This is not the published version of the article / Þetta er ekki útgefna útgáfa greinarinnar

Author(s)/Höf.:	Vilbergsdóttir, Sigurbjörg Gróa; Hvannberg, Ebba Þóra; Law, Effie Lai-Chong
Title/Titill:	Assessing the Reliability, Validity and Acceptance of a Classification Scheme of Usability Problems (CUP)
Year/Útgáfuár:	2014
Version/Útgáfa:	Post-print (lokagerð höfunda)

Please cite the original version:

Vinsamlega vísið til útgefnu greinarinnar:

Vilbergsdottir, S. G., Hvannberg, E. T., & Law, E. L.-C. (2014). Assessing the reliability, validity and acceptance of a classification scheme of usability problems (CUP). Journal of Systems and Software, 87, 18-37. doi:https://doi.org/10.1016/j.jss.2013.08.014

Rights/Réttur: Copyright © 2013 Elsevier Inc. All rights reserved.

https://doi.org/10.1016/j.jss.2013.08.014

Assessing the Reliability, Validity and Acceptance of a Classification Scheme of Usability Problems (CUP)

The aim of this study was to evaluate the Classification of Usability Problems (CUP) scheme. The goal of CUP is to classify usability problems further to give user interface developers better feedback to improve their understanding of usability problems, help them manage usability maintenance, enable them to find effective fixes for UP, and prevent such problems from reoccurring in the future. First, reliability was evaluated with raters of different levels of expertise and experience in using CUP. Second, acceptability was assessed with a questionnaire. Third, validity was assessed by developers in two field studies. An analytical comparison was also made to three other classification schemes. CUP reliability results indicated that the expertise and experience of raters are critical factors for assessing reliability consistently, especially for the more complex attributes. Validity analysis results showed that tools used by developers must be tailored to their working framework, knowledge and maturity. The acceptability study showed that practitioners are concerned with the effort spent in applying any tool. To understand developers' work and the implications of this study two theories are presented for understanding and prioritising UP. For applying classification schemes, the implications of this study are that training and context are needed.

1. INTRODUCTION

Discovering usability problems (UP) is insufficient: we must be able to explain them (Wixon, 2003), i.e. what they involve, what causes them and how they are fixed. Human-computer interaction (HCI) researchers and practitioners have long tried to build a link between a problem and its design, recognising that a problem is not rooted in a static design but in the dynamic interaction that occurs between a *user* and the user interface *design*. The motivations for analysing problem characteristics can be several: (i) To understand the nature of the problem, including what has gone wrong in the interaction, or what the deviation from the intention has incurred; (ii) To find relations between problems, such as identifying generalised patterns or types (e.g., a local fix solving only the symptoms vs. global findings of underlying causes cutting across products (Dumas, 2002) or aggregating several problems into one problem scenario (e.g., SUPEX (Lavery et al., 1997))); (iii) To understand a problem's characteristics, such as its effects on the user, and to make the problem analysis valid, as in not reporting problems that are false alarms. The ultimate goal is to see whether analysis of these problem characteristics and their relations to interactions can guide designers to devise and implement new designs that fix problems (Hornbæk, 2010).

Classifying usability problems according to characteristics is not an end itself but a means to enhance the understandability of UP, enabling developers and designers to prioritise them correctly (i.e. high priority UP are corrected first) and inspiring them to form redesign ideas. It has been suggested that researchers should look at how problem prioritisation is done in practice (Hornbæk, 2010), as the matter has not been getting enough attention in the HCI field. Several studies have been conducted to investigate the relationship between prioritisation and severity (Hassenzahl, 2000; Hertzum, 2006) but the findings are inconsistent. Two issues that make severity a difficult parameter to assess and apply are the evaluator effect (Hertzum and Jacobsen, 2001) and the vague persuasiveness of this parameter given its seemingly arbitrary nature. Other influencing factors on prioritisation include managerial conflicts, budgetary constraints, frequency of problems, location of problems and technical skills. Determining which UP to fix is a challenging issue (Bruun and Stage, 2012), but deciding how to fix them can be even harder. The generic topic of revisions has not received enough attention, and little is known about how evaluation results are interpreted and translated into revisions (De Jong and Schellens, 2000). Basically, redesign can either be symptomatic or non-symptomatic, with the former being triggered by problem fixing and the latter by new requirements. According to Wixon (2003) view that usability evaluation should not cease at the point of identifying UP but extend to include fixing UP, redesign proposals are deemed significant evaluation outcomes (Andre et al., 2003; Hornbæk and Stage, 2006). Describing UP is necessary but not sufficient for coming up with viable redesign proposals; more relevant would be providing a development team with accurate information about underlying UP causes.

Besides the motivations of making the tasks of understanding and prioritisation problems easier, and redesign, another motivation for developing classification schemes is improving the development process. Though the software engineering community has mostly addressed this issue (Dyba, 2005), there is evidence that HCI practitioners are well aware that applying analysis, design or evaluation methods can impact the development process. A survey of practitioners (168 responses) asked to identify the three most significant outcomes of methods revealed that 44% of the respondents rated impact on the development process as an important outcome of a method, landing in sixth place after such factors as customer satisfaction, new design suggestions, usability of the system, new understanding of users, and identification of usability problems, which came in first place (69%) (Bark et al., 2005).

The topic of classification schemes has been researched in the software engineering community. In a systematic literature review of papers that researched software requirement errors (Walia and Carver, 2009), the authors identified nine schemes that use error information to improve software quality. However, no scheme is tailored specifically to UP. The above motivations have driven us to develop a scheme targeting UP. This scheme, entitled "Classification of Usability Problems" (CUP), has the following main goal:
Classify usability problems (UP) further to give developers better feedback to improve their understanding of these problems, help them manage usability maintenance, enable them to find effective fixes for UP, and prevent such problems from reoccurring in the future.

To systematically evaluate the effectiveness of CUP-informed redesigns, the evaluation of design-change effectiveness is important (John and Marks, 1997), i.e. the effectiveness with which the resolution of a usability problem (UP) is implemented (downstream utility). This is an under-researched area (Law, 2006). The psychology of developers, despite a half-century attempt to bring psychological paradigms to bear on software engineering (Curtis, 1984), remains not well understood, including such areas as how developers integrate multifarious information (e.g., usability evaluation outcomes, corporate guidelines, timeframe) to draw up a problem fixing plan (cf. theoretical models for decision-making (Patel et al., 2002) and information integration (e.g., (Anderson, 1996)) and how they associate low-level coding data with high-level redesign solutions (cf. the viscosity of cognitive dimensions (Green, 2006)). Methodologically, experimental psychology lab-based paradigms seem inapplicable, as the design process is highly situated, intertwining with a cluster of personal, social, technical and organisational factors. In the ensuing text, we further explore the role of CUP in the complex issues of prioritization and redesign.

The validity and reliability of methods have been a concern in the HCI discipline for decades (De Jong and Schellens, 2000; Wenger and Spyridakis, 1989), but these characteristics are unfortunately all too seldom evaluated. The reliability analysis conducted in this research study aimed to learn whether analysts can agree while using the classification scheme, what their performance depends on and, if they do not agree, how the scheme can be improved. As with any tool provided for development, we must probe applicants on their acceptance of the tool. In the acceptance study, we asked participants about the scheme's ease of use and usefulness and their intention to use the scheme further. Though such studies have not been performed on defect classification schemes (see below), they are now considered relevant measures of developers' motivations (Baddoo and Hall, 2002) and software process improvement adoption (Green et al., 2005). A method's validity concerns whether it measures what it is intended to measure (De Jong and Schellens, 2000) or observe. As an example, if our aim was that the CUP scheme be able to measure problem understanding or prioritisation, we would want to know if there was a match between those concepts and the variables constructed for CUP. While other papers have evaluated "predictive validity" (De Jong and Schellens, 2000), where one method is compared to another, often in a laboratory setting, we were more interested in how well the developers found the metrics represented real-world activities. As a part of the validity analysis, we constructed a theory of what variables can support two activities, namely understanding usability problems and prioritising the usability problems to fix. Therefore, this paper goes beyond mere observation of the developers' understanding and prioritisation of usability problems by trying to understand why they work the way they do and how (Hannay et al., 2007). Building a theory of what influences developers' tasks allows us to form a foundation to further validate the conclusions and compare them to analogous theories in software engineering and related design disciplines.

Three main rationales underlie the current work: (1) to expand the scale and scope of studying the reliability of the CUP classification scheme with usability practitioners of different expertise levels in applying CUP and domain knowledge, (2) to assess the acceptability of CUP among its users, and (3) to establish the validity of CUP by assessing its usefulness during the redesign of two systems of interest. The above aims were addressed by studying the development of user interfaces of two systems, LMS (Learning Management System) and HS (Hydrology Recording System). This paper is organised as follows. In section two, CUP is compared with other classification schemes. Section three contains a description of CUP and its development and includes the research questions based on the three rationales introduced above. Section four describes the research methodology in detail. Section five presents the results of the study. Finally, section six of the paper discusses implications drawn from this research study on the scheme's improvements and the downstream utility of CUP.

2. CLASSIFICATION SCHEMES

Hvannberg and Law (2003) developed the CUP scheme. The basic justification behind the creation of CUP is that defects (i.e. a list of usability problems) identified in usability tests greatly enhance product quality and process improvement if precise and concrete information about these defects can be presented to the development team. Freimut (2001) proposed the structure of a Defect Classification System (DCS) which inspired CUP development. Other DCSs were also used as reference frameworks, including those found in

(Card, 1998), Chillarege et al. (1992), Grady (1992), Leszak et al. (2000) and Mays et al. (1990). Hvannberg and Law (2003) adjusted the granularity of existing attributes and their subsuming values and created new values to meet the needs of user interface developers. Attributes from other DCSs are not relevant because, in principle, when usability problems (UP) are classified, the evaluators as usability engineers do not necessarily have access to the source code of the interface.

To better understand the basis of CUP and its distinction from other DCSs, three DCSs were selected for comparison: Orthogonal Defect Classification (ODC) (Chillarege et al., 1992; Freimut, 2001), Root Cause Defect Analysis (RCA) (Leszak et al., 2000; Leszak et al., 2002) and User Action Framework (UAF) (Andre et al., 2001). These three schemes were selected because we believe they have most relevance to CUP. Furthermore, ODC and RCA have been heavily cited and UAF was selected because it has been specifically designed for problems found in user interfaces.

Before we describe the comparison of the four DCS, we will explicate how the literature review was performed. We searched for citations to the original papers describing the schemes in Web of Science® and Google Scholar, and only looked at journal and conference publications. Papers found in Web of Science® were loaded into EndNote, a reference manager, where we searched in abstracts for the keywords reliability, repeatability, validity, case study and the name of the scheme. The search within Google Scholar was different in that after receiving a set of cited papers of the original publication of the scheme, the keywords search was applied to the whole document. Regardless of the portal, the abstracts of the resulting set were read, and papers, thus indicating evaluation of reliability or validity were selected for reading. Table 1 contains a comparison of these four DCSs with respect to ten factors, providing an overview of the classification schemes and identifying their commonalities and differences. A central, common factor is that three of the four schemes have not been assessed formally for validity. However, the current CUP study attempts to do so, as described in Section 5.3 “Validity Analysis”. Although validity has not been formally evaluated for the schemes, several case studies have been reported on the use of the schemes. ODC’s successful use has been reported in a number of papers, e.g. (Bassin et al., 2002; Bhandari et al., 1993; Butcher et al., 2002; Gupta et al., 2009; Hamill and Goseva-Popstojanova, 2009; Lutz and Mikulski, 2004; Shenvi, 2009).

Among the four schemes, reliability has been systematically evaluated for CUP and UAF and partly for RCA. Stochel (2011) conducted a study of the agreement between evaluators using RCA and between an evaluator and a benchmark, which was played by either the person who fixed the fault or the technical lead. Six evaluators assessed 20 defects using three attributes, process root cause, screen failure and engineering type. The current study aims to evaluate CUP’s reliability more extensively, as described in Section 5.1 “Reliability Analysis”.

None of the schemes has been evaluated for user acceptance. All schemes, except CUP, estimated the mean effort for applying the scheme. We view this as a fundamental dimension, which the current study aims to estimate.

A few case studies have been reported on Root Cause Analysis, e.g. Ghazarian (2009) conducted a study of how defects are introduced into software. Interestingly, the study found 449 defects and concluded that 8% of the defects examined in the user interface component can be traced back to inconsistencies in the user interface. Case studies of UAF include one by Mentis and Gray (2003), who classified problems according to the four phases of UAF, i.e. planning, translation, physical action, outcome and assesment. In addition, Khajouei et al. (2011a) have compared the results of two evaluation methods against the same four stages of UAF. Furthermore, the method has been extended with severity and impact to enable better understanding of the problem, its prioritization and redesign (Khajouei et al., 2011b). Chattratichart and Lindgaard (2008) compared two heuristics evaluation methods using UAF, similar to the work of Mentis and Gay (2003) but using the whole framework.

Table 1. Comparison of Four Defect Classification Schemes (DCSs)

DCS / Comparison criteria a	Classification of Usability Problems (CUP) (Hvannberg and Law, 2003)	Orthogonal Defect Classification (ODC) (Chillarege et al., 1992)	Root Cause Defect Analysis (RCA) (Leszak et al., 2002)	User Action Framework (UAF) (Andre et al., 2003)
1.Goal	Classify UP in order to give developers better and constructive feedback on how to correct the defects found in usability evaluation	Provide feedback on all the development lifecycle, measure the effectiveness of a verification stage and find the reason a defect surfaced.	Find systematic root causes of defects. Reduce number of critical defects and reduce work cost by proposing improvement actions	Guide interaction development activities and facilitate high quality UP reporting
2. Structure & Attributes	11 attributes, 7 of which have pre-defined values(*): Defect ID Description Defect Detection Activity Trigger Impact Failure Qualifier* Expected Phase* Actual Phase* Type of Fault Removed Cause* Error Prevention	8 attributes, all of which have pre-defined values: Activity Trigger Impact Target Source Age Defect Type Defect Qualifier	6 attributes, 4 of which have pre-defined values (*): Phase Detection* Real Defect Classification* Defect Nature* Real Defect Location Defect Triggers Barrier Analysis	It uses a hierarchically structured knowledge base with 6 base levels. The flattened structure has more than 150 end-node descriptions. Each node represents a usability attribute. The path from the root to a specific node is an encoded representation of a problem type and its causes.
3. Users	Developers, usability experts	Developers	Developers	Usability Experts
4. Development lifecycle phase	After usability evaluation	Throughout the systems development lifecycle	Throughout the systems development lifecycle	Throughout the systems development lifecycle
5. Proportion of problems that the scheme was applied to in the case study	Two CUP analysts applied CUP to all the UP that were found: 39 UP in user tests and 52 UP in Heuristic Evaluation.	Applied to all the defects found by the technical team including developers, testers, and service personnel	20-40% of the total modification requests (MRs) were analysed by an analyst and the authors	Ten evaluators analysed 15 out of over 100 UP case descriptions in a database
6. Resources expended	Not stated in the study	Not stated in the study	The mean time spent on analysing a modification defect was 19 minutes	All the evaluators viewed a 20-minute tutorial on UAF. None of the evaluators took more than 90 minutes to classify the 15 UP descriptions.
7. Process Automation	Not available	Unclear	Fully automated web interface tool	Fully automated
8. Reliability	Inter-evaluator kappa between two evaluators	Not addressed	For a few variables, evaluations were compared to a standard (i.e. a technical lead or a developer fixing the fault), in Stochel, 2011	Multi-evaluator kappa among ten evaluators
9. Validity	Not addressed	Successful results of applying ODC in different projects have been reported e.g. see next line.	Not addressed	Not addressed
10. Relevant case studies of the		(Bassin et al., 2002; Bhandari et al.,	(Ghazarian, 2009)	(Chattratichart and Lindgaard, 2008)

schemes		1993) (Butcher et al., 2002; Lutz and Mikulski, 2004); (Gupta et al., 2009); (Hamill and Goseva-Popstojanova, 2009); (Shenvi, 2009)		(Khajouei et al., 2011b; Mentis and Gay, 2003)

Before proceeding further, we distinguish between related terms used in the paper. CUP is designed to be applied to usability problems (UP), which we define as “a flaw in the design of a system that makes the attainment of a goal with the use of the system ineffective and/or inefficient and thus lowers the user’s level of satisfaction with its usage” (Law and Hvannberg, 2002). In addition, we present definitions of a fault, failure and defect according to the IEEE standard 729 (IEEE, 1983). A fault occurs when a human error results in a mistake in some software product, i.e. the fault is the encoding of the human error. A failure is the departure of a system from its required behaviour. Failures can be discovered both before and after system delivery. Defects refer collectively to faults and failures.

3. CLASSIFICATION OF USABILITY PROBLEMS (CUP)

Usability specialists or developers, hereafter termed CUP analysts, apply the CUP scheme to usability problems by evaluating them according to thirteen attributes. CUP analysts do not have to discover the usability problems themselves; they can receive them electronically from a usability specialist who has conducted the usability evaluations. Nine attributes are recorded before a usability problem is fixed, which we call Pre-Fix, and four are recorded after a usability problem is corrected, called Post-Fix. The term corrected is used here to mean that the problem has been confirmed and resolved by, for examples, changing the design, followed by implementation. Whether the change removes the problem can be verified only after another round of evaluation and testing. Pre-Fix attributes describe in detail UP that are found in usability testing by representative end users in the case of user-based evaluation or usability experts in the case of usability inspection. The Pre-Fix results are presented to developers, who can then view the specific results for a selected subset of problems and/or collectively view the descriptive statistics. After correcting some or all UP, the developers record four attributes in Post-Fix. Figure 1 illustrates this process. The remainder of this section provides an overview of the attributes, including the two changes (attributes *Expected phase* and *Cause*) made to the original CUP scheme for this study before performing the reliability and validity analysis.

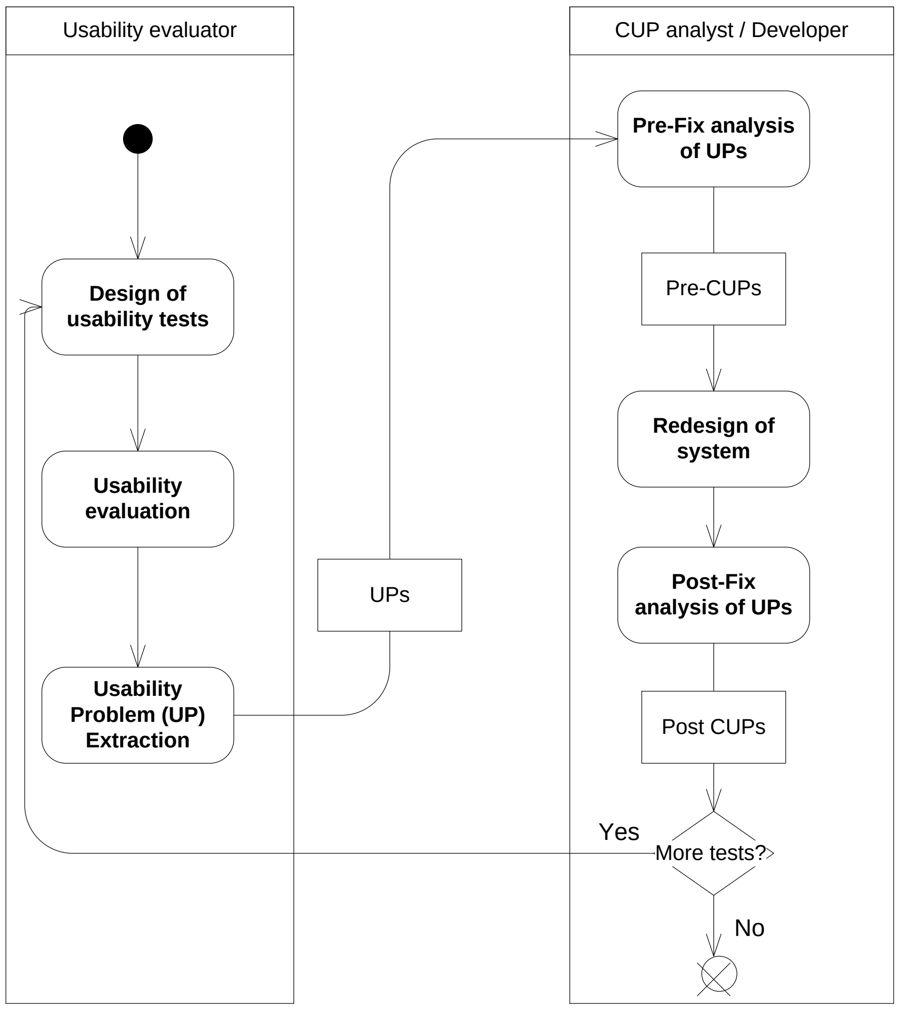

Figure 1. The CUP process

Table 2 shows brief definitions of the CUP attributes and lists their values with the four Post-Fix attributes highlighted in grey. The third column in Table 2 explains the changes made to the original scheme in the study. In response to developers' comments during the first study reported in this paper, two attributes were added to the original Pre-Fix scheme: *Frequency* and *Context*. The developers were interested in knowing how many users/experts experienced/predicted a UP. This provides developers with an idea, quantitatively, of how much different users/experts agree on that UP. In the *Context* variable a screenshot from the user interface can be used for additional information. This was added to separate the context description from the UP description and allow grouping of the UP according to context. Both variables are often found in usability evaluations. Several CUP attributes warrant a detailed description of their values and an explanation of how they are constructed from previous research. The rationale of the *Failure Qualifier* attribute is that it helps developers understand the nature of the problem and helps them fix it. The definitions of the Failure Qualifier values are inspired by a corresponding Qualifier attribute in the Orthogonal Defect Classification scheme (Chillarege et al., 1992) and are as follows:

- *Missing*: When the test participant fails to find something in the user interface that she expected to be present.
- *Incongruent mental model*: When the user interface is unclear because it does not match the test participant's mental model or previous experience.
- *Irrelevant*: When the user interface contains information/objects that do not contribute to system services and are unnecessary.
- *Wrong*: When the test participant notices that something has gone wrong, e.g. an apparent programming bug.
- *Better way*: When the test participant suggests that something in the user interface could have been done differently.
- *Overlooked*: When the test participant is given a task, but she or he overlooks an entity in the user interface, i.e. the user does not see the existing entity or fails to realise that she/he is supposed to interact with it.

Table 2. CUP attributes and their values, first Pre-Fix variables and then Post-Fix.

Attribute			**Values**	**Changes made to CUP for the current study**
Pre-Fix				
Defect ID			Identification number	
Frequency			Number of users/experts that experienced/predicted a UP	New attribute
Trigger			Describe what a user is doing when she/he discovers the UP, i.e., task scenario, heuristic, reflective question	
Context			Describe in what part of the user interface the user/expert was when the UP occurred	New attribute
Description			Concise description of the UP	
Defect Removal Activity			Usability evaluation method, e.g. user test and heuristic evaluation	
Impact	**Severity**		Indicates what effects the UP had on the user/expert: Severe, Moderate and Minor	
	Task Efficiency (%/min)			
	Task Effectiveness	**Completion rate**		
		Mean time on task (min)	Only for UP found with user tests (UT), calculations based on the tasks scenario that the user was performing	
	Instances of Frustration			
	Instances of Help Sought			
Failure Qualifier			Describes how the user/expert experienced a UP; ○ Missing ○ Incongruent Mental Model ○ Irrelevant ○ Wrong ○ Better way ○ Overlooked	One of the values was renamed from Extraneous to Irrelevant
Expected phase			Indicates in which phase of a software development lifecycle the developer thought that the UP originated before fixing it: ○ Task analysis and context of use (TAN) ○ Functional requirements (FUR) ○ Quality attribute analysis (QAN) ○ Conceptual modelling (COM) ○ Dialogue design (DIA) ○ Navigational design (NAV) ○ Presentation design (PRE) ○ Implementation (IMP)	This attribute was revised after the LMS study reported in this paper. The values were organised hierarchically into three development stages: Requirements (TAN, FUR, QAN) Design (COM, DIA, NAV, PRE) and Implementation (IMP). The descriptions of the values were improved to increase understandability.

Post-Fix		
Actual Phase	The phase where the UP was fixed; same values as Expected Phase.	
Types of Fault Removed	Describes what in the user interface was changed to fix the UP	
Cause	Aimed to understand why the developers committed the error: ○ Personal ○ Communication ○ Technical ○ Methodological ○ Managerial ○ Review	The attribute was revised based on the literature, but after the LMS study it was converted to free text, since developers found the values hard to understand and inappropriate.
Error Prevention Techniques	Ideas on what can be done in the future to prevent the fault	

The premise of the Expected phase is that if the developer knows in what development phase the problem originated, it could help him or her fix it. The design of the attribute *Expected Phase* was led by two objectives. The attribute should be appropriately detailed to give the developer sufficient information but not too detailed, which would make classification difficult. The number of values chosen can thus be critical. The phases must also reflect current practices, so they are likely to be familiar to usability engineers and developers. With this in mind, we coarsely analysed the current literature on user interface lifecycles. To meet the first goal, appropriate detail, we selected lifecycle models and integrated the phases into the *Expected Phase* variable. The lifecycle models were the Constantine and Lockwood (1999) model, Idiom (van Harmelen, 2001) and Hudson (2001) UCUML. We assumed that these models would be good candidates for our second goal, reflecting current practices, because of their wide dissemination, maturity, and acceptance in practice. The expected phase variable is described in detail below. The phases are described in the activities, which are typical actions that are performed; the deliverables are example outputs of the corresponding phases.

Requirements analysis

- **Task analysis and use context:** *Activities*: Analyse the application domain and what cognitive tasks and physical actions a user performs to achieve a goal. *Deliverables*: Use case diagrams and activity diagrams of tasks and/or hierarchical tasks. Scenarios. Descriptions of actors and the environment.
- **Functional requirements:** *Activities*: Decide the scope of the system. Select which tasks from the task analysis phase the system should perform. *Deliverables*: Use case diagrams, text description of use cases according to a template.
- **Quality attribute analysis:** *Activities*: Analysis of non-functional requirements, including usability, security, performance/efficiency, reliability/fault tolerance, interoperability between services or between devices and safety-criticality. *Deliverables*: Requirement specification.

Design

- **Conceptual modelling:** *Activities*: Analysis of the system as a set of concepts and what the main objects in the system are and its operations. *Deliverables*: list of concepts in a data dictionary, simple relationships between concepts, actions on concepts, content model or class diagram.
- **Dialogue design:** *Activities*: Divide the responsibilities and design the dialogue between man and machine. Decide how the user translates goals or intentions into actions on objects. Decide how the system responds in terms of what objects and actions are displayed as a response to an action. *Deliverables*: Essential use cases context maps, sequence diagrams.
- **Navigational design:** *Activities*: Divide the user interface objects and actions into groups, i.e. contexts. Determine navigations between contexts. *Deliverables*: User interface architecture, navigational model.
- **Presentation design:** *Activities*: Design the look of the user interface, i.e. layout of objects within a context, presentation of objects and actions as user interface elements, i.e. icons, buttons, layout, font, colour scheme. *Deliverables*: Graphic design of user interface objects.

Implementation: *Activities*: Programming *Deliverables*: programs and/or prototypes of user interface.

The above phases are not meant to imply any order, i.e. they do not imply a waterfall-like lifecycle, but could also be used in a spiral lifecycle or even an agile process where phases are not well delimited. It is necessary to keep this in mind because the uptake of agile processes has become popular in the last decade. However, empirical research has not shown consistent evidence for better product quality in agile software development (Dybå and Dingsøyr, 2008), and one study showed that teams develop a better and much more consistent user interface with traditional lifecycles (Wellington et al., 2005).

The final attribute described in this section is the *Cause* attribute from the Post-Fix attribute set. HCI researchers have been characterising causes of interaction problems from several perspectives, including cognitive, social, organisational and development process. The Post-Fix attribute *Cause* describes the development process perspective. An in-depth discussion of the attribute supports the view that a causal analysis is an important skill for a usability evaluator (Dumas, 2002p. 1106).

A cognitive approach can help explain cause. Capra (2006) developed 10 guidelines for describing usability problems through a thorough study using an open-ended questionnaire with 19 respondents and card sorting of the results with 8 participants. The guideline on cause was: "Describe the cause of the problem, including context such as the interaction architecture and the user's task. Describe the main usability issue involved in the problem. Avoid guessing about the problem cause or user's thoughts." In Capra's study (2006), problem cause was considered difficult to assess, but at the same time required. This attribute is related to two attributes of CUP, Trigger and Context.

When analysing deviations from a system task model, Paternò and Santoro (2002) recommend that information on potential causes be collected, i.e. "indicating the potential causes for the deviation considered and which cognitive faults might have generated the deviation”. The types of causes are classified according to which phases of Norman’s model (1988) can cause the problems: intention, action, execution, perception, interpretation and/or evaluation. The above work is rooted in the analysis of safety-critical software and hazard and accident analysis. Rasmussen (1986) developed a classification scheme called Skill, Role, Knowledge (SRK) for types of errors occurring during information processing. Reason (1990) identified these three classes as ranging from automatic (skill) to conscious (knowledge-based). He developed these ideas further into GEMS, a Generic Error Modeling system, which explains the switching between different types of errors (Embrey, 1995). Hollnagel’s (1993) classification distinguishes between observable error, phenotype (manifestation) and genotype (the cause). Sutcliffe and Rugg (1998) take a broader approach, grounded in previous research, including a classification of causes into social, organisational and cognitive categories. Sutcliffe and Rugg include a category of design errors in their taxonomy, as do Lavery et al. (1997), who have suggested a more development-oriented model. In this model, for example, a design fault causes a breakdown in an interaction, which results in changed user behaviour or performance. A cause in their model is not only a design fault, but it can be further rooted in a wrong knowledge requirement on the part of the user, thus acknowledging that the user is a part of the context. Lavery et al. (1997) allow evaluators to enter free text in the cause variable.

In a discussion of motivating change through usability testing, Dumas (1989) raises the awareness that user testing can improve the technical and managerial skills of product designers and managers. In one of four usability testing practices, Dumas’ recommendation is to focus not only on product-related problems but also on the underlying technical and organisational problem causes. Examples of probable causes include a lack of clear guidelines for how to create effective software or manuals, lack of co-operation between people on the product development team, and lack of effective design reviews. Sutcliffe and Rugg (1998) propose a distinct category for human failure in the design process, which has indirectly led to accidents. This design process category has six values: lack of method conformance, poor tool support, machinery failure, maintenance problems, poor design method, and inadequate support tools. In software engineering, scientists and practitioners have for several decades (Chillarege et al., 1992; Endres, 1975; Weiss and Basili, 1985) studied the cause of faults through classification. The values selected for the *Cause* attribute in CUP are Personal, Communication, Technical, Methodological, Managerial and Review (Leszak et al., 2000). A systematic literature review of 149 papers (Walia and Carver, 2009) has resulted in a taxonomy of software requirement errors that led to the actual faults, i.e, people, process and documentation errors, with a total of fourteen values. Some values in the taxonomy coincide with values of the *Cause* attribute, including Communication, Methodological and Management. Though this taxonomy is comprehensive, it remains to be seen how suitable it is as a basis for classification.

Having devised the CUP scheme and performed the initial evaluations, it became clear that a thorough evaluation of its reliability, validity and acceptance was needed. This would not only lead to improvement suggestions for the CUP scheme, but also to new knowledge on how software developers perceive data, how they make decisions, and how they create ideas. Finally, given the challenging methodology of evaluating defect schemes, the present study was expected to provide insights in this regard. Thus motivated, the following research questions, based on the three goals described in the introduction, are set forward:

Reliability

RQ1: How reliable is the Pre-Fix scheme among a group of novice CUP analysts?

RQ2: To what extent is the level of inter-evaluator agreement influenced by the CUP analysts' expertise in applying the Pre-Fix scheme and by their involvement in usability testing the application?

Validity

VQ1: Do developers think that the Pre-Fix CUP scheme aids their UP understanding?

VQ2: Do developers find the Pre-Fix CUP scheme useful when prioritising what UP to fix and deciding how to correct them?

VQ3: How well do the developers understand Pre- and Post-Fix CUP attributes and their values? Do developers propose any changes to the CUP scheme?

Acceptance

AQ1: To what extent do CUP analysts' perceived ease of use and usefulness of the Pre-Fix CUP scheme relate to their intention to adopt it in the future?

4. RESEARCH METHODOLOGY

For the study described in this paper, we have conducted three tests for different purposes. Figure 2 illustrates the flow between the main parts of the study. The four studies are as follows: (i) user tests (UT) on two applications with a total of 18 participants to obtain a list of usability problems used as a baseline in the next two tests; (ii) reliability study on a subset of the UT results with eight novice participants and two experts to assess CUP reliability; and (iii) presentation of the Pre-Fix classification to four developers in two field studies to assess scheme validity. In the reliability analysis, the participants classified 21 UP according to CUP, and the reliability was evaluated using a generalised kappa. Following the reliability study, (iv), an acceptance study was conducted with 8 novice participants. The validity analysis consisted of two field studies, where two different applications were tested for usability. After presenting developers in the first field study with the CUP results, the scheme was refined based on their views. The second field study then started, and data were collected on how developers used the results in their development. Each part of the research methodology is discussed in detail in this section.

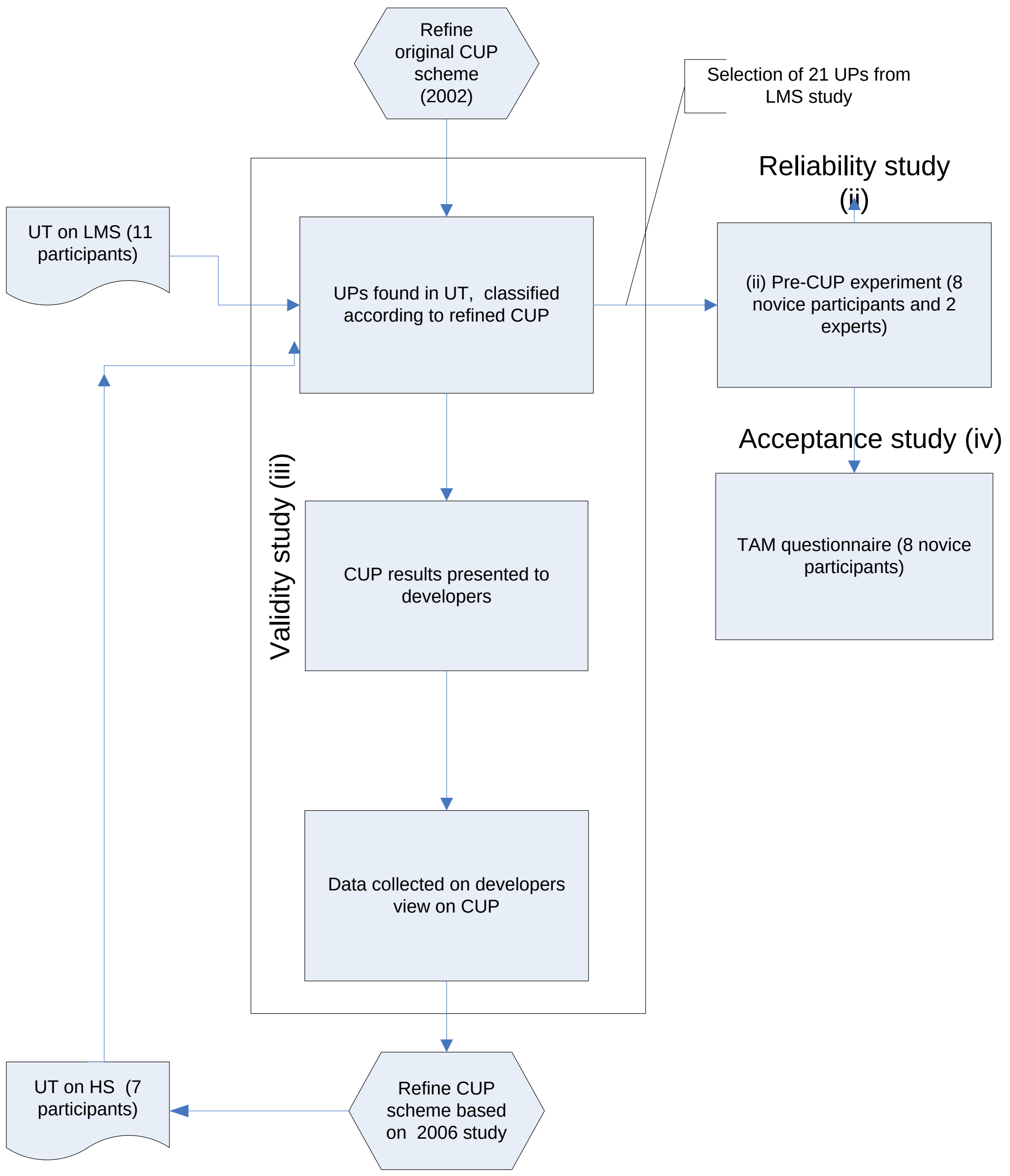
Refine original CUP scheme (2002)
Selection of 21 UPs from LMS study
Reliability study (ii)
UT on LMS (11 participants)
UPs found in UT, classified according to refined CUP
(ii) Pre-CUP experiment (8 novice participants and 2 experts)
Acceptance study (iv)
Validity study (iii)
CUP results presented to developers
TAM questionnaire (8 novice participants)
Data collected on developers view on CUP
UT on HS (7 participants)
Refine CUP scheme based on 2006 study

Figure 2. The flow between the main parts of the study

4.1 User test

The purpose of the user test reported in this section was to produce data that would be used in the two reliability and validity studies.

4.1.1 Applications

This study uncovered defects in two applications, called the LMS Teaching Web Version 2.0 and HS Version 1.9.2, with think-aloud user tests. The LMS Teaching Web is a university learning management system which allows teachers to manage their courses online (hereafter referred to as the LMS). The application has two user groups: teachers and students. Teachers can provide students with online course materials (e.g. lecture notes and web links), send announcements to their students, create online discussions and maintain an event calendar for their course. Students have fewer functions available to them and mainly use the system to access the provided learning resources. The size of the application is roughly 11 thousand lines of code, and the first version was released in the autumn of 2001. The second application, HS, is a storage and analysis system for data collected by a governmental institute (e.g. river flow data). It was developed by the institute and first released ten years before this research took place. It enables users to analyse the collected data and conduct comparisons on data collected over time.

4.1.2 Method and Procedure

Eleven participants were recruited for the LMS UT, including five teachers and six students. The student participants received a compensation of €32 for participating in the tests. No participant had previously used the LMS Teaching Version 2.0 in any courses. The user groups had different tasks that they were required to perform, but both groups had nine task scenarios covering the core functionalities of their user group. Seven regular HS users were recruited for the UT in the second field study and were required to perform five tasks. The UT procedure for both applications was kept as similar as possible. While performing the tasks, the participants were asked to think aloud. An assistant UT tester was present at all sessions, and he recorded the test participant's thinking aloud verbatim. The sessions were also recorded using Camtasia Studio (TechSmith, 2005) to capture screen activity and the thinking aloud audio.

4.1.3 Measurements

The performance measures were mainly obtained through observations, including time on task, number of screens of online help consulted, and number of instances of expressed frustration. Primarily, the test participants reported the subjective measures in a think-aloud verbatim protocol and by answering post-task and post-test questionnaires. The LMS users experienced 112 total UP. The UP were filtered by consolidating those that were the same, i.e. users experienced the same types of problems in the same parts of the application. This was done by going over the list of UP, one at a time, and checking whether a similar UP had come up previously on the list. After filtering, there were 71 unique UP, including 46 in the teachers' group and 25 in the student group. Of the 71 unique UP, there were 26 Minor, 30 Moderate and 15 Severe UP. The average number of UP experienced per user was 10.2 (SD=4.0, N=11).

The seven users in the HS UT experienced a total of 83 UP. After consolidating the UP, the list contained 65 unique UP, including 31 Minor, 20 Moderate and 14 Severe. The mean number of UP discovered per user was 11.9 (SD=5.4, N=7).

4.2. Reliability and Acceptance Studies

The reliability of the CUP scheme depends on its repeatability. It is only repeatable if there exists consistent, shared understanding of the scheme. Defect classification is a subjective exercise, and it is thus possible for different CUP analysts to classify the same defect differently. The credibility of defect data is unconvincing if such disagreement in classifications is dominant (Hvannberg and Law, 2003). The reliability analysis only addresses Pre-Fix attributes, as they constitute feedback that developers receive from the usability testing. To evaluate CUP reliability an experiment was conducted, which is described in detail below and illustrated in Figure 3. The reliability study was intended to answer the first goal (1) in the introduction, and the results are presented in Section 5.1, Reliability Analysis.

4.2.1 Selection of Usability Problems for Experiment

As reported above, 71 unique UP were identified in the LMS application UT. We developed a scheme for selecting a subset of UP to use in the reliability experiment. When developing the scheme, we kept in mind that the classification should not take more than approximately 90 minutes for the participants, as otherwise they would get tired. We anticipated that the participants would be able to rate approximately 20 UP in 90 minutes. We reviewed two selection schemes: a case study of Root Cause Analysis (Lezak et al., 2000) that used a rather complicated scheme where many items had to be selected and the User Action Framework (Andre et al., 2001) in which UP selection was based on the real-world expected frequency. Taking into account the trade-offs of these selection schemes, we developed our own scheme to select a subset of UP to be examined in the reliability experiment:

i. Sort all UP according to the UT tasks where they occur.
ii. For each UT task, first sort UP according to frequency, which is an objective parameter, and then according to severity. Two analysts should perform this severity rating, to discover inter-evaluator reliability. One analyst should have conducted the UT.
iii. Select the top half or 50% of the sorted list of UP per task.
iv. Apply the Pre-Fix scheme to these extracted UP. An analyst who conducted the UT performs the rating.
v. Identify 21 UP from those extracted in iv that cover most attribute values, i.e. at least one UP from each value of the attributes *Expected Phase* and *Failure Qualifier*.

In step i UP were classified according to 18 tasks. In step iii 40 UP were extracted. The rationale of step v was to maximise the number of UP types with different attributes and attribute values, thereby allowing the reliability test participants to work on various UP types. After step v we ended up with approximately 30% of the total UP, or 21 UP.

The two analysts who were involved in step ii are hereafter called the CUP and User Test (UT) experts. The CUP expert has better knowledge about CUP than the UT expert, but she conducted the UT. The UT expert performed the selection process and prepared the data on the 21 UP, i.e. filled out the Defect ID, Context, Description, Defect Removal Activity and Trigger attributes. The agreement between the experts on the severity rating was only fair (kappa value κ=0.303). The CUP expert received the same materials as the Pre-Fix experiment participants, who had all dealt with so-called ‘second-hand’ data about the UP because they did not conduct the UT themselves.

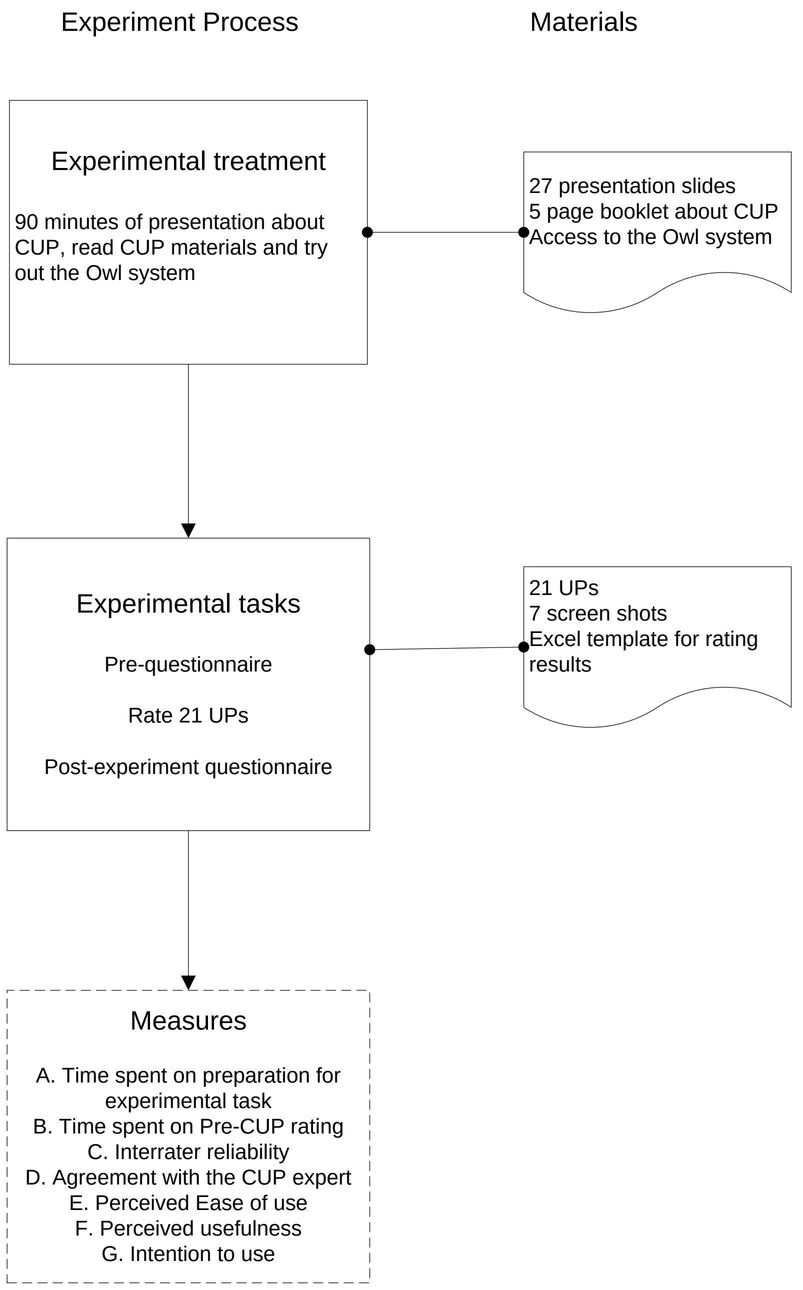
Experiment Process
Materials
Experimental treatment
90 minutes of presentation about CUP, read CUP materials and try out the Owl system
27 presentation slides
5 page booklet about CUP
Access to the Owl system
Experimental tasks
Pre-questionnaire
Rate 21 UPs
Post-experiment questionnaire
21 UPs
7 screen shots
Excel template for rating results
Measures
A. Time spent on preparation for experimental task
B. Time spent on Pre-CUP rating
C. Interrater reliability
D. Agreement with the CUP expert
E. Perceived Ease of use
F. Perceived usefulness
G. Intention to use

Figure 3 Reliability study, materials and measures

4.2.2 Participants

Eight participants were recruited to participate in the reliability experiment, who were each paid approximately €64 for their participation. There were four master's degree students, two bachelor's degree students and two developers working in the industry. Three of the master’s degree students were studying computer science, and one was studying bioinformatics. The bachelor’s degree students were from computer science and industrial engineering disciplines. One software developer held a bachelor’s degree in computer science, and the other held a master’s degree in language technology.

The participants' ages ranged from 20-50 years, and most were 20-24 years old. Half of the participants had work experience in software development, user interface design or user interface testing. Four participants had used earlier versions of the LMS system in their studies at the university for 2-3 years, two for 1-2 years, and two participants had not used the application before the experiment.

4.2.3 Experimental Treatment

The experiment (Figure 3) involved two sessions which the participants had to attend. A presentation to introduce Pre-Fix was followed by a session where participants had to classify 21 UP according to Pre-Fix (3 attributes), which were selected using the method described in section 4.2.1.

The presentation to introduce Pre-Fix covered the following items: explaining the user tests, the goal of Pre-Fix, information about the system where the UP were identified, explaining the Pre-Fix attributes and their values, example of a UP that had been classified according to Pre-Fix and instructions on what participants were supposed to do in the experiment.

All participants received the following materials during the experimental treatment: a printout of the 27 slides used in the training presentation, a five page booklet about Pre-Fix with an example of its application, the tasks used in the system's user test and access to the part of the LMS system used in the test.

4.2.4 Experimental Task

The information session for the participants about Pre-Fix was held by the CUP and UT experts, and the participants took part in the second (classification) session two or three days later. They were instructed, in the meantime, to familiarise themselves with the materials about Pre-Fix so they would be better at applying it to the UP. They were also asked to try out the LMS system between the two sessions to know the domain better and have an enhanced UP understanding. The participants were instructed to record the time they spent on these activities.

The second session of the experiment was conducted in a computer lab so the participants could examine the system as they were classifying the UP. The participants received, on paper, 21 UP where the first five Pre-Fix attributes had been filled out for each UP. The attributes that had been filled out beforehand were Defect ID, Context, Description, Defect Removal Activity and Trigger. The participants also received a printout and electronic version of the seven screenshots to which the

Context attribute referred. The participants rated three attributes for each UP: Severity, Failure Qualifier and Expected Phase. The participants then entered their results into an Excel template that was provided, where they also recorded the time they spent preparing for the session and on the CUP classification. All participants filled in a questionnaire about their age, education, previous experience with the system and whether they had software development experience.

At the end of the second session, the participants filled in a post-experiment questionnaire based on the Technology Acceptance Model (TAM) (Davis, 1989), which had been adapted to evaluate the CUP scheme as relates to the empirical examination of the CUP analysts' perceptions of the CUP scheme. This part of the study was meant to address the second goal (2), acceptability assessment. Section 5.4 discusses these results.

4.3 Validity Study

The validity analysis consisted of two field studies in real software development environments, where four developers were introduced to the CUP scheme. To strengthen the validity analysis, we presented CUP to developers in different application development environments. Two field studies, using action case study (Avison et al., 1999; Braa and Vidgen, 1999) where designers and researchers collaborate, allowed us to make changes to CUP based on what was learned in the first field study and test them in the second field study. The validity study was intended to answer the third goal (3), validity. The remainder of this section describes the environments in which the studies were performed, the data that were collected and how they were analysed. The results are presented in section 5.3, Validity Analysis.

4.3.1 Environments of Study

Two LMS software developers were involved in the study. One is the manager of the software development department, and the other is the main programmer of the LMS system. They work together on designing the application and make decisions regarding it, but only one programs the application. Both test the LMS system, mostly unit testing, and have never used the results of a UT. The programmer of the application has a bachelor's degree in computer science and had worked for the company for 15 months when the study began. The department manager had worked for the company for nine years and is self-educated.

The HS system was first released in 1995, and the work on it today is mainly maintenance work and adding new functionality as requirements are added and changed. The two developers who participated in the study are responsible for maintaining the system; one has a bachelor's degree in computer science, and the other is a systems analyst. When the study started, the developers had worked on HS for approximately 18 months.

These applications, LMS and HS, were selected for this study because they were developed in dissimilar environments. The HS developers were close to their users, as the software is maintained in-house for a limited number of expert users, but the LMS is used by a large, diverse group in several universities. The HS developers worked on the redesign solely based on CUP results, but the LMS developers additionally had a list of comments and change requests from their users, collected through a web survey.

4.3.2 Data Collection Methods

This section describes how we collected the data in the LMS and HS field studies. After the LMS field study, we reflected on the research protocol to learn what could have gone better and employed that knowledge in the HS field study. The data in the LMS study were mainly collected in the form of semi-structured interviews and meetings with system developers over a three-month period. After the developers received the CUP results from the UT, ten meetings were held with them, lasting a total of 12 hours. Three meetings were spent discussing particular UP and their CUP ratings. These meetings provided the developers with a feel for the scheme and opened discussion on what they thought about it. We also examined the available documentation for the LMS system, which was limited, as their software development process is mainly agile and does not involve a great deal of documentation. The most relevant documentation was a list of 177 non-unique items that included comments and error reports from system users. The items on the list were collected through the company's help line and running web surveys about the company's services and applications. To motivate solutions to items/problems, the developer went through this list of items, which includes suggestions for enhancements, errors, and comments about functionality, usability and performance, and then they categorised all items according to a potential solution or change in design. Next, both developers estimated how long it would take to implement the changes and prioritised them.

In the HS field study, we wanted to meet the developers more frequently to collect CUP data while the developers were using the data or shortly thereafter, while their recollection was still fresh. The researcher met the developers twice a day on the days that they were working on HS for short discussion meetings. This approach is similar to that used in agile software development (Abrahamsson and Kyllonen, 2005; Control Chaos, 2006). The first meeting was in the morning to find out what they had planned for the day and on what tasks they intended to work. The researcher then met the developers again in the afternoon; each meeting lasted approximately 10-15 minutes. The afternoon meeting was to discuss how their work went during the day, what they achieved and if they ran into any problems. Taking this approach allowed the researcher to talk to the developers directly before and after they were working on the HS application and find out more about how they employed the CUP scheme to help their development tasks. These meetings provided information about how the developers were using the CUP data after deciding on what UP they were going to work. An advantage of this approach was that the researcher talked to the developers at their work desk, where they were comfortable. In this way it was also possible to observe the developers' work, and they could easily show the researcher something on screen or paper. The researcher also had more structured meetings with the developers, both one at a time and together, at the start and end of the study to reflect on the study overall. The researcher spent eleven hours in meetings and discussions with the developers about software development and the CUP scheme.

There was no up-to-date documentation available about HS, except failure reports in a program called ProblemTracker (NetResults Corporation, 2005). This is a web-based system where users can report bugs, requests for enhancements or other similar issues. Users also talked directly to the developers, who recorded the issues in ProblemTracker. The developers said that users had not been reporting many HS-related errors or issues during the year prior to the CUP study.

5. RESULTS AND DISCUSSIONS

5.1 Reliability analysis

5.1.1 RQ1: Reliability of Novice CUP analysts

To evaluate reliability, Fleiss' (1971) generalised kappa was used (King, 2004b). The generalised kappa statistic extends the kappa statistic to include multiple analysts and is useful for measuring inter-evaluator agreement among three or more evaluators (King, 2004b). King's Microsoft Excel spreadsheet was used to calculate the generalised kappa values (King, 2004a).

Table 3 reports the generalised kappa results. The reliability is highest for the *Severity* attribute and lowest for the *Expected Phase* attribute. The reliability results for the *Expected Phase* attribute suggest that its definition and values require refinement. This was confirmed by the post-experiment questionnaire, where the participants commented that the values for this attribute needed to be defined more clearly. The reasons for this may be twofold. First, there are probably too many values defined for this attribute. Second, the novice analysts may not have been given adequate background in user interface development to be able to distinguish between the phases, e.g. navigational and dialogue design.

Table 3. Reliability measures for eight novice CUP analysts in the experiment

Reliability measure	**Severity**	**Failure Qualifier**	**Expected Phase**
Generalised Kappa	0.31	0.25	-0.27

The *Severity* attribute has the highest reliability, and part of the reason may be that it has the fewest values of the three attributes being examined. The fact that *Severity* is an ordinal scale (i.e. Minor, Moderate and Severe) may have increased the understandability of the novice analysts more than the other attributes, which are nominal scales.

5.1.2 RQ2: Inter-evaluator Agreement and Influencing Factors

As the participants were all CUP novices, we wanted to compare their ratings pairwise to those of a CUP expert. The ratings of every participant were compared to those of the CUP expert. The CUP expert was as familiar with the UP as the novice CUP analysts and received the same data on the defects. In addition, to see how much knowledge about the UTs could help a CUP analyst, we compared the ratings of the CUP expert to those of the UT expert.

In this part of the reliability evaluation, we used a kappa coefficient because it is the recommended statistical method for verifying inter-evaluator agreement for nominal scale data classified by two evaluators (Siegel and Castellan, 1988). Kappa values can be qualified into five agreement levels, ranging from "poor" to "very good" (Altman, 1991). Table 4 shows the kappa results for the comparison between the CUP analysts in the Pre-Fix experiment and the CUP expert. The *Severity* attribute had relatively the highest level of agreement: moderate (0.41-0.61) for three CUP analysts, but five CUP analysts had a poor ($\kappa<0.2$) level of agreement with the CUP expert. The *Failure Qualifier* attribute appeared to have the most consistent level of agreement, as it was rated as fair (0.21-0.40) for seven CUP analysts. The *Expected Phase* attribute had the lowest overall level of agreement, with a poor value for half of the CUP analysts and a fair value for the other half.

Table 4. Kappa for the attributes, as rated by the participants

CUP analyst	**Severity**	**Failure qualifier**	**Expected phase**
CUP analyst 1	0.37	0.38	0.10
CUP analyst 2	0.50	0.32	0.09
CUP analyst 3	0.16	0.38	0.32
CUP analyst 4	0.48	0.25	0.14
CUP analyst 5	0.13	0.33	-0.04
CUP analyst 6	0.13	0.22	0.21
CUP analyst 7	0.11	0.25	0.31
CUP analyst 8	0.44	0.20	0.38
Mean	0.29	0.29	0.19
SD	0.17	0.07	0.14

If we consider the time that the participants spent preparing for the second session of the experiment, five of them spent less than an hour on preparation, two spent one to two hours and one participant spent two to three hours. The time that the participants spent on the Pre-Fix classification itself in the second session ranged from 60-117 minutes, and the mean time was 86 minutes (SD=17.01, N=8). In comparison, the CUP expert spent 50 minutes on the Pre-Fix classification, which was considerably less than the mean and could probably be explained by the fact that she knew the CUP scheme well.

Table 5. Kappa for the agreement of the CUP expert and the UT expert

Attribute	**No. of possible values**	**Kappa**	**Agreed cases***	**Level of Agreement**
Severity	3	0.33	12	Fair
Failure qualifier	6	0.50	13	Moderate
Expected phase	8	0.22	7	Fair

*Note the total number of cases was 21

All but one of the novice CUP analysts had a poor agreement level with the CUP expert for one attribute. Specifically, Analyst 5 had poor agreement for two attributes and was also the only analyst who had a negative kappa value for the *Expected Phase* attribute, which implies that rater 5 and the expert agreed less than would be expected just by chance. Analyst 5 spent less than one hour on preparation and spent the least time of all participants on the Pre-Fix classification, i.e. 60 minutes. It is possible that Analyst 5 rushed himself too much.

The analysts who spent more than an hour on preparation were Analysts 1, 2 and 4. As shown in Table 4, Analysts 2 and 4 had the highest level of agreement for the *Severity* attribute, and Analyst 1 had the highest level of agreement for *Failure Qualifier*. However, the extra time did not seem to help these analysts in rating the *Expected Phase* attribute, as they all had a poor level of agreement.

Table 5 shows the kappa results for the CUP and UT experts. The CUP and UT experts had a fair agreement level for the *Severity* and *Expected Phase* attributes and moderate for the *Failure Qualifier*. For the *Expected Phase*, the experts only agreed in seven cases out of 21, which was the lowest level of agreement of the three attributes.

5.1.3 Discussion on Inter-evaluator Agreement

The agreement level between the eight novice analysts from the experiment and the CUP expert was overall best for the *Failure Qualifier* attribute. This concurs with the agreement between the two experts (Table 5), but as discussed above, the *Severity* attribute had the highest reliability among novices according to the generalised kappa results. The *Severity* attribute has only three possible ordinal values and is thus easier for novices to learn rather than the eight nominal values for *Expected Phase*. The results confirm that the *Expected Phase* attribute requires refinement. The agreement level between the analysts and CUP expert on the *Failure Qualifier* attribute was quite consistent, i.e. nearly always fair, which might imply that the analysts understood some of the six values well and rated them similar to the expert, but perhaps they lacked understanding and practice in the other values. The value "Incongruent Mental Model" was most commonly used by the analysts, and seven agreed most of the time with the CUP expert on this value. The novice analysts never agreed with the CUP expert on the value "Better Way", and half of them never agreed with the expert on the value "Irrelevant".

When comparing the inter-evaluator agreements of this study with those of the previous one, we noticed that the agreement between experts in this study was similar to that in Hvannberg and Law

(2003) for *Severity* (i.e. fair) and *Failure Qualifier* (i.e. moderate) in UP discovered in the heuristics evaluation. The agreement was slightly worse than in the previous study for *Failure Qualifier* in UP discovered in usability tests with good agreement. The agreement in Hvannberg and Law (2003) was moderate for the *Expected Phase*, but for this study it was only fair. We attribute this difference to the experience in applying CUP.

5.1.4 Consensus of Experts

In a study by El Emam (1999), two groups independently performed a software process assessment based on the same data. After their independent ratings, the groups would meet to reach a consensus of their ratings and thus produce a consolidated rating, which was the outcome of an assessment. Based on this idea and because the kappa value between the experts was considered only fair for two attributes, it was necessary for the two experts to have a consensus building session. To find the reason for the disagreement and learn from it, the experts revisited the UP where they had disagreed. Next, they adjusted the ratings when appropriate. After the consensus building session, the kappa was calculated for the new ratings.

As Table 6 shows, the experts achieved a very good level of agreement (0.81-1.00: Very good) after their consensus building session. The following were the main reasons for the experts' disagreements:

- The CUP expert thought the description for the UP was not precise enough and misunderstood the problem.
- The UP descriptions contained two closely related usability issues, and the experts considered them from different perspectives. In these cases, the UP might have been split into two separate ones.
- Some Failure Qualifier attribute values overlap, and both experts were correct in their rating; the same was true for Expected Phase.
- The CUP expert thought a *Context* attribute value was unclear.
- Both experts acknowledged a mistake.
- The severity rating might have been partly based on the UT expert's opinion influenced by user tests rather than solely on the description.
- The CUP expert lacked domain knowledge of the application.

Table 6. Kappa for the agreement of the CUP and UT experts after a consensus session

Attribute	No. of possible values	Kappa	Agreed cases*	Level of Agreement
Severity	3	1.00	21	Very good
Failure qualifier	6	0.94	20	Very good
Expected phase	8	0.88	19	Very good

*The total number of cases was 21

5.2 Acceptance of CUP

This section presents and discusses the results of the post-experiment questionnaire filled out by the analysts in the Pre-Fix experiment. The post-experiment questionnaire had 19 questions: 16 closed and 3 open-ended questions. The closed questions were in a randomised order, and they were all measured on a 5-point Likert scale with a positive and negative assertion at each end of the scale for each item, e.g. 5 meaning easy to use and 1 meaning difficult to use. The following three perception-based constructs were used to evaluate the scheme:

- Perceived Ease of Use (EASE): the degree to which a person believes that using the scheme would be free of effort. (five questions)
- Perceived Usefulness (USEFUL): the degree to which a person believes that the scheme would be useful. (eight questions)
- Intention to Use (INTENT): the degree to which a person intends to use the scheme. (three questions) (Moody et al., 2003)

Table 7 shows the means and standard deviations for the three constructs. Perceived usefulness had the highest mean and intention to use the lowest. This was confirmed by the responses in the open-ended questions.

Table 7. Mean of analysts' responses (N=8)

Construct	**Mean**	**SD**
Perceived Ease of Use	3.48	0.60
Perceived Usefulness	4.00	0.45
Intention to Use	3.04	0.81

To answer research question AQ1, a multiple regression analysis was performed. The relatively high multiple correlation coefficient (adjusted $R^2 = 0.74$) suggested that EASE and USEFUL taken together could contribute to the INTENT prediction. As indicated by the beta weight (b), EASE was a significant predictor ($b = 0.66$, $t = 2.48$, $p = .05$), whereas USEFUL was not. The partial Pearson correlation coefficients (r) were 0.48 and 0.22 for EASE and USEFUL, respectively. These statistics corresponded to effect size measures in partial eta squared (η_p^2), which were 0.55 and 0.21 for EASE and USEFUL, respectively. The findings imply that ease of use is a critical factor in determining whether the novice analysts will adopt the CUP scheme in their future work. This observation was not surprising, as these analysts were not involved in the entire process of applying the scheme (i.e. defect revisions and subsequent Post-Fix attribute classifications) and thus could not see its full potential usefulness. What concerned them most was whether they would be able to understand and apply the scheme rather than whether it was useful to improve the evaluated system. This finding is opposite to that of a related study where researchers aimed to understand how ease of use and usefulness of software process innovations (i.e. Personal Software Process) related to practitioners' intentions to adopt the innovations. That study could not find any relationship between ease of use and intention to adopt the innovation, a result it explained by observing that the practitioners had been trained using the innovation and had had considerable practice in using it and could have thus been past the initial innovation use hurdle (Green et al., 2005). The opposite could be said for the study of the novices reported in this paper.

When asked in an open-ended question how participants thought the scheme could be made easier to use in the future, they answered that the definitions of the *Expected Phase* values should be explained more thoroughly and more examples be provided. When asked what their main reasons for deciding to use or not use the CUP scheme in the future, they said that they thought the scheme took too long to use. The analysts' concerns about scheme effectiveness concur with the study of (Green et al., 2005), which found a significant relationship between productivity from software process improvement innovation and its usefulness. Furthermore, the analysts said they would need more confidence in their own expertise to make a decision on intention to use the scheme. This relates to the finding of Mathieson et al. (2001), who extended TAM to include a perceived resources instrument, which includes resources available to the scheme analysts. Examples of such resources include help available during use, expertise and time. Their results showed that resources available to the analyst were related to intention to use and perceived ease of use, with a weak link to perceived usefulness. When there were no constraints on resources, ease of use and usefulness could be related to the intention to use, but in cases when there were constraints, the intention to use could better be explained by the extent of these resources than the ease of use or usefulness of the scheme.

5.3 Validity analysis

Observing developers empirically during user interface evaluation and maintenance, the following work activities emerged after uncovering usability problems: understanding the problem, prioritising the problem and redesigning the user interface to prevent the problem from appearing in future user interactions. This section addresses these activities in relation to CUP.

5.3.1VQ1: Developers' understanding of the UP

Understanding a problem includes getting information about the nature and origin of a usability problem (i.e. what is the problem, where does it occur, how much difficulty has it caused the user, why does the problem occur, and who has done the tasks leading to the problem). Neither of the two organisations involved had conducted a UT on the respective system before taking part in the present study; the number of UP found surprised both development teams. The attitude towards the UP was different, however. The LMS developers did not blame their users for the problems experienced but acknowledged that the system did not provide users with sufficient support for completing the tasks. The HS developers, however, felt that approximately a third of the UP was caused by a lack of knowledge or training on the part of the users. Evidence of such a view among developers can be found in other studies (Hoegh et al., 2006). The developers considered bugs and program crashes to be the most serious UP and errors caused by the user interface as not as pertinent. The LMS developers probably had a different attitude because they had not been dealing with serious functionality bugs, e.g. program crashing, which the HS developers had from the beginning. Another influential factor could be that the LMS developers were conscious of the fact that they had a large user group (their users had different information technology skill levels) and one of their goals was that unassisted users could carry out basic tasks in the system. The LMS users were also further away from the developers; they could not handle user problems by giving a short presentation like the HS developers did.

The LMS and HS developers agreed that CUP helped them understand the UP; they focused most on the *Description* and found the screenshots useful in all cases. The LMS developers used the *Trigger* attribute less than the HS developers; the reason for this might have been that, in the LMS UP, the *Description* and *Context* usually made the user's task obvious. In HS, however, the tasks were more complicated, e.g. how they were carried out depended on the type of data entered by the user.

When it came to understanding the UP, the HS developers did not look at CUP attributes other than *Description, Frequency, Trigger, Context* and *Severity*. The LMS developers found all attributes helpful, except *Severity* and *Task Efficiency*; they used the *Description*, *Frequency*, *Trigger,* and *Context* the most. The LMS developers did not consider *Severity* to be part of gaining an understanding of the UP because they thought it was included in the description.

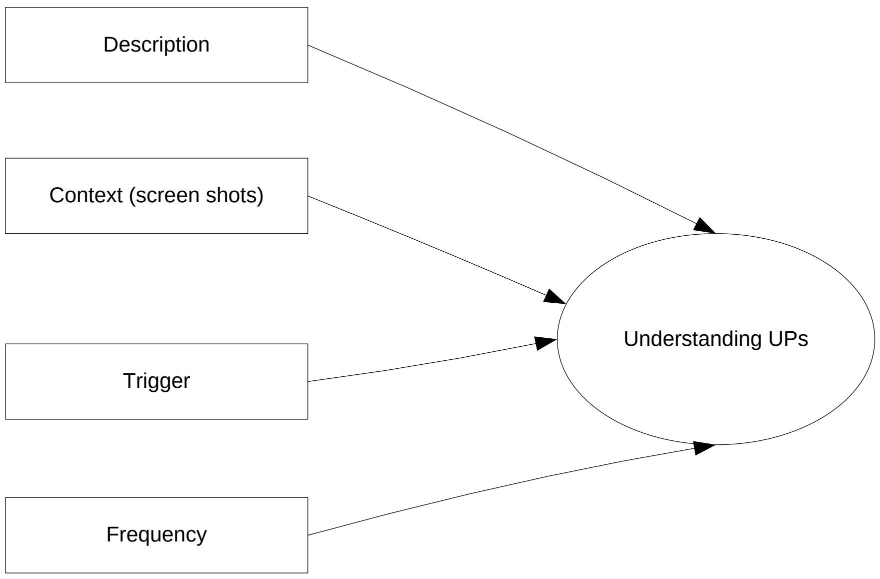

Figure 4. Attributes that the LMS and HS developers used most to understand the UP

Figure 4 shows what CUP attributes the developers used most when trying to understand a UP. *Description* was most important to them; they found the screenshots useful and that they saved time by looking at the system itself. The developers also used the *Trigger* attribute often because it provided them with information about what the user intended to do.

5.3.2 VQ2: CUP's effect on prioritising what UP to fix

Prioritisation is a method frequently addressed in software engineering in different activities. One is to prioritise *requirement selection,* i.e. what features and qualities the software product should include. When software products are *tested,* total test coverage is seldom affordable and test cases must therefore be prioritised. Next, when the problems have been discovered and reported, the development team must decide how the product can be *maintained*, i.e. what problems should be given the highest priorities to correct. Developers can prioritise usability problems according to severity and frequency. Another way to prioritise a usability problem is to look at a broader organisational view, thus considering the company, marketing and sales goals. Fadden and McQuaid’s (2003) view on prioritisation is to consider not only the users' opinions, but also the product team's opinion, i.e. the group of people making changes to the product. From the perspective of the product team, it may be desirable to fix the easy problems first. Mapping importance and difficulty may help us determine the return on investment, where importance will return revenues but difficulty will determine the cost. Greer et al. (1999) list five factors that determine prioritisation and conclude that their relative importance is adjustable: benefits, costs and risk exposure in the current and target systems as well as development process. In the HCI arena, Keenan et al. (1999) suggested ways in which problems might be prioritised based on their suggested Usability Problem Taxonomy (UPT) in conjunction with other factors, including severity, cost-to-fix and user satisfaction questionnaire. However, no formal studies have been made on the UPT’s contribution to problem prioritisation. Based on the literature, Table 8 shows the expected effects of CUP attributes on prioritisation.

Table 8. The expected effects of the CUP attributes on prioritisation

CUP attribute /	**Expected reasons for being useful for the prioritisation activity**	**Prioritisation of change**
Task completion	It is more important to fix problems for tasks which have a low success rate	Criticality
Failure qualifier	It may be less expensive to take something out than add something to the user interface. (e.g. extraneous vs. missing) Something that is incongruent to the user’s mental model may be more costly to correct.	Effort of implementation in man power
Expected phase	It is generally believed that problems originated in earlier phases are more costly to correct than in later phases	Effort of implementation in man power
Severity, Frequency	Problems with higher severity impact are more critical and it is more urgent to fix problems experienced by many	Criticality

All developers in our study agreed that *Frequency* and *Severity* had the most influence on prioritising what UP to fix. The LMS developers also mentioned the *Trigger* attribute because they considered some tasks less important. A low completion rate (*Task completion)* drew the attention of all developers; only the HS developers thought that *Failure Qualifier* could be useful in assessing UP seriousness. When planning for release increments, the developers commented on the *Context* attribute and felt that “If there are many programmers working on the system, the *Context* can be used to allocate UP among them.”

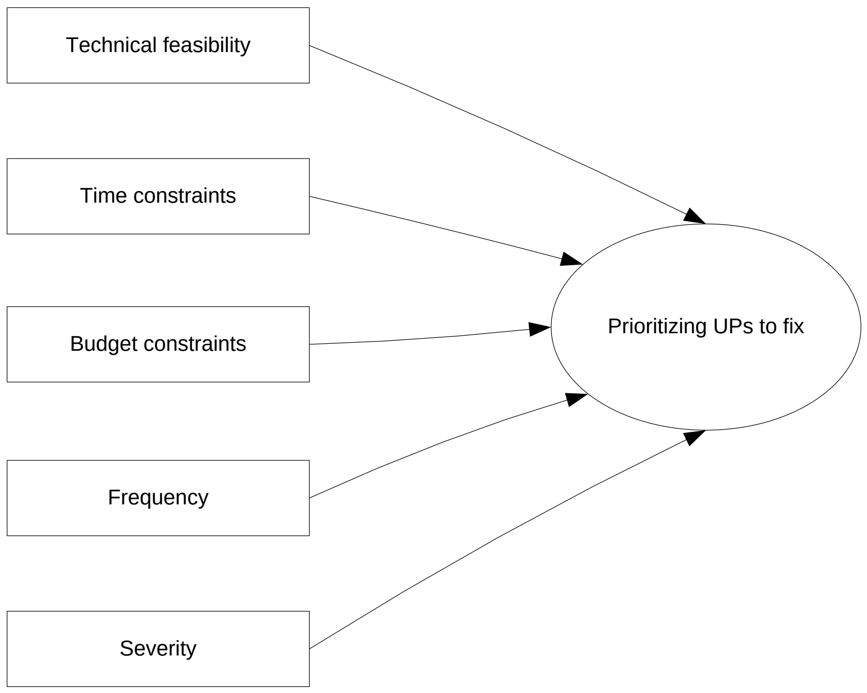

Figure 5. Influencing factors on prioritising what UP to fix in the LMS and HS field studies

Figure 5 shows these and other factors that also affect the prioritising task. At both organisations, deadline and budget constraints had a strong effect on the UP that would be fixed because there was a limit on how many person-hours could be spent on redesign. The LMS developers had a systematic way of evaluating the effort of a change: they would list the possible changes and estimate how long it would take in workdays to implement them. They based this evaluation on insight, experience and their system knowledge. They were concerned with meeting a deadline, which was the start of the university’s semester. The new LMS version had to be ready before the semester started, and they did not want to change the system in the middle of the semester. They spent all available resources on the LMS, and it was not possible to add another developer to work on it full-time.

The HS developers did not estimate the effort of a change as precisely as the LMS developers. They did not have to meet a deadline for a new version but had limited work-hours to spend on HS. In effect, the limited resources created the same pressure as a deadline. In their prioritising, the HS developers were concerned with the technical feasibility of a change to the system; they did not always appear to be confident in making changes to the code. They were maintaining a system that they did not develop, and might thus have been facing more difficult challenges. The developers were always worried that making a change in one place would have unforeseen effects. One developer said that: “When I started working here I didn’t think it would take so long to familiarise yourself with the code and the system. It has taken a long time, and it has been very difficult, much harder than I expected. I think maintenance can be one of the hardest tasks in software development.” The LMS developers did not appear to be having these problems. The organisation developed LMS from the start, and nothing seemed beyond their capabilities; it was just a matter of how long it would take them. What also affected the HS time schedule was that managers pressed the developers to work on another system. That work came up quite unexpectedly and led to less time spent on HS than originally planned.

It is intriguing to compare these results to those from the study by Boivie et al. (2003). The developers in that study said that deadlines controlled their work. "Whenever there was a priority conflict between usability and development resources, deadlines ruled - sometimes meaning that the teams had to give up on usability matters." (Boivie et al., 2003). This was also the case in the HS field study, where the developers had limited time to spend on correcting UP, and their priority was on correcting functional errors. They did not have the resources to address the root cause of some UP. The LMS study showed that the developer thought that the UP he fixed were due to a lack of development time. Other studies have reported similar findings; for example, in Hoegh et al. (2006), a project resource such as time was an overall topic throughout interviews with two developers while examining usability reports. A lack of time is common in software projects, according to several cost overrun analyses of software projects, where the average overrun was found to be between 33% and 41% (Moloekken-Oestvold et al., 2004). Further, as with the LMS development team, which had a hard delivery deadline, a tight schedule may affect the development process (Cerpa and Verner, 2009). In such circumstances, it may be difficult for developers to focus on usability, as they mostly think about getting the software functional.

5.3.3 VQ2: CUP's effect on system redesign

The third activity observed was system redesign. Indeed, redesign proposals associated with UP are deemed useful for improving the system under evaluation (Hornbæk and Frøkjær, 2005).

However, CUP lacks this redesign attribute based on two considerations. Presumably, values of the existing attributes can inspire or somewhat guide the development team to come up with redesign ideas. It is also desirable to keep scheme complexity to a manageable level to ensure its usability. Furthermore, because time is already a limited resource, the scheme should not suggest activities that may expand it. The first presumption seems corroborated by the chief developer's comment that, although CUP does not suggest correction to a UP, the *data* make the problem obvious: the data referred to are the values of the variables *Description, Failure Qualifier*, and *Expected Phase*. The conclusions of Hoegh et al. (2006) concur with the importance of the *Failure Qualifier*, emphasising that a usability problem list could be improved by giving the reason why a problem occurred, e.g. saying that "there is too much information or that a button is placed without a clear reference to its context" (Hoegh et al., 2006).

All developers focused on creating intuitively holistic solutions to a set of problems rather than identifying individual problem corrections. While Hornbæk (2010) remarks that most studies consider individual usability reports, he gives three reasons why individual usability reports should be supplemented, e.g. with feedback of a higher abstraction. One reason is that, as pointed out by Jeffries Jeffries (1994), a set of problems must be considered as a whole. Indeed, CUP could meet this need by allowing a developer to group problems together according to different categories, such as Failure Qualifier, Expected Phase or Context.

5.3.4 VQ3: Developers' understanding of the CUP attributes

The developers were asked to rate their understanding of the CUP attributes on a three-point scale: easy, moderate and hard. The LMS developers understood all attributes easily, except for *Expected phase*, *Task Efficiency* and *Cause*. The developers' view that the attribute *Expected phase* was hard to understand concurred with the outcome of the inter-evaluator agreement in the reliability study. The developers felt that it had too many values, some definitions were too technical and they found it tough to distinguish between the values. The LMS developers found the attribute *Cause* hard to understand and thought it hard not to take the *Cause* classification personally. They did not like being tied to some defined values that were perceived as unfit to the faults in their process. They also found the *Task Efficiency* hard to grasp because it was difficult to benchmark it against another value and did not have a specified range. On *Task Effectiveness,* the developers found the completion rate useful, but not the mean time-on-task. Again, this was because there was no baseline against which it could be compared.

It is interesting that the developers relied most on the *Description* attribute. However, the classifying parameters could be more useful when the *Description* quality was low or the usability problem was complex.

Based on the LMS field study and reliability analysis, some changes were made to CUP to address the most salient issues before the HS study commenced (Table 2). To improve the attribute *Expected Phase,* we took the approach of simplifying the values' structure by organising them hierarchically into three development stages: Requirement analysis, Design and Implementation. Three sub-phases belonging to the Requirement analysis are Task analysis and context of use, Functional requirements, and Quality attribute analysis. Design has four sub-phases: Conceptual modelling, Dialogue design, Navigational design and Presentation design. By structuring the values hierarchically, we hoped that rating the UP would be more accessible: the analyst first selects one of the three stages and can then focus on the sub-phases. It was not considered viable to merge values to reduce their number lest the classification be less accurate. The descriptive definitions of the *Expected Phase* values were simplified to make them more understandable. The *Cause* attribute was made free text, so the developers could describe the cause in their own words. The attribute aimed to make the developers think about the underlying cause of the problem, and it should be equally achievable with free text.

The HS developers found CUP slightly hard to grasp at first because there were many attributes to consider and they had not previously been involved with a UT. However, after they got started, the developers easily understood all Pre-Fix attributes, except for *Expected Phase* and *Task Efficiency*. They also found Failure Qualifier slightly challenging to understand and commented that the value *Incongruent Mental Model* was a bit technical. The developers did not find *Task Efficiency* a useful attribute, as they found it hard to understand because there was no baseline with which to compare. The developers said that they found *Expected Phase* hard to distinguish between its values and thought they were intertwined. One developer commented, “I think it overanalyses things and it’s too complicated and hard to learn.” They only related the main categories: Requirement Analysis, Design and Implementation to their own software development process.

When it came to the Post-Fix attributes, the HS developers found *Actual Phase* to be the most complex. One developer said that: “I had to look at the definitions many times because there are so many values. But *Actual Phase* tells you how big the fault was depending on where it was fixed.” The developers did not experience any problems with the rest of the Post-Fix attributes and found it useful to fill them out. The attribute *Cause* helped them think about the problems, and they thought it would help them in future projects. One developer said the following about *Cause*: “It is very useful to fill this out because it makes you think of what has gone wrong. Although it is sometimes hard since we have not worked on HS from the beginning.” This is a compelling point because the developers had only been maintaining HS for 18 months and thus did not know what basic design decisions had been made prior to their involvement. The developers did not have any documentation about HS development. When they were filling out *Cause*, it was hard for them to find the root cause of the fault; this could therefore explain why some of their answers specified what in the user interface caused the UP but not what in the software process caused it, i.e., what the developer did wrong.

When the *Cause* attribute is free text, it is possible to collect data on what causes UP in systems. It is appealing to collect this data for more than one system or usability evaluation of the same system, because we can then learn more about what developers think about what causes the UP. As more such data are gathered, it is even possible to develop patterns of UP root causes.

5.4 Implications for the Improvements of CUP

The developers did not suggest any significant changes to the CUP scheme, except to the *Expected* and *Actual Phase* attributes. As discussed above, the *Cause* attribute was changed to free text after the first study, and it worked better in the second study. Though the developers were not using all CUP data at once and considered one attribute more useful than another, they still wanted all the data. They focused on what could be most useful to them at some point in time. Hornbæk and Frøkjær (2005) had the same result: the developers all felt that they wanted to receive the problem descriptions and redesign proposals.

Based on what the developers thought about the *Expected Phase* values, it appeared that they might not be of any real use, even if values were simplified. The values as currently defined seem inapplicable to the development process employed at the participating organisations. A further detailed study of the reasons for this would be required. One reason could be that there was no explicit documentation to support the processes, and processes were implicit with tacit knowledge of them, which is the current trend, as a software process study has revealed (Coleman and O'Connor, 2008). In developing a defect classification scheme, it is imperative to consider that many development teams do not use user interface lifecycles at all (Gulliksen et al., 2004). Instead of presenting the engineer with a predefined lifecycle for user interface development for the *Expected Phase*, an assessment of an individual team's current practice can thus identify a lifecycle model that can then be later used to reveal root causes of a usability problem. This harmonises well with previous findings, which conclude that companies tailor standard software processes to their own contexts depending on the size of the company, market and project and system type (Coleman and O'Connor, 2008; Green et al., 2005). The obvious advantage of this approach is that developers are familiar with the *Expected Phase* attribute values, but the disadvantage is that it may not give enough information about root causes if the lifecycle is too plain.

Despite dissatisfaction with some variables, including *Expected phase*, the developers found that CUP increased their UP understanding and its attributes helped them when prioritising what UP should be fixed. The aim of the field studies was not only to study how the developers used CUP, but to come up with ways to improve it. The aim was achieved to a certain extent. The changes that were made to CUP after the first field study probably enhanced the developers' use and understanding of CUP. One HS developer made the following comment: "I think that CUP is very useful when working on a new version or improvements on a system. The reason is that it makes you work in a more disciplined and precise manner because the data are so precise. Comments that we get from users in ProblemTracker are usually too vague and hard to understand. CUP saves a lot of time and effort, makes the work more focused because knowing what the problem is makes it easier to solve."

Besides improving the definition of individual attributes of the classification scheme, the included and excluded attributes must be decided. Linking the attributes to certain work activities can lead to a more targeted and practical classification scheme. The present paper has presented such links by suggesting two theories on what influences developers' understanding and prioritisation of a UP.

Research on how problem prioritisation is done in practice is scarce, but such studies could help discover whether that process has any implication on usability evaluation methods (Hornbæk, 2010). In response to the findings of this study, i.e. that developers decide which defects to correct and their redesign is based on technical difficulty, budget constraints and holistic solutions, Table 9 suggests new variables considered useful for prioritisation. The first column proposes new CUP attributes, the second column justifies an attribute's existence and the third column links the attribute to the other attributes supposed to determine prioritisation, according to the literature. The first two new attributes relate to the technical difficulty, the third new attribute relates to a holistic solution and the final two new attributes would help users assess whether the corrections are worth their time (return on investment).

Table 9. Proposed new attributes influencing prioritisation

Proposed new CUP attributes	**Expected reason for being useful for prioritisation activity**	**Prioritisation of change**	**Genre**
Span of solution	The solution may induce more risk if a large part of the application is being changed at one time	Risk of implementation (risk factor, impact, countermeasures/controls)	Technical difficulty
Complexity of solution	The solution may induce more risk if it is complex	Risk of implementation (risk factor, impact, countermeasures/controls)	
Groups or an abstraction of problems	Instead of looking at individual problems, strive to find a group of problems for which there exists a solution module. The module could be a node in a dependency graph.	Dependency between requirement changes	Holistic
Sacrifice cost	Revenues lost by owner/buyer/user because of this problem and future loss if not corrected	Revenues, returns	Return on investment
Gained revenues if problem is corrected	Correction of problem leading to increased sales	Revenues, return on investment	

5.5. Downstream Utility of CUP

The main goal of CUP is to provide a development team with more concrete information about system UP, thereby maximising the usability evaluation outcome exploitation to improve the system. Put differently, CUP aims to facilitate interplay between usability evaluation and system redesign. Can CUP achieve this?

Hornbæk and Stage (2006) identify four challenges for improving the interplay. We call them Evaluation Artefacts, Evaluation Goals, Evaluation Products, and Evaluation Insights. Put briefly, artefacts to be evaluated should include not only low- or high-fidelity prototypes but also design products, including personas and scenarios. Evaluation goals should address the real needs of the development team as well as values to be attained by applying interest. Common feedback mechanisms include written usability reports and UT observations, but richer and more diverse evaluation products are required. Finally, redesign insights can be gained through support for analysing problems, prioritising them and recommending solutions. CUP can be seen as an alternative Evaluation Product, offering rich contextual and analytical information about UP. CUP may also be regarded as discount Evaluation Insights, providing support for understanding and prioritising UP but lacking recommendations; this graceful degradation is meant to ensure scheme usability.

Nevertheless, the downstream utility of CUP is hard to demonstrate with the two field studies conducted. Law (2006) defines “downstream utility” as *the effectiveness with which the resolution to a usability problem is implemented”*. The challenge is that there are so many confounding variables in a real software development environment that one can never conclusively attribute the improved system redesigns to any specific CUP features. For LMS, their own list of issues might have contributed even more to system improvement than the CUP list, but we cannot isolate the respective impacts. In fact, given that no systematic usability evaluation has yet been performed on the new LMS release, there is no way for us to tell whether the changes triggered by the two lists are effective or not. With the major concern about ecological validity, it is undesirable to perform any lab-based experiment to manipulate interesting variables (e.g. one group of developers is provided with CUP results, whereas another group is provided with a conventional usability report). Alternatively, we adopted action research methodology, by engaging practitioners and researchers in collaboratively identifying limits of and improvement strategies for CUP through an informal, qualitative, reflective and interpretive approach.

6. CONCLUSION AND FUTURE WORK

We have evaluated the reliability, validity and acceptability of the CUP scheme by conducting comparative experiments, qualitative studies and questionnaires. We believe that the contribution of the paper is twofold; one for HCI developers and evaluators and a second for researchers of work practices and tools. First, the study results serve as valuable insights into improving a maturing usability tool, which can strengthen the feedback between evaluation and redesign. Second, the study provides a methodological framework for assessing reliability, validity and acceptance of a tool under development. Analysing reliability is useful for improving the scheme. CUP reliability results indicated that analysts' expertise and experience are critical factors or determinants for applying this tool consistently and effectively. An expert CUP user could analyse the given UP differently than a novice CUP user. The former can analyse more accurately than the latter. Our results show that systematic training on deploying CUP is imperative. Similar results were shown in Henningsson and Wohlin (2004, 2005), but previous studies are not unanimous on this. For example, Chattratichart and Lindgaard (2008) have found no difference between the novices and experts and Capra (2006) found that for some variables there was a difference between the groups but not always. Research studies examining variations of ODC and studying novices and experts have discussed the reason for differences in repeatability of the two groups. Kelly and Shepard (2001) stated that lack of training and inexperience accounted for the largest contributions to the variations between students and professionals. Besides considering prior experiences of the evaluators, previous research has examined if familiarity or experience with the artefacts in question affects the agreement level. Henningsson and Wohlin (2005) tried to analyse whether classifiers’ experiences with the software components under inspection affected their agreement levels, but discovered that no such relationship could be found. Another way to gain experience with the artefact is to ask classifiers to work with the software during classification. In a study by Kelly and Shepard (2001), the task-directed software inspection included tasks such as finding a data dictionary while reviewing the code. To increase their knowledge about classifiers’ level of understanding and to examine what level of understanding was required by the task, Kelly and Shepard (2001) described each defect with four levels, requiring increasing understanding: C (Comparison of code to user documentation), I (Naming of variables and modules versus use), S

(Semantic or logical structure of data, active code, and modules), and L (Calculation/logic and error conditions). In a second experiment, participants in the experiments were asked to describe the defects with these four levels after completing the regular ODC-CC experiment. The conclusion from this exercise was that there was some evidence that tasks will help participants to find more L level defects, i.e. the kind of defects which require the most understanding.

The inference for future work is to identify effective training strategies to enhance novice analysts' understanding. A comparison of two equal sized groups of novices and experts would hopefully lead to a more valid study and further understanding of how the progression from novice to expert occurs. In addition, we can further break down analysts' expertise and experiences into three variables, namely, the actual involvement in conducting the usability evaluation method of interest, the domain knowledge of the system under investigation, and CUP scheme expertise. Indeed, research into expertise development in product design has shown that novices and experts differ substantially in the strategies and domain-specific and experiential knowledge employed. Using this framework, it has been shown that novices can only handle small steps, small 'chunks' of information, and need concrete phenomena to make many assumptions, but experts can handle large 'chunks', can abstract more and do not need to make many assumptions (Popovic, 2004). The difference in behaviour between experts and novices in classifying phenomena, objects or events has been attributed to their ability to understand abstraction (Snyder, 2000). For example, novices are thought to understand phenomena that are at a lower level than those that are at a higher level of abstraction and include many sub-categories. As an example, the attribute *Actual phase* is a higher level category than *Severity*. The value 'requirements analysis' (of *Actual phase*) potentially includes a number of sub-categories and many more than the value 'severe' (of *Severity*). Alexander (2012) discusses at length the subjectivity vs. objectivity of a taxonomy, comparing it to a scientific endeavour. Instead of focusing on objectivity, an alternative goal which is especially eminent in user interface design, is to focus on usefulness rather than truthfulness. Second, Alexander summarises the literature by stating that the issue of subjectivity vs. objectivity is presented as a contrast between tailorability to a specific viewpoint vs. accessibility to all. In practice, Alexander (2012) notes that there will always be a balance between making things more useable to a specific community or a target user group vs. accessible to all, or about a balance between specialisation and generalisation (Lambe, 2007). This can be a motivation for further studies into the systematic investigation of the variables' effect on CUP analysis results.

The validity analysis helped us understand whether the CUP attributes are relevant for developers' work. The study showed, interestingly, that developers focused more on creating intuitively holistic solutions to a set of problems rather than identifying individual problem solutions. This may suggest that grouping the problems or their abstraction may indeed be useful. The study also showed that project management is of immense concern and some attributes supported that task well. The developers' unfamiliarity with using the results of systematic tools, including UTs, applying problem reviews and problem tracking, might have affected the study results. The results of the *Expected Phase* attribute convinced us that tools used by developers must be tailored to their working framework, knowledge and maturity. More fine-grained attributes, such as *Failure Qualifier* and *Expected Phase*, can give richer data and provide more insight into plausible UP origins, but they may demand more expertise or initial training effort.

The results of the acceptability study showed that practitioners were concerned with the effort spent in applying any tool. However, the preliminary data on cost-effectiveness of CUP collected in this study indicated that CUP was not time-consuming and did not delay deadlines or the releases of either the LMS or HS software. Nonetheless, further data are needed to assess the direct benefit in the effort saved.

A triangulation study, like the one described here, with experiments, qualitative work in the field, and a questionnaire are all helpful in assessing and improving an intervention suggested for user interface developers. Our experience in this study shows that such an intervention can not only help us evaluate the scheme in question, but can also tell us a great deal about the current practices and constraints of developers' work.

What distinguishes this work from previous work on classification schemes is that it recognises the importance of relating such a scheme to primary work activities, the team's maturity, development constraints and the application being developed.

Future work will study work activities, especially process lifecycles, to understand better the usefulness of the *Expected Phase* variable. The results of the study motivate questions on whether there are links between process maturity and classification scheme usefulness. For example, a problem management activity taxonomy has been proposed as a part of the Corrective Maintenance maturity Model (CM^3) (Kajko-Mattsson, 2006). As the suitability of development methods may depend on applications or domains, it may be worthwhile to investigate that dependency. In particular, the comparison between domains or types of application will be done with respect to individual parameters of the scheme. How to go about tailoring such a classification scheme to individual teams and corporations remains an open question.

While the main emphasis has been on Pre-Fix variables, a more longitudinal study will give us enhanced data for a study on Post-Fix variables. With data from two different application developments, it will also be useful to study the distribution of values for individual parameters, including *Failure Qualifier* and *Expected Phase*. Finally, the methodology of developing such a classification scheme will be an interesting question to tackle. It may not be a none-or-all decision between a quantitative (i.e. pre-defined values) and a qualitative approach (i.e. free text); identifying the right balance by integrating both into the scheme is more challenging and relevant.

As this research study has shown, explaining usability problems in a language understandable to developers in an effective manner is a challenging task. Many factors are involved, including organisational motivation, development context, current practices and skills. Achieving high reliability and validity is thus demanding. In many ways, we are caught between current practices and desired outcomes. While we do want to fit the CUP classification scheme to the current context, we aim to pull practitioners to higher process standards. Process improvement is thus not only about designing innovative schemes, but also providing appropriate training (Cockton, 2006) to maximise the impact on products and processes.

The conclusion is that while such classifications may be proposed, they will be adapted by practitioners, and, as experience from ODC has shown, they will evolve as time passes. Such classifications need to be open and responsive to criticism, publicly accessible, for example, through standards, and maintain an equality of the intellectual authority of the contributors, as referred to in Alexander (2012). Thus, the aim of this paper is to put forward such a classification scheme and by evaluating its reliability, validity and acceptability, point out its strengths and weaknesses which can be a basis for further adaption and evolution.

When implementing the field study, we tackled this challenge with a well-defined evaluation plan and intensive meetings with the development team in order to teach them the details of implementing and understanding the potential results obtainable from using CUP. However, what we lack is methodologically sound measures and instruments to demonstrate the actual impact of utilising CUP. It is not only a matter of the interplay between problem discovery and redesign or problem explanation, but also the interplay between qualitative and quantitative research methods (Hughes, 1999). The HCI and software engineering communities should strive to develop innovative research tools to address these interplays.

REFERENCES

Abrahamsson, P., Kyllonen, P., 2005. Efficient Development of Mobile Software Applications. VTT Technical Research Centre of Finland.
Alexander, F., 2012. Assessing information taxonomies using epistemology and the sociology of science. Journal of Documentation 68, 7.
Altman, D., 1991. Practical Statistics for Medical Research. Chapman and Hall.
Anderson, N.H., 1996. A Functional Theory of Cognition Lawrence Erlbaum Associates New Jersey, USA.
Andre, T.S., Hartson, H.R., Belz, S.M., McCreary, F.A., 2001. The user action framework: A reliable foundation for usability engineering support tools. International Journal of Human-Computer Studies 54, 107-136.
Andre, T.S., Hartson, H.R., Williges, R.C., 2003. Determining the Effectiveness of the Usability Problem Inspector: A Theory-Based Model and Tool for Finding Usability Problems. Human Factors: The Journal of the Human Factors and Ergonomics Society 45, 455-482.
Avison, D.E., Lau, F., Myers, M.D., Nielsen, P.A., 1999. Action research. Communication of the ACM 42, 94-97.
Baddoo, N., Hall, T., 2002. Motivators of Software Process Improvement: An analysis of practitioners' views. Journal of Systems and Software 62, 85-96.
Bark, I., Følstad, A., Gulliksen, J., 2005. Evaluating user-centred methods: Results from a survey study among Nordic HCI-practitioners. SINTEF Trondheim, pp. 1-25.
Bassin, K., Biyani, S., Santhanam, P., 2002. Metrics to evaluate vendor-developed software based on test case execution results IBM Systems Journal 41, 13-30.
Bhandari, I., Halliday, M., Tarver, E., Brown, D., Chaar, J., Chillarege, R., 1993. A case study of software process improvement during development. Software Engineering, IEEE Transactions on 19, 1157-1170.
Boivie, I., Åborg, C., Persson, J., Löfberg, M., 2003. Why usability gets lost or usability in in-house software development. Interacting with Computers 15, 623–639.
Braa, K., Vidgen, R., 1999. Interpretation, intervention, and reduction in the organizational laboratory: A framework for in-context information system research. Accounting, Management and Information Technologies 9, 25-47.
Bruun, A., Stage, J., 2012. Training software development practitioners in usability testing: an assessment acceptance and prioritization, Proceedings of the 24th Australian Computer-Human Interaction Conference. ACM, Melbourne, Australia, pp. 52-60.
Butcher, M., Munro, H., Kratschmer, T., 2002. Improving software testing via ODC: Three case studies. IBM Systems Journal 41, 31-44.
Capra, M.G., 2006. Usability Problem Description and the Evaluator Effect in Usability Testing, Industrial and Systems Engineering Virginia Polytechnic Institute and State University, Virginia.
Card, D., 1998. Learning from our mistakes with Defect Cause Analysis. IEEE Software Jan/Feb, 56-63.
Cerpa, N., Verner, J.M., 2009. Why did your project fail? Communications of the ACM 52, 130-134.
Chattratichart, J., Lindgaard, G., 2008. A comparative evaluation of heuristic-based usability inspection methods, CHI '08 extended abstracts on Human factors in computing systems. ACM, Florence, Italy.

Chillarege, R., Bhandari, I.S., Chaar, J.K., Halliday, M.J., Moebus, D.S., Ray, B.K., Wong, M., 1992. Orthogonal Defect Classification – A Concept for In-Process Measurements. IEEE Transactions on Software Engineering 18, 943-956.
Coleman, G., O'Connor, R., 2008. Investigating software process in practice: A grounded theory perspective. Journal of Systems and Software 81, 772-784.
Constantine, L.L., Lockwood, L.A.D., 1999. Software for use. Addison-Wesley Massachusetts.
Control Chaos, 2006. What is Scrum?
Curtis, B., 1984. Fifteen years of psychology in software engineering: Individual differences and cognitive science, Proceedings of the 7th international conference on Software engineering. IEEE Press, Orlando, Florida, United States.
De Jong, M., Schellens, P.J., 2000. Toward a document evaluation methodology: What does research tell us about validity and reliability of evaluation methods? IEEE Trans. Prof. Commun. 43, 242-260.
Dumas, J.S., 1989. Stimulating change through usability testing SIGCHI Bulletin, pp. 37-44.
Dumas, J.S., 2002. User-based evaluations in: Jacko, J.A., Sears, A. (Eds.), The Human-Computer Interaction Handbook:, pp. 1093-1117.
Dyba, T., 2005. An empirical investigation of the key factors for success in software process improvement. Ieee Transactions on Software Engineering 31, 410-424.
Dybå, T., Dingsøyr, T., 2008. Empirical studies of agile software development: A systematic review. Information and Software Technology 50, 833-859.
El Emam, K., 1999. Benchmarking Kappa: Interrater Agreement in Software Process Assessments. Empirical Software Engineering 4, 113-133
Embrey, D., 1995. Cognitive and conceptual errors, Role of the Operator in the Safety of the Nuclear Industry, IEE Colloquium on, pp. 6/1-617.
Endres, A., 1975. An analysis of errors and their causes in system programs, Proceedings of the international conference on Reliable software. ACM, Los Angeles, California.
Fadden, S., McQuaid, H., 2003. Fixing what matters: Accounting for organizational priorities when communicating usability problems, Usability Professionals' Association 2003 Conference Proceedings.
Freimut, B., 2001. Developing and using defect classification schemes, IESE-Report No.072.01/E Version 1.0. Fraunhofer.
Ghazarian, A., 2009. A Case Study of Defect Introduction Mechanisms, in: Eck, P., Gordijn, J., Wieringa, R. (Eds.), Advanced Information Systems Engineering. Springer Berlin Heidelberg, pp. 156-170.
Grady, R., 1992. Practical Software Metrics for Project Management and Process Improvement. Prentice Hall.
Green, G.C., Hevner, A.R., Webb Collins, R., 2005. The impacts of quality and productivity perceptions on the use of software process improvement innovations. Information and Software Technology 47, 543-553.
Green, T., 2006. Aims, achievements, agenda--where CDs stand now. Journal of Visual Languages & Computing 17, 288-291.
Greer, D., Bustard, D.W., Sunazuka, T., 1999. Prioritisation of System Changes using Cost-Benefit and Risk Assessments, Requirements Engineering, 1999. Proceedings. IEEE International Symposium on. IEEE, Limerick.
Gulliksen, J., Boivie, I., Persson, J., Hektor, A., Herulf, L., 2004. Making a difference - a survey of the usability profession in Sweden NordiCHI. ACM, Tampere, Finland.

Gupta, A., Li, J.Y., Conradi, R., Ronneberg, H., Landre, E., 2009. A case study comparing defect profiles of a reused framework and of applications reusing it. Empirical Software Engineering 14, 227-255.
Hamill, M., Goseva-Popstojanova, K., 2009. Common Trends in Software Fault and Failure Data. Ieee Transactions on Software Engineering 35, 484-496.
Hannay, J.E., Sjoberg, D.I.K., Dyba, T., 2007. A Systematic Review of Theory Use in Software Engineering Experiments. Software Engineering, IEEE Transactions on 33, 87-107.
Hassenzahl, M., 2000. Prioritizing usability problems: data-driven and judgement-driven severity estimates. Behaviour & Information Technology 19, 29-42.
Henningsson, K., Wohlin, C., 2004. Assuring Fault Classification Agreement - An Emprical Evaluation IEEE Conference Proceedings International Symposium on Empirical Software Engineering Redondo Beach, California, USA, pp. 95-104.
Henningsson, K., Wohlin, C., 2005. Monitoring Fault Classification Agreement in an Industrial Context, Conference on Empirical Assessment in Software Engineering, Keele, UK.
Hertzum, M., 2006. Problem Prioritization in Usability Evaluation: From severity Assessments Toward Impact on Design. International Journal of Human-Computer Interaction 21, 125-146.
Hertzum, M., Jacobsen, N.E., 2001. The Evaluator Effect: A Chilling Fact About Usability Evaluation Methods. International Journal of Human-Computer Interaction 13, 421-443.
Hoegh, R.T., Nielsen, C.M., Overgaard, M., Pedersen, M.B., 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, 173-196.
Hollnagel, E., 1993. Human reliability analysis: Context and Control. Academic, London.
Hornbæk, K., 2010. Dogmas in the assessment of usability evaluation methods. Behaviour & Information Technology 29, 97-111.
Hornbæk, K., Frøkjær, E., 2005. Comparing Usability Problems and Redesign Proposals as Input to Practical Systems, CHI. ACM, Portland, Oregon, USA, pp. 391-399.
Hornbæk, K., Stage, J., 2006. The Interplay Between Usability Evaluation and User Interaction Design. International Journal of Human-Computer Interaction 21, 117-123.
Hudson, W., 2001. Toward Unified Models in User-Centered and Object-Oriented Design in: van Harmelen, M. (Ed.), Object Modeling and User Interface Design Addison-Wesley New Jersey, pp. 313-362.
Hughes, M., 1999. Rigor in usability testing Technical communication 46, 488-576.
Hvannberg, E.T., Law, E.L.-C., 2003. Classification of Usability Problems (CUP) Scheme, in: Rauterberg, M., Menozzi, M., Wesson, J. (Eds.), Interact, Switzerland, pp. 655-662.
IEEE, 1983. IEEE 729-1983, Glossary of Software Engineering Terminologies.
Jeffries, R., 1994. Usability problem reports: Helping evaluators communicate effectively with developers, Usability inspection methods. John Wiley & Sons, Inc., pp. 273-294.
John, B.E., Marks, S.J., 1997. Tracking the effectiveness of usability evaluation methods. Behaviour & Information Technology 16, 188-202.

Kajko-Mattsson, M., 2006. Evaluation of CM^3: Front-End Problem Management within Industry, Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on, pp. 367-368.
Keenan, S.L., Hartson, H.R., Kafura, D.G., Schulman, R.S., 1999. The Usability Problem Taxonomy: A Framework for Classification and Analysis. Empirical Software Engineering 4, 71-104.
Kelly, D., Shepard, T., 2001. A case study in the use of defect classification in inspections, Proceedings of the 2001 conference of the Centre for Advanced Studies on Collaborative research. IBM Press, Toronto, Ontario, Canada, p. 7.
Khajouei, R., Hasman, A., Jaspers, M.W.M., 2011a. Determination of the effectiveness of two methods for usability evaluation using a CPOE medication ordering system. International Journal of Medical Informatics 80, 341-350.
Khajouei, R., Peute, L.W.P., Hasman, A., Jaspers, M.W.M., 2011b. Classification and prioritization of usability problems using an augmented classification scheme. Journal of Biomedical Informatics 44, 948-957.
King, J., E., 2004a. Microsoft Excel templates for calculating a generalized kappa statistic.
King, J., E., 2004b. Software solutions for obtaining a kappa-type statistic for use with multiple raters, Paper presented at the annual meeting of the Southwest Educational Research Association, Dallas, TX.
Lambe, P., 2007. Organising Knowledge: Taxonomies. Knowledge and Organisational Effectiveness. Chandos Publishing Ltd., Oxford
Lavery, D., Cockton, G., Atkins, M.P., 1997. Comparison of evaluation methods using structured usability problem reports. Behaviour & Information Technology 4/5, 246-266.
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, 147-172.
Law, L.-C., Hvannberg, E.T., 2002. Complementarity and Convergence of Heuristic Evaluation and Usability Test: A Case Study of UNIVERSAL Brokerage Platform NordiCHI, Aarhus, Denmark, pp. 71-80.
Leszak, M., Perry, D.E., Stoll, D., 2000. A case study in root cause defect analysis, Proc. of ICSE Limerick, Ireland, pp. 428-437.
Leszak, M., Perry, D.E., Stoll, D., 2002. Classification and evaluation of defects in a project retrospective. Journal of Systems and Software 61, 173-187.
Lutz, R.R., Mikulski, I.C., 2004. Empirical analysis of safety-critical anomalies during operations. Ieee Transactions on Software Engineering 30, 172-180.
Mathieson, K., Peacock, E., Chin, W.W., 2001. Extending the technology acceptance model: the influence of perceived user resources. SIGMIS Database 32, 86-112.
Mays, R.G., Jones, C.L., Holloway, G.J., Studinski, D.P., 1990. Experiences with defect prevention. IBM Systems Journal 29, 4-32.
Mentis, H., Gay, G., 2003. User recalled occurrences of usability errors: implications on the user experience, CHI '03 extended abstracts on Human factors in computing systems. ACM, Ft. Lauderdale, Florida, USA.
Moloekken-Oestvold, K., Joergensen, M., Tanilkan, S.S., Gallis, H., Lien, A.C., Hove, S.W., 2004. A survey on software estimation in the Norwegian industry, Software Metrics, 2004. Proceedings. 10th International Symposium on, pp. 208-219.
Moody, D.L., Sindre, G., Brasethvik, T., Sølvberg, A., 2003. Evaluating the quality of information models: empirical testing of a conceptual model quality framework

Proceedings of the 25th International Conference on Software Engineering IEEE Computer Society, Portland, Oregon pp. 295-305.
NetResults Corporation, 2005. ProblemTracker
Norman, D., 1988. The Psychology of Every Day Things. Basic Books, New York.
Patel, V.L., Kaufman, D.R., Arocha, J.F., 2002. Emerging paradigms of cognition in medical decision-making. Journal of Biomedical Informatics 35, 52-75.
Paternò, F., Santoro, C., 2002. Preventing user errors by systematic analysis of deviations from the system task model. International Journal of Human-Computer Studies 56, 225-245.
Popovic, V., 2004. Expertise development in product design--strategic and domain-specific knowledge connections. Design Studies 25, 527-545.
Rasmussen, J., 1986. Information processing and human computer interaction: An approach to cognitive engineering. North-Holland, New York.
Reason, 1990. Human error. Cambridge University Press, Cambridge, England.
Shenvi, A.A., 2009. Defect prevention with orthogonal defect classification, Proceedings of the 2nd India software engineering conference. ACM, Pune, India.
Snyder, J.L., 2000. An investigation of the knowledge structures of experts, intermediates and novices in physics. International Journal of Science Education 22, 979-992.
Stochel, M., 2011. Reliability of feedback fechanism based on root cause defect analysis - case study. Annales UMCS, Informatica 11, 21-32.
Sutcliffe, A., Rugg, G., 1998. A Taxonomy of Error Types for Failure Analysis and Risk Assessment. International Journal of Human-Computer Interaction 10, 381-405.
van Harmelen, M., 2001. Designing with Idiom, Object Modeling and User Interface Design Addison-Wesley New Jersey, pp. 71-113.
Walia, G.S., Carver, J.C., 2009. A systematic literature review to identify and classify software requirement errors. Information and Software Technology 51, 1087-1109.
Weiss, D.M., Basili, V.R., 1985. Evaluating Software Development by Analysis of Changes: Some Data from the Software Engineering Laboratory. Software Engineering, IEEE Transactions on SE-11, 157-168.
Wellington, C.A., Briggs, T., Girard, C.D., 2005. Comparison of student experiences with plan-driven and agile methodologies, Frontiers in Education, 2005. FIE '05. Proceedings 35th Annual Conference, pp. T3G-18.
Wenger, M.J., Spyridakis, J.H., 1989. The relevance of reliability and validity to usability testing. Professional Communication, IEEE Transactions on 32, 265-271.
Wixon, D., 2003. Evaluating Usability Methods, why the current literature fails the practitioner, Interactions, pp. 29-34.