Proceedings of the International Workshop on:

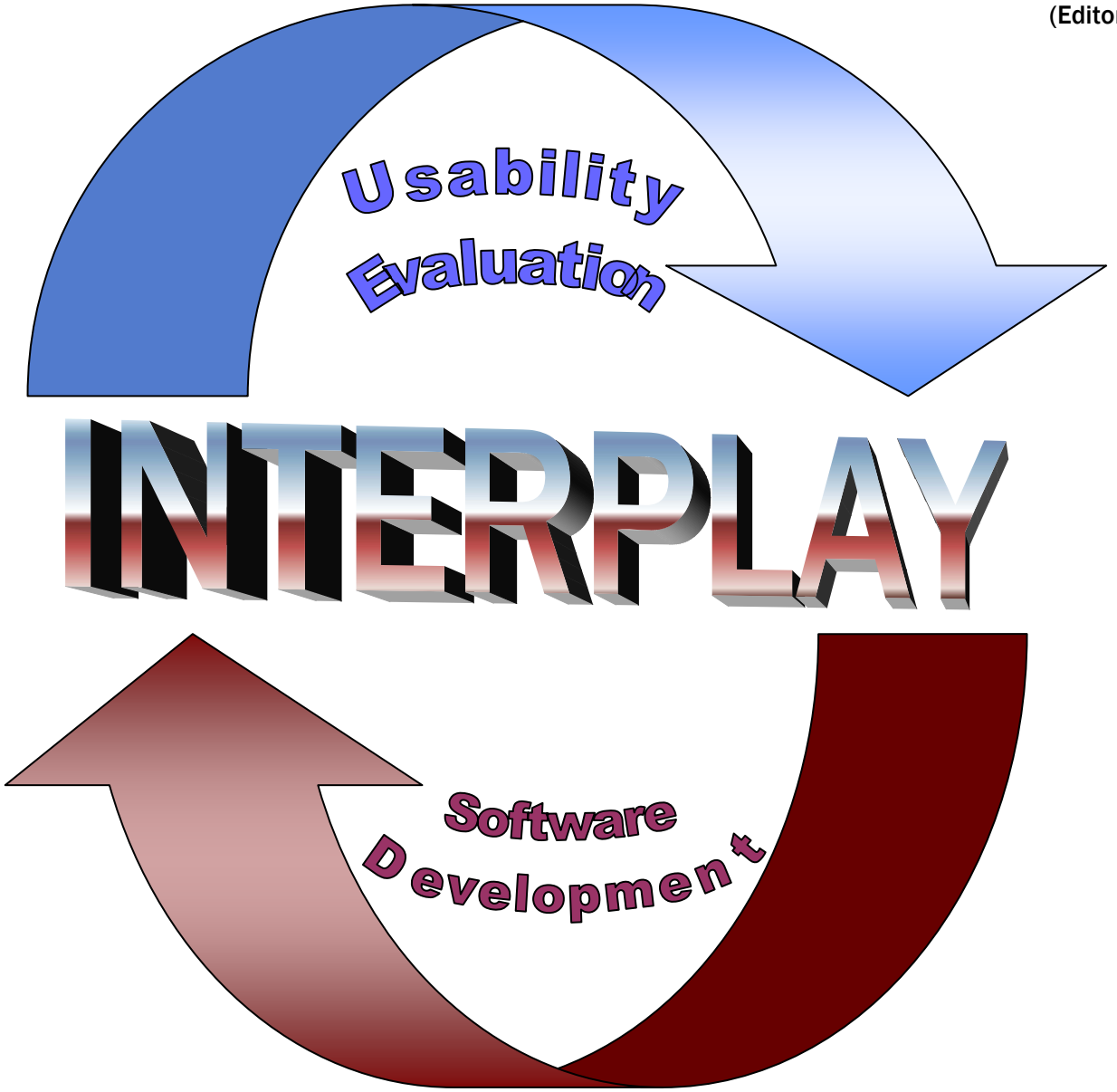
Interplay between Usability Evaluation and Software Development (I-USED 2008)

http://cost294.org/

24th September 2008, ISTI- CNR, Pisa, Italy

co-located with the 2nd Conference on Human-Centred Software Engineering (HCSE 2008), September 25-26, 2008

Silvia Abrahão
Effie Lai-Chong Law
Jan Stage
Kasper Hornbæk
Natalia Juristo
(Editors)



Title: Proceedings of the International Workshop on Interplay between Usability Evaluation and Software Development (I-USED 2008)

Editors: Silvia Abrahão, Effie L-C Law, Jan Stage, Kasper Hornbæk, and Natalia Juristo

ISSN: 1613-0073

The CEUR Workshop Proceedings Series
(http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/)

Printed by: Audio Visual Services (AVS), University of Leicester, UK

ACKNOWLEDGEMENTS

First of all, we are very grateful to the local organizers – **ISTI** (Institute of Information Science and Technologies)**, CNR** (Consiglio Nazionale delle Ricerche [the National Research Council]), Pisa, Italy, especially **Carmen Santoro** and **Fabio Paternò**, who have strongly supported us to hold our 6th COST294-MAUSE Open Workshop **"Interplay between Usability Evaluation and Software Development (I-USED 2008)"** (http://www.dsic.upv.es/workshops/i-used/). Thanks must also go to the authors of the workshop's papers, whose contributions serve as rich sources of stimulation and inspiration to explore the issues of interest from multiple perspectives. The quality of the contributions could further be ensured and improved with the generous help of the program committee members and additional reviewers. Their effective and efficient review works are highly appreciated.

Last but not least, we express gratitude to our sponsor – COST (European Cooperation in the field of Scientific and Technical Research; http://cost.cordis.lu/src/home.cfm). The COST Office operated by the European Science Foundation (ESF) provides scientific, financial and administrative support to COST Actions. Specifically, the COST Action 294 (http://www.cost294.org), which is also known as MAUSE, was officially launched in January 2005. The ultimate goal of COST294-MAUSE is to bring more science to bear on Usability Evaluation Methods (UEM) development, evaluation, and comparison, aiming for results that can be transferred to industry and educators, thus leading to increased competitiveness of European industry and benefit to the public. The current Workshop is the second open workshop implemented under the auspices of COST294-MAUSE. As with other past and forthcoming events of COST294-MAUSE, we aim to provide the participants with enlightening environments to further deepen and broaden their expertise and experiences in the area of usability.

PREFACE

International Workshop on Usability Evaluation and Software Development (I-USED 2008)
24 September 2008, Pisa, Italy

MOTIVATION

Software development is highly challenging. Despite many significant successes, several software development projects fail completely or produce software with serious limitations, including (1) lack of usefulness, i.e. the system does not adequately support the core tasks of the user, (2) unsuitable designs of user interactions and interfaces, (3) lack of productivity gains or even reduced productivity despite heavy investments in information.

Broadly speaking, two approaches have been taken to address these limitations. The first approach is to employ evaluation activities in a software development project in order to determine and improve the usability of the software, i.e. the effectiveness, efficiency and satisfaction with which users achieve their goals. To help software developers' work with usability within this approach, more than 20 years of research in Human-Computer Interaction (HCI) has created and compared techniques for evaluating usability. The second approach is based on the significant advances in techniques and methodologies for user interface design, which have been achieved in the last decades. In particular, researchers in user interface design have worked on improving the usefulness of information technology by focusing on a deeper understanding on how to extract and understand user needs. Their results today constitute the areas of participatory design and user-centered design.

In addition, the Software Engineering (SE) community has recognized that usability does not only affect the design of user interfaces but the software system development as a whole. In particular, efforts are focused on explaining the implications of usability for requirements gathering, software architecture design, and the selection of software components.

However, the interplay between these two fields and between the activities they advocate to be undertaken in software development, have been limited. Integrating usability evaluation at relevant points in software development (and in particular to the user interface design) with successful and to-the-point results has proved difficult. In addition, research in Human-Computer Interaction (HCI) and Software Engineering (SE) has been done mainly independently of each other with no in substantial exchange of results and sparse efforts to combine the techniques of the two approaches. Larry Constantine, a prominent software development researcher, and his colleagues express it this way: "Integrating usability into the software development process is not easy or obvious" (Juristo et al. 2001, p. 21).

THEME AND GOALS

The goal of this workshop is to bring together researchers and practitioners from the HCI and SE fields to determine the state-of-the-art in the interplay between usability evaluation and software development and to generate ideas for new and improved relations between these activities. The aim is to base the determination of the current state on empirical studies. Presentations of new ideas on how to improve the interplay between HCI & SE to the design of usable software systems should also be based on empirical studies. Within this focus, topics of discussion include, but are not limited to:

- Which artifacts of software development are useful as the basis for usability evaluations?
- How do the specific artifacts obtained during software development influence the techniques that are relevant for the usability evaluation?
- In which forms are the results of usability evaluations supplied back into software development (including the UI design)?
- What are the characteristics of usability evaluation results that are needed in software development?
- Do existing usability evaluation methods deliver the results that are needed in user interface design?
- How can usability evaluation be integrated more directly in user interface design?
- How can usability evaluation methods be applied in emerging techniques for user interface design?
- How can usability evaluation methods be integrated to novel approaches for software development (e.g., model-driven development, agile development).

RELEVANCE TO THE FIELD

The main contribution is the determination of state-of-the-art and the identification of areas for improvement and further research. The HCI field includes a rich variety of techniques for either usability evaluation or user interface design. But there are very few methodological guidelines for the interplay between these key activities; and more important, there are few guidelines on how to properly integrate these two activities in a software development process.

PARTICIPANTS

The authors of 15 accepted papers come from eight European countries (Denmark, Finland, Germany, Italy, Norway, Spain, Sweden, and UK) as well as from India and the USA. The workshop brings together these authors with diverse cultural and academic backgrounds and research interests to explore a very relevant topic in HCI from different perspectives. Discussions in the workshop will be very stimulating. Emerging issues thus identified will be addressed in the future work.

WORKSHOP WEBSITE

http://www.dsic.upv.es/workshops/i-used

Webmaster: Adrián Fernández, Universidad Politécnica de Valencia, Spain.

WORKSHOP CO-CHAIRS

Silvia Abrahao, Universidad Politécnica de Valencia, Spain

Jan Stage, Aalborg University, Denmark

Kasper Hornbæk, University of Copenhagen, Denmark

Natalia Juristo, Universidad Politécnica de Madrid, Spain

Effie L-C Law, ETH Zürich, Switzerland & University of Leicester, UK

PROGRAM COMMITTEE

- Scott Ambler, IBM Rational
- Nigel Bevan, Professional Usability Services, UK
- Cristina Cachero, Universidad de Alicante, Spain
- Tiziana Catarci, Università degli Studi di Roma 'La Sapienza', Italy
- Xavier Ferre, Universidad Politecnica de Madrid, Spain
- Maria Francesca Costabile, Universita' degli Studi di Bari, Italy
- Morten Hertzum, Roskilde University, Denmark
- Emilio Insfran, Universidad Politécnica de Valencia, Spain
- Nuno Jardim Nunes, University of Madeira, Portugal
- Maristella Matera, Politecnico di Milano, Italy
- Emilia Mendes, University of Auckland, New Zealand
- Philippe Palanque, IRIT, France
- Fabio Paternò, ISTI-CNR, Italy
- Isidro Ramos, Universidad Politécnica de Valencia, Spain
- Ahmed Seffah, Université Concordia, Montreal, Canada
- Jean Vanderdonckt, Université catolique de Louvain, Belgium

ADDITIONAL REVIEWERS

- Marianella Aveledo, Universidad Politécnica de Madrid, Spain
- Laura Carvajal, Universidad Politécnica de Madrid, Spain
- Rosa Lanzilotti, Universita' degli Studi di Bari, Italy
- Tobias Uldall-Espersen, University of Copenhagen, Denmark

SPONSORS

The workshop is mainly sponsored by the European COST Action n°294 MAUSE (Towards the Maturation of IT Usability Evaluation – www.cost294.org). Several members of this COST action are members of the workshop Program Committee and guarantee a large geographical and topical coverage of the workshop.

TABLE OF CONTENTS

Theme and Variation: Usability in a Post-Waterfall World

Larry Constantine
Department of Mathematics and Engineering
University of Madeira
Campus Universitário da Penteada
9000-390 Funchal, Portugal
Phone: +351 291705281
Fax: +351 291705199
lconstantine@uma.pt

ABSTRACT

The shortening of product life cycles, the advent of rapid iterative techniques, and the rise of agile methods all strain conventional approaches to usability evaluation and interaction design. To function effectively in this changing context, professionals must go beyond time-and-resource intensive conventional methods, such as elaborate ethnographic inquiry, full upfront design, and in-depth user testing to broaden their perspective and practices. This keynote will challenge conventional thinking about usability evaluation and design and consider the role of a range of streamlined approaches that might better fit modern design and development processes. These include model-driven inquiry, small-N and single-subject user testing, design metrics and predictive measures, and usability inspections and peer reviews.

BIOGRAPHY

Larry Constantine, IDSA, is a Professor in the Department of Mathematics and Engineering and Director of the Laboratory for Usage-centered Software Engineering at the University of Madeira, Portugal. An ACM Fellow recognized for his contributions to software design, he is regarded as one of the pioneers of modern software engineering theory and practice. An award-winning designer specializing in interaction design and techniques for enhancing user performance in safety-critical applications, he is a persistent innovator with a number of patents in human-machine interaction to his credit. His publications include more than 175 articles and papers and 17 books in both the human sciences and computer sciences. His papers have been widely reprinted, and his books have been translated into nine languages. He has taught in 20 countries and his clients have included leading technology companies throughout the world.

The Effect of Severity Ratings on Software Developers' Priority of Usability Inspection Results

Asbjørn Følstad
SINTEF ICT
Forskningsveien 1
0314, Oslo, Norway
+47 22067515
asf@sintef.no

ABSTRACT

Knowledge of the factors that affect developers' priority of usability evaluation results is important in order to improve the interplay between usability evaluation and software development. In the presented study, the effect of usability inspection severity ratings on the developers' priority of evaluation results was investigated. The usability inspection results with higher severity ratings were associated with higher developer priority. This result contradicts Sawyer et al. [7], but is in line with Law's [5, 6] finding related to the impact of user test results. The findings serve as a reminder for HCI professionals to focus on high severity issues.

Categories and Subject Descriptors

H5.m.Information interfaces and presentation (e.g., HCI): Miscellaneous.

Keywords

Usability evaluation, usability inspection, developers' priority, impact, severity ratings.

1. INTRODUCTION

One important indicator of successful interplay between usability evaluation and software development is the extent to which evaluation results are associated with subsequent changes in the system under development. This indicator, termed the "impact" [7] or "persuasive power" [4] of usability evaluation results, may reflect whether or not a usability evaluation has generated results that are needed in the development process.

Problem severity is a characteristic of usability evaluation results that has been suggested to affect the impact of usability evaluation results. There is, however, divergence in the literature regarding the actual effect of severity ratings on developers' prioritizing of usability evaluation results. Sawyer et al.'s [7] study of the impact of usability inspection results indicated that usability inspectors' severity ratings had no effect on the impact of the evaluation results; reported impact ratios were 72% (low severity issues), 71% (medium severity issues), 72% (high severity issues). In contrast to this finding Law [5, 6], in a study of the impact of user tests, reported a tendency towards higher severity results having higher impact; reported impact ratios were 26% (minor problems), 42% (moderate problems), 47% (severe problems). Law's findings, however, were not statistically significant [5]. Hertzum [3] suggested that the effect of severity classifications may change across the development process, e.g. high severity evaluation results may have relatively higher impact in later phases of development. Law's study was conducted relatively late in the development process, on the running prototype of a digital library. Sawyer et al. did not report in which development phases their usability inspections were conducted.

In order to complement the existing research on the effect of severity ratings on the impact of evaluation results, an empirical study of the impact of usability inspection results is presented. The data of the present study was collected as part of a larger study reported by Følstad [2], but the results discussed below have not previously been presented.

I-USED'08, September 24, 2008, Pisa, Italy

2. RESEARCH PROBLEM AND HYPOTHESIS

The research problem of the present study was formulated as:

What is the effect of usability inspectors' severity ratings on developers' priority of usability inspection results?

The null hypothesis of the study (no effect of severity ratings) followed the findings of Sawyer et al., and the alternative hypothesis (H1) was formulated in line with the findings presented by Law:

H1: High severity issues will tend to be prioritized higher by developers than low severity issues.

3. METHOD

Usability inspections were conducted as group-based expert walkthroughs [1]. The objects of evaluation were three mobile work-support systems for medical personnel at hospitals, politicians and political advisors, and parking wardens respectively. All systems were in late phases of development, running prototypes close to market. The usability inspectors were 13 HCI professionals, all with >1 year work experience (*Mdn*=5

years)[1]. Each inspector participated in one of three evaluation groups, one group for each object of evaluation. The walkthroughs were conducted as two-stage processes where (1) the individual evaluators noted down usability issues (usability problems and change suggestions) and (2) a common set of usability issues were agreed on in the group. All usability issues were to be classified as either *Critical* (will probably stop typical users in using the application to solve the task), *Serious* (will probably cause serious delay for typical users …), or *Cosmetic* (will probably cause minor delay …). The output of the usability inspections was one report for each object of evaluation, delivered to each of the three development teams respectively.

Three months after the evaluation reports had been delivered individual interviews were conducted with development team representatives. The representatives were requested to prioritize all usability issues according to the following: *High* (change has already been done, or will be done no later than six months after receiving the evaluation report), *Medium* (change is relevant but will not be prioritized the first six months), *Low* (change will not be prioritized), *Wrong* (the item is perceived by the developer to be a misjudgment). In order to align the resulting developers' priorities with the impact ratio definitions of Law and Sawyer et al., the priority *High* was recoded as "Change", and the priorities *Medium, Low and Wrong* were recoded as "No change".

4. RESULTS

The evaluation groups generated totally 167 usability issues. The three objects of evaluation were associated with 44, 61, and 62 usability issues respectively. The total impact ratio (number of issues associated with change/total number of issues [following 7 and 6]) was 27%, which is relatively low. The relationship between the developers' priorities and the usability inspectors' severity ratings is presented in Table 1.

Table 1. Usability issues distributed across developers' priorities and usability inspectors' severity ratings

	Not Classified	Cosmetic	Serious	Critical
Change	6	9	18	12
No change	46	31	26	16
Impact ratio	12%	23%	41%	43%

Visual inspection of Table 1 shows a tendency towards higher priority given to usability issues with severity ratings serious and critical. A Pearson Chi-Square test showed statistically significant differences in priority between severity rating groups; X^2=14.446, df=3, p(one-sided)=.001.

5. DISCUSSION

The presented results indicate that severity ratings may have significant impact on developers' priority of results from usability inspections. This finding contributes to our understanding of severity ratings as a characteristic of usability evaluation results that may help to identify which usability evaluation results that are needed in software development.

The finding is particularly interesting since it contradicts the conclusions of Sawyer et al. and therefore may provoke necessary rethinking regarding usability inspectors ability to provide severity assessments that are useful to software engineers.

It is also interesting to note that the results are fully in line with Law's findings related to severity ratings of user test results. The present study may thus serve to strengthen Law's conclusions. Curiously, the impact ratios of the different severity levels in Law's study and the present study are close to being identical.

Why, then, do the present study indicate that the severity ratings of usability inspection results may have an effect on the developers' priority, when Sawyer et al. did not find a similar effect? One reason may be the relatively high impact ratios reported by Sawyer et al., something that may well result in a greater proportion of low severity issues being prioritized. Another reason may be that the present study, as the study of Law, favored high severity evaluation results since the usability evaluations were conducted relatively late in the development process [cf. 3]. Sawyer et al. do not state which development phases their usability inspections were associated with, but their relatively high impact ratios suggest that their inspections possibly may have been conducted in earlier project phases.

The present study, as the study of Law, indicates that the identification of a low severity usability issue typically is of less value to software developers than the identification of a high severity issue. This should serve as a reminder for HCI professionals to spend evaluation resources on identification and communication of higher severity usability issues.

6. ACKNOWLEDGMENTS

This paper has been written as part of the RECORD project, supported by the VERDIKT program of the Norwegian Research Council.

7. REFERENCES

[1] Følstad, A. 2007. Group-based Expert Walkthrough. In: D. Scapin, and E.L.-C. Law, Eds. R3UEMs: Review, Report and Refine Usability Evaluation Methods. Proceedings of the 3rd. COST294-MAUSE International Workshop, 58-60.

[2] Følstad, A. 2007. Work-Domain Experts as Evaluators: Usability Inspection of Domain-Specific Work-Support Systems. International Journal of Human-Computer Interaction 22(3), 217-245.

[3] Hertzum, M. 2007. Problem Prioritization in Usability Evaluation: From Severity Assessments Toward Impact on Design. International Journal of Human-Computer Interaction, 21(2), 125–146.

[4] John, B.E., and Marks, S.J. 1997. Tracking the effectiveness of usability evaluation methods. Behaviour & Information Technology, 16, 188–202.

[5] Law, E. L.-C. 2004. A Multi-Perspective Approach to Tracking the Effectiveness of User Tests: A Case Study. In Proceedings of the NordiCHI Workshop on Improving the

[1] The study reported by Følstad also included separate evaluation groups with work-domain experts. The results of these groups were not included in the current study, in order to make a clear-cut comparison with the findings of Law and Sawyer et al.

Interplay Between Usability Evaluation and User Interface Design, K. Hornbæk, and J. Stage, Eds. University of Aalborg, HCI Lab Report no. 2004/2, 36-40.

[6] Law, E. L.-C. 2006. Evaluating the Downstream Utility of User Tests and Examining the Developer Effect: A Case Study. International Journal of Human-Computer Interaction, 21(2), 147-172.

[7] Sawyer, P., Flanders, A., Wixon, D. 1996. Making a Difference - The Impact of Inspections. In Proceedings of the CHI'96 Conference on Human Factors in Computing Systems, 376–382.

Usability Promotion in a Technical Project with Sparse Resources – a Case Study

Kaarina Karppinen
VTT Technical Research Centre of Finland
Kaitoväylä 1, PO Box 1100
FI-90571 Oulu, Finland
+358 40 5487 058
kaarina.karppinen@vtt.fi

Marja Liinasuo
VTT Technical Research Centre of Finland
Vuorimiehentie 3, Espoo, P.O. Box 1000
FI-02044 VTT, Finland
+358 400 912711
marja.liinasuo@vtt.fi

ABSTRACT
In this paper, we describe how the usability of software functionalities are promoted and evaluated during the design phase of a software project developing security-related functionalities in a middleware. The paper describes our work-in-progress in GEMOM project, challenges faced in the beginning of the project, and our plan to overcome those challenges with a clearly defined usability implementation plan.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management – *Software quality assurance (SQA).*

General Terms
Design, Human Factors

Keywords
software design, usability, scenarios, acceptability

1. INTRODUCTION
The usability of a software product is becoming a widely recognised quality attribute in software development [1]. However, the conception of usability realised in projects is often quite narrow and, pertaining software design, restricted to attributes that are conceived as becoming topical to produce only in the later phases of software development. Anyhow, no effective product can be designed without taking into account also the "soft" human and context-related complexities, broadly speaking human factors, already in the beginning of the system development.

Furthermore, in the Human-Computer Interaction (HCI) community, software usability has primarily been concerned with the presentation of information, more precisely with user interface [2]. User interface can denote the visible part of the system and, less frequently, the interaction part of the system, i.e., the coordination of the information exchange between the end user and the system in both directions [3]. Either way, the easily neglected fact is that the usability of the tool is not only about the interface but also depends on other attributes rooted deeper in the character of the tool, e.g. the tasks it performs.

Middleware is a specific type of computer software that connects various software components or applications together. Typically other applications are conceived as users of a middleware, and it has no direct human users, except for systems specialists or the like who usually install and maintain complex IT systems. Hence, usability in a middleware system development is harder to promote than of applications which have a direct interface towards a human user. As a consequence, the lack of "proper" users and the tradition of software development methodologies, which may see a user only as a means to elicit requirements, easily results in software development without any usability perspective. In the ongoing GEMOM project these obstacles are planned to be prevailed and one of the project's aims is better usability of the end product without risking the security of it; a task that is proved to be hard to perform [4].

2. OVERCOMING THE CHALLENGES FOR USABILITY
GEMOM (Genetic Message Oriented Secure Middleware) is a recently launched research project, lasting for 2½ years, that is co-funded by the European Commission and involves ten industry and research partners across Europe [5]. GEMOM is developing a prototype of a secure, self-organizing and resilient messaging platform, which enables reliable message sourcing and delivery in applications.

In GEMOM, five case studies, where the new PS-MOM (Publish-Subscribe variant of Message Oriented Middleware) will be used, are defined. Each case study represents a different application area with diverse demands on security and usability; hence, no common definitions can be produced.

2.1 Defining the Challenges
The challenges concerning usability promotion within the project included several issues: usability was to be promoted in a deeply technical project and among technically oriented project members; the task for usability was not clearly defined; the focus of the work, which is the development of a middleware, lacked direct end users; and additionally, a very limited amount of person months were allocated for human factor studies, thus excluding the possibility of a usability

I-USED'08, September 24, 2008, Pisa, Italy

specialist to e.g. interview or lead workshops in various countries by herself.

This working context resulted in two practical questions, involving also some matters of principle, with no direct answer for the usability expert participating in a pre-defined project. Firstly, how to motivate usability studies in a project without direct end users? Secondly, how to perform usability studies with sparse resources?

2.2 Creating Motivation

The first problem to be solved was the motivation for usability studies, related with the problem of having no direct end users for a middleware. Hence, the eventual end users as well as the outline of a plan for usability promotion had to be defined.

In order to clarify the definition of a user in our project, we started by creating a more detailed picture about the various users. The preliminary version of users was based on the usage distance between the user and the middleware. Three levels of users were found: (1) users that were provided some IT-related service, being the furthest away from the middleware; (2) users that provided the service in question; and finally (3) users that maintained the software providing the service, including the middleware, and thus being situated closest to the system.

The predefined project plan stated that user acceptance shall be obtained with the help of scenarios; the new technical solutions would be interpreted into scenarios of usage, which would then be evaluated with the users. This way user acceptance, i.e. the worth of the technical solutions planned to be realised, as experienced by the future users, could be found out. With no user interface to evaluate, a reasonable choice was to concentrate on the functionalities of the middleware as seen by the human user. This choice was also meaningful regarding the method chosen, as it is easier to describe verbally the chain of events than the attributes of a user interface.

2.3 Overcoming the Lack of Resources

The other problem, sparse resources for usability studies, could be compensated by harnessing technical experts to assist in usability evaluation. Consequently, usability study had to be planned extremely carefully as no prior knowledge of usability could be expected from other project members. In this project, case studies provide the human users for usability studies. The usability expert acts as a supervisor who plans and analyses the usability implementation and its results. For instance, she instructs the case study leaders to reflect with the user representatives what aspects regarding usability and security are important from the viewpoint of the user in their case study.

3. USABILITY IMPLEMENTATION

The theme throughout the usability plan is to realise it mainly by non-usability experts. Hence, a stepwise approach was chosen. The main idea is to perform usability studies as early in the project as possible so that the studies could have an actual effect on the middleware functionalities perceivable by human users. The process steps described below are accompanied by practical instructions produced by the usability expert so that the tasks in question can be performed.

1. Case study representatives are to define and describe who the users affected by the functioning of the middleware in their case study are.
2. Technical experts are to describe the technical solutions from the perspective of the users, i.e. the effect of the solution as can be perceived by the human users.
3. Leader of each case study is to produce the scenarios with the users. For that purpose, a description about the functionalities from the human point of view is provided.
4. The case study leaders are to send the scenarios to the usability expert who will check their meaningfulness and return the checked and possibly corrected scenarios with focused questions related with each scenario.
5. Users in each case study are to answer the questions, and the answers will be sent to the usability expert who will analyse them and produce a report about user acceptance.

So far, after having finished the first step of the process, challenges have mainly been related with the understanding of terms that have different meanings in HCI and SE (Software Engineering) approaches. Hence, special care has been taken when discussing about users or scenarios in this project. “User” means human users for HCI but may mean applications for SE. “Scenario” in turn denotes short stories describing relatively freely working process from the human user’s viewpoint in HCI [e.g. 6], compared with the system description that is more technically oriented in SE [e.g. 7].

This paper describes a work-in-progress, and more will be learned when the project is progressing.

4. REFERENCES

[1] Abran, A., Khelifi, A., Suryn, W., and Seffah, A. 2003. Usability meanings and interpretations in ISO standards. Journal of Software Quality 11, 325-338.. DOI= http://dx.doi.org/10.1023/A:1025869312943

[2] Rafla, T., Robillard, P.N., and Desmarais, M. 2007. A method to elicit architecturally sensitive usability requirements: its integration in to a software development process. Software Quality Journal, 15(2), 117-133. DOI= http://dx.doi.org/10.1007/s11219-006-9009-9

[3] Ferré, X., Juristo, N., Windl, H., and Constantine, L. 2001. Usability basics for software developers. IEEE Software 18(1), 22-29. DOI= http://dx.doi.org/10.1109/52.903160

[4] Cranor, L., and Garfinkel, S. 2005. Security and Usability. O’Reilly Media, Inc.

[5] GEMOM website (July 10, 2008): http://www.gemom.eu

[6] Go, K., Carroll, J.M., 2004. The Blind Men and the Elephant. Views of Scenario-Based System Design. Interactions 11(6), 44-53. DOI= http://dx.doi.org/10.1145/1029036.1029037

[7] Uchitel, S., Kramer, and J. Magee, J. 2003. Synthesis of Behavioral Models from Scenarios. IEEE Transactions on Software Engineering, 29(2), 99-115. DOI= http://dx.doi.org/10.1109/TSE.2003.1178048

Preparing Usability Supporting Architectural Patterns for Industrial Use

Pia Stoll
ABB Corporate Research
Forskargränd 6
SE 72178 Västerås, Sweden
Tel: +46 21 32 30 00
pia.stoll@se.abb.com

Bonnie E. John, Len Bass, Elspeth Golden
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA, USA 15213
Tel: 1+412 268 2000
bej@cs.cmu.edu, ljb@sei.cmu.edu, egolden@cmu.edu

ABSTRACT
Usability supporting architectural patterns (USAPs) have been shown to provide developers with useful guidance for producing a software architecture design that supports usability in a laboratory setting [7]. In close collaboration between researchers and software developers in the real world, the concepts were proven useful [2]. However, this process does not scale to industrial development efforts. In particular, development teams need to be able to use USAPs while being distributed world-wide. USAPs also must support legacy or already partially-designed architectures, and when using multiple USAPs there could be a potentially overwhelming amount of information given to the software architects. In this paper, we describe the restructuring of USAPs using a pattern language to simplify the development and use of multiple USAPs. We also describe a delivery mechanism that is suitable for industrial-scale adoption of USAPs. The delivery mechanism involves organizing responsibilities into a hierarchy, utilizing a checklist to ensure responsibilities have been considered, and grouping responsibilities in a fashion that both supports use of multiple USAPs simultaneously and also points out reuse possibilities to the architect.

Categories and Subject Descriptors
D.2.2 [**Design Tools and Techniques**] User interfaces; D.2.11 [**Software Architectures**]: Patterns; H.5.2 [**User Interfaces**] Theory and Methods

General Terms
Design, Human Factors.

Keywords
Software Architecture, Usability, Human-Computer Interaction, HCI

I-USED'08, September 24, 2008, Pisa, Italy

1. INTRODUCTION
The Software Engineering community has recognized that usability affects not only the design of user interfaces but software system development as a whole. In particular, efforts are focused on explaining the implications of usability on software architecture design [3, 4, 5, 6, 10].

One effort in this area is to produce artifacts that communicate usability requirements in a form that can be effectively used for software architecture evaluation and design. These *usability supporting architectural patterns* (USAPS) have been shown to improve the ability of software architects to design higher quality architectures to support usability features such as the ability to cancel a long-running command [7, 8]. Other uses of USAPs in industrial settings have been productive [2] but have revealed some problems that prevent scaling USAPs to widespread industrial use. These problems include:

2. Prior efforts have involved personal contact with USAP researchers, either face to face or in telephone conversations. This process does not scale to widespread industrial use.
3. The original USAPs included UML diagrams modifying the MVC architectural pattern to better support the usability concern. Although intended to be illustrative, not prescriptive, software architects using other overarching patterns (e.g., legacy systems, SOA) viewed these UML diagrams as either unrelated to their work or as an unwanted recommendation to totally redesign their architecture.
4. The original use of USAPs was as a collection of individual patterns. Even using one pattern involved processing a large amount of detailed information. Applying multiple USAPs simultaneously is likely to overwhelm software architects with information.

In this paper, we introduce a pattern language [1] for USAPs that reduces the information that architects must absorb and produces information at a level that applies to all architectures. We also discuss the design of a delivery mechanism suitable for industrial-scale adoption of USAPs.

2. BACKGROUND
A USAP has six types of information. We illustrate the types with information from the cancellation USAP [9]

1. A brief scenario that describes the situation that the USAP is intended to solve. For example, "The user issues a command

then changes his or her mind, wanting to stop the operation and return the software to its pre-operation state."

2. A description of the conditions under which the USAP is relevant. For example, "A user is working in a system where the software has long-running commands, i.e., more than one second."
3. A characterization of the user benefits from implementing the USAP. For example, "Cancel reduces the impact of routine user errors (slips) by allowing users to revoke accidental commands and return to their task faster than waiting for the erroneous command to complete."
4. A description of the forces that impact the solution. For example, "No one can predict when the users will want to cancel commands"
5. An implementation-independent description of the solution, i.e., responsibilities of the software. For example, one implication of the force given above is the responsibility that "The software must always listen for the cancel command."
6. A sample solution using UML diagrams. These diagrams were intended to be illustrative, not prescriptive, and are, by necessity, in terms of an overarching architectural pattern (e.g., MVC).

It is useful to distinguish USAPs from other patterns for software design and implementation. USAPs are not user interface patterns, that is, they do not represent an organization or look-and-feel of a user interface [e.g., 11]; they are software architecture patterns that support UI concerns. Nor are USAPs structural software architecture patterns like MVC; they are patterns of software responsibilities that can be applied to any structure. As such, they can be applied to any legacy architecture and can support the functionality called for in UI patterns.

3. A PATTERN LANGUAGE FOR USAPs

Through collaboration among academic and industrial researchers and usability engineers, we are constructing three USAPs for process control and robotics applications. The first author and her colleagues in the industry research team drafted an "Alarms, Events and Alerts" USAP while, independently, the last three authors drafted a "User Profile" USAP and an "Environment Configuration" USAP. When these three USAPs were considered together, it was clear that there was an enormous amount of redundancy in the responsibilities necessary for a good solution. In addition, in preliminary discussions, industry software architects reacted negatively to the large amount of information required to implement any one of the USAPs.

Our early work [4] recognized the possibility of reusing such software tactics as separating authoring from execution and recording (logging), but our subsequent work had not incorporated that notion, treating each USAP as a separate pattern. A consequence of focusing on industrial use is that reuse in constructing and using USAPs was no longer an academic thought experiment, but a necessity if industrial users are to construct and use USAPs themselves.

We observed that both the industry research team and the academic research team independently grouped their responsibilities into very similar categories. This led us to construct a pattern language [2] that defines relationships between USAPs with potentially reusable sets of responsibilities that can lead to potentially reusable code. Our pattern language relating "Alarms, Events and Alerts", "User Profile" and "Environment Configuration" is shown in Figure 1.

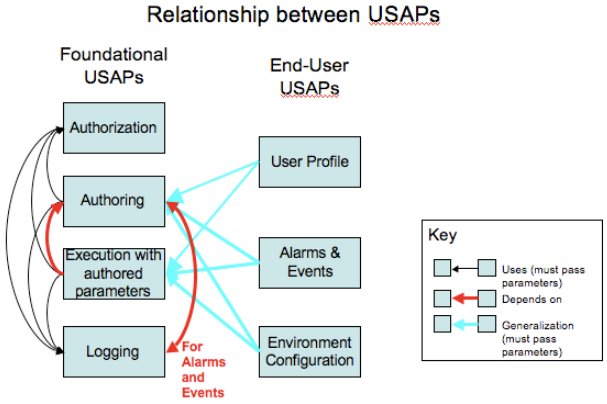


Figure 1 USAP Pattern Language for "User Profile", "Alarms, Events and Alerts", and "Environment Configuration"

The pattern language has two types of USAPs. "End-User USAPs" follow the structure given in Section 2. Their purpose from a user's point of view can be expressed in a small scenario, they have conditions under which they are relevant, benefits for the user can be expressed and they require the fulfillment of software responsibilities in the architecture design. End-User USAPs are used by the requirements team to determine which are applicable to the system being developed. In this example, they are "User Profile", "Alarms, Events and Alerts", and "Environment Configuration".

The pattern language also contains what we are calling "Foundational USAPs". These do not have the same six portions as the End-User USAPS. For example, there is no scenario, no description of conditions, and no benefits to the user for the Foundational USAPs. Rather, they act as a framework to support the construction of the End-User USAPs that make direct contact to user scenarios and usability benefits. For example, all of the End-User USAPs that we present have an authoring portion and an execution portion, that is, they are specializations of the Authoring Foundational USAP and the Execution with Authored Parameters Foundational USAP. These foundational USAPs make use of other foundational USAPs, Authorization and Logging. We abstracted the commonalities of the End-User USAPs to derive the responsibilities of the Foundational USAPs. The responsibilities in the Foundational USAPs are parameterized, where the parameters reflect those aspects of the End-User USAPs that differ.

An example of the parameterization is that the Authoring Foundational USAP and the Execution with Authored Parameters Foundational USAP each have a parameter called SPECIFICATION. The value of SPECIFICATION is "Conditions for Alarm, Event and Alerts", "User profile", and "Configuration description" for the three End-User USAPs, respectively.

In addition to parameterization, End-User USAPs explicitly list assumptions about decisions the development team must make prior to implementing the responsibilities. For example, in the "Alarms, Events and Alerts" End-User USAP, the development team must define the syntax and semantics for the conditions that will trigger alarms, events or alerts. End-User USAPs may also

have additional responsibilities beyond those of the Foundational USAPs they use. For example, the "Alarms, Events and Alerts" End-User USAP has an additional responsibility that the system must have the ability to translate the names/ids of externally generated signals (e.g., from a sensor) into the defined concepts. Both the assumptions and additional responsibilities will differ for the different End-User USAPs.

There are three types of relationships among the Foundational USAPs and these are shown in Figure 1 as different color arrows. The Generalization relationship (turquoise) says that the Foundational USAP is a generalization of part of the End-User USAP. The End-User USAP passes parameters to that Foundational USAP and, if there are any conditionals in the responsibilities of the Foundational USAP, the End-User USAP may define the values of those conditionals. The Uses relationship (black) also passes parameters, but the USAPs are at the same level of abstraction (the foundational level). The Depends-On relationship (red) implies a temporal relationship. For example, the system cannot execute with authored parameters unless those parameters have first been authored. The double headed arrow between authoring and logging reflects the possibility that the items being logged may be authored and the possibility that the identity of the author of some items may be logged.

Foundational USAPs each have a manageable set of responsibilities (Authorization has 11; Authoring, 12; Execution with authored parameters, 9; and Logging 5), as opposed to the 21 responsibilities of the Cancel USAP that seemed to be too much for our experiment participants to absorb in one sitting [7]. These responsibilities are further divided into groups for ease of understanding, e.g., Authoring is separated into Create, Save, Modify, Delete and Exit the authoring system. This division into manageable Foundational USAPs simplifies the creation of future USAPs that use them. For example, the User Profile End-User USAP requires only the definition of parameters and the values for one conditional, and pointers to the Authoring and Execution Foundational USAPs.

4. DELIVERING A SINGLE USAP TO SOFTWARE ARCHITECTS

The roadblocks to widespread use of USAPs in industry identified in the introduction were (1) the need for contact with USAP researchers in the development process, (2) reactions to examples using a particular overarching architectural pattern (MVC) and (3) an overwhelming amount of information delivered to the software architect. Data from our laboratory study and the pattern language outlined above put us in a position to solve these problems.

Our laboratory study [7] showed that a paper-based USAP could be used by software engineers[2] without researcher intervention, to significantly improve their design of an architecture to support the users' need to cancel long-running commands. Although significantly better than without a USAP, these software engineers seemed to disregard many of the responsibilities listed in the USAP in their designs. To enhance attention to all responsibilities, we have chosen to design a web-based system that presents responsibilities in an interactive checklist (Figure 2).

[2] The participants in our lab study had a Masters in SE or IT, were trained in software architecture design, and had an average of over 21 months in industry.

The design includes a set of radio buttons for each responsibility that are initially set to "Not yet considered." The architect reads each responsibility and determines whether it is not applicable to the system being designed, already accounted for in the architecture, or that the architecture must be modified to fulfill the responsibility. If "Not applicable", "Must modify architecture to address this" or "Architecture addresses this" is selected, then the responsibility's checkbox is automatically checked. If "Not considered", "Must modify architecture or "Discuss status of responsibility", is selected, the responsibility will be recorded in To-Do list generated from the website (Figure 3). We expect this lightweight reminder to consider each and every responsibility will not be too much of a burden for the architect, but will increase the coverage of responsibilities, which is correlated with the quality of the architecture solution [8].

As Figure 2 show, the responsibilities are arranged in a hierarchy, which reflects both the relationship of End-User and Foundational USAPs and the internal structure within a Foundational USAP. This hierarchy divides the responsibilities into manageable sub-parts. The checkboxes enforce this structure by automatically checking off a higher-level box when all its children have been checked off, and conversely, not allowing a higher-level box to be checked when one or more of its children are not. Thus, this mechanism simultaneously addresses the problems of providing guidance without intervention by USAP researchers and simplifying the information provided to the software architect.

Another mechanism for simplifying the information delivered to an architect is that each responsibility has additional details available only by request of the architect. These details include more explanation, rationale about the need for the responsibility and the forces that generated it, and some implementation details. This information is easily available, but not "in the face" of the software architect. As well as simplifying the presentation, this mechanism de-emphasizes the role of illustrative examples situated in reference architecture like MVC. We expect that this presentation decision will reduce the negative reactions to generic example UML diagrams. When using the tool in-house in industry, the reference architecture used in example solutions could be changed to an architecture used by that industry. This would both accelerate understanding of the examples and increase the possibility of re-using the sample solution. This presumes that the tool is constantly managed and updated by in-house usability experts and software architects, a presumption facilitated by delivering the examples in separate web pages.

Although the hierarchy of responsibilities reflects the relationship of the End-User USAPs and the Foundational USAPs, the difference between the types of USAPs is not evident in the presentation of responsibilities. It was a deliberate design choice to express each responsibility in terms of the End-User USAP's vocabulary. Thus, the responsibilities in Figure 2 are couched in terms of "User Profile", "Configuration Description", "Conditions for Alarms, Events, and Alerts" and this string replaces the parameter SPECIFICATION in the Foundational Authoring USAP.

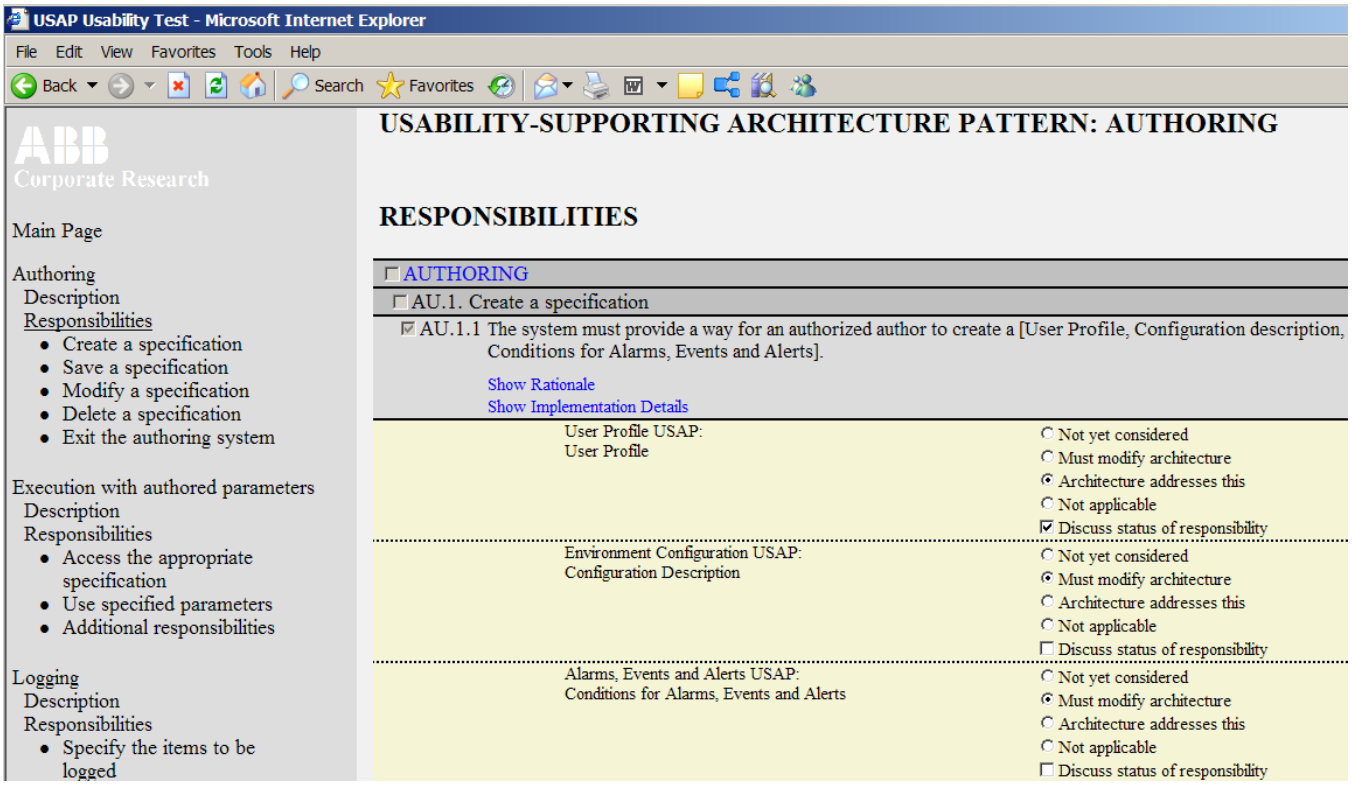


Figure 2: Prototype of a web-based interface for delivering USAP responsibilities to industry software architects.

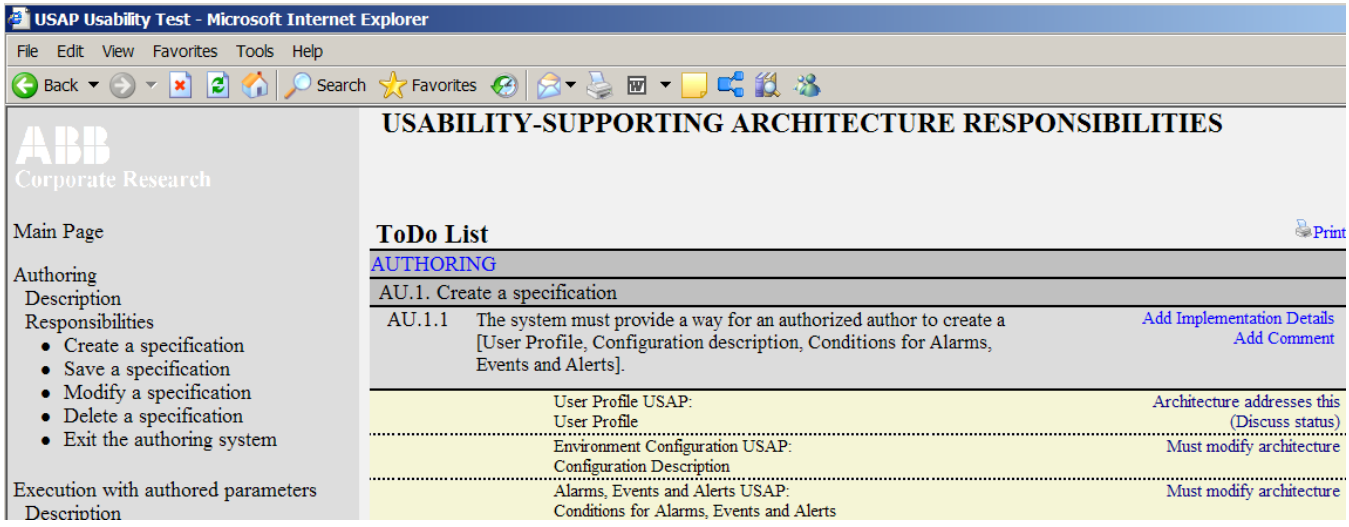


Figure 3: Prototype "to do" list produced from those responsibilities that are marked as requiring architectural modification.

In the next section, we discuss how we anticipate managing the situation when the architect chooses multiple USAPs as being relevant to the system under construction. This will allow distribute architecture teams both to record rationale for their choice and to discuss potential solutions. Attaching design rationale and discussion is optional so our delivery tool will support discussion, but not require it, keeping the tool lightweight.

At any point in the process of considering the different responsibilities, the architect can generate a "to do" list. This is a list of all of the responsibilities that have been checked as "Not yet considered" or "Must modify architecture". See Figure 3 for an example. The list can then be entered into the architect's normal task list and will be considered as other tasks are considered.

Supporting world wide distribution of the architecture team in the use of USAPs has two facets.

- Enable world wide access
- Reduce the problems associated with simultaneous updates by different members of the team.

The use of the World Wide Web for delivery allows world wide access with appropriate access control. Standard browsers support the concept of check lists and producing the "to do" lists.

Allowing simultaneous updates is not supported by standard browsers. Some Wikis do support simultaneous updates, e.g. MediaWiki [www.mediawiki.org], but we do not yet know whether these wikis directly support checklists and the generation of "to do" lists. We are currently investigating which tool or combination of tools will be adequate for our needs and what modifications might have to be made to those tools.

5. DELIVERING MULTIPLE USAPS TO SOFTWARE ARCHITECTS

Our motivation for developing the USAP Pattern Language was partially to simplify the delivery of USAPs when multiple USAPs are relevant to a particular system. We also want to indicate to the architect the possibilities for reuse. In this section, we describe how we anticipate accomplishing these two goals.

Recall that the Foundational USAPs are parameterized and each End User USAP provides a string that is used to replace the parameter. For instance, consider a responsibility from the Authoring Foundational USAP "The system must provide a way for an authorized user to create a SPECIFICATION". When three End User USAPs are relevant to the system under design, such as "User Profile", "Environment Configuration", and "Alarms, Events and Alerts", the three responsibilities are displayed to the architect as "The system must provide a way for an authorized user to create a [User Profile, Configuration description, Conditions for Alarm, Event and Alerts].

This presentation satisfies two goals and introduces one problem. Presenting three responsibilities as one reduces the amount of information displayed to the architect since every Foundational USAP responsibility is displayed only once, albeit with multiple pieces of information. This presentation also indicates to the architect the similarity of these three responsibilities and hence the reuse possibilities of fulfilling them through a single piece of parameterized code.

The problem introduced by this form of the presentation is that now the radio buttons becomes ambiguous. Does the entry "Architecture addresses this" mean that all of the three responsibilities have been addressed or only some of them? We resolve this ambiguity by repeating the radio buttons three times, once for each occurrence of the responsibility. Thus, the three responsibilities will be combined into one textual description of the responsibility but three occurrences of the radio buttons.

6. CURRENT STATUS AND FUTURE WORK

At this writing, we have developed the pattern language for three End User USAPs and four Foundational USAPs (Figure 1) and have fleshed out all the responsibilities for these seven USAPs. We have constructed a prototype delivery tools for a browser based checklist and "to do" list generator.

We plan to test the delivery mechanism in an ongoing industrial development effort. This will demonstrate strengths and weaknesses of our approach and we will iterate to resolve any problems or capitalize on any opportunities. One suggestion put forth in early industry feedback is to enhance the to-do list by assigning expected effort to each responsibility. One requirements engineer at ABB said that her perception of the effort needed to implement a scenario had been thoroughly revised just be looking at the to-do list. By adding estimated hours to the responsibilities, industry would get a better estimate of the usability improvements' translation into software implementation cost. These estimates would vary depending on many factors such as underlying architectural style, implementation language, skill of programmers, etc. but a large organization may have enough data from previous projects to make such estimates for their organization. In addition, such a feature could emphasize the savings realized by reuse; responsibility-implementations that serve multiple End-User USAPs would show up as requiring very little effort after the first implementation.

The delivery platform that we have described here, to be used by software architects, is envisioned to be the final portion of a tool chain. There are two additional roles involved in the development and use of USAPs. First, USAP developers will have to create USAPs within the stylized context of the USAP Pattern Language. Tool support for USAP developers will greatly simplify the creation of USAPs.

The second role is the requirements definers; often a team comprised of technologists and human factors engineers, usability engineers, designers, or other users or user advocates. Figure 4 shows how we envision a tool supporting this role.

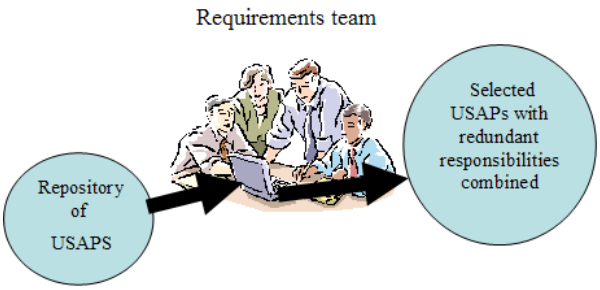


Figure 4: Tool to support the requirements elicitation process

The requirements team has available to them a repository of USAPs. They select the ones that are appropriate for the system being constructed. In our experience, the USAP end-user scenarios are very general and can be used to invoke ideas about how they apply to the system at hand. However, industrial teams would like to tailor these scenarios to match their everyday usability issues. Thus, the tool supporting requirements definers will allow them to re-write the general scenarios to suit their specific application.

The tool then creates input for the delivery tool while simultaneously combining redundant responsibilities. The output of the requirements definition process will then be presented to software architects, as described in this paper, to aid in their architecture design process.

In summary, USAPs have been proven to be useful to software architects but have also demonstrated some problems that hinder industrial use. Definition of a USAP Pattern Language and an appropriate selection of tools supporting the roles involved in the creation and use of USAPs should simplify industrial use. We are currently constructing versions of these tools and testing the extent to which they do, in fact, enable the industrial use of USAPs.

7. ACKNOWLEDGMENTS

We would like to thank Fredrik Alfredsson and Sara Lövemark for their contributions to the "Alarms, Events and Alerts" USAP. This work was supported in part by funds from ABB Inc. The views and conclusions in this paper are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of ABB.

8. REFERENCES

[1] Alexander, C. (1977). *A Pattern Language: Towns, Buildings, Construction*. USA: Oxford University Press. ISBN 978-0195019193.

[2] Adams, R. J., Bass, L., & John, B. E. (2005) Applying general usability scenarios to the design of the software architecture of a collaborative workspace. In A. Seffah, J. Gulliksen and M. Desmarais (Eds.) *Human-Centered Software Engineering: Frameworks for HCI/HCD and Software Engineering Integration.* Kluwer Academic Publishers.

[3] Bass, L., John, B., & J. Kates, (2001), "Achieving Usability through Software Architecture," Technical Report CMU/SEI-2001-TR-005, Software Eng. Inst., Carnegie Mellon Univ., 2001.

[4] Bass, L., & John, B. (2003) "Linking Usability to Software Architecture Patterns through General Scenarios," *The J. Systems and Software*, vol. 66, no. 3, pp. 187-197, 2003.

[5] Folmer, E. (2005) *Software Architecture Analysis of Usability*, Ph.D. thesis. Department of Computer Science, University of Groningen, Groningen.

[6] Folmer, E., van Gurp, J., Bosch, J. (2003) A Framework for capturing the relationship between usability and software architecture; *Software Process: Improvement and Practice,* Volume 8, Issue 2. Pages 67-87. April/June 2003.

[7] Golden, E, John, B. E., & Bass, L. (2005) The value of a usability-supporting architectural pattern in software architecture design: A controlled experiment. *Proceedings of the 27th International Conference on Software Engineering, ICSE 2005* (St. Louis, Missouri, May, 2005).

[8] Golden, E., John, B. E., Bass, L. (2005) Quality vs. quantity: Comparing evaluation methods in a usability-focused software architecture modification task. *Proceedings of the 4th International Symposium on Empirical Software Engineering* (Noosa Heads, Australia, November 17-18 2005).

[9] John, B. E., Bass, L. J., Sanchez-Segura, M-I. & Adams, R. J. (2004) Bringing usability concerns to the design of software architecture. *Proceedings of The 9th IFIP Working Conference on Engineering for Human-Computer Interaction and the 11th International Workshop on Design, Specification and Verification of Interactive Systems*, (Hamburg, Germany, July 11-13, 2004).

[10] Juristo, N., Moreno, A. M., Sanchez-Segura, M. (2007), "Guidelines for Eliciting Usability Functionalities", *IEEE Transactions on Software Engineering, Vol. 33*, No. 11, November 2007, pp. 744-758.

[11] Tidwell, J. (2006), *Designing Interfaces: Patterns for Effective Interaction Design*. O'Reilly Media: Sebastopol, CA.

Enriching Requirements Analysis with the Personas Technique

John W. Castro
Departamento de Ingeniería Informática
Universidad Autónoma de Madrid
Calle Tomás y Valiente 11
28049 Madrid, Spain
john.castro@estudiante.uam.es

Silvia T. Acuña
Departamento de Ingeniería Informática
Universidad Autónoma de Madrid
Calle Tomás y Valiente 11
28049 Madrid, Spain
silvia.acunna@uam.es

Natalia Juristo
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo s/n
28660, Boadilla del Monte
Madrid, Spain
natalia@fi.upm.es

ABSTRACT
A thorough understanding of the users that interact with the system is necessary to develop usable systems. The Personas technique developed by the human-computer interaction (HCI) discipline gathers data about users, gains an understanding of their characteristics, defines fictitious personas based on this understanding and focuses on these personas throughout the software development process. The aim of our research is to build Personas into systems development following software engineering (SE) guidelines. The benefits to be gained are an understanding of the user which is not traditionally taken into account in SE. To do this, we had to undertake two types of tasks. First, we modified the Personas technique to conform to the levels of systematization common in SE. We have called the modified technique PersonaSE. Second, we incorporated the proposed technique into the software requirements analysis process.

Keywords
Personas technique, usability, human-computer interaction, requirements analysis, software process.

1. INTRODUCTION
In recent decades, the HCI community has developed a variety of techniques for improving software systems usability, but these techniques are not very widespread in SE [17]. On the other hand, software developers only receive basic usability training [12] and do not usually have the knowledge they need to build usable software.

I-USED'08, September 24, 2008, Pisa, Italy

Two separate processes for building usable systems —one from SE to develop the system and another from HCI to improve usability— are not easily manageable. Software development and usability design cannot be controlled and synchronized separately. Additionally, the likely overlap of activities across the two processes would reduce efficiency and increase costs. Milewski [15] claims that there are still problems with SE-HCI interactions that require more research. One of the major remaining obstacles to cooperation between HCI and SE is that there is little knowledge and communication about the practices and techniques of HCI in SE and vice versa.

In this research, we propose modifying the HCI technique to assure that it is completely incorporated and assimilated in the SE development process. This step will benefit both disciplines, as it will promote an understanding between the SE and HCI activities and techniques. We have chosen the Personas technique [8] used in the HCI user analysis activity. This technique is useful for gathering, analysing and synthesizing the information related to the users interacting with the software system. Personas helps to focus software analysis and design on the features and goals of the product's end user [7]. Personas are detailed descriptions of fictitious users, stressing their characteristics and goals based on surveys of real end users. The quantitative and qualitative data that are gathered, analysed and synthesized about the users are used as background for designing the personas [10].

We have selected the Personas technique, as, even though it has not been around for long (the first HCI literature citation dates from 1999 [5]), it is a technique used routinely. Additionally, encouraging results have been reported on the use of the Personas technique in quite a few developments [2][11][4][7]. Its use is especially widespread in Web development, although it can be used to design any type of interactive software [5]. One indication of the current impact of personas is that the Microsoft MSN Personas gateway (http://advertising.msn.com/home/MSNPersonas.asp) uses this technique in its marketing strategy to attract advertisers, showing concern about who its users are.

However, the Personas technique does not include a detailed definition of the basic process elements—activity and product—, which would enable its introduction into the SE development process to enrich the requirements analysis activity.

The goal of our work is to analyse the Personas technique and make the modifications required to conform to the levels of systematization and method characteristic of SE. We have called this modification of the Personas technique PersonaSE. These modifications adapt Personas for incorporation and use in the SE development process analysis activity. Finally, we enrich the software process analysis activity by establishing the relationships between the proposed PersonaSE technique activities and the traditional SE requirements analysis activities to enable the software engineer to put Personas into routine use.

This paper has been structured as follows. Section 2 describes the Personas technique. Section 3 presents the analysis of the weaknesses of the latest version of Cooper and colleagues' Personas [8], as well as suggested modifications. Section 4 presents the proposed PersonaSE technique. In section 5, we detail the enrichment of the SE requirements analysis process, discussing the relationships between the PersonaSE and routine SE requirements analysis activities. Finally, section 6 discusses the conclusions.

2. PERSONAS TECHNIQUE

The Personas technique provides an understanding of the system user in terms of his or her characteristics, needs and goals to be able to design and implement a usable system. This method is attributed to A. Cooper [6], who later upgraded the method in [7] and [8]. Several methods for successfully creating personas have been proposed on this basis [10][11][18]. To assure that the design focus is on user considerations, this method does not take into account real users participating in the design process; it creates fictitious users, called personas. These personas specify the target user. The development efforts are focused on these personas. Personas main potential benefit is that it serves the explicit development objective [2].

The Personas technique is based on a survey of users that can be used to tightly couple the key characteristics and goals of the personas to the user data [10][11][7]. When he was working for Cooper Interactive, Goodwin [10] suggested that personas should mainly be based on qualitative data, gathered through interviews and observations. Cooper and Reimann [7] share Goodwin's view and detail the social research methods they recommend. These methods focus on user goals and take into account user domains.

The data gathered from the observations and interviews are mapped to behavioural variables. The mapping does not need to be overly precise. The important thing is for the mapping of different interview subjects to be correct. A number of interview subjects grouped within a set of behavioural variables forms a behavioural model. A behavioural model is the basis of a persona. If detailed data are added to the behavioural model, it becomes a persona. Once personas have been created, they need to be documented and shared with team members. The communication of personas has been recognized as a key factor for software project success [16][1]. In a failed application of the Personas technique, reported by Blomquist and Arvola [3], lack of communication was identified as the main ground for the failure. To prevent this failure, Cooper and Reimann [7] mention two basic deliverables for each created persona: a list of its key characteristics and a third-person narrative of the persona. Cooper and Reimann stress the importance of the persona having a name and a photograph to make it more life like. The narrative should be one to two pages long and should not cover all the observed details, as ideally the team members will have participated in the interview phase, and people outside the team do not need to know the interview details [7]. When the personas have been documented and the materials are finished, a meeting should be arranged with the team of developers to present the personas [16].

3. PERSONAS TECHNIQUE MODIFICATIONS

To be able to build personas into routine software development, the technique needs to conform to the guidelines on systematization and definition of certain elements of the SE software process. More to the point, the technique needs to be defined by its activities and the outputs associated with each activity to be fit into SE. To add these elements to Personas, first we analysed the criticisms of the latest version of Cooper and colleagues' technique [8], proposing improvements that have an impact on such weaknesses. Second, we systematized the decomposition of Personas into activities and defined an output for each activity.

To make all these modifications we selected the latest version of the Personas technique [8], because i) Cooper authored the original proposal, ii) versions by other authors were based on this proposal, and iii) it has been successfully used in different software development projects (see [11][4][18][9]).

Table 1 is an assessment of Cooper and colleagues' Personas [8] with respect to two criteria, *Procedure Definition* and *Product Formalization* and their associated attributes. The attributes of the Procedure Definition criterion are: a) *What does the procedure do*? and b) *How does the procedure work?* Criterion a) evaluates how well the technique defines what a step should do (the possible values are Implicit, Semi-Explicit and Explicit). Criterion b) evaluates how well the technique defines what techniques and procedures should be used to perform a step (the possible values are Undefined, Semi-Defined and Defined). The Product Formalization criterion also has two attributes: a) *Product Content* (the possible values are Undefined, Semi-Defined and Defined); and b) *Product Structure* (the possible values are Informal, Semi-Formal and Formal).

Table 1 is a summary of the values of the characteristics assigned to each step of the Personas technique [8] for each analysed criterion. As Table 1 shows, *What does the procedure do?* is the only attribute that takes the *explicit* value for almost all the steps of the Personas procedure, i.e. the procedure is declarative and indicates what to do in most steps. Looking at the *How does the procedure work*? attribute, we find that over 70% of the steps of the Personas technique take the value of either *undefined* or *semi-defined*. Therefore, this procedural attribute is not completely defined in most of the Personas steps. The *Product Content* attribute takes the value of *undefined* and *semi-defined* in over 70% of the Personas technique steps, reflecting, like the last attribute, weaknesses in this respect. *Product Structure* is the worst rated attribute, as almost 60% of the Personas technique steps are given the poorest rating, *informal*, for this attribute, and none of the steps have a formally defined product structure. This is evidence that the Product Formalization criterion needs more modification. Also, changes need to be made to how each

Personas step is carried out in order to reach the levels of systematization demanded by SE.

Table 1. Summary of the Assessment of the Personas Technique

CRITERION / CHARACTERISTIC / STEPS OF THE PERSONAS TECHNIQUE [8]	PROCEDURE DEFINITION		PRODUCT FORMALIZATION	
	What?	How?	Product Content	Product Structure
Step 1: Identify Behavioural Variables	Semi-explicit	Semi-defined	Semi-defined	Semi-formal
Step 2: Map interview subjects to behavioural variables	Explicit	Undefined	Semi-defined	Informal
Step 3: Identify significant behaviour patterns	Semi-explicit	Semi-defined	Undefined	Informal
Step 4: Synthesize characteristics and relevant goals	Explicit	Semi-defined	Semi-defined	Informal
Step 5: Check for redundancy and completeness	Explicit	Semi-defined	N/A	N/A
Step 6: Expand the description of attributes and behaviours	Explicit	Defined	Defined	Semi-formal
Paso 7: Designate persona types	Explicit	Defined	Semi-defined	Informal

For example, Cooper and colleagues [8] assume in Step 1 -Identify Behavioural Variables- that the users have already been interviewed and the gathered data have been organized. This is an implicit step, which should be listed as the first explicit step of the technique. To improve this aspect, we propose adding an initial activity in the personas construction process, called State Hypotheses. This new activity aims to state initial personas hypotheses and gather the data required from potential future users and then identify the behavioural variables in a later activity using the creativity-building techniques proposed in this paper (see Table 2). Additionally, we define two new documents that consist, respectively, of a justified List of Personas Hypotheses for activity 1 and a List of Behavioural Variables for activity 2 (see Table 2). In Step 5 - Check for Completeness and Redundancy-, Cooper and colleagues [8] do not specify any product associated with this step, and it is rated as N/A (see Table 1), that is, not applicable. In our version of the personas technique we suggest that participatory meetings be held to evaluate the models obtained and that they be recorded in a Validation Document (see Table 2).

The other steps of Cooper and colleagues' Personas technique [8] have been analysed similarly. This analysis is available at http://arantxa.ii.uam.es/~sacuna/PersonaSE/modificacion and is, for reasons of space, not detailed in this paper.

The aim behind the Personas technique is to adapt the system to the future system users. However, none of the steps in this technique includes usability mechanisms (e.g. provides undos, alerts, wizards, feedbacks, etc.) connected to the defined personas. In our paper, we have identified the usability mechanisms (undo, cancel, etc.), imported from [14], that the different types of personas will need according to their characteristics and what they expect of the software system. Following on from this line, the aim of which is to consider usability in the early stages of the software development process, we have set out to incorporate additional activities into the Personas technique that are helpful for this purpose. These new activities are: a) Relate behaviour patterns to usability mechanisms; b) Build use cases; and c) Build mock-ups. Both use cases and mock-ups should include the usability mechanisms selected for each created persona.

For each of the identified limitations, we have proposed a modification that can be easily incorporated into the Personas technique. These modifications implement a new version based on Cooper and colleagues' Personas technique [8] that covers the weaknesses specified in Table 1. This new proposal, called PersonaSE, is described in the next section.

4. PERSONASE TECHNIQUE

The PersonaSE technique that we propose consists of a set of interrelated activities that lead to the creation of personas and ease the incorporation of the usability mechanisms from the SE requirements analysis activities, thereby helping to improve the usability of the software system that is to be developed.

Table 2 presents all the activities making up the PersonaSE technique. For each activity we outline objectives, techniques and associated products. The new activities proposed are shown on a grey background.

In activity 1 -State hypotheses- we formulate the list of initial hypotheses for the personas that are to be created, and develop and interview the future system users. This produces the transcribed interviews, from which the information required to carry out the other activities is gathered. In activity 2 -Identify Behavioural Variables-, the full List of Behavioural Variables is identified from the Interview Synthesis.

Activity 3 -Map Interview Subjects to Behavioural Variables outputs the Ranges of Behavioural Variables and Mapping of Interview Subjects. These products are the input for activity 4 -Identify Significant Behaviour Patterns, where the Significant Behaviour Patterns are identified and the Group Percentage Table is generated. This is the source of the personas. The Personas Foundation Document is put together during activity 5 -Synthesize Characteristics and Relevant Goals-. This document contains the full definition of a persona. Activity 6 -Check for Redundancy and Completeness- is carried out to locate information gaps that need to be filled. Additional interviews may be required for this purpose. They may discover behaviours outside the behavioural spectrum, which would have an impact on other activities. The Validation Document is the input for activity 7 - Expand the Description of Attributes and Behaviours-. This activity outputs a narrative for each of the created personas, that is, a one-page document describing the persona and a typical day in the life of that persona.

In activity 8 -Relate Behaviour Patterns to Usability Mechanisms- the behavioural patterns or created personas are related to different usability mechanisms, and these relationships are justified in a Pattern-Usability Mechanism Relationship Document. All the information gathered from the above activities is used in activity 9 -Designate Persona Types- in order to associate the persona type with each persona. In activity 10 -Build Use Cases- use cases are built taking into account the relationships between the patterns and usability mechanisms. Finally, in activity 11 -Build Mock-Ups-, mock-ups (also containing the usability mechanisms for each persona) are built, and the Mock-Up Evaluation Document is generated.

The PersonaSE technique has been used to design a Web-based Flight Booking System. This application, available at http://arantxa.ii.uam.es/~sacuna/PersonaSE/aplicacion, gives a better understanding of how the PersonaSE technique works. This system searches flights based on the selection, by defined personas, of dates, destination and origin, as well as the number of adult passengers.

Table 2. Description of the PersonaSE Technique Activities

ACTIVITIES		OBJECTIVES	TECHNIQUES	PRODUCTS
ACTIVITY 1: STATE HYPOTHESES	Activity 1.1: Identify possible personas	State preliminary hypotheses about the possible personas to be created.	Based on the information gathered from the customer, the nature of the application domain and the organizational documentation gathered at the previous meeting with the customer, developers state hypotheses for personas. The technique we recommend for this purpose is brainstorming, followed by a voting round at the end of the session to determine the most creative and feasible hypotheses.	• List of Hypotheses for Personas
	Activity 1.2: Hold ethnographic interviews	Based on these hypotheses, investigate possible system users to find out their motivations and behaviours, gathering behavioural data.	The interviews for each hypothesis are conducted based on business domain knowledge and through the proposed ethnographic interviews template.	• Transcribed Interviews
ACTIVITY 2: IDENTIFY BEHAVIOURAL VARIABLES	Activity 2.1: Synthesize the Interview Responses	Synthesize the responses to all the interviews.	Analyse the results of the survey conducted in activity 1. To do this, process all the responses to the transcribed interview questions using Atlas.ti software (http://www.atlasti.com/) to output the behavioural variables.	• Interview Synthesis
	Activity 2.2: List Behavioural Variables	List all behavioural variables. Check identified hypotheses for validity.	Behavioural variables are selected by participative meetings. Then, compare these variables with the personas hypotheses to validate these hypotheses.	• List of Behavioural Variables
ACTIVITY 3: MAP INTERVIEW SUBJECTS TO BEHAVIOURAL VARIABLES	Activity 3.1: Identify the Ranges of Behavioural Variables	For each behavioural variable identify its range of possible values.	At a participatory meeting, analyse the interview synthesis to identify the ranges of each behavioural variable.	• Ranges of Behavioural Variables
	Activity 3.2: Map Interview Subjects	Represent exactly how the multiple subjects are grouped with respect to each of the significant behavioural variables.	Interview subjects are mapped according to the perception of the subjects' observations and the interview responses. To do this, place each of the respondents in different ranges for each of the identified behavioural variables.	• Mapping of Interview Subjects
ACTIVITY 4: IDENTIFY SIGNIFICANT BEHAVIOUR PATTERNS		Identify particular groups of interview subjects occurring in more than one range or variable.	Examine the mappings of interview subjects from activity 3 and build a table showing the percentage of interview subjects that share each of the behavioural variable range values. The groups with the highest percentages are the significant behaviour patterns. These are the source of the personas, which are given a name and a photograph.	• Significant Behaviour Patterns • Group Percentage Table
ACTIVITY 5: SYNTHESIZE CHARACTERISTICS AND RELEVANT GOALS		Synthesize characteristics and relevant goals. Describe the personas' personalities.	Synthesize the data for each person identified in activity 4, briefly specifying points about the behavioural characteristics identified in the synthesis of the interviews (activity 2).	• Personas Foundation Document
ACTIVITY 6: CHECK FOR REDUNDANCY AND COMPLETENESS		Check persona mappings, characteristics and goals.	Check that the important identified aspects are fully defined in the personas created and models built through participatory inspection meetings.	• Validation Document
ACTIVITY 7: EXPAND THE DESCRIPTION OF ATTRIBUTES AND BEHAVIOURS		Convey the attitudes, personality, needs and problems of the personas to other team members.	Analyse the data collected and the personas foundation document (activity 5) and synthesize the personal profile and a typical day in the life of each persona. For each created persona, write a third-person narrative.	• Narrative
ACTIVITY 8: RELATE BEHAVIOUR PATTERNS TO USABILITY MECHANISMS		Relate each behaviour pattern to usability mechanisms.	Based on information about the values of the behavioural variables for each identified persona and the interview responses, analyse the relationships between the behaviour patterns and usability mechanisms imported from [14].	• Pattern – Usability Mechanism Relationship Document
ACTIVITY 9: DESIGNATE PERSONA TYPES	Activity 9.1: Select Representative Personas to Elicit Requirements	Prioritize the created personas to determine which should be the primary design objective, that is, find just one primary persona whose needs and objectives can be completely and positively satisfied by a single interface.	Based on the description of each of the personas types and all the analyses conducted throughout the personas creation process, determine the person types (primary, secondary). Each of the created personas is associated with a personas type.	• Persona Type Association
	Activity 9.2: Enrich the System with Secondary Personas	Determine what secondary persona needs are likely to enrich the system.	Analyse the secondary persona foundation document and narrative and search for functionalities not stated by the primary persona that are useful for the system.	(Software Requirements Specification is enriched)
ACTIVITY 10: BUILD USE CASES		Materialize the usability mechanisms listed in activity 8 in the use cases.	First build the usual set of use cases, not including the usability mechanisms, and then add these mechanisms taking into account the relationship between the behaviour patterns and the above mechanisms, and the information specified in the Personas Foundation Document.	• Use Cases (with usability mechanisms)
ACTIVITY 11: BUILD MOCK-UPS	Activity 11.1: Implement Mock-ups	Build mock-ups that include the usability mechanisms.	Based on the use cases developed in the last activity and the analysis of the relationship between the created personas and usability mechanisms, build mock-ups.	• Mock-ups
	Activity 11.2: Evaluate Mock-ups	Validate mock-ups.	At participatory meetings, validate mock-ups.	• Mock-up Evaluation Document

5. INTEGRATION OF THE PERSONASE TECHNIQUE INTO THE SOFTWARE REQUIREMENTS ANALYSIS PROCESS

As PersonaSE helps to synthesize all the data available about the prospective system users and also to determine what it is that the product should do to satisfy the personas' needs and profile, the best place in the development place to incorporate this new technique is the software requirements analysis process. To be able to integrate PersonaSE into the software requirements analysis process activities, each PersonaSE technique activity has to be assigned to the activities making up the requirements analysis process. This way, the requirements analysis activities will be modified because, apart from the routine tasks, requirements analysts will also have to perform new tasks taken from the PersonaSE technique. To define the SE requirements analysis process activities, we considered SWEBOK (SoftWare Engineering BOdy of Knowledge) [13]: Requirements Elicitation, Requirements Analysis, Requirements Specification and Requirements Validation. The right-hand side of Figure 1 shows these four activities according to [13]. Each of these SE activity types is linked to one or more PersonaSE technique activities (left-hand side of Figure 1). The directed lines in Figure 1 show links between the PersonaSE technique and the four analysis activities.

The PersonaSE technique offers the **Requirements Elicitation** activity additional information sources and resources for eliciting knowledge to what are traditionally used in the SE requirements elicitation activity. The PersonaSE technique activities linked to the requirements elicitation activity and their justification follow:

- *Identify possible personas*: state hypotheses for the personas to be created to determine who the possible interview subjects will be. This is a preliminary step designed to find out things about the user.
- *Hold ethnographic interviews*: these ethnographic interviews are designed and held taking into account the stated personas hypotheses. Interviewing is a means of eliciting information. Like the other information acquisition sessions that are held to elicit requirements, these interviews also have to be transcribed.
- *Synthesize the interview responses*: interview synthesis is based on analysis, for which reason the analysis and synthesis of interviews are linked to the requirements elicitation analysis task.
- *List behavioural variables*: by synthesizing the interviews we get the list of behavioural variables that are to somehow characterize the possible users, thereby helping to find out things about the user.
- *Identify the ranges of behavioural variables*: these ranges are identified by observing how the subjects are grouped around the behavioural variables. These groups characterize possible system users, thereby providing greater knowledge of the user.
- *Relate behaviour patterns to usability mechanisms*: this relationship provides information about what the possible users need to interact with the system.
- *Select representative personas to elicit requirements*: possible users are selected to participate in the routine requirements elicitation process, thereby helping to improve the knowledge there is about the user.
- *Implement mock-ups*: building mock-ups provides information by explicitly stating what the user requires of the system depending on his or her profile. Discussing the mock-up with potential users will supply even more information.

The PersonaSE activities offer the **Requirements Analysis** activity useful conceptual tools that supplement and/or extend instruments usually used in the requirements analysis activity. They can analyse information and knowledge about the user, model the user and help to model the system. In the following, we justify the linkage between the PersonaSE technique activities and the requirements analysis activities.

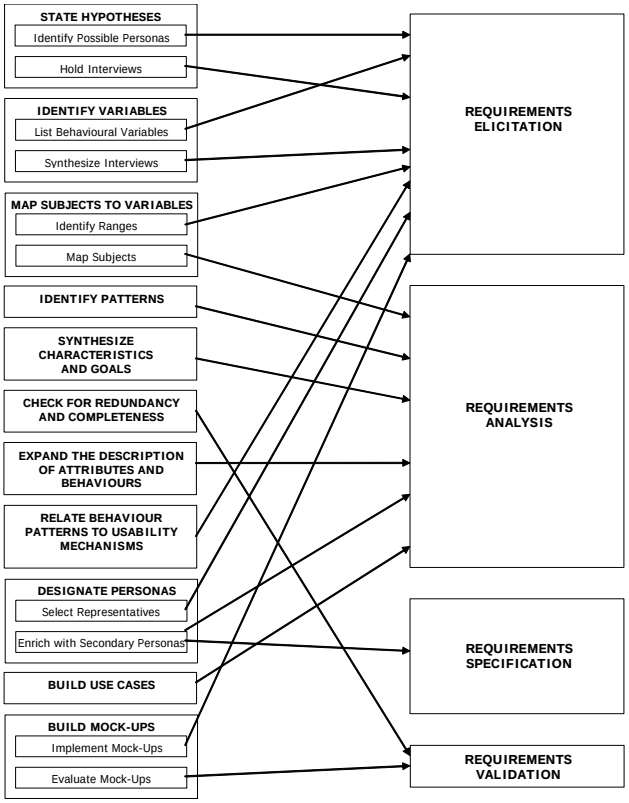


Fig. 1. Relationships between the PersonaSE activities and SE requirements analysis activities

- *Map interview subjects*: by representing how multiple subjects are grouped around the behavioural variables, we are modelling the user. This has to do with the conceptual modelling that is carried out in the requirements analysis activity.
- *Identify significant behaviour patterns*: personas (archetypal users) are the result of identifying particular groups of subjects in more than one range. This is, in the last analysis, equivalent to user modelling.
- *Synthesize characteristics and relevant goals*: this brief description of characteristics and relevant goals, which reflects the personality of the created personas, is also helpful for modelling the user.
- *Expand the description of attributes and behaviours*: the development of narratives provides a brief introduction to the persona in terms of job or life style and conveys the persona's attitudes, needs and problems to other team members. This is a user model in the shape of a narrative.
- *Enrich the system with secondary personas*: system modelling is extended by determining what functionalities the secondary personas would add to the system.

- *Build use cases*: the use cases enriched with the behaviour pattern-dependent usability mechanisms are a system model. This activity can therefore be linked to the system modelling traditionally performed in requirements analysis.

The PersonaSE activity *Enrich system activity with secondary personas* inputs information for writing requirements to the **Requirements Specification** activity, which generally has to do with drafting a document specifying the requirements that the system should comply with and is concerned particularly with the structure, quality and verifiability of that document:

- *Enrich the system with secondary personas*: by determining what functionalities (not explained by the primary persona) the secondary persona expects to find in the system, this activity inputs requirements for the Software Requirements Specification document.

The PersonaSE technique activities related to **Requirements Validation** are:

- *Check for redundancy and completeness*: mappings are checked, as are the characteristics of the personas and their goals in order to find out whether there are any important gaps that need to be filled in. This way, the developed models and products are validated in both textual and graphical format.
- *Evaluate mock-ups*: a document is drafted to record the results of the user evaluation of the mock-ups, thereby validating the set of mock-ups.

6. CONCLUSION

This work contributes towards building HCI knowledge into routine SE practice. To do this, we modified the HCI Personas technique to comply with the levels of systematization common in SE, and we enriched the requirements analysis process by incorporating the PersonaSE activities into the four routine requirements activities: requirements elicitation, requirements analysis, requirements specification and requirements validation. After adding PersonaSE to the four activities, the activities that gained most were requirements elicitation and requirements analysis, as PersonaSE introduces important innovations into these activities: i) elicit the characteristics of real users to create fictitious personas based on the understanding of these users, and ii) model these personas.

The integration of personas and requirements analysis can better identify what the software product should do and how it should behave, as it shapes a common language to help to build an understanding of the personas who are to interact with the system and match the system development to the characteristics of these personas. The next step is to determine the timeline for integrating the PersonaSE technique activities into SE's software requirements analysis process.

7. REFERENCES

[1] T. Adlin, H. Jamesen, J. Pruitt. (2002). Personas: Exploring the Real Benefits of Imaginary People. Available: http://www.chiplace.org/ techniques/show-article.jsp?id=1.

[2] H. Beyer, K. Holzblatt. (1998). Contextual Design: Defining Customer-Centered Systems. Morgan Kaufmann Publishers, San Francisco.

[3] Blomquist, M. Arvola. (2002). Personas in Action: Ethnography in an Interaction Design Team. Second Nordic Conference on Human-Computer Interaction. Proceedings of NordiCHI, pp.197-200.

[4] S. Calde, K. Goodwin, R. Reimann. (2002). SHS Orcas. The First Integrated Information System for Long-Term Healthcare Facility Management. American Institute of Graphic Arts, New York. Available: http://www.aiga.org/ resources/content/7/6/2/documents/FORUM_calde_case_032102.pdf.

[5] Cooper. (1999). The Inmates are Running the Asylum. Sams, Indianapolis.

[6] Cooper. (2003). The Origin of Personas. Available: http://www.cooper.com/insights/journal_of_design/articles/the_origin_of_personas_1.html.

[7] Cooper, R. Reimann. (2003). About Face 2.0: The Essentials of Interaction Design. Wiley, Indianapolis.

[8] Cooper, R. Reimann, D. Cronin. (2007). About Face 3.0: The Essentials of Interaction Design. Wiley, Indianapolis.

[9] J. Dong, K. Kelkar, K. Braun. (2007). Getting the Most Out of Personas for Product Usability Enhancements. Second International Conference on Usability and Internationalization. Proceeding of the UI-HCII, pp.291-296.

[10] K. Goodwin. (2002). Getting from Research to Personas: Harnessing the Power of Data. Available: http://www.cooper.com/content/insights/newsletters/2002_11/getting_from_research_to_personas.asp.

[11] J. Grudin, J. Pruitt. (2002). Personas, Participatory Design and Product Development: An Infrastructure for Engagement. Participatory Design Conference. Proceedings of the PDC, Computer Professionals for Social Responsibility, pp.144-161.

[12] Holzinger. (2005). Usability Engineering Methods for Software Developers. Communications of the ACM 48(1): pp.71-74.

[13] IEEE Computer Society Professional Practices Committee. (2004). Guide to the Software Engineering Body of Knowledge (SWEBOK- 2004 Version). IEEE Computer Society, Los Alamitos, CA.

[14] N. Juristo, A. Moreno, M. Sánchez. (2007). Guidelines for Eliciting Usability Functionalities. IEEE Transactions on Software Engineering 33: pp.744-758.

[15] Milewski. (2004). Software Engineers and HCI Practitioners Learning to Work Together: A Preliminary Look at Expectations. Proceedings of the 17th Conference on Software Engineering Education and Training CSEET´04 IEEE, pp. 45-49.

[16] J. Pruitt, J. Grudin. (2003). Personas: Practice and Theory. Available: http://www.research.microsoft.com/research/coet/Grudin/Personas/Grudin-Pruitt.pdf.

[17] Seffah, E. Metzker. (2004). The Obstacles and Myths of Usability and Software Engineering. Communications of the ACM 47(12): pp.71-76.

[18] K. Vasara. (2003). Introducing Personas in a Software Project. Master's thesis, Helsinki University of Technology, Helsinki.

Towards Industrial-Strength Usability Evaluation

Martin Schmettow
Passau University
Informations Systems II
94032 Passau, Germany
schmettow@web.de

ABSTRACT
Usability professionals may face strict economic demands on the usability process in near future. This position paper outlines a research agenda to make usability evaluation a predictable and highly efficient engineering process.

Categories and Subject Descriptors
H.5.2 User Interfaces (e.g. HCI) Evaluation/methodology

Keywords
Usability Evaluation, Measurement, Process Quality

1. MOTIVATION
Usability professionals are never tired to stress the economic impact of good usability. And indeed, there are several compelling arguments: The first may be derived from the ISO norm 9241-11: Efficiency is regarded as one of the three main criteria of usability and can directly be converted into a bargain. For example, a very efficient interface to an enterprise information system makes users do their tasks more quickly which increases overall throughput. The second argument is specific to web usability. Web users are known to be very impatient with web sites having poor usability, especially with online purchasing; consequently usability directly affects the conversion rate of e-commerce companies. The third argument is from the perspective of software development. It is a widely accepted law, that defect fixing costs overlinearly depend on how early a defect was introduced and how late it was found. This is a justification for doing intensive usability evaluation early in system development.

But, many usability professionals still act under the paradigm of discount usability. In a broad sense this denotes: usability evaluation as a best effort strategy and conducted iteratively by experts who just know what they are doing. What, if clients or employers of usability professionals start taking the above economic arguments seriously? For example: What, if a start-up company has an innovative product idea and plenty of venture capital, but usability is mission-critical and they have only one shot? Will they rely on discount usability? Will they accept the good reputation of a usability company as the only guarantee? It is more likely, that they want objective preconditions, like a proven and certified evaluation plan. And maybe they even want quantitative guarantees and proven contract fulfillment, like: There is no show stopper left in the system and at least 90% of serious problems are identified. The paradigm of discount usability is inappropriate in such cases.

Research on the usability evaluation process has seen two major debates (research agendas, respectively): The Five-Users-Is-Not-Enough debate and the Damaged Merchandise debate. The Five Users debate is about how to reliably plan and control usability evaluation studies, whereas the Damaged Merchandise debate treats the topic of how to compare evaluation methods in fair and valid way. In the following, I will argue why we must continue these research agendas, in order to make usability evaluation a well understood and highly optimized engineering activity. But, I will also claim that we have to put off some blinders.

2. WHY TO CONTINUE THE "FIVE USERS" DEBATE
The five users debate goes back to Nielsen and Landauer's suggestion to model the progress of evaluation studies as a geometric series [9]. Unfortunately, the debate was primarily carried by an oversimplification of Nielsen, who trivialized his own findings in stating that testing five users is enough in industrial practice [8]. This is, by the way, an excellent example of the discount usability paradigm, which may turn out obsolete.

In contrast, several researchers went deeper into the theoretical impact of this model: The phenomenon of variance in the process was discovered [3], good task design was found to be a major impact factor [6] and basic stochastic assumptions of the model were questioned [2]. A recent contribution was the proof that the geometric model is inherently flawed by falsely assuming that usability defects are equally visible and sessions equally effective [10]. Instead, the beta-geometric model, accounting for heterogeneity, was shown to better predict the process.

But, this is still an oversimplification that does not comprise all impact factors found in industrial studies. For example, recently I tried to fit the data reported from the CUE-4 study with the beta-geometric model – with disappointing results: The model could not sufficiently explain the overwhelming number of defects that were detected only once [7]. In consequence, there is still no reliable estimation of how many defects were left undetected. For the first, there are two options for enhancing the model in order to better fit the data and reliably plan and control usability studies: First, the study progress has to be tracked on the finer grained level of single tasks presented in a usability test (or imagined by usability inspectors). Specifically, this may help identify when a certain set of tasks is "exhausted" and replace it by new tasks that

I-USED'08, September 24, 2008, Pisa, Italy

make further defects observable. Second, the current models do not handle the problem of false alarms in evaluation studies. These may well be liable for the misfit reported above. Currently, we are working on an enhanced model to incorporate the occurrence of false alarms and varying task sets. This hopefully enables us to better estimate the number of remaining defects (misses) and to give a probability for a reported defect being a false alarm. The latter may prevent wasting development resources on would-be defects and thus has direct economic impact.

3. BEYOND "CHASING THE HE"

The Damaged Merchandise debate arouse by the harsh critique of Gray and Salzmann on the poor validity of experiments on UEMs [5]. However, my main point here is not validity, but the observation that research on designing UEMs has not made much substantial progress. Even recent well designed studies are still very restricted in their contribution to understanding the cognitive or contextual factors of finding usability defects. Instead, they make more or less marginal adaptions to common inspection methods and compare this in a two conditional experimental design to the Heuristic Evaluation (HE). The observed effectiveness gains are in many cases marginal (e.g. [4]) or non-existent [11]). This "Chasing the HE" approach has the severe drawback of restricted insight. It lets us only know which of two procedures is (slightly) better. It does not inform about the specific interplay of impact factors granting effective defect identification. But, this is a precondition to design (much) better procedures, provide adequate training and adjust the evaluation process to business goals.

Only few studies have paid attention to successful versus unsuccessful cognitive-behavioral strategies of usability experts. To give an example for a rarely recognized work that has done better: Perspective based reading is a well known technique in software inspections and raises effectiveness by reducing cognitive load. Zhang et. al. have transferred this technique to usability inspection and have found likewise improvements [13]. Another positive example is how Woolrych et. al. analyzed the knowledge resources involved in usability inspections [1]. (They also made some points on how false alarms arise.)

These are interesting and relevant results, as they may lead to methods and training concepts for increased effectiveness of usability experts. But, there still is a lack of quantitative research on such topics. Especially, defects are likely having qualitative properties that make a difference with respect to behavioral strategies and knowledge resources. Frøkjær and Hornbæk have found differing detection profiles for two inspection methods after classifying defects with the User Action Framework [4]. An promising way to go is to search for defect classes in the raw data from evaluation processes and derive an empirically valid classification Advanced statistical exploration techniques, like differential item functioning from item response theory [12] or binary cluster analysis probably apply well to this problem, in contrast to ordinary variance analysis. The strength of these techniques is that they to not require manipulating independent variables. Instead, they can reveal latent variables in existing data sets, including results from industrial studies.

These approaches may be used to profile methods according to their effectiveness regarding certain types of defects. In industrial settings this is useful for selecting a method appropriate to the development context. For example, we may purposefully choose a method for identification of task related defects early in development. Late in the process another method may serve identification of superficial design issues). Another possibility is aligning the evaluation focus to business goals, e.g. evaluating for efficiency in case a system is primarily aimed at experts.

4. CONCLUSION

Modern software engineering is well regarding economic demands: efficiency of development processes, early defect discovery and aligning software qualities to business goals. The usability profession is still dragging a little behind, but may sometimes face their customers' claims for process approval, efficiency and guarantees. The aim of this paper was to point out valuable research agendas in the past, but to also identify future directions of research: Quantitative research with refined experimental designs and advanced statistical techniques may reveal relevant properties on several levels of the usability evaluation process. Knowing the properties on process level results in better approaches to plan and control studies towards given business goals. Knowing the properties on the cognitive-behavioral level are a precondition to significantly raise effectiveness and appropriateness of evaluation processes. Much can be achieved with advanced statistical techniques on existing data sets. The minimum to get is specific and well grounded hypotheses that will inspire for well designed and elaborate experimental studies to deeply understand the anatomy of usability evaluation.

5. REFERENCES

[1] Alan Woolrych, Gilbert Cockton, and Mark Hindmarch. Knowledge Resources in Usability Inspection. In *Proceedings of the HCI 2005*, 2005.

[2] David A. Caulton. Relaxing the homogeneity assumption in usability testing. *Behaviour & Information Technology*, 20(1):1–7, 2001.

[3] Laura Faulkner. Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments & Computers*, 35(3):379–383, 2003.

[4] Erik Frøkjær and Kasper Hornbæk. Metaphors of human thinking for usability inspection and design. *ACM Trans. Comput.-Hum. Interact.*, 14(4):1–33, 2008.

[5] Wayne D. Gray and Marilyn C. Salzman. Damaged merchandise? A review of experiments that compare usability evaluation methods. *Human-Computer Interaction*, 13(3):203–261, 1998.

[6] [Gitte Lindgaard and Jarinee Chattratichart. Usability testing: What have we overlooked? In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1415–1424, New York, NY, USA, 2007. ACM Press.

[7] Rolf Molich and Joseph S. Dumas. Comparative usability evaluation (CUE-4). *Behaviour & Information Technology*, 27(3), 2008.

[8] Jakob Nielsen. Why you only need to test with 5 users. Jakob Nielsens Alertbox, March 19 2000. http://www.useit.com/alertbox/20000319.html.

[9] Jakob Nielsen and Thomas K. Landauer. A mathematical model of the finding of usability problems. In *CHI '93: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 206–213, New York, NY, USA, 1993. ACM Press.

[10] Martin Schmettow. Heterogeneity in the usability evaluation process. In David England and Russell Beale, editors, *Proceedings of the HCI 2008*, volume 1 of *People and Computers*, pages 89–98. British Computing Society, 2008. in print.

[11] Martin Schmettow and Sabine Niebuhr. A pattern-based usability inspection method: First empirical performance measures and future issues. In Devina Ramduny-Ellis and Dorothy Rachovides, editors, *Proceedings of the HCI 2007*, volume 2 of *People and Computers*, pages 99–102. BCS, September 2007.

[12] Martin Schmettow and Wolfgang Vietze. Introducing item response theory for measuring usability inspection processes. In *CHI 2008 Proceedings*, pages 893–902. ACM SIGCHI, April 2008.

[13] Zhang Zhijun, Victor Basili, and Ben Shneiderman. An empirical study of perspective based usability inspection. Technical report, University of Maryland, Human-Computer Interaction Lab, 1998.

Controlling User Experience through Policing in the Software Development Process

Mats Hellman
Product Planning User Experience
UIQ Technology
Soft Center VIII
SE 372 25 Ronneby, Sweden
+46708576497
Mats.hellman@uiq.com

Kari Rönkkö
School of Engineering
Blekinge Institute of Technology
Soft Center
SE 372 25 Ronneby, Sweden
+46733124892
Kari.ronkko@bth.se

ABSTRACT

Today the challenge in the mobile industry is User experience (UX), which is starting to affect software engineering processes. A common use or definition of the term UX is still not de facto defined. Industry and academy are both in agreement that UX definitely includes more than the previous usability definition. Our concern in this paper is how industry and manufacturers can manage to successfully get a UX idea into and through the software development cycle? Our discussion includes obvious components from usability and new UX components that are not taken into account by prevailing HCI approaches. We will discuss branding, trends and timing as vital components in that puzzle.

KEYWORDS

User Experience, usability, brand, trends, invention, software development process, mobile industry, software engineering, management

1. INTRODUCTION

Mobile phones have reached a point beyond the level where technical hot news are not enough to satisfy buyers, because today mobile devices also have to include the aspect of user experience. Apple's iPhone is one indication of this change. In recent years the mobile industry has put in a lot of efforts to grasp and develop products that can be claimed to be User Experience (UX) products. A mental shift from a usability focus toward a more UX driven requirement gathering focus and handling has occurred. One reason is that UX discourses has been ongoing for a long period, even though mostly connected to new services like web, multimedia and other media centric services. Interestingly, these are products that acquire a different experience than the mobile applications and services. Another related factor is today's improved hardware possibilities including their infrastructural developed support on the market.

Unfortunately we are convinced that many companies in the mobile sector still are stuck with outdated control mechanisms that do not adequately support the recently introduced UX focus. Today's prevailing product control mechanisms has a stronger relationship to software development costs and rationales than securing UX. Before returning to the issue of how to secure and control UX design decisions within today's prevailing product and software development approach we will sketch the UX scene and exemplify challenges following with it.

When Apple launched the iPhone the UX hype hit the roof. All competitors now saw a device with intuitive, simple finger touch interface, with fast and smooth transitions and excellent performance. This device created a lot of media as well as consumer attention even though targeting a high price range and offering for bindings to one operator to start with. It got promoted by operators without fulfilling their requirements, and operators even accepted a new economical model that would give Apple a percentage of operator's winnings. A development we have not seen earlier in the branch. Why did Apple's iPhone reach this high level of UX recognition and operator acceptance?

Symbian Ltd and UIQ Technology have for over a decade offered an OS and SW platform that support touch; and their licensees, SonyEricsson, and Motorola to mention some have launched series of different versions of phones on the UIQ platform. Touch enabled phone devices like e.g. the P800 to the latest P1i from SonyEricsson have sold in good numbers and created a lot of media covering but not close to what the iPhone did. Another company trying to gain market in the touch area is Neonode. They created a clear buzz around their product but had trouble reaching the big sales even with UX claims of their product. So why is it that well known and established companies, with long experience, don't get the same "buzz" around their products as Apple? And why doesn't new innovative and creative company like Neonode hit it of massively? What made the success possible for Apple's product iPhone? In our opinion it has to do with a number of connected reasons.

First, usability as a "hygiene" factor needs to be in place if we want to hit in a mass-market launch for a new type of device. Meaning that the functionality and performance of a device are things a user doesn't notice until they create annoyance. In this view, in a well worked up market, usability has become a dissatisfier [1]. In such market users will notice and complain about the product when the expected outcome or usage doesn't live up to their expectations. On the contrary, if the hygiene works the way it should, as expected, they won't praise the usability of it anyhow. We are convinced that most companies in

I-USED'08, September 24, 2008, Pisa, Italy.

mobile industry are in control of the level of hygiene through applying HCI usability test methods (see [11] for example).

Second, total product design is another vital component. The product must be a throughout solid and attractive design, from hardware to software design. New and hot functionality is not enough anymore, today it is the design of the total experience that sells.

Third, the brand is an important part of the total product design, just as vital and important for success as is the design itself. We argue that this is one of the reasons explaining why Apple made a direct success with their iPhone and Neonode did not. New kids on the block always have a hard time, and have to make up an own role an identity to be both understood and accepted.

Fourth, trends need to be monitored and understood. How can you predict and take into account that a "fuzz" or "buzz" in a small group of people will turn into a mass market trend? How do you foresee and market that e.g. a mobile touch screen device will become a device in "every man's" hand instead of its initial status as status device in the pockets of the businessman tribe? Trend awareness and understanding about marketing, brands and target groups have always been important, but will in the mobile UX era be vital for success.

Fifth, timing is a vital component in a successful launch of a product? There is more than one understanding of timing. If you talk to product owners etc they will argue that if a specific device misses its target release window that device could and maybe should be cancelled. This is obvious and understandable, here we talk about a specific type of timing; the maturity of the market for a device with a specific functionality. When is a specific functionality or technology mature enough to be embraced and used without any hurdles or suspicion by the market and end users? Take the e-commerce adaptation as an example from the PC world. It took some years before users found e-commerce applications comfortable and secure enough to be used for paying stuff from the internet. This even though the technology, security solutions and infrastructure had been in place and worked a long time. Could this type of user phenomena be foreseen and taken to account when to launch a product at the optimal time?

Total design, brand, trends and adequate timing are subjects in need of further understanding within today's mobile industry; both concerning how to predict coming trends and brands, when to launch products, as well as how to secure and control the resulting UX designs throughout the software development process. Regarding the former challenge we lend at taking inspiration and borrow insights from the area of innovation. Knowledge about innovation processes and framework could be used to understand and prioritize actions to create and launch products in a successful manner. When it comes to the latter challenge, we present one solution in this paper. Our solution fits the established engineering idea of splitting product complexity into smaller manageable sub-functions, and working in multidisciplinary teams. In large software development projects this splitting approach has proven successful to cut time costs.

Below is provided a hierarchic map where we place the aspects discussed in this paper in relation to the following categories: User Experience, Market, Technology, and Software Development. Here it is possible to visualize relationships such as: brands and trends exist on a market with potential consumers; brand and trend is part of the user experience that companies tries to design to pleasure users; successful match between these is highly dependent on adequate judgments of maturity and timing for a product. We can also see where the border of traditional usability efforts is today. We do not emphasize new or existing technology in this papers discussion, even though, we indicate the importance of timing and maturity also here. Our contribution called "Policing" can be found under the category Software Development under Methodology and Requirements Engineering fulfilling the role of monitoring and securing a holistic product view. The Software development methodology is in this paper refers to the engineering idea of splitting the product complexity into smaller more manageable sub-functions (and teams), i.e. a traditional software engineering development approach.

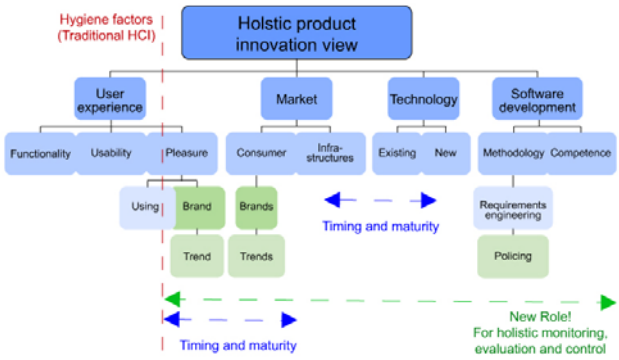


Figure 1. Overview and placement of discussed subjects

2. USABILITY AS HYGIENE FACTOR

The HCI community is nowadays agreeing that UX include more factors than defined in usability. Usability is an established part of software development even though maybe not as formalized as needed. UX on the other hand is not established throughout the development process and our belief is that when it is formalized and established it will change the way we understand and talk about requirement handling as well as product development processes and methods.

Usability as such in today's mobile business and product development is a thermometer that sets the "hygiene" level of a product. Users today take the "ease of use" part of product concepts for granted and will not praise the fact that a product or service has good usability. On the other hand users will complain loudly if the product doesn't live up to the expected level of hygiene. Usability has become a dissatisfier. Hence, the challenge for usability engineers is to collect dissatisfiers and feed them back as prioritized requirements. These will affect the product negatively if not treated as an important part of UX. In a sense dissatifiers can be perceived as the base of UX. These are aspects of a product or service that just have to work and when they do they will not be noticed by the users. Examples of the areas we talking about here are responsiveness, snappiness, learnability and visibility, effectiveness, efficiency etc. Keep in mind that handling dissatisfiers is not enough to reach a decent UX level. To do that we need to understand what pleasures a user during both use and owning a product. When we understand above it will be possible to launch products with satisfying level of UX.

3. TRENDS AND BRANDS

Today we see trends in society that emerge from and support environmental concerns. We can also see an increase in tribing activities that in one level has to do with big movements of refugees moving to other part of the world, to find "shelter from the storm" in new countries. This has created a possible growth for national groups that use violence as a toll for securing their tribal belonging. The other level of tribal behavior has more to do with groups that have found new ways to indulge themselves in their hobbies/interests. Examples of this is the late middle-aged bikers living their teenage dream as they drive down the roads as aged "hell-riders" on their Harley Davidson's.

Leading trend institutes has identified trends that need to be understood and taken into account as important aspects to succeed when developing a product with high level of UX. Below you find some trends that one well known trend institute; Faith Popcorn's BrainReserve describe on their website [12] and as they find as necessary to know and beware of when you: look for a new positioning on the market, strategic development and new product or service.

99 lives*: Too fast a pace, too little time, causes societal schizophrenia and forces us to assume multiple roles*

Anchoring*: A reaching back to our spiritual roots, taking what was secure from the past in order to be ready for the future.*

Being alive*: Awareness that good health extends longevity and leads to a new way of life.*

Pleasure revenge*: Consumers are having a secret bacchanal. They're mad as hell and want to cut loose again.*

Small indulgences*: Stressed-out consumers want to indulge in affordable luxuries and seek ways to reward themselves.*

Cashing out*: Working women and men, questioning personal/career satisfaction and goals, opt for simpler living.*

Clanning*: Belonging to a group that represents common feelings, causes or ideals; validating one's own belief system.*

Cocooning*: The need to protect oneself from the harsh, unpredictable realities of the outside world.*

Fantasy adventures*: Modern age whets our desire for roads untaken.*

In the mobile business obvious trends are staying connected and sharing content, this simultaneously with being an assessor expressing belonging and social status. To capture these types of requirements and to be able to support these kinds of trends we need to involve more than traditional usability evaluation can offer; a new UX and innovation related perspective of capturing user requirements is needed. These factors also need to be translated and incorporated in new formalized methods in the process of product and software development.

Neonode relied on the existing touch screen market as entry for their products. To their disadvantage they did not have large enough credibility among users in the market of touch phones to become a truly market success from start. Apple's iPhone had both credibility and a successful touch screen product. A product that provided the user with intuitive and responsive use, a pleasurable experience concerning the overall design, together with the pleasure of owning and showing of it as an assessor. Besides this iPhone also supported the "Mac, Apple" tribe. This new product called iPhone could actually be claimed to help these users secure their status and existing as members of precisely this tribe. This is a group of users that committed themselves to Apple's specific brand and design, a consumer group that buys for reasons of precisely experience and design (that Apple products helps them to communicate) rather than for a specific set of functionality. The fact that Apple has a very strong brand could be the difference when it comes to success or not. User have expectations and/or and experiences of Apple as "the" design company whereby the company gets a competitive advantage over other on-a-technical-level-equal-companies. Apple has the knowledge and the company culture needed in order to "live the brand". Other less brand known companies has to rely on the product without any help from a brand expectation or experience. One reason to this could be as Richard Mulholland states in his article; Fuck. Love. Brand: [13] "You see, "*brand" is a word open too much interpretation, a corporate ID executive sees it as the face of the company they designed, HR sees it as the people, marketers see it as the marketing they create, and management thinks it's the physical manifestation of the mission, vision, and values. This is the problem, in order to build "X", all your builders need to first understand what "X" is and here's the thing, it ain't rocket science. Once we realize that the word "brand" is a place-marker, we simply need to find out what we're replacing."* From our point of view the strategic work of building up a brand needs to be integrated in all levels in a company, relate to vision, goals and be a vital part of a holistic product view.

4. THE TIMING COMPONENT

"The winner gets it all", "It's only first place that counts and will be remembered". These are expressions that color us from upbringing and society and in many respects also true on a tough market. The timing aspects of releasing a new product is in many cases as important as the product it self. The right timing will give an advantage against competitors. But it is hard to judge when to launch a product; users or consumers on a market must be mature enough to appreciate the product to its full extent. Its functionality could be too advanced or just a bad copy of already existing product. Symbian and UIQ has produced Touch supported Software platforms for mobile phones for many years and delivered to customers like SonyEricsson and Motorola. These products has sold good in the business segment of the mobile world. It could be claimed that Sony Ericsson and Motorola over the years of delivering phones with touch enabled screens actually created both the marketplace as well as the user acceptance and user mature-ness for touch phones. If we compare with Apple's iPhone that was a hit direct, they besides using their extremely strong brand (se previous section 4) delivered with a good timing in a mature enough touch market.

5. UX AND PREVAILING SD PROCESSES

Good UX understanding an input is one side of the coin, how to organize with respect this understanding and input is the other side. As previous argumentation revealed it has become more and more important to deliver UX products. This is not enough, these products has to be developed faster and faster, whereby it also becomes vital for an organization to continue to keep the development time short.

"Everything about mobile phone design and production has to be quick, so it's months from when there is an idea for a phone to the

roll out on the market," said James Marshall, Sony Ericsson's head of product marketing, who is in Las Vegas this week for the trade fair. "The market moves very quickly, so you have to minimize development times."[4]

One approach that many organizations, including UIQ Technology AB, have chosen to apply to both secure quality and focus on deliveries, and meet the time challenge is to work in parallel multidisciplinary teams (see Hellman and Rönkkö 2008 [11] for details). The solution is a typical software engineering solution, i.e. to make complex things manageable through splitting up the problem in separated parallel work tasks during the development process.

Engineers often approach complexity through splitting the product complexity into smaller more manageable sub-functions. In the end all the sub-functions are put together and a product appears, hopefully as the designer or the idea maker intended. Deviations from the intended product idea are handled through iterated defect reporting and defect handling until the product is judged to have sufficient product quality. Hence, monitoring product quality is conducted by processes in which milestone criteria are measured mainly by different ways of controlling defect levels and defect status. So far this approach has been sufficient enough when striving to secure a product's quality from a task and goal perspective (classic usability view from HCI), but still no guarantee for enhancing the user experience (that increases the chances of product success on the market). In the goal and task view three canonical usability metrics have dominated, i.e. effectiveness, efficiency and satisfaction. Where the latter, satisfaction, has been a term capturing the felt experience on a very high level, i.e. without further dividing it into its diverse constituent elements. Today the UX level of quality needs to be handled. Handling this quality forces us to divide satisfaction into other soft values such as exemplified by fun, pleasure, pride, intimacy, joy, etc. [8, 4].

A risk with dividing is that the product owners (often Product Managers whining the company) will have an even harder time knowing that the intended product is the one that will turn up when all "bits and pieces" are assembled again to constitute the product.

Figure 2 visualizes above described work in multidisciplinary teams. Here the separateness of a product vision into many divided requirements means risks of not monitoring UX in a holistic way; it also represent today's goal and task oriented development models. The outcome/product includes the risk of becoming something that was not intended.

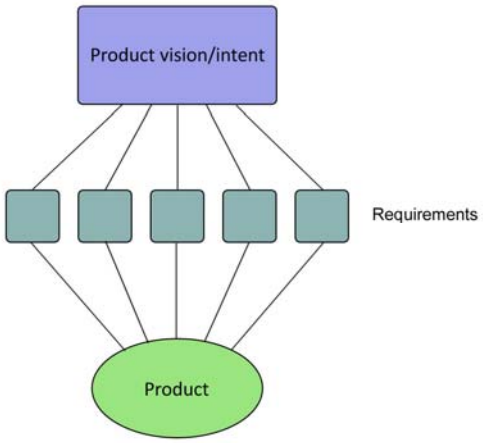


Figure 2. Split of product complexity

One problem that follows when splitting the product into smaller manageable sub-functions in the production process is the risk of losing a holistic product view. In the quality of user experience apparently small changes made in different subparts can actually constitute a huge user experience change when put together in the final product. It is also difficult to predict the effects of such separately handled changes. Applications in mobile products have in the past been more or less separate entities or islands in a product. And opportunities have existed for application designers and engineers to apply their own solutions and create their own application specific components with "isolated" specific behavior to support a use case (see [12] for an example). Such isolated behavior can and will be a big threat to the total UX of a product.

Pushing out ownership and responsibility to the separate parts is a common management strategy. It can be questioned if organizational models that push ownership out to the leaves in organization really are effective in the mobile industry? Doesn't this model encourages handling risks via a focus on each constituent part rather than a holistic view on the end product? Are there better and more efficient ways of making an idea appear in a product? Ways that could shorten the time to market, minimize the risk of fragmentation of the product, and in effective ways help organizations to prioritize and secure successful UX in products. Can we maintain a holistic perspective despite multiple splits of functionality during development? In this era with a growing need for high level monitoring of UX in products we are still left with the goal and task oriented development models. For the goal and task related usability paradigm dividing and delegating has been successful. Today we have to realize that good quality on different parts is not enough, not a guarantee for a successful product. In parallel with understanding and handling UX we need to find new ways to measure and monitor UX quality aspects during development. To support UX qualities efficiently a process with a clear product focus is needed in parallel with the up to today successful split application development approach. Otherwise, because of the prevailing task and goal tradition within software development, there is a risk that we talk about a holistic product view but in practice end up monitoring small identities. Still, we believe the engineering approach of separation is powerful and necessary in large projects. So - what are the possible approaches for ensuring an idea appears throughout the prevailing engineering approach of separating the development?

The introduction of an overall UX control process is the solution we advocate.

In order to secure the vision of product intent, in complex and multi requirement projects, the organization needs to acknowledge the need for what we call policing (actually having real cups in mind doing police(ing) work in the positive sense appreciated by citizens). Not just defect levels, but also and maybe even more important, the holistic product intent throughout the development cycle and in all different teams participating in the development process. This is needed to secure an efficient and effective way of working towards a successful product.

There is a risk of losing the UX intent of a product if no support structure is in place. In order to keep the organization "mean and lean" and at the same time deliver UX focused products we need to secure the vision of a product throughout the development process. Today many companies have developed methods to validate concepts of the final product with end users. UIQ technology AB uses for instance their UTUM method. [3], [10]. Unfortunately these kinds of validation activities are too often handled by and within a UI Design/Interaction Design group and not as part of the overall design process, e.g. as ad hoc help in the design work at different stages. Our suggestion is that companies organize in such a way so that UX requirements developed by end user understanding and use knowledge are monitored throughout the development cycle. This can be done by having UX guards in leading positions in the development process. People that monitor the holistic view of the product and who have the mandate take necessary actions whenever it is needed to secure the overall product intent.

6. POLICING UX

Even though most companies have both verbal and written UX statements and visions on their walls as lead goals for their business, an overall UX strategy are often missed out. A products quality definition is still related to different sub-levels, measurements and predictions of defects as criteria, and seldom includes usability and/or UX quality criteria. This means there is no connection or possible way of measuring the "temperature" of UX in the product during the development between vision and final product. There is also a embarrassing divergence between UX quality and existing product quality, meaning that we have processes and means from traditional software engineering to monitor product quality by defects, which do not constitute the wished for guarantee to achieving an envisioned high level of UX in the final product.

Therefore a complementary way to also inject UX quality assurance into the development process would be by:

1. Gaining acceptance of a **vision** through user research with end users by means of methods like early prototype testing.

2. **Policing** the vision throughout the development process by internal review methods to secure UX product quality. UX quality criteria and milestones should be included in an overall design process influencing the development process. A new quality assurance role needs to be created for UX experts to act as guardians for the UX quality.

3. **Validating** the product and evaluating the result against the vision, again by formalizing existing methods like UTUM [3], [10] in the development process. This is also visualized in figure 3.

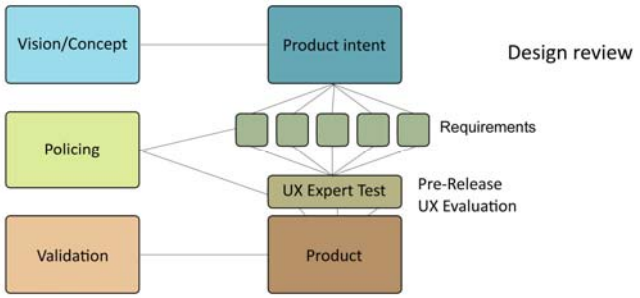


Figure 3. Policing UX requirements

An organizational set up like the one described in figure 3 would be a better guarantee that the product vision and intent is what will be delivered in the end compared with the organizational set up presented in figure 2. Meaning that the whole of the organization needs to understand and prioritize the end result. Project Managers need to acknowledge, understand and take these new UX criteria's into their plans. The way to secure product quality and to include UX into the product quality aspect has to be to introduce "UX guards" in all levels of development. Their role would need to be to police the fulfillment of the UX quality criteria in the process defined and decided checkpoints. These checkpoints could e.g. be expert reviews of requirements and expert UX reviewers to get the authority to set a pass/not pass stamp on the intended delivery. This needs to be agreed and formalized into the development process.

7. DISCUSSION

It is identified that the academic fields of Software Engineering (SE), Human Computer Interaction (HCI), and Participatory Design (PD) to a large extent developed divided from each other [Juristo et al. 2001, Kensing 3003]. Each area is highly challenging and has today decades of important documented knowledge; SE has significant successes in requirements gathering related to software development organization, HCI in usability evaluation and PD in techniques and methodologies for user participation. Industry has picked and applied parts from the different fields despite the academic separation. Five years ago a mix of the knowledge inherent in these fields was considered to provide a good enough foundation for building successful development process. In recent years the mobile industry has started to compete with what can be claimed to be User Experience (UX) products. Hence a fourth aspect called UX appears that also needs to be integrated on the top of these aspects.

When will the above mentioned areas develop to support also the understanding of UX, so that we can find better ways to capture and monitor when a market is mature enough for appreciating a product or service? We need to widen our understanding of users also in the UX aspect. Find ways to monitor UX requirements throughout the process. UX should be the backbone of product development today and not as in many cases something that is added as a final finishing procedure of a product. Such approach is just a "lipstick on a chicken" approach and will not lead to a

successful launch of a product. We need to change existing development processes to be built around UX definitions, and not just incorporate UX as add on to already existing processes. UX is a new perspective we have to apply in order to successfully launch products at the right point of time within an "open" market window; in which it supports new and existing trends, and of course deliver satisfactory levels of hygiene. Hence, UX will change how we perceive and perform product development.

As future work we will continue to look for inspiration and knowledge within the area of innovation. Denning and Dunham [2] make a clear distinction between invention and innovation meaning that invention is the idea as such but with the absence of adaptation applied. An innovation on the other hand is an invention that covers the entire way from idea to adaptation and sustainability of that idea making sure that all is done for that idea, artifact or process to make it successful in the intended marketplace. One indication of the power and control over user innovation is that companies like Apple with control over their products from hardware to software throughout the marketing process seem to have better chance than smaller not so well known companies that has to rely on the market allowance or a better chance than companies that uses sub-contracting as a way to produce their product? The WeBIS [5] project is a research attempt started in the spring 2008 that aim to address some of the in this paper mentioned innovation aspects, and also to create a user centric and user innovation driven method; a method to support early decision making, if an idea, product, service is worthwhile going for or if there are too high risk of failure.

8. CONCLUSIONS

Today we monitor and define product quality by measuring defects levels in different ways. This will still be needed but must be complemented by UX quality measurements. The product quality definition needs to be increased and widened to include measurements from the UX area and new quality criteria need to be accounted for with actually higher priority than previous sub-quality criteria. More organizational effort should be spent on developing Metrics and KPI's for monitoring and securing UX product quality.

When we decide to prioritize UX in products a new development approach is needed as well, this to ensure that the intended UX appears in products at the market. On a high level there needs to be a cultural shift into a more UX oriented and UX driven mentality within the whole product development organization. On an organizational level UX quality assurance needs to be established by recognizing and given authority to UX expertise that can secure the total UX product quality in all levels of development. In this paper we suggest UX "guardians", see figure 4 below, for policing the UX throughout the product development.

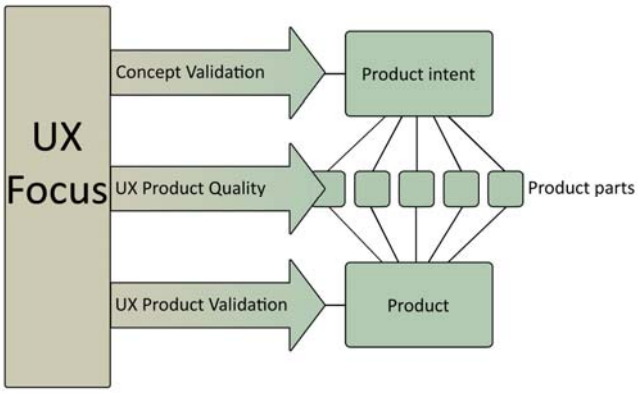


Figure 4. Focusing UX

In order to introduce this approach a cultural widening or increased knowledge among existing SWD roles of e.g. Project managers and Product Managers is needed in order to break the traditional cultural views of monitoring and planning project deliveries. We think it is possible and the suggested approach can be well integrated in traditional SWD processes, but emphasis the need for other competences and milestones than present today e.g. project Managers, UX experts, market experts & Product Managers and technical expertise have to cooperate to a much larger extent than in most large companies in mobile industry today.

9. ACKNOWLEDGEMENTS

We wish to thank Gary Denman from UIQ Technology AB for providing valuable support and insights. This work was partly funded by Vinnova (a Swedish Governmental Agency for Innovation Systems) under a research grant for the project WeBIS [5].

10. REFERENCES

[1] Patrick W. Jordan 2000. Designing Pleasurable Products. CRC Press, Taylor & Frances.

[2] Peter J. Denning and Robert Dunham. Innovation as language action. Communications of the ACM, May 2006/vol. 49, No.5

[3] UIQ Technology. UIQ Technology Usability Metrics, UIQ Technology, http://uiq.com/utum.html. 2008-05-22

[4] International Herald Tribune/Technology and Media. *Iphone pushes mobile makers to think simpler*. By Eric Sylvers. http://www.iht.com/articles/2008/01/09/technology/wireless10.php, 2008-05-22

[5] WeBiS, http://www.webis.se

[6] Juristo, N., Windl, H., & Constantine, L. (2001), "Introducing usability", IEEE Software, 20-21.

[7] Juristo, N., Moreno, A. M., Sanchez-Segura, M. (2007), "Guidelines for Eliciting Usability Functionalities", IEEE Transactions on Software Engineering, Vol. 33, No. 11, November 2007, pp. 744-758.

[8] Kensing, F. *Methods and Practices in Participatory design*. Doctoral Thesis, The ITU University of Copenhagen, Press: ITU, ISBN 87-7949-038-7, 2003.

[9] Hellman and Rönkkö. *Is User Experience supported Effectively in Existing Software Development Processes?* Proceedings of the International Workshop on Meaningful Measures: Valid User Experience Measurement (VUUM 2008). ISDN:978-2-917490-02-0

[10] Blekinge Institute of Technology. *UIQ, Usability test*. 2008 [cited 2008-08-29]; Available from: http://www.youtube.com/results?search_query=UIQ%2C+Usability+test&search_type=&aq=f

[11] Rönkkö, K., Hellman, M., Kihlander, B. and Dittrich, Y., 2004. Personas is not Applicable: Local Remedies Interpreted in a Wider Context, Proceedings of the Participatory Design Conference, PDC '04, Artful Integration: Interweaving Media, Materials and Practice, Toronto, Canada, (July 27-31, 2004), 112-120.

[12] Faith Popcorn's Brain reserve; Available from: http://www.faithpopcorn.com/

[13] Richard Mulholland, UX Magazine; http://www.uxmag.com/strategy/95/fuck-love-brand

Problems of Consolidating Usability Problems

Effie Lai-Chong Law
University of Leicester/ ETH Zürich
LE1 7RH Leicester/ Institut TIK
UK/Switzerland
+44 116 2717302
law@tik.ee.ethz.ch

Ebba Thora Hvannberg
University of Iceland
107 Reykjavik
Iceland
+354 525 4702
ebba@hi.is

ABSTRACT
The process of consolidating usability problems (UPs) is an integral part of usability evaluation involving multiple users/analysts. However, little is known about the mechanism of this process and its effects on evaluation outcomes, which presumably influence how developers redesign the system of interest. We conducted an exploratory research study with ten novice evaluators to examine how they performed when merging UPs in the individual and collaborative setting and how they drew consensus. Our findings indicate that collaborative merging causes the absolute number of UPs to deflate, and concomitantly the frequency of certain UP types as well as their severity ratings to inflate *excessively*. It can be attributed to the susceptibility of novice evaluators to persuasion in a negotiation setting, and thus they tended to aggregate UPs leniently. Such distorted UP attributes may mislead the prioritization of UPs for fixing and thus result in ineffective system redesign.

Categories and Subject Descriptors
H.5.2 [**User Interfaces**]: *Evaluation/Methodology*

General Terms
Measurement, Performance, Experimentation, Theory

Keywords
Usability problems, Merging, Filtering, Consensus building, Downstream utility, Severity, Confidence, Evaluator effect

1. INTRODUCTION
The extent to which UPs identified by different users/analysts overlap seems unpredictable, despite the persistent research efforts of formalizing the cumulative relation between the numbers of users/analysts and UPs ([7], [8], [10]). The practical implication of these concerns is to recruit as many users/analysts as the project's resources allow, thereby maximizing the probability of identifying most, but impossibly all, UPs.

I-USED'08, September 24, 2008, Pisa, Italy

One concomitant procedure of involving multiple users/analysts in usability evaluation is to consolidate UPs identified by different users/analysts to produce a master list. Such a consolidation process can serve two purposes: (i) providing a design team with neat and clean information to facilitate system redesign, and (ii) enhancing the validity of comparing the effectiveness of different (instances of) usability evaluation methods (UEMs). This process consists of two phases [1]: The first step is known as *filtering*, that is, to eliminate duplicates *within* a list of UPs identified by a user when performing a certain task with the system under scrutiny or by an analyst when inspecting it. The second step is *merging*, that is, to combine UPs *between* different lists identified by multiple users/analysts, to retain unique, relevant ones, and to discard unique, irrelevant ones. While such consolidation procedures are commonly practised by usability professionals and researchers, little is known about how it is exactly done and what impact it can have on final evaluation outcomes and eventually on system redesigns, especially when severity ratings play a non-trivial role in the prioritization strategy for UP fixing ([2], [3]).

In the HCI literature, the UP consolidation procedure is mostly described at a coarse-grained level. Nielson [9], when addressing the issue of multiple users/analysts, highlighted the significance of merging different UP lists, but he did not specify how this should be done. Connell and Hammond [1], in comparing the effectiveness of different UEMs, delineated the merging procedure at a rather abstract level. Further, Hertzum and Jacobsen [4] coined the notion of *evaluator effect* that has drawn much attention from the HCI community towards the reliability and validity issues of usability evaluation. Nonetheless, their work focused on problem extraction on an *individual* basis rather than problem merging on a *collaborative* basis. More recently, a tool for merging and grouping UPs has been developed [5], which, however, supports the work of individual evaluators but neglects the collaborative aspect of usability evaluation.

In summary, the actual practice of UP consolidation is largely open, unstructured and unchecked. With the major goals to examine the impact of the UP consolidation process and to understand the mechanism underlying the consensus building process, we have conducted a research study. In this paper we summarize the main findings on the first issue while leaving out the second one as the data are still being analyzed.

2. RESEARCH METHODS
The empirical study was conducted at a university in the UK. Ten students (one female) majored in computer science were recruited. All have acquired reasonable knowledge of HCI and

experience in user-based evaluation through lectures and projects. They were grouped into five pairs. An e-learning platform was usability evaluated (i.e. think aloud) with representative end-users one year ago. Among different types of data collected, we employed for this current study the observational reports written by the experimenter who was present throughout the testing sessions and registered the users' behaviours in very fine detail. We also developed several structured forms to register the participants' findings in the different steps of our study. All the participants had to attend two testing sessions: In the first one they performed *Individual Problem Extraction* and *Individual Problem Consolidation*, and about a week later, they paired up to perform *Collaborative Problem Consolidation.*

2.1 Individual Problem Extraction

Each participant was given the narrative observational reports (printed texts) how the users P1 and P2 performed Task 1 (T1) "Browse the Catalogue" and Task 2 (T2) "Provide and Offer a Learning Resource". For each UP extracted, the participant was required to record in a structured analysis form five attributes:

1. Develop UP identifier with a given format;
2. Provide a UP description as detailed as possible;
3. Select criteria from a given list to justify the UP;
4. Judge the severity level of UP: minor, moderate, severe;
5. How confident the evaluator was that the UP identified was true: 1 lowest – 5 highest;

After completing the analysis form for T1, the participant was asked to apply the same procedure to P1's T2, and then to P2's T1 and T2 (Figure 1). In other words, each participant was required to analyse four sets of data (P1-T1, P1-T2, P2-T1 and P2-T2).

2.2 Individual Problem Consolidation

With the four lists of extracted UPs, the participant was required to filter out any duplicate within the lists and then merge similar UPs, resulting in two sets of UPs (i.e. P1-T1 and P2-T1 as one set; P1-T2 and P2-T2 as another set). Unique UPs identified would be retained or discarded during this process. The participants were asked to record the outcomes in the same form for problem extraction, but they needed to indicate explicitly in the column UP-identifier which UPs were combined. Severity and confidence levels could also be adjusted. No time limit was imposed.

2.3 Collaborative Problem Consolidation

With a break of several days, two participants of a group came together to merge their respective lists of UPs prepared in the individual sessions into a master list. They could access all the materials used in the earlier sessions. They were asked to track every item (i.e., a single UP or combined UPs) in their own consolidated list by recording in a structured form which of the three possible changes was made - merged (with which one), retained or discarded. No time limit was imposed on any of the above procedures. While individual and collaborative problem consolidation basically involved similar sub-tasks, the latter was conducted to observe how the collaborative setting influenced an individual's merging strategies.

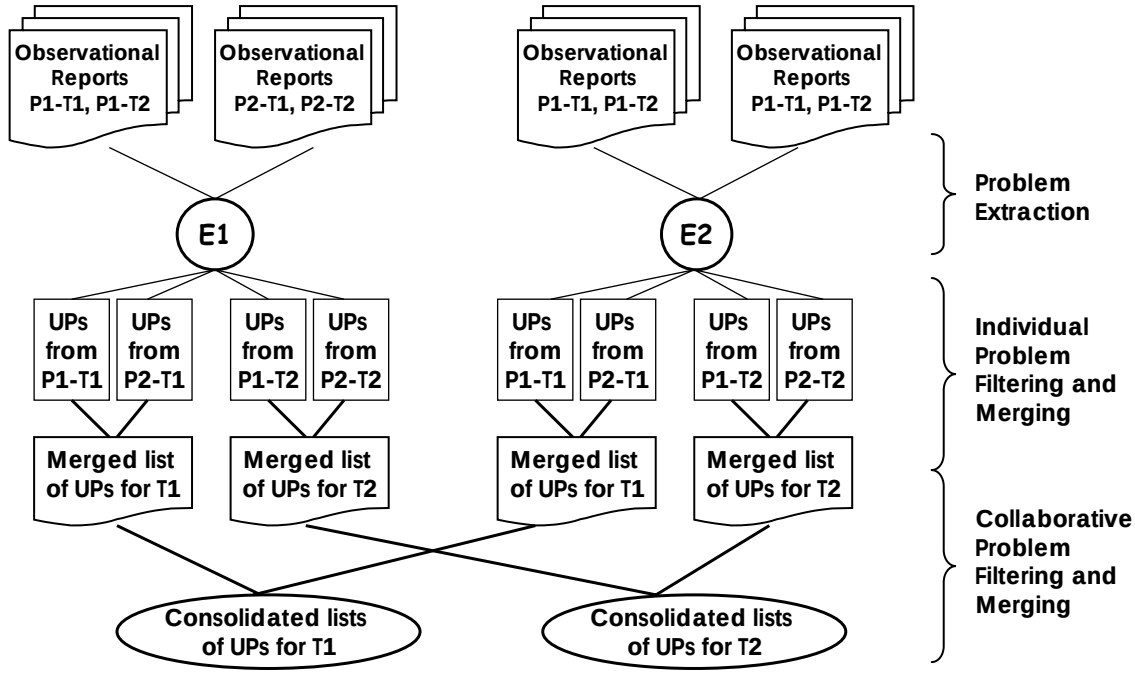


Figure 1: **The workflow of problem consolidating process**

3. RESULTS

3.1 Individual Problem Consolidation

The ten participants extracted from the observational reports altogether 98 and 81 UPs for T1 and T2 over the two users (P1 and P2), respectively. Furthermore, they individually consolidated their UPs. Table 1 shows the extent to which the participants merged, discarded and retained the UPs extracted.

Table 1. **Distribution of outcomes in the individual filtering**

	Merged	Discarded	Retained
T1	39%	13%	48%
T2	51%	10%	39%

For the merged and retained UPs, there were changes in severity ratings and/or confidence levels or no changes at all. To simplify the results, we collapse different degrees of increase/decrease (e.g. minor → moderate/severe or vice versa) into INC or DEC, respectively, and denote no change with SAME.

Table 2. **Severity/confidence changes in merged UPs (Indiv.)**

	Severity		Confidence	
	T1	T2	T1	T2
DEC	4 (10%)	3 (7%)	6 (15%)	4 (10%)
SAME	20 (53%)	29 (71%)	15 (40%)	18 (44%)
INC	14 (37%)	9 (22%)	17 (45%)	19 (46%)

The same notations are applied to the confidence level. In merging the UPs, the participants tended to increase the severity ratings by one or two degrees (i.e. 37% for T1 and 22% for T2; Table 2). In contrast, it seemed they did not bother to adjust the severity of the UPs retained (i.e., 2% and 6% for T1 and T2, respectively). In the post-filtering interviews, most participants explained that when a UP was both identified in P1 and P2, it could indicate that the UP was more severe than originally estimated and that it rectified the realness of the problem, thereby boosting their confidence. Interestingly, the correlation between the original severity ratings and confidence levels (r = 0.25, n = 179, p = 0.001) was found to be significant, implying that the participants were more confident that they judged the severe UPs correctly but less so when judging minor or moderate UPs. In contrast, the correlation between the changes in both variables (r = 0.19, n = 26) was insignificant. In other words, changing the severity of a UP does not imply that the participant has become more (or less) confident about the realness of the UP.

3.2 Collaborative Problem Consolidation

In comparison, the participants demonstrated an even stronger tendency to merge UPs in a collaborative setting (Table 3), which is higher than that (cf. 39% vs. 81% for T1; 51% vs. 77% for T2) observed in an individual session. The participants tended to negotiate at a higher abstract level where broad problem types can accommodate a variety of problem instances, thus mitigating direct confrontation with partners over controversial similarities. The participants tended to receptive to their partners' proposals, especially when the agreement thus reached would not cause any actual economic or personal gain (or loss). When negotiating to merge or retain UPs, the participants adjusted the severity and confidence ratings. For each aggregate we averaged the ratings of the original set of to-be-merged UPs and compared it with the corresponding final ratings. Table 4 displays the results for the merged UPs. Similar patterns to Table 1 were observed.

Table 3. **Distribution of outcomes in the collaborative filtering**

	Merged	Discarded	Retained
T1	81%	10%	9%
T2	77%	15%	8%

Table 4. **Severity/confidence changes in merged UPs (collab.)**

	Severity		Confidence	
	T1	T2	T1	T2
DEC	2 (5%)	2 (7%)	2 (5%)	3 (11%)
SAME	23 (52%)	16 (57%)	22 (50%)	13 (46%)
INC	22 (43%)	10 (36%)	19 (45%)	12 (43%)

4. DISCUSSION

The empirical findings of this study enable us to draw comparisons between the individual and collaborative UP consolidation processes, which presumably involve the core mechanism of judging similarity among UPs. One notable distinction is the lenience towards merging in the collaborative setting, as shown by the high merging rate. Indeed, quite a number of participants combined UPs that had not been merged in their individual sessions to merge with their partners'. It may be attributed to social pressure that coerces them to reach consensus. The data indicate that as a result of the merging process, severity ratings of UPs tend to inflate and the number of UPs tends to deflate excessively in the collaborative setting. In contrast, confidence levels, in which personal experience plays a role, do not fluctuate with the merging process. Previous research studies indicate that severity ratings influence how developers and project managers prioritize which UPs to fix ([3], [6]). Invalid severity ratings presumably lead to the fixing of less urgent UPs. Consequently, the quality of the system may still be undermined by more severe as well as more urgent UPs.

The implication for the future work is to look into relevant theories on similarity (an age-old issue), communication, and social interaction. Further, we aim to extend our empirical studies by systematically comparing merging through negotiation (i.e. the consolidation procedure is to be implemented by a group of two or three usability specialists or a group of developers or an integrated team) versus merging through authority (i.e. only one person-in-charge is to combine different lists of UPs). The quality of the consolidated usability outcomes will be compared, thereby enabling us to identify valid and reliable methods for consolidating UPs and to develop objective measures of the cost-effectiveness of such methods. Findings thus obtained will also contribute to our ongoing research endeavour on downstream utility.

5. REFERENCES

[1] Connell, I., & Hammond, N. (1999). Comparing usability evaluation principles with heuristics: Problem instances vs. problem types. *Proc. INTERACT 1999*.

[2] Hassenzahl, M. (2000). Prioritizing usability problems: data-driven and judgement-driven severity estimates. *Behaviour & Information Technology, 19*(1), 29-42.

[3] Hertzum, M. (2006). Problem prioritization in usability evaluation: From severity assessments toward impact on design. *International Journal of Human Computer Interaction (IJHCI), 21*(2), 125-146.

[4] Hertzum, M., & Jacobsen, N.E. (2003). The evaluator effect: A chilling fact about usability evaluation methods. *IJHCI, 15(*1).

[5] Howarth, J. (2007). *Supporting novice usability practitioners with usability engineering tools*. PhD thesis (VT).

[6] Law, E. L.-C. (2006). Evaluating the Downstream Utility of User Tests and Examining the Developer Effect: A Case Study. *International Journal of Human Computer Interaction* (IJHCI), 21(2), 147-172.

[7] Law, E. L-C., & Hvannberg, E. T. (2004). Analysis of combinatorial user effect in international usability test. *Proc. CHI 2004*

[8] Lewis, J.R. (1994). Sample sizes for usability studies: Additional considerations. *Human Factors, 36*(2), 368-378.

[9] Nielsen, J. (1994). Heuristic evaluation. In J. Nielsen & R.L. Mack (Eds.), *Usability inspection methods*. New York: Wiley

[10] Virzi, R.A. (1992). Refining the test phase of usability evaluation: How many subjects is enough? *Human Factors, 34*(4), 457-468

User Experience Metric and Index of Integration: Measuring Impact of HCI Activities on User Experience

Anirudha Joshi
Indian Institute of Technology Bombay
Mumbai 400076, India
+91 9820345569
anirudha@iitb.ac.in

Sanjay Tripathi
Tech Mahindra Ltd.
Pune 411004, India
+91 9922963298
stripathi@techmahindra.com

ABSTRACT
We propose two metrics to demonstrate the impact integrating human-computer interaction (HCI) activities in software engineering (SE) processes. User experience metric (UXM) is a product metric that measures the subjective and ephemeral notion of the user's experience with a product. Index of integration (IoI) is a process metric that measures how integrated the HCI activities were with the SE process. Both metrics have an organizational perspective and can be applied to a wide range of products and projects. Attempt was made to keep the metrics light-weight. While the main motivation behind proposing the two metrics was to establish a correlation between them and thereby demonstrate the effectiveness of the process, several other applications are emerging. The two metrics were evaluated with three industry projects and reviewed by four faculty members from a university and modified based on the feedback.

Categories and Subject Descriptors
D.2.8 [**Software Engineering**]: Metrics – *process metrics, product metrics.*

General Terms
Measurement, Design, Human Factors

Keywords
User experience metrics, HCI-SE integration.

1. INTRODUCTION
Large contracted software development companies with tens of thousands of employees are often involved in a wide variety of software development projects, often in an off-shore mode. Managers of user experience (UX) groups in such companies need to track progress of each project and ensure the quality of deliverables. They are often required to juggle across projects a limited resource – the time of their best UX professionals. While there are numerous usability metrics to evaluate specific projects, there are few that allow organizations to easily track progress across projects. Our first proposal is product metric (UXM) that measures the user's experience with a product. The objective is to provide a summary measure of the user experience of a product that is independent of the domain, context of use, platform or the software development process, so that the manager is able to make judgments across projects.

Another challenge faced by UX groups is integrating HCI in established SE processes. The field of HCI has a large amount of literature on user-centred design methods, techniques and processes [1], [3], [17], [23] etc. These proposals are excellent demonstrations of how user centred design can result in improved user experience design. Unfortunately, there continue to exist major gaps between HCI and SE, in academics, literature and industrial practice. The IFIP working group 2.7/13.4 on User Interface Engineering remarks that 'there are major gaps of communication between the HCI and SE fields: the architectures, processes, methods and vocabulary being used in each community are often foreign to the other community' [7]. For example, while SE literature admits that communication with the customer is an unsolved problem, even recent editions of standard text books on software engineering such as [13] and [20] do not suggest use of established user study techniques like [1] during communication. Example projects shown in [13] and [20] seem to take HCI design lightly, prematurely and without following any process. A detailed critique of SE literature from an HCI perspective is presented in [11]. There have been several proposals to integrate HCI in SE process models (for example, [5], [12], [21]) but none have become popular in the industry. One reason could be concerns about return on investments. Though there is plenty of evidence of the a return on investment of usability activities in general [2], there is no direct evidence that shows that better integration of HCI activities in SE processes will lead to better products at less cost.

Contracted software companies often promise a level of process compliance to their clients. UX managers need summary measures of process compliance of their projects to ensure that the company lives up to its promise. One way would be to measure how integrated were the HCI activities with SE processes. Our second proposal is a process metric (IoI) that would be one such measure. If validated, IoI and UXM can also be used to demonstrate the return on investment on integration of HCI with SE – if higher IoI consistently leads to higher UXM, it makes sense to invest in better integration of HCI with SE.

I-USED'08, September 24, 2008, Pisa, Italy

The main objective of this paper is to share with other metrics researchers the lessons we have learned from attempting to incorporate UXM and IoI in live industrial projects.

We begin with an introduction to different attempts done in recent years on applying metrics in HCI. Next, our metrics proposals are described. Finally, the evaluation methodology used so far to analyse the results of study is described.

2. METRICS IN HCI

Metrics are thoroughly discussed in software engineering literature. Fenton and Pfleeger [4] describe measurement as "the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules". Pressman [20] highlights the subtle difference between measurement and metrics – measurement occurs as the result of collection of one or more data points, while a software metric tries to relate the measures in some way. IEEE Standard Glossary [6] defines a metric as "a quantitative measure of the degree to which a system, component or process possesses a given attribute".

Though the word 'metric' itself is seldom used in the practice of usability, several measures are often used. Seconds taken to withdraw money from the ATM, the number of keystrokes to enter a word in a complex script, the number of errors made to complete a banking transaction or the percent of users who abandon the shopping cart on checkout are all examples of quantitative measures of the user experience afforded by the product. However, none of these are summary measures that can be used for apple-to-apple comparison across projects varying in domains, platforms and contexts. While several research papers talk about metrics related to usability and HCI, this paper only focuses on those that give a summary measure.

Lewis [14] used a rank based system of assessing competing products based on user's objective performance measure and subjective assessment. The metric is useful for a relative comparison between like products with similar tasks but it does not result in a measure that can be used across unlike products.

McGee [18] derives a single usability scale across users by including additional reference tasks. However McGee does not suggest how to derive a single measure for usability from measures for the different tasks. Further, this work is completely dependent on the technique of usability evaluation. This is not always practical in a global contracted software company striving to move up the HCI maturity ladder. The other limitation of this method is that it relies only on perception of users and ignores perspectives of other stakeholders, particularly the goals of business stakeholders.

Lin et al [15] propose the Purdue Usability Testing Questionnaire based on eight HCI considerations to derive a single weighted average score for usability. While the approach does lead to a single usability score, the selected eight considerations (compatibility, consistency, flexibility, learnability, minimal action, minimal memory load, perceptual limitation and user guidance) seem to be a mix of usability goals and heuristics that achieve those goals. Secondly, the weightage for parameters is to be assigned by the evaluator during the evaluation without consulting stakeholders. Thirdly, the listed eight considerations and the questions listed under each of them seem to be limiting and do not leave room for project-specific goals (e.g. "do it right the first time").

Sauro et al [22] proposed a 'single, standardized and summated' usability metric for each task by averaging together four standardized values for task time, errors, completion and satisfaction. Their calculation however is based on the equal weightage. Tasks, domains, users, contexts and platforms vary a lot and it does not make sense to give equal weightage in all contexts. Moreover, the metric ignores some aspects such as learnability and ease of use, which might be important in some contexts.

Measuring the wider notion of user experience (as opposed to usability) is relatively new concept in HCI and is attracting attention of the academic as well as the industrial world. Usability parameters are typically related to the processing of information or completion of tasks. However, affective reactions and emotional consequences play important role in the overall user experience [16]. In some product contexts, we may need to consider visceral, behavioural and reflective elements [19], aesthetics [25], enjoyment [10] and creativity [24].

None of the summary metrics mentioned above measure the experience of a product with reference to all user and business goals relevant to a product. Many are too complex to compute practically on an on-going basis in the industrial practice. They lack the flexibility required to serve the needs of a wide variety of projects or to mature with the UX group. And finally, there seems to be almost no work on measuring integration of HCI activities with SE processes.

3. USER EXPERIENCE METRIC

Fenton and Pfleeger [4] emphasise the importance of goals in metrics: "a measurement program can be more successful if it is designed with the goals of the project in mind". User experience goals are very important in driving the design of interactive products. They help speed up the design process, make the design activity more tangible and help evaluate the design. User experience goals can be understood easily, even by non-UX-professionals, and they have a significant overlap with business goals. Stakeholders outline the user experience goals and UX professionals fine-tune them on the basis of their knowledge and findings from user studies. User experience goals are (and should be) available early in a project – another plus when it comes to metric calculation in a practical situation.

We propose user experience metric (UXM), a product metric that measures the quality of user experience. The motivations are:

- to measure the user experience of a product in reference to its user experience goals
- to develop a flexible metric that can be applied across a variety of projects, irrespective of domain, context, platform, process model or usability technique
- to develop a flexible metric that that will mature with the organization
- and to compute the metric with minimal additional costs and efforts.

UXM is product metric on a scale of 0-100, where 100 represents the best user experience possible and 0 represents the worst.

UXM consists of these distinctions:

Goals: High-level user experience goals guide the design of interactive systems.

Parameters: Each high-level user experience goal is broken down into a set of parameters that help the designer to achieve and measure the higher-level goal in a direct manner. For example Learnability can have parameters like Conceptual model clarity, Language understandability, Minimal training time, Consistency with earlier version etc.

Weightage: Each goal has a weightage between 0-5 where 0 represents that the goal is not important, 2 represents the typical importance and 5 represents that it is very important. Further, each parameter under a goal also has a weightage attached.

Score: Each parameter has a score between 0-100, where 0 represents the worst possible user experience on account of that parameter and 100 represents the best possible user experience.

Guidelines: The purpose of the guidelines is to help evaluators assign a score to the parameters. Guidelines let the goal-setters express themselves better and interpret goals for the context of a project – e.g. *"'Consistency with earlier version' means all frequent and critical tasks from earlier version are unchanged."* Further, guidelines tell the evaluators when to assign which score: *"The interface clearly communicates the correct conceptual model. Strongly agree = 100, Weakly agree = 75, Neutral = 50, Weakly disagree = 25 and Strongly disagree = 0".*

Goals and parameters are a way to express the desired user experience and performance in the design. Though expressing user experience goals is a common activity in HCI design, there is no standard way of doing so. There are many ways to describe the high level user experience goals. For example, ISO 9126-1 describes usability in terms of understandability, learnability, operability and attractiveness [8]. ISO 9241 on the other hand defines usability as the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use [9]. Shneiderman [23] describes goals for user interface design in terms of five human factors central to evaluation: time to learn, speed of performance, rate of errors by users, retention over time and subjective satisfaction.

We adopt goals from Shneiderman as the high-level user experience goals for a product and express them as: learnability, ease of use, speed of use, error-free use, retention and subjective satisfaction. We added 'ease of use' to the list as we thought that it is an important user experience goal distinctly different from the other factors such as speed. We also generalized the expressions of some of the goals. For example we express 'time to learn' as 'learnability' since it allows for expression of other concerns such as understandability of language or clarity with which the interface communicates the conceptual model in addition to the time users take to learn the interface. We believe that this allows the designers to express a wider set of goals.

Our proposal of an initial set of goals and parameters are listed in Table 1. However, we must highlight that this is not a prescribed, exclusive set. We give the evaluators and stakeholders freedom to derive additional, relevant parameters that express their goals. Goals and parameters could be added, removed or combined according to the context of the project, the needs of the users, the vision of the stakeholders and UX professionals and to fit the terminology that the product development team is familiar and comfortable with. The initial list is meant to give users a starting point, while the flexibility is meant to allow the metric to mature with the experience of the organization using UXM.

As Shneiderman [23] states, 'a clever design for one community of users may be inappropriate for another community' and also, 'an efficient design for one class of tasks may be inefficient for another class'. Weightages express the relative importance of goals and parameters in the context of a project. For example, a product meant to be used several times a day by a call-centre agent is likely to have higher weightage for 'speed of use'. A one-time use product like a web site for visa application for a tourist might insist on learnability and error-free use. On the other hand, a life-critical product to be used in a operation theatre is likely to rate highly error-free use and may sacrifice learnability. A game would perhaps give highest weightage to 'subjective satisfaction'.

The evaluators and stakeholders assign the weightage to set the context of the project. Goal-setters should be aware that while it may be tempting to set a high weightage to each goal, it may not be necessary, practical, or even possible to achieve such a design. The weightages should reflect the priorities of the stakeholders and users. The weightage would also help prioritize usability evaluation activity – the highest rated goals and parameters must be evaluated more thoroughly, while the lower weighted goals could be perhaps evaluated by a discount method.

The process for computing UXM for a product has these steps:

Goal Setting: Early in the project, typically just after user studies but before design, an HCI professional and stakeholders identify goals and parameters for each goal, assign weightage to each goal and parameter and decide evaluation guidelines for the parameters.

Scoring: Immediately after a usability evaluation, one or more independent HCI professionals assign a score to each parameter of each goal. The usability evaluation could be either user-based (e.g. a usability test) or review-based (e.g. heuristic evaluation).

UXM Calculation: UXM is the sum of the weighted average of the scores of all goals. UXM = $\sum (W_g \times S_g / \sum W_g)$, where W_g is the weightage of a goal and $S_g = \sum (W_p \times S_p / \sum W_p)$ where W_p is the weightage of a parameter and S_p is the score of that parameter.

Scores of some of the parameters can be directly linked to the findings of the usability evaluations (for example, % of users who did not make errors while doing benchmark tasks, or % of users who thought the product was engaging). Other parameters may not be so easily linked numerically (e.g. conceptual model confusions discovered during a think aloud test or problems identified during heuristic evaluation). In such cases, evaluators consider the guidelines and their own experience to arrive at a score for each parameter. If there are multiple evaluators, a simple average across evaluators is deemed to be the score for a given parameter. Multiple evaluators assign scores independently to begin with. If there is a significant variation in their scores, the evaluators discuss the parameter and have the opportunity to converge their scores before the average is calculated.

In case of applications with multiple user profiles, separate UXM should be calculated for each profile. Calculation of UXM could be a part of every usability evaluation of the project, but we recommend that it should certainly be a part of the final usability evaluation, beyond which no design changes are planned for.

Table 1. An example UXM calculation

Goals	Weightage	Score
Learnability	4	78.6
Speed of use	2	77.1
Ease of use	3	65.6
Error free use	3	67.5
Retention	1	75.0
Subjective satisfaction	2	78.6
UXM Value		73.3

Goal Parameters	Weightage	Score
Learnability		78.6
Conceptual model clarity	3	75
Language understandability	0	0
Minimal learning time	5	75
Consistency with earlier version	1	50
Visibility of choices and data	4	100
Consistency with other products	1	50
Speed of use		77.1
User control and freedom	3	75
No memory and cognitive load	4	100
Internal consistency	4	75
Customization	0	0
Automation and shortcuts	1	0
Ease of use		65.6
Minimal user task load	5	75
Automation of routine tasks	3	50
Error free use		67.5
Good feedback	4	75
Error tolerance	3	50
Error recovery	3	75
Retention		75.0
Retention	3	75
Subjective satisfaction		78.6
Visceral appeal	2	75
Behavioural appeal	4	75
Reflective appeal	1	100

Table 1 shows an example UXM calculated by a team for an Indic text input interface for novice users on a mobile phone. The team was given a default set of higher level goals, parameters and example parameter evaluation guidelines. The team first assigned the weightages for higher level goals (shown in the second column of the upper part of Table 1). Next, they broke down goals into parameters and assigned them weightages (shown in the second column of the lower part of Table 1). The team's experience from previous Indic text input projects and mobile phone projects helped them arrive at these weightages.

The team then evaluated the interface and assigned scores to each parameter (shown in the third column of the lower part of Table 1). A weighted average of parameter scores gave the score for each goal (shown in the light grey cells of the third column of lower part of Table 1). A weighted average of the goal scores gave the UXM value (shown in the dark grey cells of the upper part of Table 1). Parameter evaluation guidelines have not been listed in this paper due to space constraints.

4. INDEX OF INTEGRATION

We conceive Index of Integration (IoI) as an empirical process metric, nominally on a scale of 0-100, where 100 represents the best possible integration of HCI activities in the software development activities and 0 represents the worst. The metric consists of these distinctions:

Software Engineering Phases: These are the broad phases as described in a software engineering process model.

HCI Activities: These are prescribed for each phase of the software engineering process model.

Weightage: Each HCI activity will have a given weightage on the scale of 0-5 where 0 represents that the activity is not important, 3 reflects the typical importance in most projects and 5 indicates that this activity is very important in the context of that project.

Score: Each activity has a score associated with it. The score is given on a rating of 0-100, where 100 represents the best case situation where the activity was done in the best possible manner, in the most appropriate phase of software development and with the best possible deliverables. 0 represents the worst case situation where the activity was not done at all.

Activity evaluation guidelines: These spell out considerations that help the evaluation of each activity.

Software engineering phases have been extensively described in literature. For example, the phases of the waterfall process model are Communication, Planning, Modelling, Construction and Deployment [20].

On the other hand, no widely accepted industry-wide specifications of HCI activities for given SE phases have emerged so far. But there have been a few proposals. For example, [12] prescribes that the Communication phase of the waterfall model should have these HCI design activities: Contextual user studies and user modelling, Ideation, Product definition and Usability evaluation and refinement of product definition. Figure 1 summarizes the HCI activities suggested for the waterfall model phases based on these recommendations.

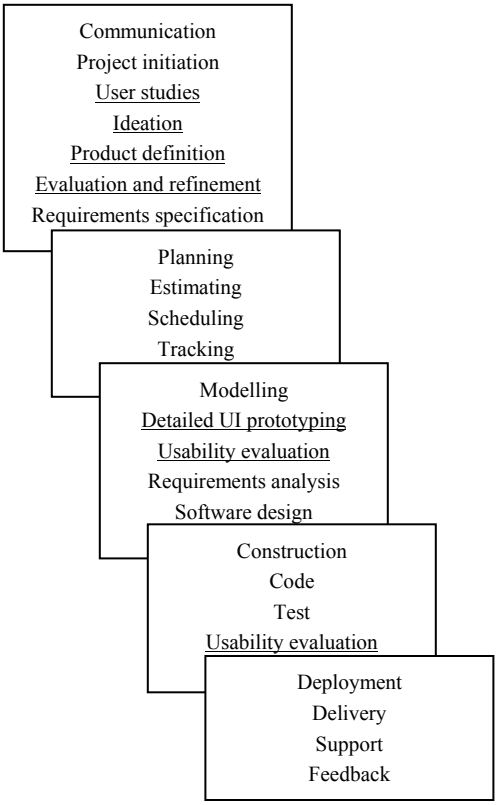


Figure 1. Integration of HCI activities with the phases of the waterfall model [12]. The HCI activities corresponding to each phase have been underlined.

Weightage of some HCI activities could vary within a range in the context of a project. For example, if the domain or users are unknown to the UX team, it may be very important to do contextual user studies in the communication phase (weightage = 4). On the other hand, if the UX team is already very familiar with the context and the domain and if they have a lot of experience designing similar products, it may be less important (weightage = 2). Table 2 summarises loosely recommended weightages for HCI activities for the waterfall model.

Guidelines may define the techniques used to carry out activities, the skills and experience levels of the people doing the activities, the deliverables and other parameters that affect the fidelity of the activity. For example, following are the guidelines for the activity of Contextual user studies and user modelling in Table 2:

1. *Organizational data gathering and user studies were done before requirements were finalised*
2. *User studies were done in the context of the users by the method of contextual inquiry*
3. *User studies were done with at least 10 users in each profile*
4. *User studies were done by people with experience in user studies in a similar domain in at least 2 projects*
5. *The findings including user problems, goals, opportunities and constraints were analyzed, documented and presented in an established user modelling methodology such as personas, work models, affinity diagram or similar*
6. *Competitive products and earlier versions of the product were evaluated for potential usability problems by using discount usability evaluation methods such as Heuristic Evaluation or better*
7. *User experience goals were explicitly agreed upon before finalizing requirements*

100 = All the above are true; 75 = At least five of the above are true, including 7, 50 = At least three of the above are true, including 7; 25 = At least two of the above are true, 0 = None of the above are true

Table 2. An example IoI calculation

SE Phases and HCI Activities	Weightage	Score
Communication		
Contextual user studies and user modelling	4	39
Ideation	2	6
Product definition	3	75
Usability evaluation and refinement of product definition	1	63
Modelling		
Detailed UI Prototyping	5	53
Usability Evaluation and Refinement of the Prototype	4	44
Construction		
Development support reviews by usability team	3	29
Usability evaluation (summative)	1	46
IoI Value		45

The process for computing IoI for a project has these steps:

Company HCI Process Prescription: The HCI group in the company prescribes what HCI activities should be done in which phase of SE process, expected deliverables from each activity, suggested weightages for each activity and suggested activity evaluation guidelines. As it often happens, a contract software development company may follow not one SE process, but several. In that case the HCI design process needs to be integrated with each SE process. The prescribed process also suggests a weightage for each HCI activity and guidelines to score each activity.

Project HCI Process Definition: After getting a project brief, the UX professional fine-tunes the weightages for the prescribed HCI activities after considering the domain, the users and project context. For example, if the HCI team has recently done contextual user studies in the same domain for a similar product and is already very knowledgeable about the context, then he may reduce the weightage of contextual user studies. On the other hand if the team is less knowledgeable, he may increase the

weightage. He should consult colleagues in the development team and business stakeholders before finalizing the weightage.

Process Evaluation: After the project is over, a group of independent UX professionals review the HCI activities and evaluate them for process compliance and give a score for each activity on a scale of 0 to 100. They may reduce the score if an activity was done with lower fidelity, resulted in poor quality deliverables or was done later than prescribed. In case of multiple evaluators, an average across evaluators is deemed to be the score.

IoI Calculation: The metric is found by computing the weighted average of the scores of all activities: $IoI = \sum (W_a \times S_a / \sum W_a)$, where W_a is the weightage for a particular HCI activity, S_a is the score (from 0-100) for that activity. In case there is a lot of divergence in scores of a particular HCI activity, the activity is discussed and reviewers are given a chance to change their score before an average is taken.

Table 2 shows calculation of IoI for an example project. First, senior UX professionals defined the HCI activities, activity weightages and evaluation guidelines that the company should be following. Then a project that had recently ended was selected for retrospective review. The project manager and the UX professionals working on the project fine-tuned the weightages for the project context. The second column of Table 2 contains these weightages. A group of reviewers comprising of some project insiders and outsiders reviewed and rated the HCI activities in the project and its IoI was calculated. The third column of Table 2 contains the average scores assigned to each activity by the reviewers.

Guidelines for evaluating all HCI activities listed in Table 2 have been created. More guidelines for evaluating HCI activities as part of extreme programming process have been created as well. Both have been omitted here due to space constraints.

5. METRICS EVALUATION

The authors evaluated the metric in two ways. First, UXM and IoI metrics were computed for retrospectively projects in Tech Mahindra, a large contracted software development company. In each case, the metrics computation was done by HCI professionals from the project, independent HCI professionals and project stakeholders. At the end of metrics computation, feedback was taken from participants of each project about the metrics.

Second, the metrics were presented to a group of faculty members from a reputed university and their comments were noted. Three of these were faculty members from the computer science discipline. One was a faculty member from a design school who is also an expert in cognitive psychology.

5.1 Findings

It typically took about 3 hours to compute both IoI and UXM for each project. The time included explaining the two metrics, weightage assignment and scoring. This seemed to be optimum time, longer meetings were difficult to schedule. The projects performed similarly in IoI and UXM scores – the one project that had a high UXM value also had a high IoI value. Participants, particularly project stakeholders, were at home with the activity of metric calculation. To them, the activity seemed to bring HCI closer to SE. It seemed to create lot of buy-in for HCI activities from the project stakeholders. One project stakeholder said *"I never thought we could think so much [about user experience]."* The activity seemed to be more successful in projects where several stakeholders from the project participated as it stimulated discussion among stakeholders. While the participants appreciated the organizational perspective, the metrics seemed of less use to the projects as the projects were already over. Participants suggested that metrics should be calculated mid-way through the project while course correction was still possible.

Specifically, UXM helped the HCI designers and project stakeholders to make goals explicit. One HCI designer remarked, *"Had we done this earlier, I would have known where to focus."* The teams usually added a few goal parameters (typically 2-3 per project) and adjusted weightage to suit UXM to their project. They confirmed that this flexibility is indeed desirable. Though parameter evaluation guidelines for UXM helped, more details were desired. Participants did not make changes to the parameter evaluation guidelines except when new parameters were added. Giving examples of HCI goals (learnability, ease of use etc.) helped participants to set goal parameters and weightages. One stakeholder remarked: *"without these inputs it would have been difficult to [assign weightage and scores]."*

In case of a few UXM parameters, divergent scores emerged for some parameters in each project. Usually variations happened in parameters where the evaluation guidelines were not understood well or were interpreted differently by evaluators. In such cases, it was felt, that it was better to let participants discuss the parameter and change ratings to converge scores if they so desire. Reducing the number of steps in scoring a parameter (e.g. 0-25-50-75-100) helped reduce variation among scores. More detailed UXM parameter evaluation guidelines with examples will further help in reducing divergence.

Computing IoI was useful for project stakeholders as they could see the importance of HCI activities in the SE context. The HCI activities integrated in SE process models were acceptable as suggested. Though they were explicitly prompted, none of the project stakeholders wanted changes to the prescribed HCI activities, their weightage or evaluation guidelines. An important feedback was need for process models specifically targeted to redesign projects. Process models typically discuss new product development. Given that many industry projects are *"next version of X"* type, process models must be specifically adapted for them.

Walking through the activity evaluation guidelines helped in scoring as all stakeholders were not aware of all HCI activities. It was felt that IoI should be computed before computing UXM as this minimizes bias.

The metric descriptions presented in this paper are a result of iterative modifications that reflect the feedback and lessons learnt.

6. DISCUSSIONS

It is important to discuss the limitations and risks of the two metrics proposed. Both UXM and IoI are summary measures that leave out much information. They allow a drill-down to constituent components, but do not point to specific problems or give suggestions for improvement. But summary measures are useful in many contexts, particularly for comparison across

projects. Such comparisons can help UX groups understand what works and what doesn't and improve performance year-on-year.

Perhaps most important limitation of UXM comes from the ephemeral nature of a 'user experience'. Any attempt to numerically embody such an abstract phenomenon is bound to be subjective and measures could differ according to the interpretation of the evaluators. Further, large companies are involved in software development projects for many clients, across domains, platforms, users, use contexts and task complexity, frequency and criticality.

Finally, there is a risk that because UXM measures are low cost, organizations may be tempted to sacrifice all user-facing activities (such as usability tests or field studies) in its favour. We do not recommend this at all. The purpose of UXM is not to replace these established methods but to supplement them and to help them mature.

In spite of these limitations, we believe that UXM is useful. UXM shows the extent to which user experience goals were achieved in a particular project. We found that breaking up abstract notions of user experience into specific goals and parameters helped evaluators focus on one issue at a time and reduced the subjectivity in measurement. Making the evaluation criteria explicit and averaging across several evaluators further reduced the subjectivity in judgement. The risk of variety in products (the apples-and-oranges risk) was partly mitigated by selecting goal-parameters relevant to each project and giving custom weightage to each parameter.

The main limitation of IoI is that it does not measure the absolute process quality of the project, rather how compliant was a project to the prescribed process. There are no widely-accepted integrated process models at this stage. Yet, IoI in conjunction with UXM may be used to verify the effectiveness of new process model proposals. If UXM and IoI are correlated, the new proposal seems acceptable. On the other hand if the UXM and IoI do not show a correlation, it questions the prescribed process model.

UXM and IoI have an organizational perspective and make more sense while looking across hundreds of projects rather than within each project individually. They have very low additional overheads on the process and are easy to integrate in the process.

Overall feedback indicates that UXM and IoI are useful and practical in evaluating products and processes. There was a lot of buy-in from project stakeholders calculating metrics as there was a lot of willingness to track, control the user experience of the product. The aspects that metric calculation was light-weight and independent of specific usability methods were particularly liked.

7. FUTURE WORK

In future, we plan to use metrics prospectively throughout the duration of projects and demonstrate their usefulness during the project. We will be building more elaborate tools and guidelines to improve the consistency of weightages and scores. We also propose to do a rigorous validation of the two metrics in experimental and industrial situations.

In its current form, UXM goals, parameters and weightages have to be chosen on the basis of experience of individuals. However, it is possible to design tools in future that will collate experience of several practitioners to help in choices of future goals, parameters and weightages. A similar tool for IoI can also evolve the specification of processes.

8. ACKNOWLEDGMENTS

We thank the anonymous reviewers for invaluable comments on improving our presentation of this material. In particular, we thank Ved Prakash Nirbhay and Deepak Korpal from Tech Mahindra for allowing us to interact with their team members while testing the metrics in different software projects. We also thank members of User Interaction Design Group at Tech Mahindra Ltd. for participating in User Experience metrics evaluation. We also thank Pramod Khambete for his continuous support and appreciation. We thank Prof. NL Sarda, Prof. UA Athavankar, Prof. Umesh Bellur and Prof. S Sudarshan for their continuing guidance and suggestions in developing the two metrics.

9. REFERENCES

[1] Beyer, H., Holtzblatt, K., Contextual Design: Defining Customer Centered Systems, Morgan Kaufman (1998)

[2] Bias, R., Mayhew, D. (Eds), Cost-Justifying Usability, Second Edition: An Update for the Internet Age, Morgan Kaufmann (2005)

[3] Cooper, A., Riemann, R., About Face 2.0 the Essentials of Interaction Design, Wiley (2003)

[4] Fenton, N.E., Pfleeger, S.L., Software Metrics – A Rigorous and Practical Approach, Thomsan Brooks/Cole (2002)

[5] Göransson, B., Lif, M., Gulliksen, J., Usability Design – Extending Rational Unified Process with a New Discipline. International Workshop on Interactive Systems Design, Specification, and Verification (2003)

[6] IEEE Standard Glossary of Software Engineering Terminology, IEEE, 1993

[7] IFIP working group 2.7/13.4 on User Interface Engineering, Bridging the SE & HCI Communities: http://www.se-hci.org/bridging/index.html (2004), accessed August, 2008

[8] International Organization for Standardization, ISO/IEC 9126-1:2001 Software Engineering - Product Quality (2001)

[9] International Organization for Standardization, ISO 9241-1:1997 Ergonomic requirements for office work with visual display terminals (VDTs) (1997)

[10] Jordan, P. W., Designing pleasurable products, Taylor & Francis (2000).

[11] Joshi, A: HCI in SE Process Literature, Indo-Dan HCI Research Symposium, IIT Guwahati (2006)

[12] Joshi, A., Sarda N.L.: HCI and SE: Towards a 'Truly' Unified Waterfall Process. HCI International '07 (2007)

[13] Kroll, P., Kruchten, P., The Rational Unified Process Made Easy, Pearson Education (2003)

[14] Lewis, J., A Rank-Based Method for the Usability Comparison of Competing Products. Human Factors and Ergonomics Society 35th Annual Meeting 1312--1316 (1991)

[15] Lin, H. Choong, Y. Salvendy, G. A proposed index of usability: a method for comparing the relative usability of different software systems. Behaviour & Information Technology (1997)

[16] Mahlke, S., Understanding users' experience of interaction, in Marmaras, N., Kontogiannis, T., Nathanael, D. (Eds.), Proc. EACE '05 (2005). 243-246.

[17] Mayhew, D., The Usability Engineering Lifecycle: A Practitioner's Handbook for User Interface Design; Morgan Kaufmann; 1998

[18] McGee, M., Master usability scaling: magnitude estimation and master scaling applied to usability measurement, Proc. CHI'00, ACM Press (2004) 335-342.

[19] Norman, D.A., Emotional Design: Why We Love (or Hate) Everyday Things, Basic Books (2004).

[20] Pressman, R.: Software Engineering – a Practitioner's Approach. 6th edition. McGraw Hill (2005)

[21] Pyla, P. S., Pérez-Quiñones, M. A., Arthur, J. D., Hartson, H. R.: Towards a model-based framework for integrating usability and software engineering life cycles. Interact 2003 Workshop on "Closing the Gaps: Software Engineering and Human Computer Interaction" 67--74 (2003)

[22] Sauro, J., Kindlund, E., A method to standardize usability metrics into a single score. CHI '05 401-409 (2005)

[23] Shneiderman, B.: Designing the User Interface, Strategies for Effective Human-Computer Interaction. 4th edition. Addison Wesley (2004)

[24] Swallow, D., Blyth, M., Peter, W., Grounding experience: relating theory and method to evaluate the user experience of smart-phones, Proc. 2005 Annual Conference on European Association of Cognitive Ergonomics (2005) 91-98.

[25] Tractinsky, N., Katz, A. S. & Ikar, D.. What is beautiful is usable, Interacting with Computers, 13 (2000) 127-145

Eclipse Plug-in to Manage User Centered Design

Yael Dubinsky
Dipartimento di Informatica e Sistemistica "A. Ruberti"
SAPIENZA - Università di Roma
Via Ariosto - 25, 00185, Roma, Italy
dubinsky@dis.uniroma1.it

Shah Rukh Humayoun
Dipartimento di Informatica e Sistemistica "A. Ruberti"
SAPIENZA - Università di Roma
Via Ariosto - 25, 00185, Roma, Italy
humayoun@dis.uniroma1.it

Tiziana Catarci
Dipartimento di Informatica e Sistemistica "A. Ruberti"
SAPIENZA - Università di Roma
Via Ariosto - 25, 00185, Roma, Italy
catarci@dis.uniroma1.it

ABSTRACT
User-centered design (UCD) approach guides the design of user interface (UI) and its evaluation by integrating user experience as part of the software development process. Involving users during the development process by applying UCD techniques minimizes risks and increases the product quality. One of the challenges towards this is to automating the management of UCD activities during the development time thus to steer and control the UCD activities within the development environment of software projects. In this paper, we present a plug-in for Eclipse development platform to manage UCD activities at the Integrated Development Environment (IDE) level. We develop and evaluate the plug-in with teams that work according to the agile software development approach. Using this plug-in, the development teams can manage UCD activities at IDE level hence developing high quality software products with adequate level of usability.

Categories and Subject Descriptors
H5.2 [**User Interfaces**]: User-centered design, K.6.3 [**Software Management**]: Software development, software process.

General Terms
Management, Measurement, Design.

Keywords
User-centered design (UCD), user experience, agile software development, Eclipse plug-in.

1. INTRODUCTION
The user-centered design (UCD) approach [5, 15] is used to develop software products by positioning the real users of the system at the centre of design activities, e.g. by representing or modeling users in some way like scenarios and personas; through users testing of prototypes (either paper or working prototype); by involving users in making design decisions (e.g. thorough participatory design). The approach focus is on the increase of usability for the users by involving them in design and development activities. UCD activities aim at reducing the risks of the software project and increasing the overall product quality.

I-USED '08, September 24, 2004, Pisa, Italy

Variations in activities arise in different UCD methods [5, 15], and still the Human-Computer Interaction (HCI) community lacks to agree upon a precise definition of UCD methods or process [3, 8]. However, in [8] there is a set of definition of twelve principles for designing and developing systems with focus on UCD that is obtained as: *"User-centered system design (UCSD) is a process focusing on usability throughout the entire development process and further throughout the system life-cycle"* (p. 401). The International Organization for Standardization (ISO) has also defined the standard guidelines to deal with different aspects of HCI and UCD; in particular, ISO/DIS 13407[3] provides the guidance on user-oriented design process. Other relevant ISO standard guidance are ISO 9241-11[4], ISO TR 16982[5]. A detailed discussion about the methods, processes, guidelines, and prototype activities in UCD can be found in ISO standards and in [5, 15].

Lack of usability and inefficient design of the end-product are common causes amongst the others for failure of software products [12, 14]. The software products are developed for the users, and normally fail if the users find it difficult to operate them due to the lack of usability and inappropriate design. The software development teams usually work with the users either at the start of project for getting the requirements or at the end of project to test and evaluate the developed product. Furthermore, normally in the testing phase, the project teams focus more on checking the functionalities of the product (such as performance, reliability, security, robustness, etc) rather than its usability and design aspects. Checking usability or solving defects at the end of the development process needs more time, efforts, and money hence causing the failure of many software projects As a result, involving the users in the design phases is a good practice to identify lack of usability and design defects early in development time, in order to avoid any possibility of product failure at the end. The UCD approach, described above, provides methods and techniques for involving users at early stages of development [15]. So, integrating the UCD approach within software development processes gives the benefit of including the user experience as part of the development process for producing quality products with adequate level of usability.

Analyzing the current software design practices, we identified a lack of *UCD management* which we define for a specific software

[3] ISO/DIS 13407: Human Centered Design for Interactive Systems
[4] ISO 9241–11: Ergonomic requirements for office work with visual display terminals (VDTs)
[5] ISO TR 16982: Ergonomics of human-system interaction - Human-cantered lifecycle process descriptions

project as the ability to steer and control the UCD activities within the development environment of the project. This management of UCD activities at IDE level is important as it will help to integrate and automate *UCD activities* across different development life-cycle phases. By automating the management of UCD activities within development environment we decrease the time and cost to test each unit and improve the overall product quality.

In this paper, we present an Eclipse plug-in to manage the user involvement for different UCD activities in software development that can work with any software development process life-cycle. Managing UCD activities while working according to the agile software development approach [1] was already suggested [2, 4, 10, 11, 13, 7]. Our contribution is by automating UCD management at IDE level to enable, for example, creating experiments, adding users, analyzing results, and tracing the code.

The remainder of this paper is as follows. In Section 2 we describe our framework to integrate the user experience in the process of software development. Section 3 presents and explains how we can use our developed Eclipse plug-in to manage UCD activities at IDE level during software development. We conclude in Section 4.

2. USER EXPERIENCE AND THE DEVELOPMENT PROCESS

A software development approach that has been emerging in the last decade is the agile approach that is used for constructing software products in an iterative and incremental manner; where each iteration produces working artifacts that are valuable to the customers and to the project. This is performed in a highly collaborative fashion in order to produce quality products that meet the requirements in cost effective and timely manner [1].

Based on our experience with guiding the implementation of the agile approach [16, 6, 9], and the integration of UCD techniques in the last three years in agile projects in academia [7], we gather the cases in which UCD can be supported within the IDE.

The main characteristics of the integrated approach of agile and UCD that we use are:

3. **Iterative design activities** - In many cases, when user-centric techniques are used, the design of the system is refined according to the users' evaluations and this is performed mainly during the design phase. When introducing the agile approach, the design is updated regularly as the product evolves. When combining UCD activities with the agile approach, the user evaluation is fostered by performing UCD tasks in each iteration of two to four weeks, and the design is updated according to the evaluation of on-going outcomes that are considered as refactoring tasks.
4. **Measures** – Taking measurements is a basic activity in software development processes. The agile approach emphasizes it and suggests the tracker role. When combining agile and UCD, the set of evaluation tools is built and refined during the process and is used iteratively as a complement to the process and product measures.
5. **Roles** – Different roles are defined to support software development environments. The agile approach adds roles for better management and development of the project. Combining agile and UCD adds the UCD roles, like for example the UI Designer role.

Using our integrated approach of UCD and agile with software teams in the academia, we have gathered use cases to establish the plug-in specifications. Following are six representative use cases that are categorized in three themes: the development process, the evaluation activity, and the design improvement.

2.1 Development Process

There is a need to involve the users in the process of development.

Following are examples for use cases that relate to this category:

- One of the tasks during the first planning session is as follows: 'Explore who are the kinds of users who should use the product that we develop; what are their characteristics; what are their needs; what are their expectations from the product.' The customer explains that this is an important task since he cannot represent all users and actually he does not know for sure what their exact needs are (though he is sure they will like it a lot). One of the teammates asks to be assigned to this task and estimates it as 10 hours of work for this iteration. Presenting her results after two weeks, she opens her development environment in the database of the *User Perspective* and shows the list of 20 users she talked with (names, titles, contact details), main issues that were learned, and one new task that has emerged for future iterations: 'Prepare and run a questionnaire that will enable us to extract users' needs.' The customer sets high priority for this new task.
- The project manager reviews the subjects for the coming reflection session, and sees that one of the subjects is 'ways to assess the usability of our product'. She then sends invitations to seven users from the two different kinds of users to join this meeting. During the reflection session, one decision is made that two users will participate in each iteration planning session and their responsibility will be to give feedbacks on what presented. In addition they will help in defining three measures that will be automated thus enable teammates an immediate feedback during development.

2.2 Evaluation

There is a need to perform user evaluation and to manage it along the process of development.

Following are examples for use cases that relate to this category:

- The team leader browses over the details of the user experiments that are planned for tomorrow. He sees the number of users that will arrive, the names, and responsibilities of the teammates that will take care of these experiments. He checks the variables that were set and the experiments flow.
- One of the teammates sees that the *User Perspective* flushes meaning new data have arrived. He clicks on it and sees that the results of the user experiments that were conducted yesterday are in. He is surprised to find a new problem with

high severity ranking. Examining results from previous experiments, he observes that this is a new problem and adds a note about it in the discussion area. During the next iteration planning, the experiments' results are presented and among others, a measure is presented that shows two problems that emerged from the users; one in normal severity and the other one in high severity.

2.3 Design Improvement

There is a need to improve the design of the user interfaces based on the evaluation results.

Following are examples for use cases that relate to this category:

- The designer of the user interface views the latest design diagrams and tries different changes that adhere to the new task in this iteration. The task was added due to the last problem that was found by the users. Thinking of different options, she talks with two users and receives their feedbacks. She shows them the possible drawings of the new interface and asks them to simulate trying it while thinking aloud. She summarizes the results and sets her decision.
- One of the teammates browses over the system reports and looks for each user experiment, which was conducted in the last two releases, what were the results and what were the implications on design. For each implication, he sees the development tasks that are related.

We suggest that the combination of the agile and UCD approaches should be supported by an extension to a contemporary development environment in order to be used in a natural manner. This is elaborated in the next section.

3. THE UCD MANAGEMENT PLUG-IN

3.1 The Project

A team of six developers in a project based course in the academia has developed the UCD plug-in that is presented in this paper[6]. The project took five and a half months and was composed of 4 iterations, three of 5 weeks each and one of 3 weeks. Table 1 shows the durations and the main themes of each iteration.

Table 1. The iterations – duration and themes

Iter.	Duration (weeks)	Themes
1	5	Experiments and Roles - End-to-end experiments: define the experiment, execute it, results view - Evaluation manager role-perspective - UI designer role-perspective - Work items can be created, assigned - The system has one data repository
2	5	Users' interface and user experience (UX) automation - Users' management and permissions - A user interface to run the experiment - Client / Server architecture for running the experiment - Support automatic measures for user evaluation that are derived from user experience - On line help - Traceability – experiments should be part of a specific project that we develop; each specific development task that is derived from one or more experiments results should be associated with the appropriate code parts that implement them
3	3	Stability - Testing and Refactoring - Development refinement
4	5	Heuristics and User Profiling - Support Nielsen heuristics technique - Support user profiling - Scale with more *end* projects

[6] This project was developed as part of the "Annual Project in Software Engineering" course that is instructed by the first author at the Computer Science Department at Technion IIT.

3.2 Using the UCD Management Plug-in

The main feature of the UCD management plug-in is the ability to create and deploy user experiments from within the Eclipse IDE. Focusing on a specific software project, we can define different kinds of experiments. One kind for example is a task-based user experiment in which the participant uses the target product and receives the tasks to perform along the experiment. During this experiment the system measures different performance times. Another kind of experiment is questionnaire-based experiment in which the user specifies the level of his/her agreement with the presented set of statements. The development team chooses the set of experiments according to the nature of software project and then selects appropriate users from the pool of target users to perform these tasks.

We illustrate the definition of a task-based user experiment using a view that is presented by Figures 1 and 2.

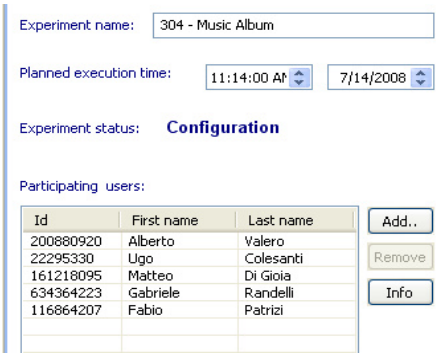


Figure 1. Defining the experiment (left hand side)

In the left hand side of the view (Figure 1) we can see the options of setting the experiment schedule and the users who are involved. In the right hand side (Figure 2) we can see the options of adding tasks to the experiment, save the experiment, and execute it.

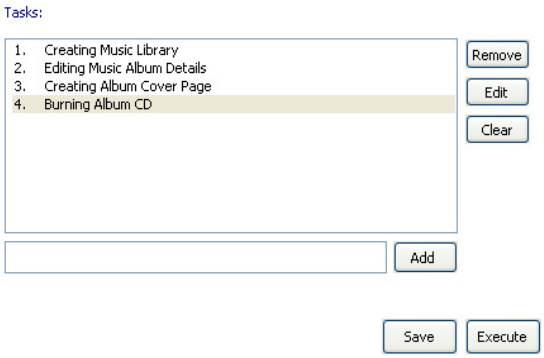


Figure 2. Defining the experiment (right hand side)

The experiment can run locally i.e., on the server on which the data is stored, or remotely. Setting the remote option causes the enlisted users to receive email with the experiment files attached, so they can perform the experiment in a way that the results are stored in the server. Figure 3 shows the results view of a specific experiment. Different kinds of experiments were developed that support appropriate results views.

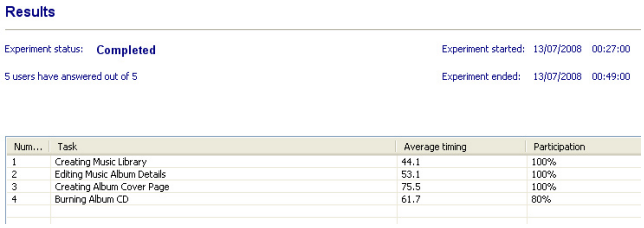


Figure 3. The experiment's results view

Experiment Explorer is available to support the experiments of a specific project (Figure 4). Experiments can be shared among different projects.

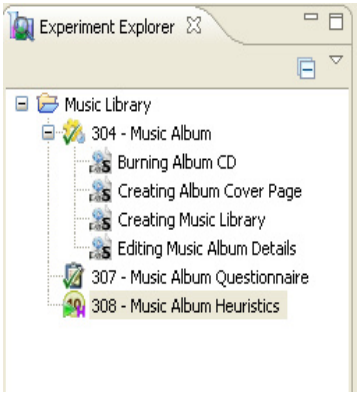


Figure 4. The Experiment Explorer

Managing the experiments, new development tasks are derived. These tasks are the results of the already conducted experiments. The plug-in enables associating code part/s to the appropriate task/s and vice versa so traceability is kept. Figure 5 shows how a code segment can be associated.

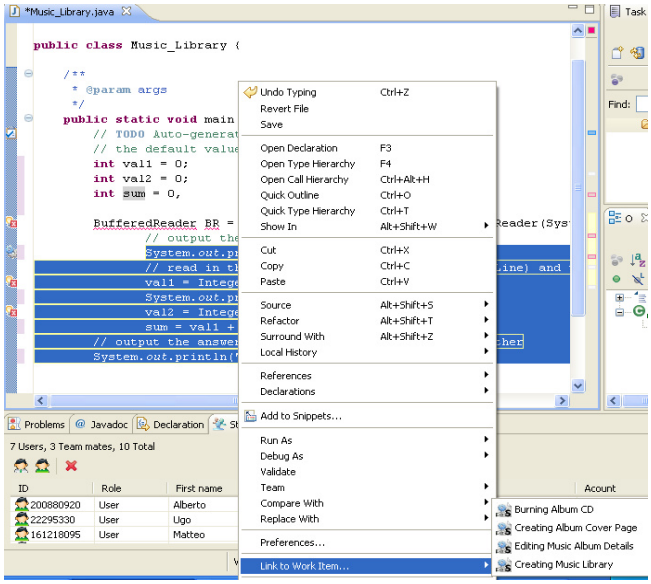


Figure 5. Associating code to a development task

Figure 6 shows how this code is marked (left side bar) and highlighted.

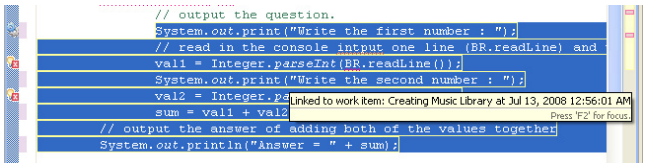


Figure 6. Associated code is marked

3.3 Evaluating the Plug-in

As part of the third iteration, the team was asked to evaluate its own product (the UCD management plug-in) using itself ("eating own cookies"). Following is the plan and the results of this preliminary evaluation.

The evaluations goals as written by the team were:

- Examining suspicious issues like adding new users to the system and analyzing the experiments' results (specifically for the questionnaire-based experiments).
- Receiving feedback on the graphical user interface (GUI) and how intuitive it is.
- Examining the plug-in on a large scale project.

Two kinds of experiments were defined by the team for the evaluation of the plug-in. The first experiment was a task-based experiment and the second was a questionnaire-based experiment.

The participants were 3 students from another team in the same course, and in addition all the six developers performed both kinds of experiments. Each participant performed the experiment by himself / herself while one observer was sitting aside for writing notes.

Results

Experiment status: **Running**

3 users have answered out of 6

Experiment started: 17/03/2008 11:32:00

Experiment ended: ---

		Strongly Disagree	Disagree	Agree	Strongly Agree
1.	Logging in to the system is simple.	0	0	2	1
2.	Adding a user or a teammate to the system is simple	0	1	0	2
3.	Switching between teammates is fast and simple.	1	2	0	0
4.	The configuration page is intuitive.	0	0	2	1
5.	The Questionnaire result page displays the level of agreement (p...	0	0	3	0
6.	The Questionnaire result page displays the usability problems disc...	0	2	1	0
7.	The different editors and views of the plug-in are uniform and foll...	0	0	1	2
8.	The different editors and views of the plug-in blend seamlessly in...	0	1	2	0
9.	I would use this plug-in to test the usability of an application in de...	0	0	1	2

Figure 7. Results of questionnaire-based experiment – participants from another team

Results

Experiment status: **Completed**

6 users have answered out of 6

Experiment started: 17/03/2008 10:56:00

Experiment ended: 17/03/2008 13:39:00

		Strongly Disagree	Disagree	Agree	Strongly Agree
1.	Logging in to the system is simple.	0	0	0	6
2.	Adding a user or a teammate to the system is simple.	1	3	1	1
3.	Switching between teammates is fast and simple.	3	2	0	1
4.	The configuration page is intuitive.	0	0	2	4
5.	The Questionnaire result page displays the level of agreement (p...	0	0	1	5
6.	The Questionnaire result page displays the usability problems disc...	3	2	0	1
7.	The different editors and views of the plug-in are uniform and foll...	0	1	4	1
8.	The different editors and views of the plug-in blend seamlessly in...	0	0	3	3
9.	I would use this plug-in to test the usability of an application in de...	0	1	2	3

Figure 8. Results of questionnaire-based experiment – team members are the participants

We focus on the questionnaire-based experiment and comments of the observers and show an example of a derived task that emerged for further development. The questionnaire included the following statements:

1. Logging in to the system is simple.
2. Adding a user or a teammate to the system is simple.
3. Switching between teammates is fast and simple.
4. The configuration page is intuitive.
5. The Questionnaire result page displays the level of agreement (per statement) in a clear way.
6. The Questionnaire result page displays the usability problems discovered in a clear way.
7. The different editors and views of the plug-in are uniform and follow a similar theme
8. The different editors and views of the plug-in blend seamlessly into the eclipse.
9. I would use this plug-in to test the usability of an application in development.

Figures 7 and 8 show the results of the three participants from another group and the results of the developers themselves respectively.

Following are few comments, for example, that were presented by the observers:

1. "In the questionnaire-view that is presented to the participant, long tasks appear truncated."
2. "The participant did not know how to save the changes in the result page. He searches for a save button like appears in other screens."
3. "The names of the operations in the menu of the experiment view are not clear."

Analyzing the results of both experiments, associations to the specific results were presented for each conclusion, and then suggested development tasks were associated to the conclusions.

One of the finding, for example, was detailed as follows:

"It was found that there is a difficulty in identifying problems in the product out of the information that is presented in the 'results page'. Participants find it hard to associate the results (as presented in the 'results page') to the experiment goals and to the practical problems that were discovered."

"Association to the results:

- In the questionnaire-based experiment the two teams marked 'Disagree' for statement 6 [The questionnaire result page displays the usability problems discovered in a clear way].
- In the task-assignments experiment, it took long time, 84 and 177 seconds in average for the two groups, to complete task 5 [According to the experiment goals, try to assess the

number of usability problems indicated by the results, and write that number as a conclusion to this experiment]."

The development task that was defined using the plug-in is as follows: "Enable determining thresholds for success and failure in an experiment and present them clearly in the 'results page'."

4. CONCLUSION

In this paper, we present our Eclipse plug-in to automating the process of managing UCD activities at the Integrated Development Environment (IDE) level during the development time of software projects. To develop the framework we were inspired by use cases that emerged when performing UCD activities with the agile teams. Using this plug-in, the software project team can create experiments, adding users, analyzing results and tracing back it to code for their developed or in-progress product. By automating the process of managing UCD activities the chances of creating quality products with adequate level of usability become high, as it helps to get benefits of user experience during development time.

In future, we intend to continue work on the developed plug-in to manage more UCD activities. Further, we also intend to evaluate the developed product on big scale with different size of software development teams.

5. ACKNOWLEDGMENTS

Our thanks to the plug-in developers from Technion IIT whose product is presented in this paper: David Ben-David, Tomer Einav, Yoav Haimovitch, Barak Nirenberg, Laliv Pele, and Alon Vinkov.

6. REFERENCES

[1] Agile Alliance 2001. Manifesto for Agile Software Development. Technical Report by Agile Alliance, http://www.agilealliance.org.

[2] Blomkvist, S. 2005. Towards a Model for Bridging Agile Development and User-Centered Design. Published as a book chapter: Seffah, A., Gulliksen, J., and Desmarais, M., (eds.). Human-Centered Software Engineering – Integrating Usability in The Development Process. Springer, Dordrecht, The Netherlands, 217-243.

[3] Blomkvist, S. 2006. User-Centered Design and Agile Development of IT Systems. IT Licentiate theses, Department of Information Technology, Uppsala University.

[4] Detweiler, M. 2007. Managing UCD within Agile Projects. ACM Interactions May-June, 40 – 42.

[5] Dix, A., Finlay, J.E., Abowd, G.D., and Beale, R. 2003. Human Computer Interaction, 3rd Edition, Prentice Hall.

[6] Dubinsky, Y. and Hazzan, O. 2005. The construction process of a framework for teaching software development methods, Computer Science Education, 15:4, 275–296.

[7] Dubinsky, Y., Catarci, T., Humayoun, S., and Kimani, S. 2007. Integrating user evaluation into software development environments, 2nd DELOS Conference on Digital Libraries, Pisa, Italy.

[8] Gulliksen, J., Goransson, B., Boivie, I., Blomkvist, S., Persson, J. and Cajander, A. 2003. Key principles for user-centered systems design. Behaviou & Information Technology, Vol. 22, No. 6, 397–409.

[9] Hazzan, O. and Dubinsky, Y., Agile Software Engineering, Undergraduate Topics in Computer Science Series, Springer-Verlag London Ltd, 2008, in press.

[10] Hudson, W. 2003. Adopting User-Centered Design within an Agile Process: A Conversation. Cutter IT Journal, (16), 10 http://www.suntagm.co.uk/design/articles/ucdxp03.pdf

[11] Hwong, B., Laurance, D., Rudorfer, A., and Schweizer, A. 2004. User-Centered Design and Agile Software Development Processes. Siemens Corporate Research http://www.scr.siemens.com/en/pdf/se_pdf/rudorfer-1.pdf.

[12] Landauer, T. K. 1995. The trouble with computers: usefulness, usability, and productivity, MIT Press.

[13] McInerney, P., and Maurer, F. 2005. UCD in agile projects: Dream team or odd couple?. ACM Interactions, 12(6), 19 - 23.

[14] Norman, D. 2006. Why Doing User Observations First Is Wrong, ACM Interactions, July-August 2006.

[15] Sharp, H., Rogers, Y., and Preece, J. 2007. Interaction Design: Beyond Human-Computer Interaction. 2nd Edition. Willey.

[16] Talby, D., Hazzan, O., Dubinsky, Y. and Keren, A. 2006. Agile software testing in a large-scale project, IEEE Software, Special Issue on Software Testing, 30-37.

Integrating Software and Usability Engineering through Jointly-constructed, Event-based Stories

John Teofil Paul Nosek
Temple University
Rm. 316 Wachman Hall
Philadelphia, PA 19122
215-204-7232

nosek@temple.edu

ABSTRACT

This position paper proposes that event-based stories appear to have the potential to provide a simple, but powerful technique for users and developers to communicate emotional and informational needs, redesign processes, and structure the user interface design within the agile development paradigm. Informal evaluation of the use of event-based stories in several development projects suggest that event-based stories could be useful in integrating software and usability engineering. Controlled experiments, in addition to more formal case analyses are the next steps.

Categories and Subject Descriptors

D.2.2 Design Tools and Techniques H.5.2 User Interfaces

General Terms

Design, Human Factors

Keywords

Events, stories, scenarios, usability engineering, software engineering

1. POSITION PAPER

Software engineering has focused on functionality, i.e., the system must do "x" [2]. However, much of software development focuses on usability issues [7]. Usability Engineering focuses more on how easy the developed system is to learn and use, but these divisions are artificial. The earlier user feedback begins and the more it can be maintained throughout the development process, the better [3]. Nosek & Ahrens [4], Nosek & Schwartz [5], Nosek & Roth [6] explored techniques that can be used by users and developers over the translation process from problem statement to developed system. Such techniques must be powerful enough for users to express needs that can be ultimately translated into code by developers. Through experience, end-users have not found most technically-oriented techniques, such as data flow diagrams, easy to learn and use. Juristo et al recommend applying elicitation patterns to garner usability requirements "after a preliminary version of the software requirements has been created [1]." This position paper explores the use of jointly-constructed, event-based stories as a powerful, flexible communications technique among end users and developers. Stories can be employed from the earliest stages of development and in concert with applying other techniques, such as, elicitation patterns. For example, agile-based development recognizes the real-world demands that development work be divided in time-segmented portions of completed deliverables, which includes, code, testing, and interface design. Rosson and Carroll use scenarios and a process of refinement of the scenarios from problem to activity design to information design to interaction design [7]. However, they lack sufficient granularity to easily identify the problems and track the refinement of the scenarios from problem description through interaction design. This is because a single scenario as employed by Rosson and Carroll can include multiple events and activities and incorporate wordier, less directly relevant task descriptions that may make for a more interesting story but adds complexity and reduces clarity. Events help to organize the problem space for software development and the construction of stories based on these events may help to integrate software and usability engineering. Developing stories for single events provides finer granularity and makes it easier to track refinements of the scenarios.

Information-based techniques by their nature filter out any emotional aspects discovered in the initial information gathering process. Through the initial story of the problem statement, users may be able to place themselves within the story and judge whether the developer understands both emotional and informational aspects. For example in the sample story below, the user can observe that the developer has incorporated the emotion of worry in the problem scenario and the reduction of this worry in the activity design. Emotions add strength or importance to a situation. Specifically recognizing emotions within stories validates the user's contribution and may make the user more confident that the developer accurately understands the situation. Users that can read have the necessary capabilities to modify, and therefore, should be able to co-construct the stories without additional training in any particular technically-oriented technique.

I-USED'08, September 24, 2008, Pisa, Italy

In the next phase, developers can incorporate process redesign in refining the problem statement to incorporate the new activities with the proposed system. This process can be refined through information and interaction design. Figure 1 shows how Event-based stories can proceed in tandem with technically-oriented techniques, which focus on coding and testing. Figure 2 shows how the solution space can be subdivided by Event-based stories and Technically-oriented techniques. An example is given for a how stories may be refined around a single event.

Event 1: Prof. Bob London missed the flight after a conference and so had to cancel the class next day. (Instructor alters the class schedule)

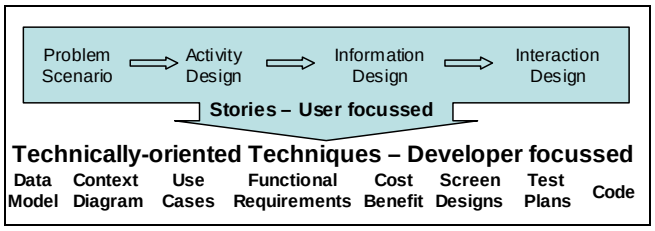


Figure 1: Event-based Stories in Software and Usability Eng.

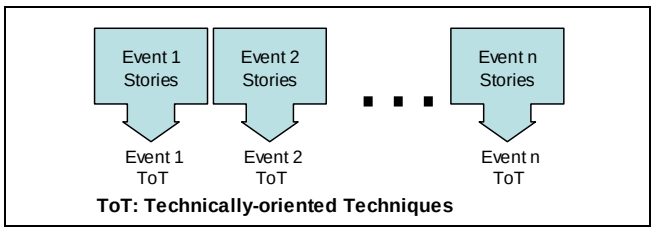


Figure 2: Division of Solution Space by Events

Problem Scenario for Event 1: Prof. Bob London was in Houston, TX on Tuesday for a conference and missed the last flight. He realized that he would miss the class next morning and was worried that students would show up to class confused and angry at him. However, he had neither the list of students for the class nor their contact information with him. So he sent an email to Ms. Tika Farrell, the dept.'s secretary, asking her to let the students know that the class for Wednesday morning is cancelled. Ms. Tika was annoyed with having to do one more thing and didn't have the contact information for the students. So she replied saying that she will post a note in the class room. Prof. B L searched through his emails and found a student's email. He made an educated guess that the student was in the Wednesday morning class and emailed him saying that the Wednesday class is cancelled and that he let other students know. It was already late in Philadelphia and Prof. B L was not sure if the student read his email that night.

(Process Redesign) Activity Design Scenario from Problem Scenario for Event 1: (same as above …) He realized that he would miss the class next morning. B L was not worried and didn't have to bother the secretary. He connected his laptop to the Internet and logged into the Online Instructional Support System and sent out an announcement to the students to the effect that the class is cancelled. Wednesday morning, Prof. B L checked the system and found that 12 out of 15 students read the announcement. He easily sent a reminder to the 3 students who didn't read the announcement.

Information Design Scenario from Activity Design Scenario for Event 1: (same as above …) Prof. B L connected his laptop to the Internet and logged into the Online Instructional Support System. He noticed that there were a couple of alerts, which he decided to ignore for the time being. Prof. B L had predefined groups of students in various courses. These were in his personal address book. He started to compose a new 'announcement'. A window similar to composing an email got displayed. Prof. B L typed in the announcement and sent it to the predefined group of students in his Wednesday class. Wednesday morning, Prof. B L checked the system and found that 12 out of 15 students read the announcement.

Interaction Design Scenario from Activity Design Scenario for Event 1: (same as above - interaction refinements are underlined) Prof. B L connected his laptop to the Internet and logged into the Online Instructional Support System. He noticed that there were a couple of alerts, which he decided to ignore for the time being. So, he clicked on the "remind me later" button. The main menu showed up. Prof. B L selected "messaging" option. He selects the "new announcement" item. A window similar to composing an email shows up. In the compose window, he selected the "To" field; right clicked and selected "predefined groups". The predefined groups in his personal address book showed up. He selected the group that corresponded to the students in his Wednesday class. The "To" field got populated with the group information. He typed in the announcement information in the "message" field and pressed the "send" button to send the announcement.

2. SUMMARY

This position paper proposes that event-based stories appear to have the potential to provide a simple, but powerful technique for users and developers to communicate emotional and informational needs, redesign processes, and structure the user interface design within the agile development paradigm. Informal evaluation of the use of event-based stories in several development projects suggest that event-based stories could be useful in integrating software and usability engineering. Controlled experiments, in addition to more formal case analyses are the next steps.

3. ACKNOWLEDGMENTS

George Mathew developed the initial story example. I thank the anonymous reviewers for their thoughtful and constructive comments.

4. REFERENCES

[1] Juristo, N., Moreno, A. M., Sanchez-Segura, M. (2007), "Guidelines for Eliciting Usability Functionalities", IEEE Transactions on Software Engineering, Vol. 33, No. 11, November 2007, pp. 744-758.

[2] Larman, C. (2004) Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design, Prentice Hall, New York.

[3] Nosek, J.T., Sherr, D. (1984) "Getting the Requirements Right vs. Getting the System Working - Evolutionary

Development," in Bemelmans, T.M.A. (ed.) Beyond Productivity: Information Systems Development for Organizational Effectiveness, North Holland, New York.

[4] Nosek, J.T., Ahrens, J. (1986), "An Experiment to Test User Validation of Requirements: Data Flow Diagrams v. Task-oriented Menus" (with J. Ahrens), International Journal of Man-Machine Studies, Vol. 25, No 6, December, 675-684.

[5] Nosek, J.T., Schwartz, R. (1988), "User Validation of Information System Requirements: Some Empirical Results", IEEE Transactions on Software Engineering, Vol. 14, No. 9, September, 1372-1375.

[6] Nosek, J.T., Roth, I. (1990), "A Comparison of Formal Knowledge Representation Schemes as Communication Tools: Predicate Logic vs. Semantic Network", International Journal of Man-Machine Studies, Vol. 33, 227-239.

[7] Rosson, M.B. & Carroll, J.M.(2002) Usability Engineering: Scenario-based Development of Human Computer Interaction, Morgan Kaufmann Publishers Inc., San Francisco, CA.

Reducing Risk through Human Centred Design

Nigel Bevan

Professional Usability Services
12 King Edwards Gardens
London W3 9RG, UK
www.nigelbevan.com
mail@nigelbevan.com

ABSTRACT

The National Academy of Science's report on Human-System Integration in the system development process (NAS HSI report) [12] explains how human needs can be integrated into system design using an incremental systems engineering development process that continually assesses risks at each phase of the system development. This paper suggests how appropriate Human Centred Design (HCD) methods can be selected to mitigate risks to project success.

1. RISKS IN SYSTEMS DEVELOPMENT

The NAS HSI report points out that the ultimate goal of system development is to produce a system that satisfies the needs of its operational stakeholders (including users, operators, administrators, maintainers and the general public) within acceptable levels of the resources of its development stakeholders (including funders, acquirers, developers and suppliers). Operational stakeholders need a system that is effective, efficient and satisfying [1]. Developing and delivering systems that satisfy all of these success-critical stakeholders usually requires managing a complex set of risks such as usage uncertainties, schedule uncertainties, supply issues, requirements changes, and uncertainties associated with technology maturity and technical design.

Boehm and Lane [4] suggest five principles for managing these risks:

1. *Stakeholder satisficing*; identifying the success-critical stakeholders and their value propositions (what is offered at what cost); negotiating a mutually satisfactory set of system requirements, solutions, and plans; and managing proposed changes to preserve a mutually satisfactory outcome.
2. *Incremental growth of system definition and stakeholder commitment*: incremental discovery of emergent human-system requirements and solutions using such methods as prototyping, testing with users, and use of early system capabilities.
3. *Iterative system development and definition*: cyclic refinements of requirements, solutions, and development plans. Such iteration helps projects to learn early and efficiently about operational and performance requirements.
4. *Concurrent system definition and development*: that includes concurrent engineering of requirements and solutions without waiting for every requirement and subsystem to be defined.
5. *Risk management – risk driven activity levels and anchor point milestones*. The level of detail of specific products and processes should depend on the level of risk associated with them.

Principles 2, 3 and 4 are consistent with approaches to human centred design, such as recommended in ISO 13407.

The other two principles (*Stakeholder satisficing* and *Risk management*) provide a means to determine which human centred design activities and methods are needed in a project to be confident that the final system will be acceptable to the operational stakeholders.

This contrasts with existing approaches to human centred design, which are commonly based on a one-size-fits-all methodology (e.g. [5], [14]) that may be justified by a cost benefit analysis to assess the potential business benefits of producing a more usable system [3].

The additional expenditure needed for human centred activities is often difficult to justify because the budget holder for project development usually does not personally gain from longer-term benefits such as increased sales or reduced whole life costs.

Project managers are much more likely to be influenced by the risks of not achieving stated project objectives. It is thus useful to recast the potential cost benefits of usability as risk reduction strategies. Table 1 restates the list of cost benefits in [2] as potential project risks.

2. HUMANCENTRED DESIGN ACTIVITIES

Looking for advice on which methods to use for human centred design can be bewildering.

Ferré [6] analyzed the methods contained in six popular HCI textbooks and identified 96 categories of HCD techniques. Individual textbooks each contained between 21 and 43 of these categories of technique:

I-USED'08, September 24, 2008, Pisa, Italy

Table 1. Risks mitigated by HCD

A: Increased development costs to produce an acceptable system

Not detecting and fixing usability problems early in the development process
Increasing the cost of future redesign or radical change of the architecture to make future versions of the product more usable
Increased costs due to unnecessary functionality
Increased costs due to additional documentation
Product fails

B: Web site usability: poor web sales

Users cannot find products that they want to purchase
Users cannot find additional information (e.g. delivery, return and warranty information)
Dissatisfied users do not make repeat purchases
Users do not trust the web site (with personal information and to operate correctly)
Users do not recommend the web site to others
Web site fails to increase sales through other channels
Increased support costs

C: Product usability: poor product sales

Competitors gain advantage by marketing competitive products or services as easy to use
Dissatisfied customers do not make repeat purchases or recommend the product to others
Poor ratings for usability in product reviews
Brand damage

D: Poor productivity: risks to purchasing organisation

Slower learning and poorer retention of information
Increased task time and reduced productivity
Increased employee errors that have to be corrected later
Increased employee errors that impact on the quality of service
Increased staff turnover as a result of lower satisfaction and motivation
Increased time spent by other staff providing assistance when users encounter difficulties

E: Increased support and maintenance costs

Increased support and help line costs
Increased costs of training
Increased maintenance costs

Author	*Number of categories*
Constantine [5]	31
Hix [7]	21
Mayhew [10]	31
Nielsen [11]	25
Preece [13]	43
Shneiderman [15]	29

ISO PAS 18152 contains an exhaustive list of 125 human systems (HS) activities that are needed for all aspects of systems development. These were derived from an analysis of best practice in human centred design in civilian and military systems. The categories of activity are:

HS.1 Life cycle involvement activities

HS.1.1 HS issues in conception
HS.1.2 HS issues in development
HS.1.3 HS issues in production and utilization
HS.1.4 HS issues in utilization and support
HS.1.5 HS issues in retirement

HS.2 Integrate human factors activities

HS.2.1 HS issues in business strategy
HS.2.2 HS issues in quality management
HS.2.3 HS issues in authorisation and control
HS.2.4 Management of HS issues
HS.2.5 HF data in trade-off and risk mitigation
HS.2.6 User involvement
HS.2.7 Human-system integration
HS.2.8 Develop and re-use HF data

HS.3 Human-centred design activities

HS.3.1 Context of use
HS.3.2 User requirements
HS.3.3 Produce design solutions
HS.3.4 Evaluation of use

HS.4 Human resources activities

HS.4.1 Human resources strategy
HS.4.2 Define standard competencies and identify gaps
HS.4.3 Design staffing solution and delivery plan
HS.4.4 Evaluate system solutions and obtain feedback

In [12] Table 3-A-1, the HS activities in ISO PAS 18152 have been categorised by type of system development activity:

1. Envisioning opportunities
2. System scoping
3. Understanding needs
4. Requirements
5. Architecting solutions
6. Life-cycle planning
7. Evaluation
8. Negotiating commitments
9. Development and evolution
10. Monitoring and control
11. Operations and retirement
12. Organizational capability improvement

An elaborated version of the table is included as an annex to this paper.

3. SELECTING HUMAN CENTRED DESIGN METHODS

The steps needed to select human-centred methods for an individual project are thus:

1. Identify the success-critical stakeholders.
2. Identify which potential consequences of poor usability affect the success-critical stakeholders.
3. Assess the likelihood and impact of these consequences.
4. Identify which categories of HS activities can reduce the risks.
5. Identify which HCD methods in each category are most cost-effective. The alternative methods should be assessed against criteria such as:

- To what extent will each possible method address the activities that have been identified as important?
- How cost effective is each method likely to be, given the time and effort required and constraints such as available skills, access to stakeholders and other users, etc.?

The needs for usability evaluation in particular should be judged in the broader context of the relative importance of usability evaluation in relation to other HS activities. For example, when designing and developing for a new context of use, the major risks might be associated with requirements, so that the majority of HCD resources might be devoted to early life cycle activities (which could include evaluation of early concepts and competitive evaluation).

4. CONCLUSIONS

This paper suggests how HCD can be justified as part of systems development and how the most appropriate HCD methods can be selected on a project-by-project basis.

This will enable HCD resources to be used most effectively for individual projects. The author would be happy to advise on or support the application of this approach to selecting HCD methods in a real development project.

The prerequisites for successfully using this approach include having usability experts in the development team who:

- can convince the project of the specific risks associated with poor usability;
- have sufficient experience to be able to select the most cost effective HCD methods; and
- have the expertise and resources to apply a wide range of different types of methods.

5. REFERENCES

[1] Bevan, N. (1999). Quality in use: meeting user needs for quality, *Journal of Systems and Software*, **49**(1), pp 89-96.

[2] Bevan, N. (2005). *Cost benefits framework and case studies.* In [3].

[3] Bias, R.G. & Mayhew, D.J. (eds) (2005). *Cost-Justifying Usability: An Update for the Internet Age*. Morgan Kaufmann.

[4] Boehm, B. and Lane, J. A. (2007). *Using the Incremental Commitment Model to Integrate System Acquisition, Systems Engineering, and Software Engineering*. CrossTalk, October 2007. Available at: www.stsc.hill.af.mil/crosstalk/2007/10/0710BoehmLane.html

[5] Constantine, L. L. and Lockwood, L. A. D. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centred Design*. Addison-Wesley, New York.

[6] Ferré, X, Juristo, N, Moreno, A.M. (2004). *STATUS Deliverable D.6.6. Final results on the Integrated Software Process.* Universidad Politécnica de Madrid.

[7] Hix, D. and Hartson, H.R. (1993). *Developing User Interfaces: Ensuring Usability Through Product and Process*. John Wiley and Sons.

[8] ISO 13407 (1999). *Human-centred design processes for interactive systems.* ISO.

[9] ISO PAS 18152 (2003). *A specification for the process assessment of human-system issues.*

[10] Mayhew, D. J. (1999). *The Usability Engineering Lifecycle*. Morgan Kaufmann

[11] Nielsen, J. (1993). *Usability Engineering*. AP Professional,

[12] Pew, R. W. and Mavor, A. (eds.) (2007). *Human-System Integration in the System Development Process: A New Look*. National Academies Press. Available at: books.nap.edu/openbook.php?record_id=11893

[13] Preece, J., Rogers, J., Sharp, H., Benyon, D., Holland, S., Carey, T. (1994). *Human-Computer Interaction*. Addison Wesley

[14] Schaffer, E. (2004). *Institutionalization of Usability: A Step-by-Step Guide.* Addison-Wesley.

[15] Shneiderman, B. (1998). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley.

Annex A. Examples of methods that can be used to support HS best practices

Activity category	Best practices for risk mitigation with ISO 18152 clause reference	HCD methods and techniques
1. Envisioning opportunities	•Identify expected context of use of systems [forthcoming needs, trends and expectations]. •Analyze the system concept [to clarify objectives, their viability and risks].	-Future workshop -Preliminary field visit -Focus groups -Photo surveys -Simulations of future use environments -In-depth analysis of work and lifestyles
2. System scoping	•Describe the objectives which the user or user organization wants to achieve through use of the system.	-Participatory workshops -Field observations and ethnography -Consult stakeholders -Human factors analysis
	•Define the scope of the context of use for the system.	-Context of use analysis
3. Understanding needs a) Context of use	•Identify and analyze the roles of each group of stakeholders likely to be affected by the system. •Describe the characteristics of the users. •Describe the cultural environment/ organizational/ management regime. •Describe the characteristics of any equipment external to the system and the working environment. •Describe the location, workplace equipment and ambient conditions. •Decide the goals, behaviours and tasks of the organization that influence human resources •Present context and human resources options and constraints to the project stakeholders.	-Success critical stakeholder identification -Field Observations and ethnography -Participatory workshop -Work context analysis -Context of use analysis -Event data analysis -Participatory workshops -Contextual enquiry
b) Tasks	•Analyze the tasks and worksystem.	-Task analysis -Cognitive task analysis -Work context analysis
c) Usability needs	•Perform research into required system usability.	-Investigate required system usability -Usability benchmarking -Heuristic/expert evaluation
d) Design options	•Generate design options for each aspect of the system related to its use and its effect on stakeholders. •Produce user-centred solutions for each design option.	-Early prototyping & usability evaluation -Develop simulations -Parallel design (tiger testing)
4. Requirements a) Context requirements	•Analyze the implications of the context of use. •Present context of use issues to project stakeholders for use in the development or operation of the system.	-Define the intended context of use including boundaries
b) Infrastructure requirements	•Identify, specify and produce the infrastructure for the system. •Build required competencies into training and awareness programs. •Define the global numbers, skills and supporting equipment needed to achieve those tasks.	-Identify staffing requirements and any training or support needed to ensure that users achieve acceptable performance
c) User requirements	•Set and agree the expected behaviour and performance of the system with respect to the user. •Develop an explicit statement of the user requirements for the system. •Analyze the user requirements. •Generate and agree on measurable criteria for the system in its intended context of use.	-Scenarios -Personas -Storyboards -Establish performance and satisfaction goals for specific scenarios of use -Define detailed user interface requirements -Prioritize requirements (eg QFD)
5. Architecting solutions a) System architecting	•Generate design options for each aspect of the system related to its use and its effect on stakeholders. •Produce user-centred solutions for each design option. •Design for customization. •Develop simulation or trial implementation of key aspects of the system for the purposes of testing with users. •Distribute functions between the human, machine and organizational elements of the system best able to fulfil each function. •Develop a practical model of the user's work from the requirements, context of use, allocation of function and design constraints for the system. •Produce designs for the user-related elements of the system that take account of the user requirements, context of use and HF data. •Produce a description of how the system will be used.	-*Function allocation* -Generate design options -Develop prototypes -Develop simulations
b) Human elements	•Decide the goals, behaviours and tasks of the organization [that influence human resources] •Define the global numbers, skills and supporting equipment needed to achieve those tasks. •Identify current tasking/duty •Analyze gap between existing and future provision	-Work domain analysis -Task analysis -Participatory design -Workload assessment -Human performance model -Design for alertness

	•Identify skill requirements for each role •Predict staff wastage between present and future. •Calculate the available staffing, taking account of working hours, attainable effort and non-availability factor •Identify and allocate the functions to be performed Functional decomposition and allocation of function. •Specify and produce job designs and competence/ skills required to be delivered •Calculate the required number of personnel. •Generate costed options for delivery of training and/or redeployment •Evolve options and constraints into an optimal [training] implementation plan (4.3.5) •Define how users will be re-allocated, dismissed, or transferred to other duties. •Predict staff wastage between present and future. •Calculate the available staffing, taking account of working hours, attainable effort and nonavailability factor. •Compare to define gap and communicate requirement to design of staffing solutions.	-Plan staffing
c) Hardware elements	See a) System architecting.	-Prototyping and usability evaluation -Physical ergonomics -Participatory design
d) Software elements	See a) System architecting.	-User interface guidelines and standards -Prototyping and usability evaluation -Participatory design
6. Life-cycle planning a) Planning	•Develop a plan to achieve and maintain usability throughout the life of the system. •Identify the specialist skills required and plan how to provide them.	-Plan to achieve and maintain usability -Plan use of HSI data to mitigate risks
b) Risks	•Plan and manage use of HF data to mitigate risks related to HS issues. •Evaluate the current severity of emerging threats to system usability and other HS risks and the effectiveness of mitigation measures. •Take effective mitigation to address risks to system usability.	-HSI program risk analysis
c) User involvement	•Identify the HS issues and aspects of the system that require user input. •Define a strategy and plan for user involvement. •Select and use the most effective method to elicit user input. •Customize tools and methods as necessary for particular projects/stages. •Seek and exploit expert guidance and advice on HS issues.	-Identify HSI issues and aspects of the system requiring user input -Develop a plan for user involvement -Select and use the most effective methods -Customize tools and methods as necessary
d) Acquisition	•Take account of stakeholder and user issues in acquisition activities.	-Common Industry Format
e) Human resources	•Implement the HR strategy that gives the organisation a mechanism for implementing and recording lessons learnt •Enable and encourage people and teams to work together to deliver the organization's objectives. •Create capability to meet system requirements in the future (conduct succession planning) •Develop and trial training solution to representative users. •Deliver final training solutions to designated staff according to agreed timetable. •Provide means for user feedback [on human issues].	
7. Evaluation a) Risks	•Assess the health and well-being risks to the users of the system. •Assess the risks to the community and environment arising from human error in the use of the system. •Evaluate the current severity of emerging threats to system usability and other HS risks and the effectiveness of mitigation measures. •Assess the risks of not involving end users in each evaluation.	-Risk analysis (process and product)
b) Plan and execute	•Collect user input on the usability of the developing system. •Revise design and safety features using feedback from evaluations. •Plan the evaluation. •Identify and analyze the conditions under which a system is to be tested or otherwise evaluated. •Check that the system is fit for evaluation. •Carry out and analyze the evaluation according to the evaluation plan. •Understand and act on the results of the evaluation.	-Obtain user feedback on usability -Use models and simulation
c) Validation	•Test that the system meets the requirements of the users, the tasks and the environment, as defined in its specification. •Assess the extent to which usability criteria and other HS requirements are likely to be met by the proposed design.	-Compare with requirements -Common Industry Format for usability reports -Performance measurement
d) HSI knowledge	•Review the system for adherence to applicable human science knowledge, style guides, standards, guidelines, regulations and legislation.	

e) Staffing	•Decide how many people are needed to fulfill the strategy and what ranges of competence they need. •Develop and trial training solution to representative users. •Conduct assessments of usability [relating to HR]. •Interpret the findings •Validate the data. •Check that the data are being used.	*HR*
8. Negotiating commitments a) business case	•Contribute to the business case for the system. •Include HS review and sign-off in all reviews and decisions	-Program risk analysis
b) requirements	•Analyze the user requirements. •Present these requirements to project stakeholders for use in the development and operation of the system. •Identify any staffing gap and communicate requirement to design of staffing solutions.	-Value-based practices and principles (identify success critical stakeholder requirements) -Common Industry Specification for Usability Requirements -Environment/organization assessment
9. Development and evolution	•Maintain contact with users and the client organization throughout the definition, development and introduction of a system. •Evolve options and constraints into an implementation strategy covering technical, integration, and planning and manning issues. •	-Risk analysis (process and product) -User feedback on usability -Use models and simulation -Guidelines: Common Industry Format for usability reports -Performance measurement
10. Monitoring and control	•Analyze feedback on the system during delivery and inform the organization of emerging issues. •Manage the life cycle plan to address HS issues. •Take effective mitigation to address risks to system usability. •Take account of user input and inform users. •Identify emerging HS issues. •Understand and act on the results of the evaluation. •Produce and promulgate a validated statement of staffing shortfall by number and range of competence.	-Organizational and environmental context analysis -Risk Analysis -User feedback -Work context analysis
11. Operations and retirement a) Operations	•Analyze feedback on the system during delivery and inform the organization of emerging issues. •Produce personnel strategy. •Review the system for adherence to applicable human science knowledge, style guides, standards, guidelines, regulations and legislation. •Deliver training and other forms of awareness-raising to users and support staff. •Assess the effect of change on the usability of the system. •Review the health and well-being risks to the users of the system. •Review the risks to the community and environment arising from human error in the use of the system. •Take action on issues arising from in-service assessment. •Perform research to refine and consolidate operation and support strategy for the system.	-Work context analysis -Organizational and environmental context analysis
b) Retirement	•Collect and analyze in-service reports to generate updates or lessons learnt for the next version of the system. •Identify risks and health and safety issues associated with removal from service and destruction of the system. •Define how users will be re-allocated, dismissed, or transferred to other duties. •Plan break-up of social structures. •Debriefing and retrospective analysis for replacement system.	
12. Organizational capability improvement a) HSI capability data collection, analysis, and improvement	•Identify and use the most suitable data formats for exchanging HF data. •Have a policy for HF data management. •Perform research to develop HF data as required. •Produce coherent data standards and formats. •Define rules for the management of data. •Develop and maintain adequate data search methods. •Feedback into future HR procurement, training and delivery strategies.	-Assess and improve HSI capability
b) Organizational skill/career and infrastructure development planning and execution	•Define usability as a competitive asset •Set usability, health and safety objectives for systems •Follow competitive situation in the market place •Develop user-centred infrastructure. •Relate HS issues to business benefits. •Establish and communicate a policy for human-centeredness. •Include HR and user-centred elements in support and control procedures. •Define and maintain HCD and HR infrastructure and resources. •Increase and maintain awareness of usability.	-Develop and maintain HSI infrastructure and resources -Identify required HSI skills -Provide staff with HSI skills -Establish and communicate a policy on HSI -Maintain an awareness of usability

	•Develop or provide staff with suitable HS skills. •Take account of HS issues in financial management •Assess and improve HS capability in processes that affect usability, health and safety. •Develop a common terminology for HS issues with the organization. •Facilitate personal and technical interactions related to HS issues. •Feedback into future HR procurement, training and delivery strategies. •Create capability to meet system requirements in the future (conduct succession planning) •Identify any opportunities for redeployment. •Develop a strategy for [HR] data gathering	

Users' Practices and Software Qualities: a Dialectical Stance

Alessandro Pollini
Interaction Design Area,
Communication Science Dpt.,
University of Siena
Via Roma 56, 53100, Siena
0039 0577 270565
pollini@media.unisi.it

ABSTRACT
The Ubiquitous Computing technology in practice is often characterized by users that experience recurring breakdowns, standards' incompatibility and a proliferation of interfaces when using, accessing and trying to connect different devices (e.g. PCs, cameras, printers, and phones). Such interconnected devices populate ordinary Ubiquitous Computing scenarios.

The focus of the present research is on how software architecture can support Ubiquitous Computing applications and how people might use these technologies to enhance their practices and reach personal goals. Architectural support is indeed needed for designing embedded, distributed, intelligent and interactive systems, which need communication through middleware components.

Use practices and Architectural Qualities have been investigated in the Active Surfaces case study. Active Surfaces is an embedded and modular system of tiles aimed at supporting therapeutic use practices and special needs. The design and developmental process is articulated on the relationship and the exchange between key users practices and architectural qualities.

Categories and Subject Descriptors
D.3.3 [**Programming Languages**]: Language Contructs and Features – *abstract data types, polymorphism, control structures.*

General Terms
Design, Performance, Experimentation.

Keywords
Software architecture, Ubiquitous computing, Usability, User requirement, Participatory Design.

1. INTRODUCTION
Design and development of software architectures for ubiquitous systems have been a major concern in academic research and industry [1] and how architectures impact *real use* and *usability* have also become issue of research interest. Usability benefits have been widely applied to individuals performing on a desktop computer but need now to be re-examined within the context of distributed, interactive, networked and embedded applications. Usability studies, which traditionally approach aspects specific to a given task or application, have to be reinterpreted and adapted to Ubiquitous Computing application systems, wherein networks of laptops, PDAs, wearable computers, mobiles and other distributed devices are constructed, de-constructed and integrated.

Designers and developers must also find ways in which sensitive, responsive and intelligent UbiComp technology can also become *usable*, i.e. noticeable, comprehensible, adaptable and easy to control. That is why usage and usability concerns need to be reconsidered outside of the desktop metaphor. Achieving usability traditionally depended on how the functions provided by the system were understandable and clearly visible through the user interface. In this paradigm users have many input and output peripheral devices and the overall system *interface* must be adequate for their needs. There is a multitude of interfaces and usability issues for each mobile device of the distributed and ubiquitous system, and this requires a unique and enabling *software architecture* that must be designed according to users' needs.

In this paper we primarily discuss the interplay between software architecture development and users practices by focusing on the *architectural qualities* peculiarity of designing ubiquitous systems for users with special needs and diverse abilities through the case of Active Surfaces, a modular system of tiles used for play and therapy in water.

Active Surfaces relies on the service-oriented architecture developed in the EU funded IP PalCom, Palpable Computing [2]. We will discuss the interplay between users' practices and software architecture development by experimenting with the Active Surfaces with therapists and children with special needs.

By focusing on those attributes that support *palpable use* of technology, that we henceforward call *Qualities*, we also consider the architectural attributes required by *usable* ubiquitous technology.

2. ARCHITECTURE AND USE
The software architecture has been explored and experimented in different application prototypes related to the Health Care and the Landscape Architecture domains [2]. Each *general scenario* is characterized by an application prototype in which the architecture, or part of it, has been experimented. The application prototypes served as testbeds for the development of the

I-USED'08, September 24, 2008, Pisa, Italy

architecture and as case studies that provide the requirements coming from the field studies.

The *architectural qualities* have been introduced and described as the meeting point between architecture and use/application. The peculiarity of these non-standard architectural qualities is that they both evolve and gain meaning through software development and investigation of key *user practices* to account for.

In fact the *architectural qualities* and the *user practices* are tightly coupled and represent the two perspectives adopted in this research: the software architecture engineering and the interaction design perspective.

In order to better focus on users and architecture it is necessary to describe the Active Surfaces application prototype. The concept, the system and the architecture are described below.

2.1 Active Surfaces

Active Surfaces is a modular system constituted by physical and interactive units, the tiles. They are interactive modules that *activate* the surfaces of the swimming pool by making the environment featured with a network of distributed interactive components [3][4]. In particular, the prototype as developed in this research affords the horizontal configuration of the tiles on the water surface.

Figure 1. The current Active Surfaces prototype

The tiles constitute a network of physical (and software) objects that communicate and exchange data. Each Active Surfaces tile is thought of as a modular unit that can communicate with the others through its six sides. These entirely homogeneous devices, the tiles - which have exactly the same physical characteristics and computation and communication resources - are assembled. Each tile is an independent, physical, tangible object that can be picked up and moved around, and the interaction between the tiles is coherent and straightforward: all the tiles can communicate with their adjacent neighbours. They are, in fact, able to recognize their relative position as being essentially positioned and orientated in a sequence of tiles.

The Active Surfaces is highly scalable in respect to computational power and number of components. In fact it can scale up or down (vertically) by adding or removing resources to a single node in a system, typically involving the addition or removal of CPUs or memory to a single tile. Active Surfaces can also scale out (horizontally) by the addition of more nodes to a system, such as adding new tiles to the distributed system.

The concept emphasizes issues related both to the *use*, such as physical manipulation, positioning and emergent uses of the system, and the *architectural platform*, like the networking and dynamic assembly of tiles that is configured purposely [3][4].

3. SPECIAL NEEDS AND USERS' PRACTICES

In order to better focus on use practices as they emerge in the Active Surface application prototype it is thus necessary to describe the target users profiles - that is, the therapists and caregivers together with the disabled children - their needs, wishes and abilities [4].

Together with the study of the domain and a survey of the enabling technologies [5], fieldwork has been carried out with the aim of directly exploring the field of therapeutic intervention in water. The fieldwork has been conducted in two settings for psychomotor therapy in water, the Disabled Children Parents Association, Siena and the D. Chiossone Institute in Genova. We adopted ethnographic methods - such as field observation and interviews - and design methods - such as user workshops and creative brainstorming. The ethnographic activities attempted to observe and reveal relevant issues related to the environment (the features of the water, the physical structure of the swimming pool), the actors (therapists, disabled children, parents), the tools (objects, toys and water noodles) and, above all, the activities (the procedures, the different phases, the practices). We have addressed the whole practice starting from the *planning*, entering the *activity* and proceeding with the *evaluation* phase [5].

We will exclusively focus here on the overall description of users' needs and therapists practices in order to understand the implications they have on software architecture development.

Figure 2. Playing domino like games with Active Surfaces

The main actors of this therapeutic setting are the children with special needs. Children with very diverse profiles actually benefit from therapeutic play in the water. The users we have observed can be summarized in three main groups described below:

Autistic Spectrum Disorders and Other Affective and Socio-Relational Disturbances. People with autism have impaired social interaction and social communication and have a limited range of imaginative activities. People with autism have a tendency toward repetitive behaviour patterns and resistance to any change in routine. They need to be instructed and supported during the game, otherwise they very quickly return to their own solitary 'obsessive activities'.

Physical and Motor Disabilities and Cerebral Palsy. These children have limitation or an impossibility of movement, restrictions in force, abnormal postures, the presence of

neurological movement disorders such as dystonia, tremor, ataxia, etc. Children with cerebral palsy can be severely impaired in playing by their motor disability, but also by speech and communication disabilities, and sensory impairments (visual and/or hearing).

Mental Retardation/ Intellectual Disabilities/ Learning Disabilities. Children with mental retardation (also referred to as intellectual disabilities or learning disabilities, for example children with Down's syndrome), have a reduced capacity for attention and might not understand the meaning of the proposed activity. They might not understand the meaning of language and many of them have speech limitations too.

3.1 Key Practices

The therapists and trainers are the other main actors of this setting. They essentially have the role of facilitating the playful physical, social and emotional experience. They have to mediate the social relationships, the experience in the water and offer a reassuring presence to the child. They are the scaffolds that allow the child to express and freely explore the space of the pool. The therapists have to facilitate the activity, and not impose rules or, on the opposite extreme, abandon the child without a guide. Even when the child would like to explore by herself the therapist should also be present and support her independent action. The intervention is considered successful when the therapist interprets the meanings of the behaviors of the child. Having an intimate knowledge of the child is central to achieving this interpretation.

The outcomes of this activity resulted in key observations that have informed the whole design process. They can be summarized as follows:

Looking for creative solutions: The therapists usually deal with dynamic settings and changing conditions. This implies the ability to manage and rearrange the available resources in purposeful and creative ways.

Dynamic configuration of the tools: In dealing with continuously changing conditions and rehabilitation demands, the therapists should always find new solutions for adapting their tools and the environment to the patients and for maintaining their attention throughout the session. Consequently a core characteristic is that the tools have to be easily re-configurable and adaptable to this evolving situation.

Resource availability and opportunities for action: The therapist needs to feel in control of the available resources and how they might be adopted, changed and exploited. As in many workplaces, since their attention is exclusively directed to the patients, the resources the therapists use have to be ready at hand and immediately understandable.

Exploration and performance: This practice facilitates and encourages exploratory experimentation by users. Tools have to be used, customized and altered according to established degrees of freedom and constraints.

The key therapist practices are among the outcomes of the field exploration of the application sites and have continuously informed the development of the software architecture.

4. RESEARCH METHODOLOGY

Dealing with diverse and special users requires that methods and experimental environments would be appropriate, i.e. non-obtrusive, able to be personalized, adaptable, and capable of anticipating emerging user needs [6].

A wide variety of methods have been used throughout the iterative design life cycle [5]. These methods pertain to Human Computer Interaction, Participatory Design and Software Architecture Engineering. In particular we integrated a participatory design perspective with a co-evolutionary approach to interaction design and we explored this methodology in the domain of software architecture design. The process is co-evolutionary since architectural development, site exploration, activity analysis and concept design have been carried out in parallel so that each path of the process can inform, without constraining, the others.

We especially highlight on how the use of scenarios helped the structuring of data gathered through activity analysis, the envisioning of the role and functionalities of the system, and the assessing and validating the envisioned solutions from an architectural perspective (see [7][8] for scenario-based evaluation methods).

Throughout this research the scenarios are used to step through the software architecture and to document the consequences of architectural solutions from a user perspective. Different kinds of scenarios drove the research process: Activity scenarios, Envisioning scenarios, Prototype scenarios and Qualities scenarios [5]. We will focus here on Activity and Qualities scenarios that better represent the dialogue between Application and Architecture.

Activity scenarios stem from the fieldwork and activity analysis. They are grounded and built on data collected with ethnographic observation and user research. Activity scenarios account for concrete use episodes and key practices. We used the Activity Scenarios to understand, as thoroughly as possible, what is relevant and appropriate in the specific domains of use, which in this case study was the therapeutic practice in the water. These issues have thus been evolved into user requirements that informed the definition of the envisioned solutions at the software architectural level.

The key User Practices also were the criteria to define the experimental plan with the architectural prototype and the evaluation framework. In fact in this research *experimental architectural prototypes* have been used to conduct experiment on the architectural qualities that we have analyzed, in particular those observable at run-time (like performance) [9]. The experimental architectural prototypes allowed concrete measurements to be made under a range of different situations that might be also defined in terms of Qualities scenarios. They will be described in Par. 6.1.

Qualities scenarios consist of a slight adaptation of the quality attribute scenarios [1][10] that are a way to make the Qualities for palpable systems operational. They are short technical scenarios referred to specific Qualities. Qualities scenarios provide a way to concretely measure whether the architecture fulfils the requirements of the scenario. It states measurable properties of an architecture by defining metrics to be used in performance testing of the architecture. These scenarios allowed us to experiment with and evaluate specific features of the technology by testing the Qualities of the software architecture.

5. ARCHITECTURAL QUALITIES

In the multiple iterative cycles of the process followed in this research, scenarios have been used to bridge the use practices and the architectural development. The key practices have been discussed in terms of system use and from the software architecture perspective. The Architectural Qualities are summarized below:

USERS PRACTICES	*ARCHITECTURAL QUALITIES*
Looking for creative solutions	Assemblability
Dynamic configuration of the tools	Adaptability
Resource availability and opportunities for action	Resource Awareness
Exploration and performance	Experimentability

Table 1. From Users Practices to Architectural Qualities

Each Quality comes from an iterative design and development in which user participation and technological challenges were interwoven strands of the whole process.

Assemblability. Each Active Surfaces tile is identical and interchangeable and can run any piece of code that is passed to it through a neighbour, included the game logics. They can be assembled in many different formations that take into account the tiles' communication capabilities and the surfaces on which they have to be placed. Each formation of tiles is instantiated as a functional and physical Assembly of devices and services. The Assembly takes form as the users construct it by means of the Assembler Tile. The Assembly can then be dynamically altered and adapted over time. Despite the stability it has when it is created, the Assemblies can be easily deconstructed and re-constructed in a different formation being supported by flexible ad-hoc networks that can be controlled and configured by end users.

Adaptability. The Active Surfaces system consists of a set of tiny, resource constrained computers that can be arranged together to create a physical network. Because the tiles can only communicate with their close neighbours, there is an explicit and consistent discovery and communication framework underpinning the whole system. The tiles can be arranged in three-dimensional patterns, like squares in a crossword puzzle, and tiles, which are stacked one on top of the other, communicate through the top and the bottom. The network can be easily reconfigured by picking up a tile and moving it; this movement immediately changes the feedback that is provided.

Resource Awareness. The tiles are embedded systems with powerful and limited resources at the same time, such as available energy, available memory or communication bandwidth. Because of the limitations of these devices they represent a concrete challenge for the developers of the software architecture. In Active Surfaces a game application can exist within a network, rather than on a single unit or a central mainframe. Through the networking among the tiles and the instantiation of the assembly, they can discover the resources present in the system and debug the behaviour of such resources in order to overlook malfunctions or degraded individual or generalized performance. The resources are monitored and managed throughout time.

Experimentability. Active Surfaces can be thought of as a toy problem to experiment the software architecture because of its peculiar characteristics, as a modular system made of small easy to handle units. The tiles can be experimented with and tested without altering the structure of the system or causing any malfunctions or error. Indeed, Active Surfaces has to operate even despite the presence of an error in the use. An error is a condition of exception resulting from some deviation from the expected behaviour, which leads to a fault or failure, and the design of the architecture aims at minimizing the eventual adverse consequences of accidental or unintended actions.

These Qualities should not be considered in isolation, but rather as interwoven contributory factors that exhibit dependencies and influences on one another. The purpose of the Qualities is to capture the essence of what defines the nature of usable, easily perceivable and understandable (in a word, *palpable*) ubiquitous computing applications.

6. EXPERIMENTING WITH THE SOFTWARE ARCHITECTURE

The goal of this experimental phase is to describe the behaviour of the Active Surfaces system by measuring the performance of the architectural prototypes. The Qualities scenarios help in describing the performance in terms of more informative detailed statements. These statements allow *quantifiable arguments* about a system to be made [10].

Empirical testing is possible when relevant requirements and architectural components have been identified and prototypes have been developed. In particular the Active Surfaces architectural prototypes, described in the following paragraph, were used to observe, explore and evaluate the Architectural Qualities.

6.1 Architectural Prototypes

Prototypes of software components with different levels of accuracy and completeness have been used throughout the process. Their usage in architectural development provided the opportunity to have intermediate embodiments of the systems' functionality even if not supposed to represent any final or complete stage.

The Active Surfaces system underwent a concurrent development either within the Simulation Framework and the Hardware Platform. The hardware platform selected for the Embedded Architectural prototype is the UNC20 microcontroller. With such small microprocessor only the PalVM, the Virtual Machine developed within the PalCom project [2], is supported as a runtime engine.

The embedded architectural prototype has been built to learn about the PalVM platform and the serial communication over IR. The testing aims at discriminating whether there are restrictions in the PalCom open architecture or if the constraints are due to the current hardware implementation (e.g IR communication implemented over serial port).

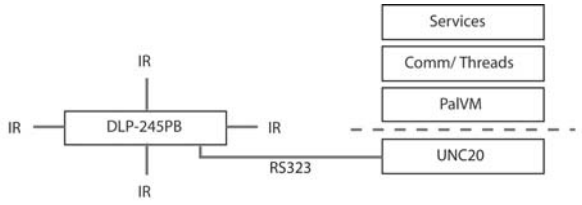


Figure 3. PalCom tile stack

The middleware management layer, which consists of managers handling resources, services, assemblies, and contingencies, requires too great a memory footprint to fit into the 8MB memory of the UNC20. Therefore, the software for the tiles has been developed to run on a standard PC with simulated infrared communication in concurrence with the development of the hardware for the tiles and the optimization of the middleware management layer. On the desktop machine the simulated framework runs on top of Sun's JavaVM.

The tiles deployed as simulated devices on a desktop machine are expected to have an optimal performance and can still exhibit a certain level of experimentability through the simulated game with a graphical user interface. In fact the therapists had the valuable opportunity to exploit the opportunities provided by the middleware managers, even if within the simulation framework The architecture experienced on the Simulated Framework was likely to inform the development of the embedded applications.

6.2 Performance Testing

The Performance Testing have been organized around tasks designed in order to translate the Qualities, and therefore with a relation to the Users' Practices, in measures observable via execution. The tasks aim at demonstrating how the existing architectural components would behave in performing the Active Surfaces scenario, e.g. performing the assigned activities.

The performance testing is based on a user-oriented perspective and assumes human practice in the therapeutic setting. In particular time responses, delays or frequency of errors have been observed with respect to the requirements coming from the activity analysis. For what regards timeliness, the major requirements from the therapeutic activity in the water are the duration of the whole session (45 minutes), the pace of the interaction (cycles of 3 to 5 minutes games to the utmost) intervened by the restless time pauses (2-3 minutes). These data allowed us to define the baseline for the experiments [5].

In order to determine whether there are restrictions in the software architecture or if the eventual constraints are due to the current hardware implementation, we have organized testing around two different conditions: 1) Tasks in which the performance is influenced mainly by the software architecture currently running; 2) Tasks in which the performance is both influenced by the architecture and mostly by the current hardware implementation [10].

In particular the experimental tasks can be grouped into the following areas. Each area represents a way to translate the Architectural Qualities (in brackets) into less conceptual and more verifiable evaluation tasks.

Communication and Discovery (Assemblability and Resource Awareness)

Task (a), (a1): 1+1 tiles, one is still, the other is rotated to reach the correct orientation for the side connection. In one case (a) Two tiles are put together, in the other (a1) two correctly connected tiles are kept apart.

Task (b), (b1): 1+2 tiles, one is still, the other two are rotated to reach the correct orientation at the same time. In (b) three tiles are put together, in (b1) three correctly connected tiles are kept apart.

Task (c), (c1): 1+3 tiles, one is still, the other three are rotated to reach the correct orientation at the same time. In (c) four tiles are put together, in (c1) four correctly connected tiles are kept apart.

The tasks are designed as two series each consisting of 10 repetitions of the tasks. In the first series the tasks are interrupted by re-boot of the game services (Re-boot series), in the other series the tasks are carried out continuously over time (Over time series). The former case represent the normal performance the tiles have on these tasks. The latter evidences how the performance in these specific tests varies over time.

Re-configuration (Adaptability)

The tiles currently can run either *fixed* GameServices, like the Jigsaw Puzzle Fish game (see Figure 1) and the Domino game (see Figure 2); or *open* GameServices where the tiles are in programming mode and learn how to configure by physical programming-by-example. The tiles also run FeedbackServices, like the actual LEDService or the possible VibrationService and SoundService that can be developed in the future.

The Re-Configuration tasks can either mean: choosing among existing pre-defined GameServices or the flexible use of single services related to game configuration, e.g. tiles' sequence, sensing and feedback.

In one case the system should allow shifting between pre-defined GameServices, i.e. different games that have already been configured. In the second case the system should allow running more services at the same time

That's why we launched different services in parallel simulating the two conditions described above. We are able to compare the task under two different conditions represented by the *ist.palcom.tiles.test.fish.prc* services, which involves IR communication among the tiles; and *ist.palcom.tiles.test.timer.prc* which doesn't involve the use of IR communication.

Performance (Experimentability)

Performance comprises 1 task performed under both the experimental conditions, *with* and *without* the use of communication. Thus there is a set of 2 tasks that consist of observing two GameServices running for 30 min.

As mentioned above, the overall session lasts 45 minutes and the duration of a single game situation can be assumed to be 30 minutes at the very most. In fact even if it is possible that children find some games very engaging, it is very hard to carry out the same game for almost the whole session. Furthermore game dynamics usually last few minutes.

7. RESULTS

In this paragraph a short summary of the gathered data is presented. For an extensive overview of the results see [5].

The results related to Communication and Discovery are presented regarding the two series of gathered data (Re-boot and

Over Time series), the two main actions (Put Together and Put Apart) and the scalability factor represented by the number of tiles utilized (2, 3 or 4 tiles).

Conditions	*Tasks*	*2 Tiles*	*3 Tiles*	*4 Tiles*
Re-Boot	Put together	3.2	7	6.9
	Put apart	7.6	9.8	12.5
Over Time	Put together	3.5	7.6	7.5
	Put apart	8.1	10.6	13.3

Table 2. Communication and Discovery. Summary of Results

The comparison among *Communication and Discovery* between 2 Tiles, among 3 Tiles and 4 Tiles, also gives a quantitative measure of how horizontal scalability affects the performance of the tiles system. Active Surfaces is conceived and designed as a modular system that in future implementation will be made of 12 units. The experimental data suggest that the performance of PalVM and the PalCom Communication components should be improved to meet the requirements of a highly scalable system and guarantee acceptable time responses as the number of the modules increase.

Tasks related to *Re-configuration* show how the system supports several services running in parallel and also creative combinations and adaptations of the tiles system. This can be done by shifting among these pre-defined solutions or by flexibly combining single services related to game configuration (e.g. game logics, sensing and feedback).

The eventual shifting among GameServices would be affected by the time required by new services to start, about 10 sec. As we observed through the activity analysis, the pace of the activity in the Active Surfaces scenario would impose a quicker response time for the re-configuration of the system. It is estimated to be no more than 10 sec in order to really provide the user with the experience of ready-at-hand tools. The results show that there is still not adequate support for the multiple services combination, i.e. more than three services running).

Regarding the combination of services, all the VM versions well support two services running in tandem both in tasks involving the use of IR communication or not. Simultaneously running two services, the system coherently exhibits the behaviours defined by the two services. Three services running in parallel are also supported but it seems to affect the behaviour of the tiles by decreasing the overall performance of the PalVM-release. These results are close to what happen with running one service alone and this could prove valuable support for re-configuration.

Tasks regarding the *Performance* over long periods of time show that the current implementation restricts the overall performance of the tiles. In fact, the performance through the LightUp GameService proved to be optimal, while tasks involving the communication modules resulted in a series of malfunctions that negatively affected the overall performance.

The results of the experiments allowed us to revise and elaborate on the initial formulation of the Qualities. For the Architectural Qualities revised see [5].

8. CONCLUSION

In discussing software architecture development and users' practices we have described the integration among the traditional ethnographic studies, participatory design methods and naturalistic experiments to inspire, inform and evaluate the design of software architectures [9]. This approach has already been adopted for the design of ubiquitous computing technologies [11] while it seems to be still fully appreciated in software architecture design [11].

Recently there has been a growing interest in understanding specific evaluation problems that arise from the use of Ubiquitous Computing systems [12]. In such paradigm software and hardware resources are distributed throughout the physical world and this impacts individual and social behaviours. Different evaluation criteria have been outlined, user attention (focus and overhead), the adoption of the system (value and availability) and the qualities of the interaction (physically embeddedness, dynamic input/ output, multiple devices, multiple users). Criteria related to the use and the person, such as understanding, control, accuracy, appropriateness, and customization, are also discussed.

This study helped us to figure out the complexity of such intricate stage where persons and computational resources influence one each other. With this research we wanted to highlight on multifaceted aspects interwoven in the *interplay* between real use and software development.

We observed that the introduction of UbiComp technology affected and changed users' activities and that, at the same time; they became responsible for maintaining, controlling and changing it. The system *architecture* / *use* relationship is dialectical since on one hand, technology enhance certain practices by enabling novel use opportunities, on the other hand user-specific dynamics provoke, inspire and inform the emergence of unpredicted architectural solutions.

In this paper we showed how such interplay took place through the whole research process, i.e. through design and development strategies that accounted for the special needs of the involved users and challenged the development of the system architecture. We wanted to give a feeling of this multiplexed process by describing the design of the experiments and the results. Data gathered during the activity analysis and activity modeling provided the backbone to define the experimental plan and the baseline for the evaluation of the system.

We empirically investigated the dialogue between user studies and software development by means of operative choices. We tried to bridge these two different fields and to take advantage of the methods of each domain. This study also resulted in the investigation of newly emergent interwoven processes that make *use* and *architecture* meeting at the edge, where software Qualities and Users' Practices juxtapose and evolve tightly coupled.

9. ACKNOWLEDGMENTS

Thanks to Prof. Patrizia Marti, Alessia Rullo and Erik Grönvall as they were a part of the research group for the duration of the PalCom project. Thanks to the collegues at the Computer Science Dpt, Univeristy of Aarhus that played a fundemental role in the software architecture development.

10. REFERENCES

[1] John, B. E.; Bass, L. (2001) *Usability and software architecture*. In Behaviour and Information Technology, 20 (5) pp. 329-338

[2] Palcom Project Website, http://www.ist-palcom.org/

[3] Grönvall, E., Marti, P., Pollini, A., Rullo, A. (2006) *Active surfaces: a novel concept for end user composition*, NordiCHI 2006, Oslo, Norway, 14-18 October, 2006.

[4] Pollini A., Grönvall E. (2006) *Constructing assemblies for purposeful interactions*. In Proceedings of Mobile Interaction in the Real World Workshop, MUIA06 at MobileHCI 2006, 8th International Conference on Human Computer Interaction with Mobile Devices and Services. 12 September 2006. Espoo, Finland.

[5] Pollini A., (2008) *Experimenting with an Ubiquitous Computing Open Architecture.* Ph.D. Thesis, University of Florence, Italy, 2008.

[6] Emiliani, P. L., Stephanidis, C. (2005) *Universal access to ambient intelligence environments: opportunities and challenges for people with disabilities*. IBM Syst. J. 44, 3 (Aug. 2005), 605-619.

[7] Kazman, R., Barbacci, M., Klein, M., Carriere, S. J., Woods, S. G. (1999) *Experience with Performing Architecture Tradeoff Analysis*, In proc. of the 1999 International Conference on Software Engineering, pp. 54-63, 1999.

[8] Bengtsson, PO. (2002) *Architecture-Level Modifiability Analysis.* ISBN: 91-7295-007-2, Blekinge Institute of Technology, Dissertation Series No 2002-2, 2002.

[9] Bardram, J. E., Christensen, H. B., and Hansen, K. M. (2004) *Architectural Prototyping: An Approach for Grounding Architectural Design and Learning*. In Proc. 4th Working IEEE/IFIP Conference on Software Architecture, pp. 15-24, 2004.

[10] Bass, L., John, B. E. (2003) *Linking usability to software architecture patterns through general scenarios*. Journal of Systems and Software, 66 (3), 187-197.

[11] Edwards, W. K.; Bellotti, V.; Dey, A. K.; Newman, M. (2003) Stuck in the Middle: The challenges of user-centered design and evaluation for infrastructure. ACM Conference on Human Factors in Computing Systems (CHI 2003); 2003 April 5-10; Fort Lauderdale; FL. NY: ACM; 2003; 297-304

[12] Scholtz, J., Consolvo, S. (2004) Toward a Framework for Evaluating Ubiquitous Computing Applications. IEEE Pervasive Computing Magazine, Vol. 3, No. 2 (Apr-Jun 2004), pp. 82-8.

Fostering Remote User Participation and Integration of User Feedback into Software Development

Steffen Lohmann
University of Duisburg-Essen
Interactive Systems and Interaction Design
Lotharstr. 65, 47057 Duisburg, Germany
steffen.lohmann@uni-due.de

Asarnusch Rashid
Research Center for Information Technology
Information Process Engineering (IPE)
Haid-und-Neu Str. 10-14, 76131 Karlsruhe, Germany
rashid@fzi.de

ABSTRACT
Permanent involvement of end users in software development is both highly recommended and highly challenging. Against the background of our results and experiences from two research projects, we summarize several key issues and design concerns that need to be considered when integrating users and their feedback into software development.

Categories and Subject Descriptors
K.6.3 [Software Management]: *Software development.* D.2.1 [Requirements/Specifications]: *Elicitation methods*. D.2.2 [Design Tools and Techniques]: *User interfaces*. H.5.2 [User Interfaces]: *User-centered design.* H.1.2 [User/Machine Systems]: *Human factors.* I.3.6 [Methodology and Techniques]: *Interaction techniques.*

General Terms
Design, Human Factors

Keywords
Remote User Participation, User-centered Software Development, Distributed Participatory Design, User Interface Annotation

1. INTRODUCTION
Nowadays, software development is increasingly characterized by evolutionary processes and short development cycles. Modern software systems usually need continuous updating, improvement, and customization. Perpetual usability evaluations and user surveys are crucial to guarantee that a software system meets the users' needs. Development concepts such as *Participatory Design in Use* [2] emphasize the importance of continuous user participation. However, the spatial and temporal distribution of system users often limits the possibilities for co-located methods of participatory design. In many cases, user participation is only remotely possible, e.g. via computer-mediated forms of communication.

I-USED'08, September 24, 2008, Pisa, Italy

Within the research projects *SoftWiki* [8] and *CallaBaWue* [3], methods and tools have been developed that ease remote participation of end users in the software development process, in particular with respect to requirements elicitation and usability evaluation. The basic toolset in both projects consists of a collaboration platform and participation channels that enable users to make suggestions for improvements concerning a certain software product (cp. [4, 6]). During the development of these methods and tools as well as in three usability tests and two case studies (one short-term and one long-term) including over 50 participants in total, we got valuable insights regarding successful forms of remote user participation as well as drivers for the integration of user feedback into software development. In the following, we summarize some key issues and design concerns that need to be taken into account when involving distributed users in software development.

2. DIMENSIONS FOR REMOTE PARTICIPATION
Several important issues and conceptual aspects regarding the integration of distributed users to improve software systems, such as the reporting of bugs or remote usability evaluations, are discussed in related work (e.g., [5, 1, 2]). On a general level, we identified three dimensions that appeared to be central when it comes to the implementation of computer-mediated user participation in software development: *degree of autonomy*, *number of users*, and *level of collaboration*.

The degree of autonomy can be divided in the two opposite approaches of *autonomous* and *event-driven participation*. Autonomous participation means that the user decides on his own when to participate. A typical scenario would be that the user expresses requirements whenever they appear in his daily use of a software system. Event-driven participation forms, in contrast, explicitly invite users to participate in certain situations or at particular points in time. Our favorite solution is a combined approach that regularly reminds the users that they can influence the system design and inspires participation by providing certain topic frames and at the same time allowing them to contribute at any time, independently of the particular development status. Especially the last aspect seems to be crucial, since we experienced that test users very much liked the possibility of being able to express requirements immediately whenever they occur while interacting with the system.

In addition, the optimal form of participation support relies largely on the *number of users* that are expected to be actively

involved in the development. The higher the number of participants, the more important are mechanism that guarantee systematic and structured elicitation and analysis that can handle large amounts of requirements. For instance, within the tool *OpenProposal* [6], we made promising experiences with the digital annotation of user interfaces that are automatically saved as screenshots and send to the developers. In cases where large numbers of users participate, automatic utilization of user annotations proofed to be useful as in the tool *Softfox* [4] that supports direct linking of user input to the application structure or underlying system models, in particular if a model-driven development [7] approach is being used.

A third design dimension concerns the *level of collaboration,* both among users and between users and developers. A central question is whether users provide requirements individually and independently or if the requirements are collaboratively developed and improved. Within our research, we came to the result that sophisticated solutions should collect the user at the point he is willing to participate. For instance, a web platform can provide collaboration support such as commenting, discussion, or cooperative editing features as well as possibilities to rate, vote, and link requirements. Embedded participation channels can be moreover provided for those users who are willing to participate but are not willing to deal with the collaboration platform. However, some kind of awareness regarding already existing requirements should be given in any case – this reduces the effort for the user as he does not need to formulate a requirement a second time that has already been expressed. Furthermore, the amount of redundant requirements is reduced leading to lower effort in analyzing the requirements.

3. FURTHER ASPECTS

Next to these basic design dimensions, we identified several aspects that we regard as highly valuable for successful implementation of remote user participation in software development. In the following, we summarize further key issues that can help to lower the participation barrier and to better link user feedback with the software product.

3.1 Reducing the Participation Barrier

Integration into the user's environment: The participation interfaces should at best be directly embedded into the user's system environment to establish an affordance always reminding the user that involvement and thus system improvement is possible. For instance, some kind of 'participate'-button can be constantly visible on the desktop or can be integrated into the interface of the web browser or application of interest.

Lightweight Participation: It should be possible for the user to participate whenever an idea for improvement comes to his mind, resulting in only a marginal interruption of his actual activity or workflow. At best, the user should decide what and how much information he wants to provide. The initial input should be based on a lightweight and informal process that can later be refined and elaborated.

Simplicity and Assistance: All interactions with the user interface should be as simple and self-explaining as possible in order to encourage users getting involved. The interface should not require to login each time the users express a requirement; appropriate interaction support, such as automatic form filling or system suggestions, should moreover be provided. The user should furthermore not be enforced to provide extensive data or make classification decisions that are cognitively challenging. However, too much assistance, such as pre-defined templates or automatic system proposals, can also have a negative impact on the creativity of the user.

Transparency: In every situation, it must be clear to the user what data is captured along with his input. Ideally, the user can continuously track the progression of his requirements in the development process. The user's motivation is heavily based on the fact that he recognizes how the system is improved as a consequence of his input, which might lead to a personal benefit.

2.1 Linking User Input to Software Artifacts

Most user requirements refer to specific artifacts of the software system. We found that both – users and developers – can benefit from options allowing to implicitly or explicitly link requirements to parts of the software system.

A key feature of our tools that has been rated as highly valuable in user tests is the possibility to directly refer to elements of the graphical user interface while formulating requirements. This is either realized by digital annotation (in case of *OpenProposal*) or by direct selection of web elements (in case of *Softfox*). The assumption of this feature is that many software artifacts have a representation in the user interface, in particular artifacts that end-users refer to. On the one hand, references to the user interface ease the requirements formulation for the user as he does not need to textually describe the interface elements but can directly point at them. Furthermore, this concretizes and illustrates his ideas for improvement and can reduce typical problems that often arise from text-only communication such as misconceptions due to wrong word choice, incomplete data, or descriptions that are too elaborate. On the other hand, the application context can provide valuable assistance in systematically analyzing the user requirements; the analyst can, for instance, inspect all requirements at once that refer to a certain element of the user interface.

4. CONCLUSION AND OUTLOOK

This position paper reported several aspects we experienced as valuable to foster user participation in distributed settings and help to integrate feedback in the software development process. However, we have not discussed in what ways developers have to rethink and change their habits to make remote user participation successful. This remains a topic for future work.

5. REFERENCES

[1] Castillo, J.S., Hartson, H.R., Hix, D. 1998. Remote Usability Evaluation: Can Users Report Their Own Critical Incidents? In CHI'98 Human Factors in Computing Systems, 253-254.

[2] Draxler, S., Stevens, G. 2006: Getting Out of a Tailorability Dilemma. In Informatik 2006 – Informatik für Menschen, 1, LNI P-93, 576-579.

[3] CollaBaWue – research project, funded by the Landesstiftung Baden-Wuerttemberg Foundation, see http://www.collabawue.de/

[4] Lohmann, S., Ziegler, J., and Heim, P. 2008. In Engineering Interactive Systems 2008, LNCS 5247, 221–228, in press.

[5] Nichols, D.M., McKay D., Twidale, M.B. 2003. Participatory Usability: Supporting Proactive Users. In Proc of 4th ACM SIGCHI NZ Symposium on Computer-Human Interaction (CHINZ'03), 63-68.

[6] Rashid, A., Wiesenberger, J., Meder, D., Baumann, J. 2008. In Proc of the PRIMIUM Subconference at the Multi-konferenz Wirtschaftsinformatik (MKWI), CEUR-WS 328.

[7] Schmidt, D.C. 2006. Model-Driven Engineering. IEEE Computer 39(2).

[8] SoftWiki – research project, funded by the German Federal Ministry of Education and Research (BMBF), see http://softwiki.de

Designing Usable Applications based on Web Services

Fabio Paternò, Carmen Santoro, Lucio Davide Spano
HIIS Laboratory – ISTI-CNR
Via Moruzzi 1
56126 Pisa, Italy
{fabio.paterno, carmen.santoro, lucio.davide.spano}@isti.cnr.it

ABSTRACT
One trend in software development is to implement application functionalities through Web services. This eases the possibility of developing interactive applications exploiting functionalities implemented by others. In this paper we discuss the issues raised when designing user interfaces for these types of applications. In particular, we describe a possible approach to address them based on the use of model-based user interface descriptions with the possibility of obtaining versions adapted to different types of interactive devices. The development of the final user interface is supported by a semi-automatic process in which at first the designers take benefit of an authoring tool able to automatically generate the first version of the user interface and then, after an evaluation phase of of the resulting user interface, they can manually make further modifications and refinements in order to obtain highly usable user interfaces.

Categories and Subject Descriptors
H5.m. Information interfaces and presentation (e.g., HCI).

General Terms
Design,

Keywords
Model/based design. Usability, Web services.

1. INTRODUCTION
One current trend in software development is the use of Web services. They have been introduced to better support software–to–software communication. This is achieved through the WSDL (Web Service Description Language) description associated with each service, an XML-based description of the possible operations, and input/output parameters. The basic idea is to ease the development of applications based on the SOA (Service–Oriented Architecture) approach in which often applications developers have to design access to services and their compositions developed by others.

Meanwhile, recent years have also assisted to a renewed interest in model-based user interface design and development because logical descriptions allow designers to better manage the complexity of multi-device environments (see for example [1], [4]). This is usually obtained by exploiting XML logical descriptions and associated transformations for the target devices and implementation languages. Having the possibility of specifying a user interface at different levels has several advantages: it helps designers because the separation in different abstraction levels provides different "views" of the same user interface, and the selection of the most appropriate view is performed by the designers depending on the specific aspects they are currently interested in, and/or on their specific skills. In addition, it is worth pointing out how not only designers can be involved in the approach, but also other stakeholders can play a relevant role in the process. Indeed, as the method is supposed to be iterative and refinement-based with a semi-automatic approach there is enough room for even an early intervention of evaluation in the process, to the aim of identifying usability problems and include the design of their solutions as soon as possible in the user interface software lifecycle.

In addition, the information contained in the models can be exploited both at design and at run time. Therefore, the use of models does not pose any particular constraints to when and how the models should be used. Maintaining links among the elements in the various abstraction levels enables e.g. linking semantic information (such as the activity that users intend to do) with more concrete levels, up to the implementation levels, and this can be exploited in many ways. For instance, such links can be automatically supported by suitable transformations, which are useful for obtaining a description in a specific abstraction level, once a description in a different level is available, not forcing designers to build all the different descriptions or to use any specific model.

If we consider the abstract level, it is generally recognised that the main benefits in using an abstract description of a user interface is for the designers of multi-device interfaces, because they do not have to learn all the details of the many possible implementation languages supported by the various devices. Thus, one advantage of using the abstract levels of a user interface is that designers can reason in abstract terms without being tied to a particular platform/modality/implementation language. In this way, they have the possibility to focus on the 'essence' of the interaction (e.g.: what is the intended effect the interaction wants to achieve/support?), regardless of the details and specificities of the particular environment considered. In addition, considering the abstract level of the user interface appears to be particularly useful when the user interfaces are aimed at handling Web Services. Indeed, WSDL files provide a description of the operations supported by the Web Services. The relationships of such

I-USED'08, September 24, 2008, Pisa, Italy

descriptions of the operations (contained in WSDL files) with the abstract user interface objects expected to support them in the resulting user interface (contained in the abstract user interfaces) is an interesting issue to investigate, and appears to be promising in helping to solve the problem of generating user interfaces for Web Services.

This is a novel problem. Indeed, most model-based approaches have not addressed the specific issues related to Web-service based applications. Work on generating user interfaces for Web services but without using model-based approaches has been carried out at Dresden [6] and Yonsei [5] universities. The limitation is that such works usually consider direct mappings between Web services functional interface and an implementation language for user interface, which cannot be exploited when devices supporting different implementation languages are considered.

abstract interface, concrete interface, implementation) does not seem particularly effective in this context for various reasons. One is that designers and developers have to create interactive applications accessing application functionalities developed by others. Thus, they have to focus their effort on how to take into account the specific requirements that the application interface of the existing Web services pose. In addition, they also have to indicate how to compose functionalities implemented in different Web services.

Our approach (see Figure 1) is to have first a bottom-up step in order to analyse the Web services providing functionalities useful for the new application to develop. In this phase an analysis of the operations (OP1, .. OP4 in Figure 1) and the data types (DT1, .., DT4 in Figure 4) associated with input and output parameters is carried out in order to associate them with abstract interaction objects suitable to support presentation of their values and their modification.

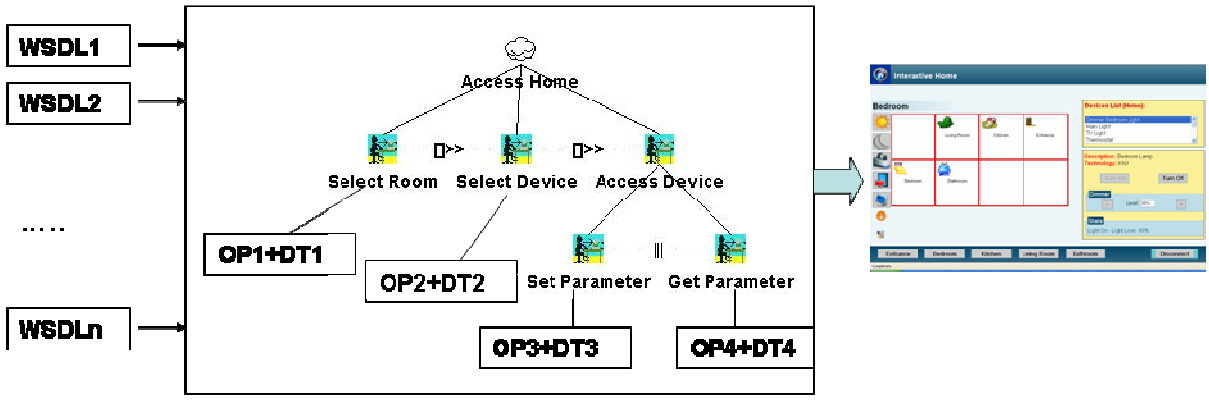


Figure 1. The Proposed Approach.

Work by Vermeulen et al. [7] aimed to solve such issues by extending Web services with OWL-S combined with task and layout model. This approach requires a lot of manual work by the designers.

In general, the type of issue that we have to solve is how to associate information regarding the data types and information on the user interface. Indeed, while the semantic Web has mainly focused on the data semantics through the use of ontologies and languages that allow for more intelligent processing, user interface models allow designers to consider the semantics of interaction, which is related to the tasks to support in order to reach the users' goals. Thus, we need to link these two types of information.

In this paper we discuss how to address the issues introduced by proposing a specific approach and a language and the associated environment, which builds on our previous experiences but aims to provide better support when designing interactive applications based on Web-services, we also report on how the approach can be applied in an application in the home domain and how it supports the interplay between software development and usability evaluation.

2. METHODOLOGICAL APPROACH

A traditional top-down approach going through the various abstraction layers that have been considered useful in HCI (task,

For example, a Boolean can be represented by a button, an enumeration type by a list or a radio button depending on its cardinality. Thus, for each Web services we can have a corresponding abstract description of the user interface.

Then, we can use the task model expressed in ConcurTaskTrees (CTT) [3] for describing the interactive application and how it assumes that tasks are performed. This notation is a standard de facto in the area of task model representations, and it also under consideration for standardization in the W3C consortium. CTT provides a first classification of tasks depending on the agent performing them: the user (in case of only internal cognitive activity, such as making a decision about how to carry on a session), the system (completely automatic task) or interaction (involving both the user and the system). Web services are application functionalities, thus they are associated with system tasks. Another issue is what level of granularity to reach in the task decomposition. There are mainly two possibilities: associating the system basic tasks to the web services or reach a further detail in order to associate each system basic task with the operations of the web services. Thus, if a Web Service supports three operations, then there would be three basic system tasks. The latter solution allows for a more detailed and flexible specification.

The next step is to obtain first an abstract, and then a concrete, platform-dependent, user interface description of the user

interface. To this end we have to consider information derived from the task models and the various pieces of abstract interface associated with the various Web Services. The information coming from the task model is useful in order to identify how to structure the presentations of the interactive applications and define the navigation model through them. The information coming from the abstract interface excerpts are mainly useful to identify the interface elements to include in each presentation and their type. Defining the structure of a presentation mainly means to identify the logical groups of elements inside it, and whether there are particular relations among some of such groups The structure of the Web services can be useful for this purpose because we can think of 'groupings' associated with each operation (indicating how to represent their input and output parameters), and higher level groupings associated with the Web services.

The use of an automatic tool and, consequently, automatic transformations, has several advantages: it allows for generating usable and consistent user interfaces by incorporating already in the transformation rules some design guidelines/rules for obtaining usable interfaces. In addition, it also allows for ensuring that some minimal consistency overall the pages is automatically kept (eg: ensuring the consistency of the title label or the style of the presentations overall an entire user interface application). However, very often the results of fully automatic authoring tools for user interfaces are not very satisfactory from a usability point of view, even when some good design rule have been incorporated in the transformations.

To this aim in our approach a semi-automatic process has been proposed. Such a process provides also space for (a manual) intervention, an evaluation phase carried out on an initial version of the user interface that has been automatically generated. Therefore, in order to improve the usability of the final results, an evaluation feedback from a HCI expert can be envisaged so that a consequent manual refinement and modification of the user interface which has been automatically obtained with the authoring tool can be carried out accordingly.

3. MARIA

In order to obtain a more powerful description language able also to satisfy the new requirements posed by service/oriented architectures, and modelling the new forms of human-computer interaction, we are developing a new UI specification language, which will take also into account the new technical requirements raised by the issue of generating usable interfaces for Web Services. The new language name is MARIA (Model-bAsed descRiption of Interactive Applications) XML and it can be used for the abstract and concrete user interface definition. Its development takes into account our previous experiences with a previous language (and the associated tool) for designing multi-device user interfaces, TERESA XML [4].

There are many differences between TERESA XML and MARIA XML. For example, MARIA supports also an abstract description of the underlying data model of the application. The interactors (namely, the elements of the abstract or concrete user interface) which compose an abstract (resp.:concrete) user interface, can be bound to a type or an element of a type defined in the abstract model. In this way, a change of application status is modelled as a change of one or more values in the abstract data, which will be reflected on the interface (abstract or concrete) status. This is a powerful feature that can be used to express in a natural manner aspects such as correlation between the value of interface elements, conditional presentation connections, conditional layout of interface parts, etc.

The data model is described using the XSD type definition language. In MARIA there is the possibility to define the data manipulated by the user interface both at the abstract level (through an abstract data model) and at the concrete level (a concrete data model, which is a refinement of the abstract one). The introduction of a data model at the abstract level also allows for having more control on the operations that will be done on the different data types. In addition, the data model is also useful for specifying the format for values: the format specification for a value can be expressed in MARIA by bounding the concrete data model with the editing interactor used for getting the input value from the user: if the editing object is bound with a date, the underlying implementation will have the needed information for validating the value that will be provided by the user. The MARIA data model can be the same as the types part of the WSDL description of the service, or it can be mapped on a more UI oriented description using an XSLT style sheet. The new authoring environment for MARIA XML is currently being developed to support this operation. The problem of mapping fields to services parameters is also supported by the MARIA environment: the mapping is obtained by performing the inverse operation of the process described before: another XSLT style sheet performs the mapping from the AUI data model to the WSDL one.

Figure 2 shows a graphical representation of an abstract interactor of type *only_output* in MARIA XML abstract user interface description (the description has been unfolded only for the higher levels).

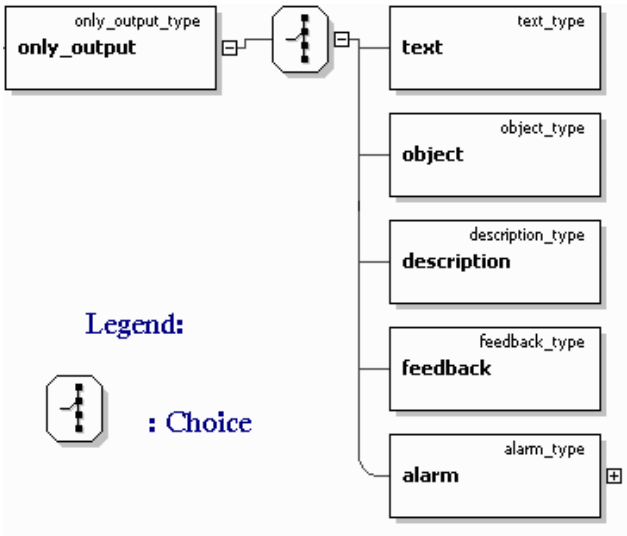


Figure 2. The specification of the *only_output* element

The *only_output* interactor models the possibility for the user to receive information from the application, and, depending on the

type of information received (text, object, description, feedback, alarm), suitable interactors should be used.

In the same way, within MARIA XML an abstract interactor allowing the user to interact with the underlying application is refined into different objects depending on the type of activity which is supported: selection (select an object within a set of objects), edit (modifying an object), control (activating a functionality), and interactive description (a combination of both *only_output* and interaction objects).

Figure 3 presents a graphical representation of an abstract interactor of type *interaction* in MARIA XML abstract user interface description (the description has been unfolded only for the higher levels).

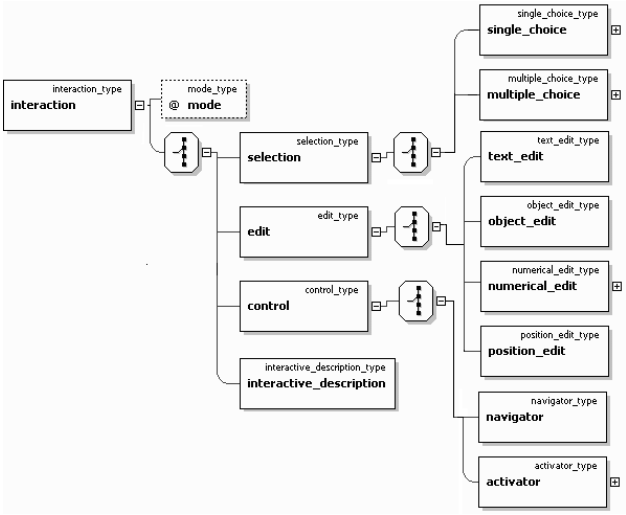


Figure 3. The specification of the *interaction* element

The corresponding excerpt of MARIA XML Schema for the abstract user interface description of the abovementioned interaction object (only for the first level) is visualised, together with the specification of the two possible types an interactor can assume (interaction and only_output):

```
<xs:complexType name="interaction_type">
  <xs:choice>
    <xs:element name="selection" type="selection_type"/>
    <xs:element name="edit" type="edit_type"/>
    <xs:element name="control" type="control_type"/>
    <xs:element name="interactive_description"
type="interactive_description_type"/>
  </xs:choice>
  <xs:attribute name="mode" type="mode_type" fixed="input"/>
</xs:complexType>

<xs:complexType name="interactor_type">
  <xs:choice>
    <xs:element name="interaction" type="interaction_type"/>
    <xs:element name="only_output" type="only_output_type"/>
  </xs:choice>
        …
</xs:complexType>
```

Figure 4 shows how it is possible, with MARIA XML, modelling a concrete user interface object (for the desktop platform) allowing for editing a textual value. More in detail, in the figure it is visualised the hierarchy of concrete interactors unfolded only for the branch of textfield objects, which allow editing text-based values. Textfields have a number of attributes, label (the label of the interactor), length (the length of the field), and the information about whether the field is aimed at accepting passwords (therefore the object should have a special behaviour in the feedback -eg: in a graphical platform it will not visualise the inserted value).

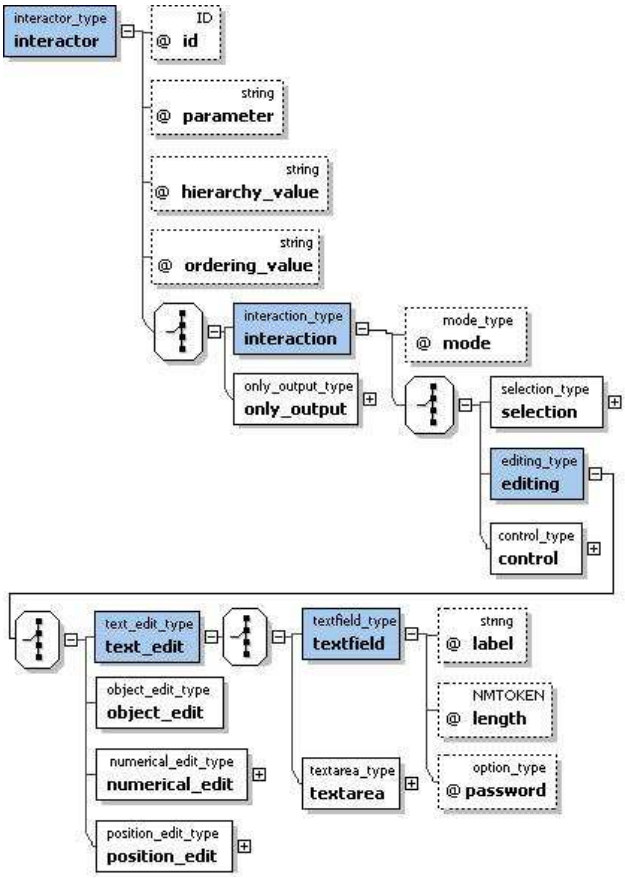


Figure 4. The specification of the *textfield* element

In Figure 4 below the objects derived from refining the *interactor* object down to the *textfield* object have been highlighted with a different colour, in order to make clearer to the reader such decomposition.

Below there is the corresponding MARIA XML specification excerpt for the textfield object

```
<xs:complexType name="textfield_type">
<xs:simpleContent>
<xs:extension base="xs:string">
<xs:attribute name="label" type="xs:string" use="required" />
<xs:attribute name="length" type="xs:NMTOKEN"
use="required" />
<xs:attribute name="password" type="option_type"
use="required" />
</xs:extension>
 </xs:simpleContent>
</xs:complexType>
```

4. AN EXAMPLE APPLICATION

We have applied our approach to the design of a home application. This is an application domain that is raising an increasing interest because our houses are becoming more and more populated with interactive, intelligent devices. In this case we have used a home server able to support interoperability among home devices supporting various communication protocol (X10, UpnP, Konnex, …). The functionalities of the domestic appliances are made available through a standardised set of Web services exposed by a home server [2]

This case study has also provided us with the possibility to define an algorithm for generating a default user interface for accessing a Web service operation. The idea is that the generated UI can then be refined by a designer with an authoring environment, but we want to create heuristics to minimize the need of human intervention.

For the sake of simplicity, in order to illustrate the algorithm we take into account a single service with a finite set of operations and data types.

The algorithm has a first step that aims to extract an object oriented model of the operations: each operation is associated to a data type (the type "owner" of the operation) defined in the types part of the WSDL file, checking the operation parameters.

After that the constructed model is reviewed for identifying the input and output operations that read or write the same properties. For instance we can take into account a Sensor data type that contains an element *status*. The WSDL has two operations :

- SetSensorStatus(Sensor s, boolean status)
- Boolean GetSensorStatus(Sensor s)

These two operations are bound to the Sensor data type, the first one is an input operation that writes a value in the *status* field and it is marked as input, while the second is a read operation and it is marked as output (it delivers as a result a Boolean data, as you can see from its specification). When the parameter that represents the value of the input operation matches with the read value of an output operation, their names are checked using the following heuristic: if the names are similar enough, they are merged into a single input/output operation: as a consequence, the same interactor will be used for supporting the input and output operations. Otherwise the two operations will remain distinct and different interactors will be used for accessing the two operations.

As an explanatory example of the above concept, we could consider a mobile user interface in which screen space is limited, and therefore it may be useful to have a single interactive element able to cover both aspects (possibility of changing the state and show actual state). In order to identify such cases, we have developed a heuristic indicating that when in the WSDL we find two methods having complementary structures (such as *set*<value> and *get*<value>, like e.g. *setSensorStatus* and *getSensorStatus* before) associated to one device, then they are mapped onto one element able to support both methods instead of two separate interface elements.

Enumeration data type, with high cardinality	

Abstract User Interface	`<selection>`
	`  <single_choice`
	`cardinality="high"/>`
	`</selection>`

Concrete Desktop	`<selection>`
	`  <single`
	`  cardinality="high">`
	`  <list_box`
	`  alignment="…">`
	`  <choice_element`
	`  label="[elementName]">`
	`  elementName`
	`  </choice_element>`
	`  [Other elements]`
	`  </ list_box >`
	`  </single>`
	`</selection>`

Concrete Mobile	`<selection>`
	`  <single`
	`  cardinality="high">`
	`  <drop_down_list`
	`  alignment="…">`
	`  <choice_element`
	`  label="[elementName]">`
	`  elementName`
	`  </choice_element>`
	`  [Other elements]`
	`  </drop_down_list>`
	`  </single>`
	`</drop_down_list>`

Concrete Vocal	`<selection>`
	`  <single`
	`  cardinality="high">`
	`  <message_menu`
	`  message="…"`
	`  nomatch_event="[nomatchmsg]"`
	`  noinput_event="[noinputmsg]"`
	`  help_event="[helpmsg]" >`
	`  <message>`
	`  [elementName]`
	`  </message>`
	`  [Other elements]`
	`  </message_menu>`
	`  </single>`
	`</selection>`

Figure 5. Examples of mappings

The next step is the creation of the abstract user interface using the collected operation information. The table shown in Figure 5 describes an example of the main mapping rules in the case of an enumeration data type with high cardinality, by showing an abstract *single_choice* element and the corresponding concrete elements for the desktop, mobile, and vocal platforms.

In this way it is possible to obtain an application able to support access through multiple types of devices. For example, it is possible to generate versions for a PDA and a desktop system, but other device types can be considered as well. In the case of the PDA access, we consider the possibility of generating an application in C#, even able to support libraries for vocal and multimodal access. This application has to be downloaded and installed in the mobile device. In the case of a desktop access, we consider the generation of a Web application able to support access, through some servlets, to the web services associated with the domestic appliances. Whatever interaction device is actually used, then the user can freely choose one domestic device and perform the desired information, usually check the state of some parameters (such as temperatures or alarms) or change some of their values.

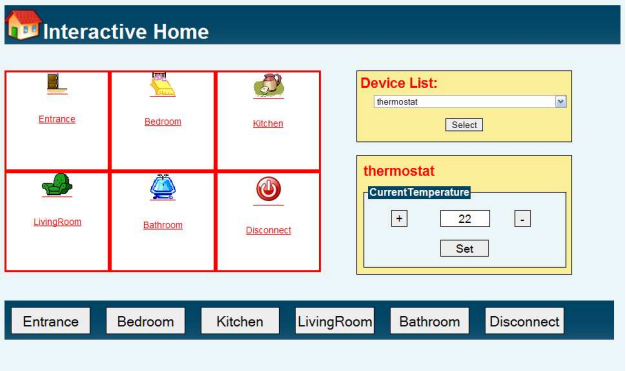


Figure 6. The Desktop User Interface.

The different screen space implies substantial differences in the generated user interfaces. In the desktop interface (see Figure 6) it is possible to show the various rooms, select a device and access the associated controls in *one single* presentation.

Figure 7. The PDA User Interface.

All these possibilities are still available in the PDA interface (see Figure 7) but they require multiple presentations and the addition of navigation capabilities among them.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have discussed a method for the model-based design of interactive applications based on the use of Web services. We have also briefly reported on the development of a new XML specification language, and the associated authoring environment, to better support the method, and, more generally, to provide more flexible support to UI designers. We have also illustrated the approach with a specific example in the home domain.

In addition, we pointed out how our approach leverages an easy coupling between on the one hand software design and development and, on the other hand, usability evaluation. Indeed, the approach is supported by a semi-automatic process in which an initial version of the user interface is expected to be obtained through the use of automatic tools in which some guidelines for good UI design are already incorporated (eg within the transformation rules). Therefore, the initial results automatically obtained should already be compliant with principles of good design if they have been incorporated in suitable transformations (which, if not hard coded in the automatic tool can be even subject of an usability evaluation as well). Afterwards, the preliminary versions of the user interfaces so obtained are supposed to be analysed and evaluated by HCI experts: the feedback of such an evaluation can be included through a manual refinement which can affect (and, hopefully improve) the result not only at the final UI level but also at more abstract UI levels, depending on their skills.

Future work has been planned for applying the presented approach to more complex case studies, in order to test the generality and the flexibility of the method.

6. ACKNOWLEDGMENTS

This work is partly supported by the ServFace EU ICT project.

7. REFERENCES

[1] Abrams, M., Phanouriou, C., Batongbacal, A., Williams, S., Shuster, J. UIML: An Appliance-Independent XML User Interface Language, Proceedings of the 8th WWW conference, 1999.

[2] Miori V., Tarrini L., Manca M., Tolomei G. - An open standard solution for domotic interoperability. In: IEEE Transactions on Consumer Electronics, vol. 52 (1) pp. 97-103. IEEE, 2006.

[3] Paternò F., Model-based Design and Evaluation of Interactive Applications, Springer Verlag, November 1999, ISBN 1-85233-155-0.

[4] Paternò F., Santoro C., Mantyjarvi J., Mori G., Sansone S., Authoring Pervasive MultiModal User Interfaces, International Journal of Web Engineering and Technology, N.2, 2008

[5] Song, K., Lee, K.-H., 2007. An automated generation of xforms interfaces for web services, Proceedings of the International Conference on Web Services, 856-863.

[6] Spillner, J., Braun, I., Schill, A., 2007. Flexible Human Service Interfaces, Proceedings of the 9th International Conference on Enterprise Information Systems, 79-85.

[7] Vermeulen J., Vandriessche Y., Clerckx T., Luyten K. and Coninx K., Service-interaction Descriptions: Augmenting Services with User Interface Models, Proceedings Engineering Interactive Systems 2007, Salamanca, Springer Verlag.

Direct Integration: Training Software Developers and Designers to Conduct Usability Evaluations

Mikael B. Skov and Jan Stage
Aalborg University
Department of Computer Science
DK-9220 Aalborg East
Denmark
{dubois,jans}@cs.aau.dk

ABSTRACT
Many improvements of the interplay between usability evaluation and software development rely either on better methods for conducting usability evaluations or on better formats for presenting evaluation results in ways that are useful for software designers and developers. Both approaches involve a complete division of work between developers and evaluators, which is an undesirable complexity for many software development projects. This paper takes a different approach by exploring to what extent software developers and designers can be trained to carry out their own usability evaluations. The paper is based on an empirical study where 36 teams with a total of 234 first-year university students on software development and design educations were trained in a simple approach for user-based website usability testing that was taught in a 40 hour course. This approach supported them in planning, conducting, and interpreting the results of a usability evaluation of an interactive website. They gained good competence in conducting the evaluation, defining task assignments and producing a usability report, while they were less successful in acquiring skills for identifying and describing usability problems.

Categories and Subject Descriptors
H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *Graphical user interfaces (GUI), Theory and methods.*

General Terms
Measurement, Experimentation, Human Factors.

Keywords
Usability evaluation, user-based evaluation, training of software developers, dissemination of usability skills, empirical study

1. INTRODUCTION
Usability evaluation and user interaction design are two key activities in the development of an interactive system. The two activities are mutually dependent, but in practice there is often too little or no fruitful interplay between them [6].

Considerable efforts have been devoted to improve the interplay between usability evaluation and software development. A substantial part of these efforts reflect two typical approaches.

The first approach focuses on better methods. The aim is to improve the products of usability evaluations through use of methods that provide better support to evaluators that carry out usability evaluations. During the last 20 years, a whole range of methods have been developed within this approach. A prominent and influential example is Rubin [15] that covers all activities in a usability evaluation. There are many others that cover all or some selected evaluation activities.

The second approach focuses on better feedback. The aim is to improve the impact of usability evaluations on user interaction design. This is achieved in a variety of ways, typically by improving the format that is used to feed the results of usability evaluations back into user interaction design. The classical format for feedback is an extensive written report, but there have been numerous experiments with alternatives to the report; see [7] for an overview.

Compared to both of these approaches, website development is, however, particularly challenging. Websites exhibit a huge and unprecedented amount of information, services and purchasing possibilities, and the users of websites are a tremendously heterogeneous group that use websites for a multitude of purposes any time, any place. Due to this, website developers must accommodate a massive variety of user preferences and capabilities.

Many contemporary websites suffer from problems with low usability, e.g. an early investigation of content accessibility found that 29 of 50 popular websites were either inaccessible or only partly accessible [17]. This is in line with the suggestions that usability evaluations of websites should focus on the extent to which users can navigate the website and exploit the information and possibilities for interaction that are available [16].

A conventional usability evaluation that involves the prospective users of an interactive system facilitates a rich understanding of the actual problems that real users will experience [15]. The main drawback of user-based usability evaluations is that they are exceedingly demanding in terms of time and other resources; some researchers have reported that duration of one month and efforts amounting to around 150 person-hours are not unusual [11][12][13]. These figures are simply not feasible for many website projects. The projects do not have this amount of

I-USED'08, September 24, 2008, Pisa, Italy

resources, and they cannot wait for the usability evaluators to conduct the evaluation and provide relevant feedback.

The two approaches that were emphasized above share a key characteristic, as they involve a complete division of work between developers and evaluators. The software is produced by the developers, and its usability is assessed by the evaluators. This division of work is undesirable or impossible in many fast-paced projects. The division of work necessitates handovers between the two groups, and this will increase project complexity and tend to lengthen development time. Thus the division between developers and evaluators is a main obstacle for integrating usability evaluation into most website development projects.

This paper presents results from an empirical study of a course where first-year students in software development and design educations were trained to conduct their own user-based usability evaluations. The aim of the approach behind this course is to facilitate direct integration of usability evaluation into software development by removing the division between evaluators and developers. In the study, we explored whether designers and software developers who had received a 40 hour training course could conduct a usability evaluation of a reasonable quality. In the following section 2, we present previous work related to our study. In section 3, we describe the study in detail. The results of the study are presented in section 4, and section 5 discusses additional aspects of the results. Finally, section 6 provides the conclusion.

2. RELATED WORK

The idea of reducing the gap between software development and usability evaluation by broadening the skills of software developers is not new. It has been suggested that education of software developers in usability engineering could contribute to reduce the problems with the usability that characterize many software products. This suggestion focused on a general awareness of usability issues and on the early activities in a development project [9].

It has also been discussed on a more general level how development teams can best be trained to use fundamental techniques from the usability engineering discipline. This requires systematic empirical studies of the true costs of learning and applying usability engineering techniques [8].

We conducted a search on the web on training of software developers in usability engineering. We found a group of companies that offer training courses for software developers in various methods from the usability engineering discipline. The two most common methods were the so-called discount usability evaluation techniques (expert inspection and walkthrough) and user-based empirical testing based on a think-aloud protocol. There were much fewer and mostly shorter courses on general usability topics.

Such courses for practitioners respond to the request for training of practitioners in usability topics [9]. Unfortunately, they are not complemented by the research studies of cost and effects that were also requested [8]. In fact, we have only been able to find very few systematic studies of efforts to train software developers in key topics from usability engineering.

A notable exception to this limited amount of research is an empirical study of training of software engineering students in a language for describing and analysing user interface designs [3]. This study measured the effect of a training course and also provided improved insight to the way experts work in this area.

3. METHOD

We have conducted an empirical study of a training course that is intended to teach software developers and designers to conduct usability evaluations. The aim of the study was to provide the participants with skills in formative usability evaluation.

Table 1. The 10 class meetings of the training course

#	Lecture	Exercises
1	Introduction to the course and basic website technology	Pilot test: Each team conducts simple pilot usability tests of websites to train their practical skills in usability evaluation. The teams choose the website themselves. Experience with conducting tests and the results achieved are discussed afterwards.
2	Basic introduction to usability issues and guidelines for interaction design	
3	The think-aloud protocol and how to set up a test scenario. User groups and their different needs	
4	Application of questionnaires for collecting data and how to use different kinds of questions	
5	Computer architecture and website technology	Usability evaluation: The teams conduct a usability evaluation of the Hotmail website according to a specification provided by the course instructors. The usability evaluations are conducted at the university in assigned rooms for each team. After the usability test sessions, the teams analyze the empirical data and make a usability report that describes the identified usability problems.
6	Describing the usability testing method and how to collect and analyze empirical data	
7	Other usability evaluation methods and how to conduct a full-scale usability test session	
8	Website structures, information search and web surfing	
9	Guidelines for website design and principles for orientation and navigation	
10	Principles for visual design and different interaction styles	

3.1 Training Course

We studied the training course in a first year university curriculum. The course included ten class meetings, cf. Table 1, each lasting four hours that was divided evenly between two hours of lecture, and two hours of exercises in smaller teams. The course required no specific skills in information technology which is the reason why class meeting number one and five included introductions to technological issues. The purpose of the exercises was to practice selected techniques from the lectures. In the first four class meetings, the exercises made the students conduct small usability pilot tests in order to train and practice their practical skills with selected methods. The exercises in the last six class meetings were devoted to conducting a realistic usability evaluation of a specified website.

The course introduced a number of methods for usability testing. The first was the conventional method for user-based testing with the think-aloud protocol [14][15]. The second method was based on questionnaires that test subjects fill in after completing each task and after completion of the entire test [16]. The students were

also introduced to additional methods such as interviewing, heuristic inspection, cognitive walkthroughs, etc.

The students were required to document their work by handing in a usability report. The instructors suggested to the students that the usability report should consist of 1) executive summary (1 page), 2) description of the usability evaluation method applied (2 pages), 3) results of the evaluation, primarily a list and detailed description of the identified usability problems for the website that was evaluated (5-6 pages), and 4) discussion of the method that was applied (1 page). The report would typically amount to around 10 pages of text. It was further emphasized that the problems identified should be categorized, at least in terms of major and minor usability problems. In addition, the report should include appendices with all data material produced such as log-files, tasks assignments for test subjects, questionnaires etc. A prototypical example of a usability report was given to the students.

3.2 Website

We chose www.hotmail.com as the website for our study. This website provides advanced interactive features and functionalities appropriate for an extensive usability test. Furthermore, it facilitates evaluations with both novice and expert test subjects due to its vast popularity. Finally, it has been used in other usability evaluations that have been published, which enabled us to compare the results of the student teams in our study with other result (this is further explained below under Data Analysis).

Table 2. Team and test subject data

Total number of students	Total number of teams	Team size *Average*	Team size *Min / Max*
234	36	6.5	4 / 8
Number of test subjects *Average*	Number of test subjects *Min / Max*	Age of test subjects *Average*	Age of test subjects *Min / Max*
3.6	2 / 5	21,2	19 / 30

3.3 Participants

The participants were first-year university students enrolled in four different studies at a faculty for natural sciences and engineering. The first of the four studies was informatics, which is a user-oriented IT education with focus on software development but also with elements of design in general. The other three studies were architecture and design, planning and environment, and chartered surveyor, which all shared a focus on design in general but also had elements of software development. All four groups of students participated together in the course described in this paper. None of the participants had any experience with usability evaluation prior to the study.

36 teams involving a total of 234 students (87 females, 37%) participated in the course and our study. Each team was required to distribute the roles of test subjects, loggers, and test monitor among themselves. This was done before the second class meeting, well before they started the evaluation of the Hotmail website. 129 (55%) of the students acted as test subjects, 69 (30%) as loggers, and 36 (15%) as test monitors, cf. [15]. The average team size was 6.5 students (SD=0.91). The average number of test subject in the teams was 3.6 (SD=0.65), and their average age was 21.2 years old (SD=1.58). 42 (33%) of the 129 test subjects had never used www.hotmail.com before the evaluation, whereas the remaining 86 subjects had varied experience with the website. These data are summarized in Table 2.

3.4 Setting

Due to the pedagogical approach of the university, each team had their own office equipped with a personal computer and Internet access. Most teams conducted the tests in their office, while the rest did it in one of their homes. After the tests, the entire team worked together on the analysis and identification of usability problems and produced the usability report.

3.5 Procedure

The student teams were required to apply the techniques presented in the course. After the second class meeting, the test monitor and loggers of each team received a two-page scenario specifying the web-based mail service www.hotmail.com that they should focus on in the usability evaluation. The scenario also specified a comprehensive list of features that emphasized the specific parts of www.hotmail.com they were supposed to evaluate. The test monitor and the loggers examined the system, designed tasks, and prepared the evaluation, cf. [15]. The use of www.hotmail.com as the website to be evaluated in the study was kept secret to the test subjects until the actual test was conducted.

3.6 Data Collection

The main data collected in the study was the usability reports that were handed in by the teams. The 36 reports had an average length of 11.4 pages (SD=2.76) excluding the appendices, which had an average length of 9.14 pages (SD=5.02). 30 (83%) of the 36 teams provided information on task completion times for 107 (83%) of the 129 subjects, and they had an average session time (with one user) of 38.10 minutes (SD=15.32 minutes).

We did not collect any data on the way the students performed during the evaluation, and we did not monitor or record how they carried out the evaluations.

3.7 Data Analysis

All reports were analyzed, evaluated, and marked by the two authors of this paper according to the following three steps.

Step 1. We designed a scheme for the evaluation of the 36 reports by analyzing, evaluating and marking five randomly selected reports out of the total of 36 reports. Through discussions and negotiations we came up with an evaluation scheme with 17 variables as illustrated in Table 3. The 17 variables were divided into the following three overall categories: evaluation (the way the evaluation was conducted), report (the presentation of the evaluation and the results), and results (the outcome of the usability evaluation). Finally, we described, defined, and illustrated all 17 variables in a two-page marking guide.

Step 2. We worked individually and marked each of the 36 reports in terms of the 17 variables by using the marking guide. The markings were made on the following scale of 1 to 5: 1= wrong

answer or no answer at all, 2=poor or imprecise answer, 3=average answer, 4=good answer, and 5=outstanding answer.

Table 3. The 17 experimentally identified variables used in the assessment of the 36 usability reports.

Category	Variable
Evaluation	1) Conducting the evaluation 2) Task quality and relevance 3) Questionnaires/interviews quality and relevance
Report	4) Test procedure description 5) Data quality 6) Clarity of usability problem list 7) Executive summary 8) Clarity of report 9) Report layout
Results	10) Number of identified usability problems 11) Usability problem categorization 12) Practical relevance of usability problems 13) Qualitative results overview 14) Quantitative results overview 15) Use of literature 16) Conclusion 17) Test procedure evaluation

We also counted the number of identified usability problems in each of the 36 usability reports. We defined a usability problem as something in the user interaction that prevents or delays users in realizing their objectives. Each time a report would described such an obstacle or delay, we would count that as a usability problem. Finally, we specified intervals for grading of the identification of usability problems based on their distribution on the following scale: 1=0-3 problems, 2=4-7 problems, 3=8-12 problems, 4=12-17 problems, and 5>17 problems.

Step 3. All reports and grades were compared and a final assessment on each variable was negotiated. In case of disagreements on a grade, we employed the following procedure: 1) if the difference was one grade, we would renegotiate the grade based upon our separate notes; 2) if the difference was two grades, we would reread and reassess the report together focusing only on the variable in question. For our study, no disagreement exceeded two grades. For each report, we also went through the set of usability problems that each of us thought they had identified. We negotiated each team's list of usability problems until we had consensus on that as well.

To examine the overall performance of the students, we included two additional sets of data in the study. Firstly, we compared the student reports to usability reports produced by teams from professional laboratories. These reports were selected from a pool of usability reports produced in another research study where nine usability laboratories received the same scenario as we used and conducted similar usability tests of www.hotmail.com, cf. [10][11]. Of the nine usability reports, we discarded one because it was only based on heuristic inspection, which was different from our focus on user-based evaluation. The remaining eight usability reports were analyzed, assessed, and marked through the same procedure as the student reports. Secondly, we calculated a combined score for each team based on the grades that the individual team members had obtained in other courses they attended in the same semester. This was done to explore the correlation between the overall skills of the students and their ability to conduct a usability evaluation.

4. RESULTS

The overall results show that the student teams did quite well. It is not surprising that the professionals did better on most variables. It was, however, surprising to us that on some variables, the students had a comparable performance and on a few variables they even performed better than the professional teams.

Table 4. Results for conducting the evaluations. Boldface numbers indicate significant differences between the student and professional teams.

	Evaluation		
Teams	Conducting the evaluation	Task quality and relevance	Questionnaire/ Interviews
Student (N=36)	3.42 (0.73)	3.22 (1.05)	2.72 (1.00)
Professional (N=8)	**4.38 (0.74)**	3.13 (1.64)	3.50 (1.69)

4.1 Evaluation

These three variables relate to the way the usability evaluation was conducted, see Table 4. On variable 1, conducting the evaluation, the professional teams have an average of 4.38 (SD=0.74). This is almost one grade higher than the student teams and a Mann-Whitney U Test shows strong significant difference between the student teams and the professional teams ($z=-2.68$, $p=0.0074$). On variable 2, task quality and relevance, the students performed slightly better than the professionals, but this difference is not significant ($z=0.02$, $p=.984$). No significant difference was found on variable 3, questionnaire/interviews quality and relevance ($z=-1.63$, $p=0.1031$).

4.2 Report

These six variables relate to the quality of the usability report that was the tangible result of the usability evaluations, see Table 5.

Table 5. Results for the usability reports. Boldface numbers indicate significant differences between the student and professional teams.

	Report					
Teams	Test description	Data quality	Clarity of problem list	Executive summary	Clarity of report	Layout of report
Student (N=36)	3.03 (0.94)	**3.19 (1.33)**	2.53 (1.00)	2.39 (0.80)	2.97 (0.84)	2.94 (0.89)
Professional (N=8)	**4.00 (1.31)**	2.13 (0.83)	**3.50 (0.93)**	**3.38 (1.06)**	**4.25 (0.71)**	3.25 (0.71)

Table 6. Results for the outcome of the usability evaluations. Boldface numbers indicate significant differences between the student and professional teams.

	Results							
Team	Number of problems	Problem categorization	Practical relevance	Qualitative results overview	Quantitative results overview	Use of literature	Conclusion	Evaluation of test
Student (N=36)	2.56 (0.84)	2.06 (1.22)	3.03 (1.00)	3.03 (1.00)	2.28 (1.14)	3.08 (0.81)	2.64 (0.90)	2.44 (1.08)
Professional (N=8)	**4.13 (1.13)**	3.25 (1.75)	**4.25 (1.49)**	**3.75 (1.16)**	2.00 (1.51)	3.13 (0.35)	**3.88 (0.64)**	2.88 (1.13)

The student teams did not perform as well as the professionals on the description of the test, and this difference is significant (z=-2.15, p=0.0316). On the other hand, the student teams actually performed significantly better than the professional teams on the quality of the data material in the appendices (z=2.07, p=0.0385).

On the clarity of the usability problem list, we found a strong significant difference in favour of the professional teams (z=-2.98, p=0.0029). There is also a significant difference on the teams' executive summary, where the professionals are better (z=-2.27, p=0.0232), and a strong significant difference on the clarity of the entire report (z=-3.15, p=0.0016). Finally, no significant difference was found for the layout of the report (z=-1.02, p=0.3077) although the number for the professional teams is slightly higher.

4.3 Results

The pivotal result of the usability reports was the usability problems that were identified and the descriptions of them. There are eight variables on this category, see Table 6.

On the number of problems identified, the student and professional teams performed rather differently. The student teams were on average able to identify 7.9 usability problems (in the marking scale: Mean 2.56, SD 0.84) whereas the professional teams on average identified 21.0 usability problems (in the marking scale: Mean 4.13, SD 1.13). A Mann-Whitney U Test confirms strong significant difference between the student and professional teams on this variable (z=-3.09, p=0.002). It is, however, interesting that the professional teams actually performed very dissimilar on this variable, as they identified from 7 to 44 usability problems. Thus the professional team that identified the lowest number of usability problems actually performed worse than the average student team.

The professional teams performed better than the student teams on categorization of the usability problems that were identified, but the difference is not significant (z=-1.84, p=0.0658). On the practical relevance of the identified usability problems, the professional teams performed better, and this difference is significant (z=-2.56, p=0.0105).

On the overview of the qualitative results, the professional teams did significantly better than the students (z=-1.99, p=0.0466). On the other hand, the student teams provided better overview of the quantitative results, but this difference is not significant (z=0.90, p=0.3681).

There is no significant difference on the use of literature (z=-0.05, p=0.9601). The conclusions are better in the usability reports from the professional teams, and this difference is strong significant (z=-3.13, p=0.0017). No significance was found for the teams' own evaluations of the test procedure they employed (z=-1.00, p=0.3173).

4.4 Usability Problem Correlations

The strong differences between the student teams and the professionals in the production of results, e.g. the usability problem identified, made us conduct a more detailed analysis of potential causes.

A Spearman Rank Correlation shows a weak positive correlation between the way the evaluation was conducted and the number of identified usability problems, but this correlation is not significant (marking (r^2=0.061, p>0.718), actual (r^2=0.089, p>0.599)). The same can be concluded for the correlation between the quality and relevance of the tasks and the number of identified usability problems (marking (r^2=0.239, p>0.157), actual (r^2=0.235, p>0.165)). Thus, our study indicates that the student's competence in planning and conducting a usability test does not necessarily influence the outcome of the evaluation in terms of the number of usability problems identified.

When looking at the corresponding variables for the professional teams, we find that there is a high correlation between the quality and relevance of the tasks and the number of identified usability problems for the professional teams and this correlation is significant (r^2=0.741, p<0.05). Furthermore, a weak correlation exists between the way the evaluation was conducted and the number of identified usability problems, but this correlation is not significant (r^2=0.336, p>0.374).

Introducing more test subjects in usability evaluations will usually (at least in theory) generate a higher number of identified usability problems. In our study, the average number of test subject was 3.6 (SD=0.65), ranging from one team using only two test subjects to one team using five test subjects. However, we found only a negligible positive correlation between the number of test subjects and the number of identified usability problems, as this correlation was not significant (marking (r^2=0.247, p>0.143), actual (r^2=0.238, p>0.159)). The test subjects had a rather varied experience with www.hotmail.com, but there is no significant correlation between the number of novice subjects and the number of identified problems (marking (r^2=0.119, p>0.482), actual (r^2=0.119, p>0.481)).

Correlations between the length of the tests and the number of identified usability problems for the 36 teams (grading and actual numbers) are illustrated in Figure 1. Considering the total time spent on all tests in each team, we identify a great variation ranging from 56 minutes to 225 minutes (mean=113.26 minutes,

SD=65.59 minutes). A minor correlation exists between the total time spent on the test and the number of identified problems, but the correlation is not significant (r^2=0.280, p>0.098). This is also the case when looking at the actual number of problems against time spent (r^2=0.329, p>0.051). This correlation is, however, close to being significant.

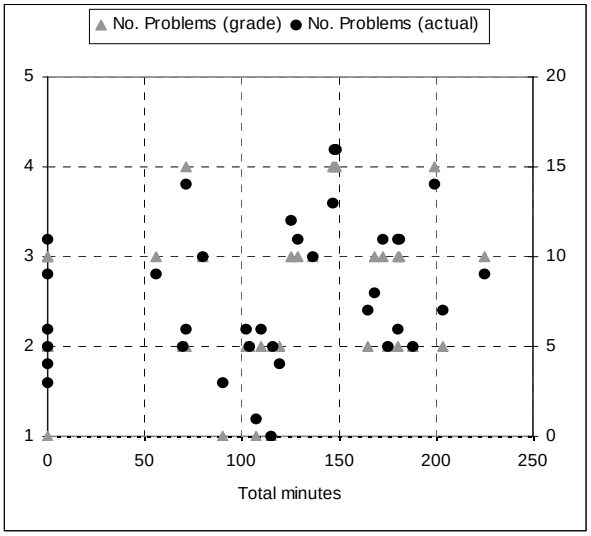


Figure 1. Correlation between the length of all tests in the 36 teams and the number of identified usability problems (reported as grading 1-5). Six teams did not report the time spent on the tests.

As a complementary perspective, we analyzed the basic skills of the students and their performances in other university activities in the same semester. We examined the correlation between the combined grade obtained by each of the 36 teams (based on the individual grades of team members) in other major coursework and the number of identified usability problems.

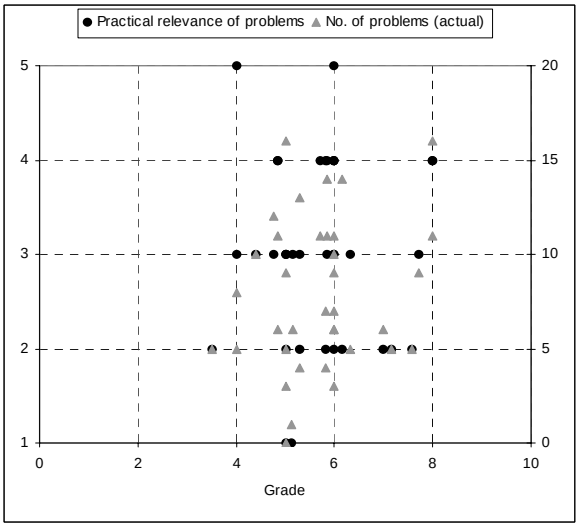


Figure 2. Correlation between the team grading (reported as zero to nine) and the number of identified usability problems (reported as grading 1-5 and the actual number identified).

The grade is reported on a scale from zero (not satisfactory) to nine (outstanding). A Spearman Rank Correlation Test shows only a slight positive correlation between the grade of the students and the number of identified usability problems (marking (r^2=0.103 p>0.542), actual (r^2=0.130, p>0.441)). This correlation between grades and identified number of usability problems is illustrated in Figure 2.

5. DISCUSSION

As emphasized in the introduction, several studies have found that many websites suffer from low usability [17]. The purpose of our study was to explore to what extent people working with software development and design but with no formal training in usability engineering could be trained to conduct website usability evaluations of a reasonable quality. If that was possible, such a training programme could help designers and developers face the challenges of and reduce the amount of usability problems on the websites they produce.

One of our key findings concerns identification and categorization of usability problems. The student teams identified significantly fewer problems than the professional teams. On average, the student teams found 7.9 usability problems, whereas the professional teams on average found 21 usability problems. This difference is important since uncovering of usability problems is a key purpose of a formative usability evaluation. The student teams did, however, perform rather differently on this variable. One student team identified no problems at all. This team might have misunderstood the assignment, but we cannot tell from their usability report, which was the basis for our analysis. The best performing students were two teams that identified 16 problems. Most of the student teams identified no more than 10 problems.

The professional teams also performed rather differently. It has been shown before that usability evaluators find different problems; this has been denoted as the evaluator effect [5]. Yet we also found a substantial difference in terms of the *number* of problems identified, and this is perhaps more surprising. One professional team identified 44 usability problems whereas another team identified only seven problems. The latter is actually rather disappointing for a professional team. We have analyzed the problems they found in more detail. The professional teams identified several critical problems on the website, but some of the critical problems were identified by relatively more student teams than professionals. For example, it was discovered by relatively more student teams that test subjects were unable to locate the functionality to change password. Thus, even though the student teams identified significantly fewer problems, they still identified some of the most severe problems on the website.

Another variable that exhibits a remarkable difference is the practical relevance of the problem list. This variable measures the extent to which the descriptions of the usability problems identified are useful for a software developer that will solve the problem. The student teams are almost evenly distributed on the five marks of the scale, and their average is 3.2. When we compare this to the professional teams, there is a clear difference. The professionals score an average of 4.6, and 6 out of 8 teams score the top mark. This difference can, at least partly, be explained from the experience that the professionals have acquired in describing usability problems in a way that make them relevant to their customers.

Another reason for the differences between student teams and professionals in identifying and describing usability problems

may be the specific design of the training course. We might have focused too little on discussing the nature of a usability problem and provided too few examples. We could also have treated this in more detail by presenting specific examples of relevant and irrelevant problems. Our analysis of the reports from the student teams clearly suggests that this topic received too little attention.

6. CONCLUSION

This article has presented the results from a study of a course that was employed to train software developers and designers in conducting usability evaluations of a website. The idea behind this effort was that if developers can conduct their own usability evaluations, the gap between usability evaluation and software design will disappear.

The course was based on a simple approach to usability testing that quickly teaches fundamental usability skills. Whether this approach is effective has been explored through a large empirical study where 36 student teams from the first year of software development and design-oriented educations were trained in and applied the approach to evaluate the usability of the Hotmail website.

The overall conclusion is that the student teams were able to conduct usability evaluations and produce usability reports of a reasonable quality and with relevant results. However, when compared to professional evaluator teams, there were clear differences. The student teams performed well in defining good tasks for the test subjects, and the data material in their reports was significantly better than the professionals. They were less successful on several of the other variables, and they performed clearly worse when it came to the identification of problems, which is a main purpose of a usability test. It was also difficult for them to express the problems found in a manner that would be relevant to a software developer working in practice.

Time pressure is a key reason why established knowledge and methodologies are ignored in many website development projects [2]. Website developers experience a strong push for speed and users of websites rapidly change preferences and patterns of use, and new ideas for design and functionality emerge constantly. This makes customers and management demand development cycles that are considerably shorter than in traditional software development [1][4]. The aim of the training course we have presented in this paper is to enable software developers and designers to conduct their own website usability evaluations. The students who were trained in the approach gained a significant step towards the level of expert evaluators. However, they still lacked competence in some of the key areas. Thus we see the training course as a relevant complement to classical usability testing conducted in a formalized manner in advanced laboratories by highly specialized experts.

Our study is limited in a number of ways. First, the environment in which the evaluations were conducted was in many ways not optimal for the usability test sessions. In some cases, the students were faced with slow Internet access that might have influenced the results. Second, motivation and stress factors could prove important in this study. None of the teams volunteered for the course and the study, and none of them received any payment or other kind of compensation. All teams participated in the course because it was a mandatory part of their curriculum, but they did not have to pass an exam in the course itself. This implies that students did not have the same kinds of incentives for conducting the usability test sessions as the evaluators from the professional usability laboratories. Thirdly, the demographics of the test subjects are not varied with respect to age and education. Most test subjects were approximately 21 years of age with approximately the same school background and recently started on an IT or design-oriented education.

The use of university students as a substitute for real software developers and designers working in practice has often, and rightly, been criticized. Yet in this case, it is less questionable. With a group of software developers from practice, it would be difficult to distinguish between their experience and the effect of the training course. With students who have basic knowledge about software development but no practical experience, that empirical problem vanishes.

Having said that, it could still be very interesting to conduct a similar study with real website developers and designers. This might be combined with a longitudinal study of the long-term effect on the quality of the websites developed. The main shortcoming that came up in our analysis was the students' lack of skill in identifying and describing usability problems. A different study could be based on a training course that was changed to focus directly on identification of usability problems.

7. ACKNOWLEDGMENTS

We would like to thank the students for their participation in the study.

8. REFERENCES

[1] Anderson, R. I. (2000) Making an E-Business Conceptualization and Design Process More "User"-Centered. *interactions*, 7(4) (July-August):27-30.

[2] Baskerville, R., and Pries-Heje, J. (2001) Racing the E-Bomb: How the Internet is Redefining Information Systems Development Methodology. In N. Russo et al. (eds.), *Realigning Research and Practice in Information Systems Development*, Kluwer, 49-68.

[3] Blandford, A., Buckingham Shum, S. J., and Young, R. M. (1998) Training software engineers in a novel usability evaluation technique. *International Journal of Human-Computer Studies*, 49(3):245-279.

[4] Broadbent, S., and Cara, F. (2000) A Narrative Approach to User Requirements for Web Design. *interactions*, 7(6):31-35 (November-December).

[5] Hertzum, M. and Jacobsen, N. E. (2003) The Evaluator Effect: A Chilling Fact About Usability Evaluation Methods. *International Journal of Human Computer Interaction*, 15(1):183-204.

[6] Hornbæk, K. and Stage, J. (2006) The Interplay Between Usability Evaluation and User Interaction Design. *International Journal of Human-Computer Interaction*, 21(2):117-124.

[7] Høegh, R. T., Nielsen, C. M., Overgaard, M., Pedersen, M. B. and Stage, J. (2006) The Impact of Usability Reports and User Test Observations on Developers' Understanding of Usability Data: An Exploratory Study. *International Journal of Human-Computer Interaction*, 21(2):173-196.

[8] John, B. E. (1996) Evaluating usability evaluation techniques. *ACM Computing Surveys*, 28(4es):139 (December).

[9] Karat, J. and Dayton, T. (1995) Practical education for improving software usability. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'95)*, 162-169. ACM Press.

[10] Molich, R. (undated) Comparative Usability Evaluation Reports. Available at http://www.dialogdesign.dk/cue.html.

[11] Molich, R., Ede, M. R., Kaasgaard, K. and Karyukin, B. (2004) Comparative Usability Evaluation. *Behaviour & Information Technology*, 23(1):65-74.

[12] Molich, R., and Nielsen, J. (1990) Improving a Human-Computer Dialogue. *Communications of the ACM*, 33(3): 338-348.

[13] Nielsen, J. (1992) Finding Usability Problems Through Heuristic Evaluation. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'92)*, 373-380. ACM Press.

[14] Nielsen, J. (1993) *Usability Engineering*. Morgan Kaufmann Publishers.

[15] Rubin, J. (1994) *Handbook of Usability Testing. How to Plan, Design, and Conduct Effective Tests*. John Wiley & Sons.

[16] Spool, J. M., Scanlon, T., Schroeder, W., Snyder, C., and DeAngelo, T. (1999) *Website Usability. A Designer's Guide*. Morgan Kaufmann Publishers.

[17] Sullivan, T., and Matson, R. (2000). Barriers to Use: Usability and Content Accessibility on the Web's Most Popular Sites. *Proceedings of Conference on Universal Usability*, 139-144. ACM Press.

European Science Foundation provides and manages the scientific and technical secretariat for COST

COST is supported by the EU RTD Framework Programme