Proceedings of MoMM2007 & iiWAS2007 Workshops*, volume 231, pages 115–120, Jakarta, Indonesia, 2007.

[52] Zahid Hussain, Martin Lechner, Harald Milchrahm, Sara Shahzad, Wolfgang Slany, Martin Umgeher, Thomas Vlk, and Peter Wolkerstorfer. User interface design for a mobile multimedia application: An iterative approach. In *ACHI '08: Proceedings of the International Conference on Advances in Computer-Human Interaction 2008*, pages 189–194, 2008. Published 1st International Conference on Advances in Computer-Human Interaction (ACHI 2008) February 10-15, 2008 - Sainte Luce, Martinique.

[53] Zahid Hussain, Martin Lechner, Harald Milchrahm, Sara Shahzad, Wolfgang Slany, Martin Umgeher, and Peter Wolkerstorfer. Agile user-centered design applied to a mobile multimedia streaming application. In *USAB 2008*, volume 5298/2008 of *Lecture Notes in Computer Science*, pages 313–330. Springer Berlin / Heidelberg, November 2008.

[54] Zahid Hussain, Martin Lechner, Harald Milchrahm, Sara Shahzad, Wolfgang Slany, Martin Umgeher, and Peter Wolkerstorfer. Integrating extreme programming and user-centered design. In *PPIG 2008, The 20th Annual Psychology of Programming Interest Group Conference, Lancaster University, UK. 10th - 12th September 2008*, 2008.

[55] Zahid Hussain, Martin Lechner, Harald Milchrahm, Sara Shahzad, Wolfgang Slany, Martin Umgeher, and Peter Wolkerstorfer. Concept and design of a

contextual mobile multimedia content usability study. In *ACHI*, pages 277–282. IEEE, 2009.

[56] Zahid Hussain, Harald Milchrahm, Sara Shahzad, Wolfgang Slany, Manfred Tscheligi, and Peter Wolkerstorfer. Integration of extreme programming and user-centered design: Lessons learned. In *XP*, volume 31 of *LNBIP*, pages 174–179. Springer, 2009.

[57] Zahid Hussain, Wolfgang Slany, and Andreas Holzinger. Current state of agile user-centered design: A survey. In *USAB 2009*, volume 5889 of *Lecture Notes in Computer Science*, pages 416–427. Springer Berlin / Heidelberg, November 2009. USAB 2009 - HCI and Usability for e-Inclusion, Linz, Austria, 9-10 November 2009.

[58] Emilio Insfran and Adrian Fernandez. A systematic review of usability evaluation in web development. In *WISE '08: Proceedings of the 2008 international workshops on Web Information Systems Engineering*, pages 81–91, Berlin, Heidelberg, 2008. Springer-Verlag.

[59] Melody Y. Ivory and Marti A Hearst. The state of the art in automating usability evaluation of user interfaces. *ACM Comput. Surv.*, 33(4):470–516, 2001.

[60] Timo Jokela and Pekka Abrahamsson. Usability assessment of an extreme programming project: Close co-operation with the customer does not equal to good usability. In *5th International Conference, PROFES '04*, pages 393–407, 2004.

[61] Joost. Free online tv. http://www.joost.com. Last visit: 01.11.2007.

[62] Satu Jumisko-Pyykkö and Jukka Häkkinen. Evaluation of subjective video quality of mobile devices. In *MULTIMEDIA '05: Proceedings of the 13th Annual ACM International Conference on Multimedia*, pages 535–538, New York, NY, USA, 2005. ACM.

[63] Jumpcut. Be good to your video. http://www.jumpcut.com. Last visit: 01.11.2007.

[64] Plinio Thomaz Aquino Junior and Lucia Vilela Leite Filgueiras. User modeling with personas. In *CLIHC '05: Proceedings of the 2005 Latin American Conference on Human-Computer Interaction*, pages 277–282, New York, NY, USA, 2005. ACM.

[65] Eeva Kangas and Timo Kinnunen. Applying user-centered design to mobile application development. *Communications of the ACM*, 48(7):55–59, 2005.

[66] Jesper Kjeldskov and Jan Stage. New techniques for usability evaluation of mobile systems. *International Journal of Human-Computer Studies*, 60(5-6):599–620, May 2004.

[67] Hendrik Knoche, John D. McCarthy, and M. Angela Sasse. Can small be beautiful?: Assessing image resolution requirements for mobile tv. In *MULTIMEDIA '05: Proceedings of the 13th Annual ACM International Conference on Multimedia*, pages 829–838, New York, NY, USA, 2005. ACM Press.

[68] Bill Krebs. Shodan 2.0 input metric survey. http://agile.csc.ncsu.edu/survey. Last visit: 03.08.2011.

[69] Philippe Kruchten. *The Rational Unified Process: An Introduction, 3 edition.* Addison-Wesley Longman Publishing Co., 2003.

[70] Amy Law and Raylene Charron. Effects of agile practices on social factors. In *HSSE '05: Proceedings of the 2005 Workshop on Human and Social Factors of Software Engineering*, volume 30, pages 1–5, New York, NY, USA, July 2005. ACM Press.

[71] Sang Ho Lee, Sung Jin Kim, and Seok Hoo Hong. On URL normalization. In Osvaldo Gervasi, Marina Gavrilova, Vipin Kumar, Antonio Lagan, Heow Lee, Youngsong Mun, David Taniar, and Chih Tan, editors, *Computational Science and Its Applications - ICCSA 2005*, volume 3481 of *Lecture Notes in Computer Science*, pages 1076–1085. Springer Berlin, Heidelberg, 2005. 10.1007/11424826_115.

[72] Michael Leitner, Peter Wolkerstorfer, Reinhard Sefelin, and Manfred Tscheligi. Mobile multimedia: Identifying user values using the means-end theory. In *Proceedings of the 10th International Conference on Human Computer Interaction with Mobile Devices and Services*, pages 167–175, Amsterdam, The Netherlands, 2008. ACM.

[73] Clayton Lewis, Peter Polson, Cathleen Wharton, and John Rieman. Methodology for theory-based design of walk-up-and-use interfaces. In *Proceedings of Conference on Human Factors in Computing Systems (CHI90)*, pages 235–242. ACM, 1990.

[74] Clayton Lewis and John Rieman. *Task-Centered User Interface Design: A Practical Introduction.* Shareware book, University of Colorado, 1993.

[75] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, January 2003.

[76] LinesOfCodeWichtel. Linesofcodewichtel. http://www.andreas-berl.de/linesofcodewichtel/en/index.html. Last visit: 25.03.2008.

[77] Joe Marasco. Software development productivity and project success rates: Are we attacking the right problem? http://www.ibm.com/developerworks/rational/library/feb06/marasco/index.html, Feb 2006. Last visit: 15.01.2008.

[78] Deborah Mayhew. *Principles and Guidelines in Software User Interface Design.* Prentice-Hall, 1991.

[79] John D. McCarthy, M. Angela Sasse, and Dimitrios Miras. Sharp or smooth?: Comparing the effects of quantization vs. frame rate for streamed video. In *CHI '04: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 535–542, New York, NY, USA, 2004. ACM.

[80] Steve McConnell. *Code Complete, Second Edition.* Microsoft Press, Redmond, WA, USA, 2004.

[81] Paul McInerney and Frank Maurer. UCD in agile projects: Dream team or odd couple? *Interactions*, 12(6):19–23, 2005.

[82] Laurianne McLaughlin. Next-generation entertainment: Video goes mobile. *IEEE Pervasive Computing*, 06(1):7–10, 2007.

[83] Marc McNeill. User centred design in agile application development. http://www.thoughtworks.com/pdfs/agile_and_UCD_MM.pdf. Last visit: 30.03.2008.

[84] Thomas Memmel, Fredrik Gundelsweiler, and Harald Reiterer. Agile human-centered software engineering. In *BCS-HCI '07: Proceedings of the 21st British CHI Group Annual Conference on HCI 2007*, pages 167–175, Swinton, UK, UK, 2007. British Computer Society.

[85] Thomas Memmel, Harald Reiterer, and Andreas Holzinger. Agile methods and visual specification in software development: A chance to ensure universal access. In Constantine Stephanidis, editor, *Universal Access in HCI*, volume 4554 of *LNCS*, pages 453–462. Springer, 2007.

[86] John Mendonca and Jeff Brewer. *Lean, Light, Adaptive, Agile and Appropriate Software Development: The Case for a less methodical Methodology*, pages 42–52. IGI Publishing, Hershey, PA, USA, 2003.

[87] G. Meszaros and J. Aston. Adding usability testing to an agile project. In *Agile Conference*, 2006.

[88] Gerard Meszaros. *XUnit Test Patterns: Refactoring Test Code.* Addison-Wesley, 2007.

[89] Huaikou Miao, Zhongsheng Qian, and Bo Song. Towards automatically generating test paths for web application testing. In *TASE '08: Proceedings of the 2008 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering*, pages 211–218, Washington, DC, USA, 2008. IEEE Computer Society.

[90] National Institute of Standards and Technology. NIST web metrics. http://zing.ncsl.nist.gov/WebTools/. Last visit: 11.10.2010.

[91] Jakob Nielsen. *Usability Engineering.* Morgan Kaufmann, 1993.

[92] Jakob Nielsen and Robert L. Mack. *Usability Inspection Methods.* John Wiley, 1994.

[93] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *Proceedings of Conference on Human Factors in Computing Systems (CHI90)*, pages 249–256. ACM, 1990.

[94] Jakob Nielsen and Kara Pernice. *Eyetracking Web Usability.* New Riders Press, 2009.

[95] Kenton O'Hara, April Slayden Mitchell, and Alex Vorbau. Consuming video on mobile devices. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 857–866, New York, NY, USA, 2007. ACM.

[96] William F. Opdyke and Ralph E. Johnson. Refactoring object-oriented frameworks. Technical report, 1992.

[97] Jeff Patton. Hitting the target: adding interaction design to agile software development. In *OOPSLA 2002 Practitioners Reports*, Seattle, Washington, 2002. ACM.

[98] Stefan Propp, Gregor Buchholz, and Peter Forbrig. Integration of usability evaluation and model-based software development. *Adv. Eng. Softw.*, 40(12):1223–1230, 2009.

[99] Jeffrey Rubin and Theresa Hudson. *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests.* John Wiley & Sons, Inc., New York, NY, USA, 1994.

[100] Barbara Schmidt-Belz and Matt Jones. Mobile usage of video and tv. In *MobileHCI '06: Proceedings of the 8th Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 291–292, New York, NY, USA, 2006. ACM.

[101] Ken Schwaber. *Agile Project Management With Scrum.* Microsoft Press, 2004.

[102] Reinhard Sefelin, Manfred Tscheligi, and Verena Giller. Paper prototyping - what is it good for?: A comparison of paper- and computer-based low-fidelity prototyping. In *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*, pages 778–779, New York, NY, USA, 2003. ACM Press.

[103] Ahmed Seffah, Mohammad Donyaee, Rex Bryan Kline, and Harkirat Kaur Padda. Usability measurement and metrics: A consolidated model. *Software Quality Journal*, 14(2):159–178, 2006.

[104] Ahmed Seffah, Jan Gulliksen, and Michel Desmarais. An introduction to human-centered software engineering: Integrating usability in the development process. In *Human-Centered Software Engineering Integrating Usability in the Software Development Lifecycle*, pages 3–14. Springer Netherlands, 2005.

[105] Selenium Project. Selenium 2.0 and WebDriver. http://seleniumhq.org/docs/09_webdriver.html. Last visit: 30.04.2010.

[106] Helen Sharp and Hugh Robinson. An ethnographic study of XP practice. *Empirical Software Engineering*, 9(4):353–375, 2004.

[107] Carolyn Snyder. *Paper Prototyping.* Morgan Kaufmann, 2003.

[108] S.R. Subramanya and Byung K. Yi. User interfaces for mobile content. *IEEE Computer*, 39(4):85–87, April 2006.

[109] Michael Tamm. Fighting layout bugs. http://code.google.com/p/fighting-layout-bugs/. Last visit: 30.04.2010.

[110] Sakari Tamminen, Antti Oulasvirta, Kalle Toiskallio, and Anu Kankainen. Understanding mobile contexts. *Personal Ubiquitous Computing*, 8(2):135–143, 2004.

[111] Bjornar Tessem. Experiences in learning XP practices: A qualitative study. In *XP*, pages 131–137, 2003.

[112] Harold Thimbleby. *Press On — Principles of Interaction Programming.* MIT Press, 2007.

[113] Paolo Tonella and Filippo Ricca. Dynamic model extraction and statistical analysis of web applications. In *WSE '02: Proceedings of the Fourth International Workshop on Web Site Evolution (WSE'02)*, page 43, Washington, DC, USA, 2002. IEEE Computer Society.

[114] Paolo Tonella and Filippo Ricca. Dynamic model extraction and statistical analysis of web applications: Follow-up after 6 years. In *10th International Symposium on Web Site Evolution*, pages 3–10. IEEE Computer Society, 2008.

[115] Anders Toxboe. Introducing user-centered design to extreme programming. http://blog.anderstoxboe.com/uploads/16082005_XP_and_UCD.pdf, May 2005. Last visit: 30.03.2008.

[116] Alexandre N. Tuch, Javier A. Bargas-Avila, Klaus Opwis, and Frank H. Wilhelm. Visual complexity of websites: Effects on users' experience, physiology, performance, and memory. *Int. J. Hum.-Comput. Stud.*, 67(9):703–715, 2009.

[117] TVEyes. Tveyes. http://www.tveyes.com. Last visit: 01.11.2007.

[118] Usability.gov. Step-by-step usability guide. http://www.usability.gov/. Last visit: 18.08.2008.

[119] Jean Vanderdonckt, Abdo Beirekdar, and Monique Noirhomme-Fraiture. Automated evaluation of web usability and accessibility by guideline review. In *Lecture Notes in Computer Science, Volume 3140/2004*, page 762, Berlin / Heidelberg, 2004. Springer.

[120] W3C. Notes on user centred design process (UCD). http://www.w3.org/WAI/EO/2003/ucd, April 2004. Last visit: 19.01.2009.

[121] Laurie Williams, Lucas Layman, and William Krebs. Extreme programming evaluation framework for object-oriented languages version 1.4. Technical report, North Carolina State University, Department of Computer Science, June 2004.

[122] Peter Wolkerstorfer, Manfred Tscheligi, Reinhard Sefelin, Harald Milchrahm, Zahid Hussain, Martin Lechner, and Sara Shahzad. Probing an agile usability process. In *CHI '08: CHI '08 Extended Abstracts on Human Factors in Computing Systems*, pages 2151–2158, New York, NY, USA, 2008. ACM.

[123] World Wide Web Consortium. W3C open source software. http://www.w3.org/Status.html. Last visit: 30.04.2010.

[124] World Wide Web Consortium. Web accessibility evaluation tools. http://www.w3.org/WAI/ER/tools/. Last visit: 30.04.2010.

[125] World Wide Web Consortium. Web content accessibility guidelines (WCAG) 2.0. http://www.w3.org/TR/WCAG20/. Last visit: 2.10.2010.

[126] XPlanner. Xplanner: (XP) project planning and tracking tool. http://www.xplanner.org/. Last visit: 04.01.2008.

[127] Mobile YouTube. Mobile youtube. http://m.youtube.com. Last visit: 01.11.2007.

Acknowledgments

First of all, I would like to thank my parents and my girlfriend Fanny for the support they provided during the writing of this thesis.

I thank my supervisor, Professor Dr. Wolfgang Slany, for the help and encouragement he offered during my research.

Also, I owe gratitude to my external supervisor, Professor Dr. Manfred Tscheligi, for his support and guidance.

Many thanks to my colleagues at the Institute for Software Technology in Graz and at the Center for Usability Research and Engineering in Vienna.

Martin Umgeher

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am ……………………………   ………………………………………………..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

……………………………   ………………………………………………..
date   (signature)