Improving Reporting of Usability Defects in Open Source Software Projects

by

Nor Shahida Mohamad Yusop

A Thesis Submitted for the Degree of
Doctor of Philosophy
at Faculty of Science, Engineering and Technology

Swinburne University of Technology
John Street, Hawthorn – 3122
Australia

2018

Dedication

I dedicate this thesis to

My late father

Mohamad Yusop Zakaria, who always believing in me and made this entire journey possible.

My mother

Che Su Hashim, whose affection, love, encouragement and prays of day and night make me able to get such success and honor.

This thesis is for you.

Abstract

Although reporting usability defects seems straightforward, usability defects are more difficult to validate and report than functional defects. This is because usability aspects involve a user's feelings and emotions, and "struggling with the software" are not objectively measurable and solely depend on human judgment. In the context of open source software (OSS) development, where usability design and qualitative assessment are handled by both technically savvy and non-savvy users with varying levels of knowledge and skills, reporting effective and useful usability defect descriptions becomes a challenge. Furthermore, current defect reporting tools such as Bugzilla that use a generic unstructured textual form to collect different types of defects make it impractical to produce high quality usability defect reports. In particular, the information requested in a generic form is often not contextually relevant and ignores the need for details pertinent to a defect type.

This thesis extends the body of knowledge in software defect reporting with a focus on improving the reporting of usability defects in OSS projects. The thesis presents (i) a systematic literature review that highlights a number of limitations in existing defect report formats – mixed data, inconsistency terms and values of usability defect data, and insufficient attributes to describe usability defects; (ii) factors that influence the quality of the defect reports and the information needs for reporting usability defects identified using a set of survey instruments; (iii) a revised usability defect classification model that only considers information available in the context of OSS development; and (iv) an in-depth investigation of the structure and content of existing defect report forms and a new design for a usability defect reporting form that is informed by the survey findings.

The revised open source usability defect classification model allows a better understanding of the trend of different types of usability defects, the impact of the problem on a user, and the nature of the failure qualifier of the problem. The new defect reporting forms and associated recommendations capture content adaptively and integrate better usability/ human-computer interaction principles. In particular, our findings if implemented can benefit less technical users by allowing the capture of contextualized usability-related attributes.

Acknowledgements

First and foremost I would like to express my hearty thanks to my supervisors – Associate Professor Jean-Guy Schneider, Professor John Grundy, and Professor Rajesh Vasa for their help, guidance, encouragement, support, patience, and dedication during the last four years. From them I learned to stay positive, be confident, believe in myself, be humble, and always appreciate others. You are my inspiration to become a better academician!

My special thanks go to Professor Denny Meyer and Swinburne Statistics Clinic team for their valuable time in consulting and advising my statistics matter. Not to forget, Mrs Aruna Vasa for her time to read and doing some editorial works on several chapters – thank you very much!

A big thank you to all the participants- especially to my colleagues at University Teknologi MARA, former colleagues at Motorola Malaysia, as well as Swinburne Software Innovation Lab team members. These studies were only possible because of you.

My highest gratitude to my mother, siblings, and in-laws for all their endless prayers, love, and support to me to go through the ups and downs of this journey. Especially to my siblings - Kak Peah, Kak Jaah, Unggai, and Asu - thank you for looking after mum while I'm away completing this studies. I owe you a lot!

Thank you to my loving husband, Abdul Hadi Othman, for all his sacrifices, understanding, love, care, encouragement and support that always force me to finish this journey. With you, writing a PhD is a sweet dream rather than a nightmare. I also thank my three beautiful angels for being supportive! – Aira and Alya you did a very good job taking care of your little sister, Auni. I love you more than words can say.

I would also like to acknowledge the sponsor of my study – Ministry of Higher Education and Universiti Teknologi MARA (UiTM) that enabled the research to take place at Swinburne University of Technology, Melbourne. My appreciation also goes to the Swinburne Software innovation Lab, Data61, and Deakin Software and Technology Innovation Lab for providing the financial support necessary to attend international conferences, which were a memorable part of this PhD journey.

Finally I would like thank all my Malaysian friends in Melbourne for their friendship, love and help for my family throughout our stay in the Australia.

Declaration

I herewith declare that I have produced this thesis with my on work without the prohibited assistance of third parties. This study has not previously been presented as a thesis. To the best of my knowledge, this thesis contains no material previously published or written by another person except where due reference is made in the text of this thesis.

Nor Shahida Mohamad Yusop,
May 2018

The Author's Publications

1. N. S. M. Yusop, "Understanding Usability Defect Reporting in Software Defect Repositories," in *Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference on - ASWEC ' 15 Vol. II*, 2015, pp. 134–137.
2. N. S. M. Yusop, J. Grundy, and R. Vasa, "Reporting Usability Defects : Limitations of Open Source Defect Repositories and Suggestions for Improvement," in *Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference*, 2015, pp. 38–43.
3. N. S. M. Yusop, J. Grundy, and R. Vasa, "Reporting Usability Defects – Do Reporters Report What Software Developers Need ?," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, 2016.
4. N. S. M. Yusop, J.-G. Schneider, J. Grundy, and R. Vasa, "What Influences Usability Defect Reporting ? – A Survey of Software Development Practitioners," in *2016 23rd Asia-Pasific Software Engineering Conference (APSEC)*, 2016.
5. N. S. M. Yusop, J. Grundy, and R. Vasa, "Reporting Usability Defects: A Systematic Literature Review," *IEEE Trans. Softw. Eng.*, vol. PP, no. 99, p. 1, 2016.
6. N. S. M. Yusop, J.-G. Schneider, J. Grundy, and R. Vasa, "Analysis of the Textual Content of Mined Open Source Usability Defect Reports", in *2017 24th Asia-Pasific Software Engineering Conference (APSEC)*, 2017.

Contents

List of Tables

List of Figures

1 Introduction

This chapter provides an overview of the research in this thesis. It presents the background and outlines the research areas. The research questions are introduced, as well as the problem to be investigated. The expected contributions of the research are also presented. This chapter ends with an overview of the thesis structure.

1.1 Background

Software usability is one of the prominent software quality characteristics that determines acceptance of a software product in today's competitive market. When software does not perform in a way that is expected or has an adverse impact on the user, the software is said to have a defect. We define a *usability defect* as an *unintended behaviour* by the software that is *noticed by the use*r and has an *effect on user experience*. The presence of usability defects will not only frustrate the user, but will also increase the project's costs and timeline. Thus, making an effort to find usability defects and fix the issues at the earliest point in the software development lifecycle are key steps towards reducing software development cost while increasing the user's satisfaction with software. In most common software development approaches, usability defects originate from two primary sources, as shown in Figure 1.1.

The first source that uncovers usability defects is from usability evaluations. Examples of usability evaluations are usability testing, usability inquiry, heuristic evaluation, analytical modelling and simulation methods [1]. A usability evaluator in such usability studies is referred to as the *expert evaluator* who has had formal education, experience and knowledge of usability and/ or human computer interaction (HCI). During usability evaluation, raw usability data in the form of video and audio, sometimes supported with images and notes, are collected. Based on this raw data, the usability evaluator will then write usability defect descriptions to be included in the final usability evaluation reports, either in the form of written documents or recorded via a digital system.

Research in the field of formative usability evaluation reports concentrates mainly on finding effective ways of discovering usability defects. To date there has been given relatively little attention to the description of the discovered usability defects [2]. While there are various formats to describe software defects, there is no clear guidance on how to describe usability defects. The use of these formats often depends on the usability evaluator's preferences and suitability of the usability evaluation method. It is thus not surprising that usability defect descriptions are often ad hoc and difficult to understand [3].

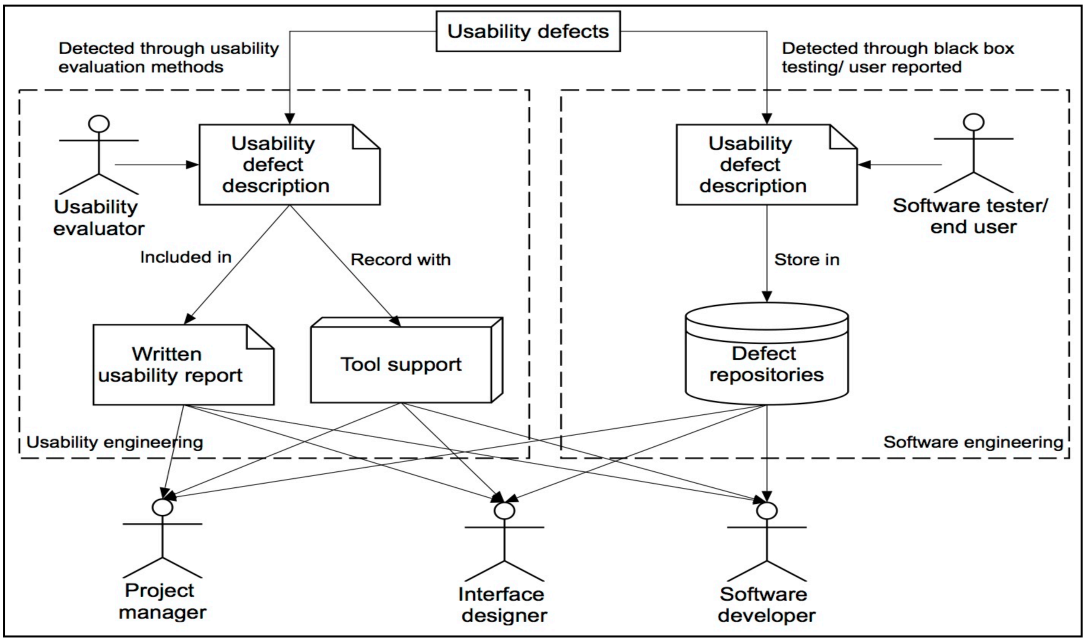


Figure 1.1. An overview of usability defect discovery and reporting in the usability and software engineering fields.

The second source for discovering usability defects is through black box testing performed by software testers. In closed source software development, the software tester is referred to as a *non-expert evaluator* with often no or limited formal education, training and experience in usability or HCI. They assess the usability of the software product indirectly while verifying the product functionality. In this case, the actual users are not involved because the main purpose is not to test the software's usability. If a software tester discovers any frustrating or confusing task, they will report it as a *defect*. This defect report is usually submitted and managed digitally via common software defect reporting tools such as Bugzilla, Jira or Trac. In this way, the information can be shared with different stakeholders (i.e., software developer, interface designer, and project manager) and the defect correction actions can be synchronously tracked and managed.

Similar to closed source software development, defect reporters in open source software (OSS) have limited knowledge about usability. The difference is that anyone in OSS development – user or contributor - can report problems they find when using the software. According to Ko and Chilana [4], OSS contributors consist of core and active developers, defect reporters and passive users. Their contributions to the OSS development are varied depending on their skills and experiences. Given the fact that OSS project development is both very diverse and distributed, usability defect management can be limited.

There are several reasons as to why usability defect management in OSS projects is often neglected [5]. First, many of those who volunteer to contribute to software projects are programmers and many have limited knowledge and skills required for usability evaluation. For passive users who can be non-technical savvy users, creating technical defect reports for technical people is a challenge to them [6]. Second, in order to formally conduct formal usability evaluations, extra commitment from contributors is necessary. However, volunteers may not be able to, or want to spend more time on doing this. Therefore, to employ a usability methodology and expecting them to follow heavyweight usability processes is unreasonable if not impossible [6]. Third, the lack of HCI expert involvement may delay detection of critical factors relating to usability in OSS projects [7].

Regardless of how usability defects are found and who is reporting the usability defects, usability defect reports should provide abundant information for different roles and responsibilities of stakeholders, from both managerial and technical aspects. From the managerial aspect, usability defect reports serve as a source of information for managing project schedules and resource allocation [8], team members selection, and guides the selection of appropriate techniques and methods. In contrast, interface designers and software developers use usability defect information for the purpose of defect correction. They are seeking information that can give them support for the defect correction process, such as event traces, proposed solutions [9] and steps to reproduce the defect.

Ideally software testers and software developers should be co-located and work as single team to share responsibility for improving software quality. If defect reports are unclear, they can meet up and discuss the issue together. However, in the OSS development context where contributors are globally distributed, face-to-face conversations are almost impossible and even remote one to one communication is often limited. Therefore, having clear and meaningful usability defect descriptions is important to avoid information loss and confusion. Furthermore, in the absence of HCI experts in OSS development to consult on usability issues, the probability of usability defects being fixed is highly dependent on how well the usability defect descriptions can convince software developers that the reported defect is worth fixing. However, previous studies have shown that many usability defect reports contain unclear and incomplete descriptions [10]–[13]. The lack of features in existing defect reporting tools for capturing usability defect attributes does not help, nor encourage reporters to submit a high quality and useful usability defect report [10]. These in turn lead to defect reporters often providing irrelevant, incorrect, and incomplete evidence.

In order to empower and motivate OSS users to proactively submit high quality usability defect descriptions, we need to consider the needs of different OSS user backgrounds. While previous studies have demonstrated that even usability practitioners have difficulties in identifying critical usability defects [14]–[16] and recording important usability data, OSS contributors with different knowledge and experience may certainly not know what to report. They may also describe usability defects at different levels of abstraction and in different ways. Since information reported about a defect is a key element to ensure defects are rectified effectively, designing defect report forms to effectively capture usability defect data is a very important step in producing quality usability defect reports by OSS users who often have limited knowledge and experience in less time.

1.2 Challenges in Reporting Usability Defects

Reporting usability defects can be a challenging task, especially in convincing the software developers that the reported defect actually requires attention. The subjective nature of usability defects that cause confusion or problems for some people only [10] may require stronger evidence in the form of specific details. For example, a small clickable area of website hyperlinks on a touch screen device may only cause problems for those who have large fingers. To convince software developers that this finger touch problem is indeed a real problem, additional information is often needed.

While it is recognized that usability defects are different to many other kinds of software defects, little has been done to understand these differences in terms of the way such usability defects should be reported. In fact, the effectiveness of existing defect reporting tools to support usability defects has been little explored. Based on our preliminary investigation of open source usability defect reporting practices [17] and previous studies [7], [10], we identified several challenges of reporting usability defects, as discussed below.

Challenge 1 – Generic defect report forms used for all types of defects. The universal use of a generic defect report form for all types of defects does not assist defect reporters to report clear usability defect descriptions [17]. Research has shown that current open source defect reporting tools are inconvenient for reporting usability defects [6], [7], [10], [18]. For instance, common open source defect reporting tools such as Bugzilla only contain general attributes (i.e., component, version, description, attachment) that do not specifically capture information related to different types of defects.

While some open source defect reporting tools offer fully customizable field lists, such as Bugzilla and JIRA, the focus of such fields towards software developers does not make them very friendly for many end users. As claimed by several respondents in our survey, the overwhelming number of fields to fill out in these defect forms are found to be very troublesome [17]. In fact, many of the fields are not relevant for usability defect reporting at all [7], [17], [19].

Challenge 2 - Insufficient attributes to capture usability defect information. Every defect reporting tool contains a number of mandatory attributes to ensure that defect reporters give a complete description of the defects they encountered. From the perspective of software developers, *steps to reproduce, actual output,* and *expected output* are considered important when fixing software defects in general [20], [21]. However, in the context of reporting usability defects, we argue that this information is inadequate to make usability defects stand out as equally important as any other defect. Usability defect reports need to highlight other important information, such as user and task difficulty, emotion and feeling of the user, impact, violated heuristics or design principles, and proposed solutions. These show how the issue is indeed a real problem and that software developers should treat them appropriately.

Furthermore, the use of a single *description* field as unstructured text in current open source defect report forms does not assist defect reporters to provide information that is important for different types of defects. The unstructured text used to capture defect descriptions results in mixing different kinds of important information. Furthermore, some usability information that is related to a user's feeling, emotions and "struggles" with an interface are difficult to explain textually [6], [17], [22]. This is because such information is subjective in nature and dependent heavily on human judgment. Without specific prompts for this usability defect-specific information, it can be challenging for non-technical users to provide such information in useful ways.

Another drawback of using unstructured textual formats is the uncertain level of detail of the information. The generic *description* field causes many defect reporters to be unsure about just what information they need to report or how the information should be reported. For inexperienced reporters, for example, they may think that their reports are complete, but in the absence of specific attributes they may actually be providing irrelevant or inadequate information. Moreover, meaningless defect information usually leads to lengthy discussions between reporters and developers that effect the defect resolution time [23].

The limitations of unstructured textual and lack of attributes for describing usability defects impedes research on finding what information is required to best explain usability defects, and how this information should be best be captured and presented.

Challenge 3 - Inappropriate terminology to describe usability defects. The lack of vocabulary and terms specific to different types of defects makes no distinction between usability defects, functional defects, or miscellaneous tasks. The inappropriate use of terms to describe usability defects, in our view, is one of the reasons where usability defects often do not get appropriate attention from software developers [23]. Incomplete descriptions to highlight difficulties faced by the users, violated usability heuristics or design principles, and the impact of the issues on business goals, user task, and human emotion, all cause usability defects to get less priority as compared to functional defects. While OSS projects have reporters with varying level of usability knowledge, it is generally difficult for them to apply usability terms, HCI principles, and interface design principles when describing usability issues, especially in the absence of proper guidelines and references. Sometimes defect reporters use very different terms for the same things to express their issues [24]. This in turn causes many usability defect reports produced by non-usability reporters to contain a wide range of non-standard usability terms that complicates usability defect discussion and resolution. Critically, finding and triaging duplicate issues is likely impossible for usability defects, as the same issue may be expressed in different terms from person to person.

Furthermore, the existing generic defect reporting tools that are not specifically designed for usability defects often lead to misunderstandings and disagreement between reporters and software developers [17]. Often reporters have to provide screenshots and emails or written reports to get the developers understand the issues and get them fixed. On the other hand, the use of generic defect classifications such as Orthogonal Defect Classification (ODC) and Root Cause Defect Analysis

(RCA) to identify the relative priorities of software defects in the classic software development lifecycles does not help to prioritize the importance of usability problems [23]. As a result, usability defects are often fixed later as compared to other functional code defects. Introducing more usability terms and options in defect report forms may better facilitate usability-specific defect priority classifications that can enhance the understanding of usability problems and enable developers to better prioritize them. It is worth developing strategies for implementing usability or HCI principles so that they can be applied effectively in the particular context of a software engineering approach. For example, we can explore the customization of the Usability Taxonomy Problem (UPT) and ODC to classify usability defects in OSS projects.

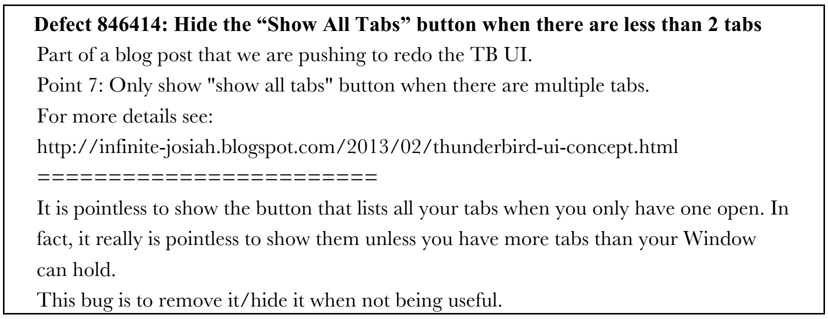
Challenge 4 – Diversity of knowledge and experience between defect reporters. Defect reporters are usually the person that experienced and reported the problem with the software product. They can be a software developer (who uncovers defects while developing a software) but might also be an end-user with varying degrees of experience [4]. Often, the defect reporters' ability to clearly describe and translate the defect determines the probability of defect resolution. Previous studies argued that the current defect reporting tools, which are developed under the direction and are primarily for the use of software developers, are not suitable for certain user groups. Ignoring the mental model of the end users and requesting too many irrelevant attributes on a generic form ignores the need for details pertinent to a defect type [7], [17].

1.3 Research Motivation

There are several important reasons to have high quality defect reports in software development. First, developers use these reports to debug and fix the defects reported by users. Since usability defects are subjective in nature, explaining the defects in appropriate ways is important so that software developers can be convinced about the importance and relevance of a reported problem. Furthermore, the early presence of certain information can speed up the defect correction process.

To get a deeper insight into how defect information can influence defect resolution time, consider usability defect 846414 in the Mozilla Thunderbird project, as shown in Figure 1.2. This defect was opened on February 2013, but only responded by Developer A after a year with the reason that the defect is not high priority and they have a lot of other things to work on. This defect report is very minimal, containing just a summary of the problem extracted from a blog post and the reporter's expectation. Possibly, the unclear description of user difficulty and solution to the problem makes this issue slip to the bottom of the list of things to fix.

Surprisingly, a patch to the issue was ready by Developer B on the same day Developer A expressed their idea to solve the issue. This suggests that including all of the relevant information when a defect is first reported is important to help software developers prioritize the defect, fix the defect and speed up the defect resolution time. However, the current unstructured free-text defect report form may not help reporters to report such information at the initial report submission [10], [17], [24]. Furthermore, without specific prompts for this information, it can be challenging for non-technical users to be aware what kind of information should be provided.



Defect 846414: Hide the "Show All Tabs" button when there are less than 2 tabs

Part of a blog post that we are pushing to redo the TB UI.
Point 7: Only show "show all tabs" button when there are multiple tabs.
For more details see:
http://infinite-josiah.blogspot.com/2013/02/thunderbird-ui-concept.html
=======================
It is pointless to show the button that lists all your tabs when you only have one open. In fact, it really is pointless to show them unless you have more tabs than your Window can hold.
This bug is to remove it/hide it when not being useful.

Figure 1.2. Mozilla Thunderbird usability defect 846414.

A second key reason for better usability defect reports is that in empirical studies researchers use defect data to analyze and evaluate software development approaches for future improvement [25]–[28] . For example, defect data such as severity, priority, version, and component are commonly used for estimating defect resolution time and predicting the number of defects in the next software release. In the context of usability defects, however, this data (i.e., priority and severity) is not relevant to use for such studies because usability defects are often biased to receive low priority [23]. Furthermore, the subjective nature of describing usability defects using various terms for the same issue makes it more difficult to use them when comparing multiple reported defect similarity.

For these reasons we feel it is necessary to investigate a better way of reporting usability defects, especially from the perspective of non-technical users, and to find factors that influence the quality of usability defect reports. We also wanted to revise the existing usability defect classification model to only consider information available in defect reports and the limitation of usability evaluation processes in OSS projects development. We then wanted to apply this improved usability defect classification model to develop a prototype wizard-based, guided usability defect report form.

1.4 Research Scope and Context

The term "usability defects" used in this research refers primarily to graphical user interface (GUI) related defects (i.e., GUI design and layout, look and feel of the software application, and appearance), defects that are related to the ease of use of the software, and defects resulting from interaction of functionality with the user. The usability defects that we studied in this research were based on the Bugzilla defect reporting tool structure. Other open source defect reporting tools such as GitHub, Redmine, Mantis BT, and WebIssues that are outside the scope of this research. However, we believe our results using the Bugzilla repository usability defect reports will generalize to these tools, as well as commercial defect management tools such as JIRA.

In this research, our focus is to understand the information needs by software developers when fixing usability defects, and the information that is usually provided by defect reporters when the usability defect is reported. Based on the findings, we have proposed a set of usability-related attributes to be included in the prototype of a usability defect report form. This form is targeted at defect reporters – who can be software developers (who found an issue while coding), testers (who found an issue while testing a system), end users (who found an issue while using a system), or customer supports (who logged a defect report on behalf of customers' complaint) - to report usability defects found in open source software.

In evaluating the effectiveness of the proposed usability defect report form, we only focus on the quality of information aspect. Our evaluation is only based on the defect readers' perspective to assess the richness of information collected through the proposed forms, rather than the effect of the information in improving defect resolution time or ease of use of the forms in eliciting usability defect information. Although longer defect descriptions are considered to speed up defect resolution time [23], [29], this may not happen when fixing usability defects. In our opinion, longer usability defect descriptions are useful to assure good understandability and clarity of the usability defects, but not at providing quick solutions for fixing defects. Therefore, in this work we focus on improving the way valuable usability defect information can be collected so that usability defects can be better understood and evaluated even though the usability defect will possibly be assessed and fixed later.

1.5 Summary of Research Methodology

The research method used in this PhD thesis research consisted of three main stages: an initial stage, an intermediate stage, and a final stage, which were devised to address the key research questions. These are illustrated in Figure 1.3.

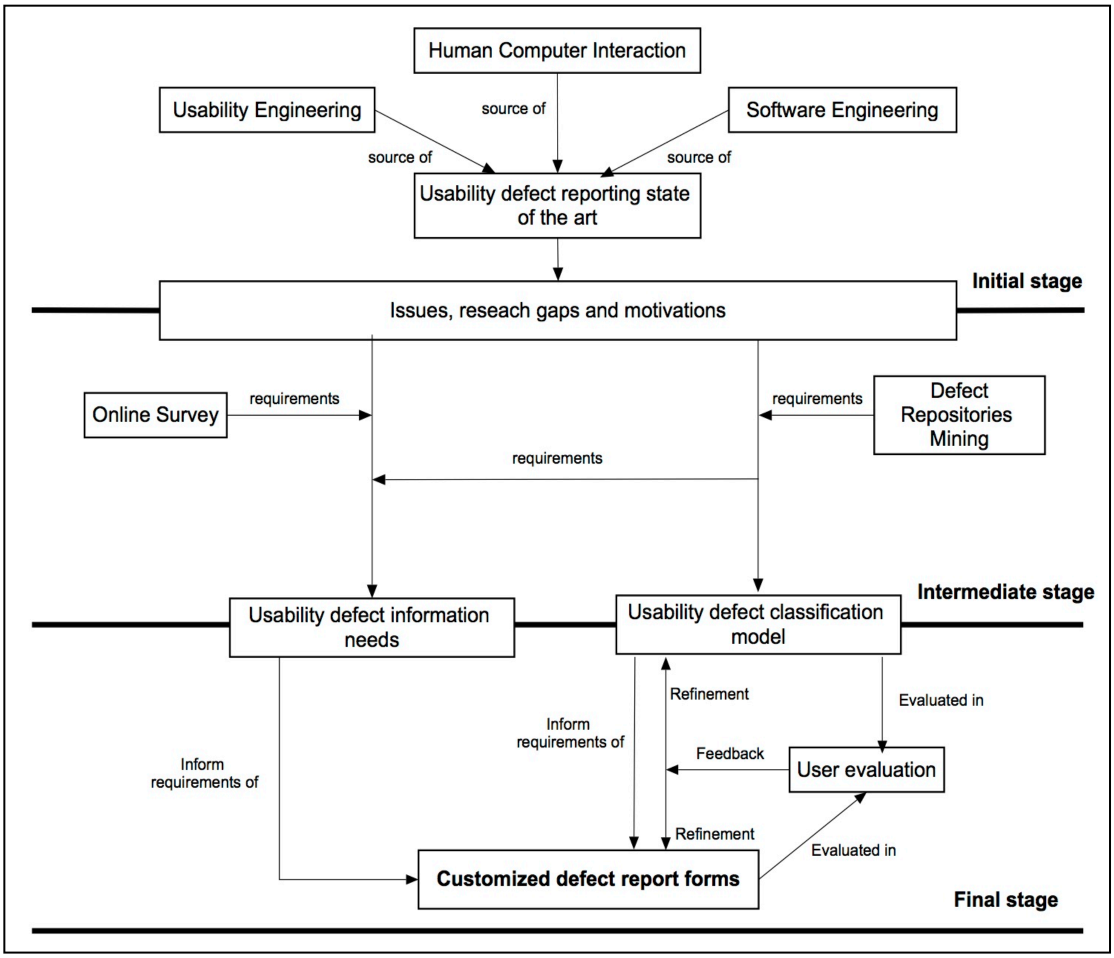


Figure 1.3. Summary of research methodology.

In the initial stage, the usability defect reporting *state of the art* was identified from three main sources: usability engineering, HCI, and software engineering studies. The reviews of these studies were used to reveal the issues, similarities, commonalities and differences in the way usability defects are described by different communities. This information was used to support our research motivations and highlight research gaps.

In the intermediate stage, we identified current practices, limitations, and needs of usability defect reporting through (i) an online survey of software development practitioners, and (ii) mining reports from OSS defect repositories. We used this information to extract desirable features of new defect report forms and develop an OSS usability defect classification model.

The final stage involved the improvement of existing defect classification taxonomies and the design of new defect reporting forms to better support usability defect reporting. Finally, our proposed new usability defect report forms and open source usability defect classification taxonomy was evaluated to identify practical aspects of the forms in reporting usability defects.

Details of the research strategies were described in different studies presented in Chapters 3 through 7 in this thesis.

1.6 Research Goals, Research Questions, and Approach

The main goal of this research was to improve the defect reporting approach for capturing usability defects in the context of OSS development. This main goal was divided into four sub goals: (1) investigate the *state of the art* of research in usability defect reporting; (2) identify the usability defect information that software development practitioners emphasize in current usability defect reporting; (3) investigate how open source usability defects can be categorized; and (4) develop a customizable usability defect reports. These research sub goals were addressed in five research questions, in which each research question was studied in a small research study with specific methods and research study objectives. Table 1.1 summarizes these research sub goals, research questions, associated research studies and methods, and outputs.

Table 1.1. Research goals, questions and steps of the approach mapped to phases.

Research Goal	Research Question	Study	Method (s)	Output (s)
RG1- Investigate the *state of the art* of research in usability defect reporting.	**RQ1** - To what extent has usability defect reporting been considered in existing usability and software engineering research?	**Study 1** – Review the literature to develop an understanding of work related to usability defect reporting and to investigate to what extent usability defect description is being studied to support defect management.	• Systematic Literature Review • Analysis	• Synthesis of usability defect report mechanisms, formats, and guidelines. • Use of usability defect data in software/ usability engineering empirical research. • Key challenges in usability defect reporting. • Recommendations for improving usability defect reporting and defect management.
RG2 – Identify what usability defect information do software development practitioners emphasize in current usability defect reporting.	**RQ2** - What usability defect information do software developers and reporters emphasize in current usability defect reporting?	**Study 2** – Conduct an online survey of software development practitioners to discover different opinions about the usability defect information that is important to reporters and software developers. The same survey also was used to investigate factors that influence usability effect reporting.	• Online survey • Software defect repositories mining • Analysis	• Identifications of usability defect data that is important to OSS development practitioners • Descriptions of limitations and suggestions of usability defect reporting • Descriptions of factors influencing usability defect reporting
	RQ3 - What factors influence the description of usability defects?	**Study 3** – Perform software defect repositories mining to understand the way usability defects are described in current open source defect repositories.		
RG3 – Investigate how open source usability defects can be categorized.	**RQ4** - How should open source usability defect reports be classified so that they can be effectively reported?	**Study 4** - Develop usability defect taxonomy based on the available information reported in open source defect reports.	• Software defect repositories mining • Analysis • Design a taxonomy • Evaluation	• An Open Source Usability Defect Classification taxonomy
RG4 – Develop a customizable usability defect reports.	**RQ5** - How can open source usability defects be more effectively reported?	**Study 5** – Develop defect report forms based on the inputs from Study 1 – Study 4	• Design defect report forms • Evaluation	• Usability defect report forms

1.7 Research Contributions

The high-level goal of this research is to improve open source usability defect reporting. Specifically, this research has made several contributions to the field of empirical software engineering, particularly in the area of software defect management. The main contributions of this research are:

- We conducted a comprehensive literature review to understand the state of the art in usability defect reporting, and most importantly to find gaps in an empirical software defect reporting. We used a Systematic Literature Review (SLR) approach to systematically review both software engineering and usability engineering studies. We identified some key areas for future research to improve the state of the art in usability defect reporting. Through this study, researchers can find a review of current practices, key open issues and limitations, and important areas for future research with respect to reporting usability defects. In addition, evidence from the SLR helps open source communities to understand which information is important when describing usability defects.
- The results from the survey of software development practitioners and observation of open source usability defect reports present empirical evidence on the nature of describing usability defects. The triangulation studies reveal evidence on the information mismatch between what defect reporters think they usually provide and what is actually reported, and what information defect reporters think is important for software developers to fix the defects.
- We revised the usability defect classification taxonomy by incorporating cause-effect relationship, not previously considered in the usability engineering studies. The proposed open source usability defect classification (OSUDC) can be used in open source defect management to track usability defect trends over time for individual categories. In fact, the usability defect reports that are classified using OSUDC may facilitate the triaging process, specifically in determining which usability defects have significant effects that require prompt attention, and the software developer who is suitable to resolve the defects.
- We proposed a list of key textual usability defect attributes that need to be captured to help software developers get a better understanding of this kind of defects. These attributes are presented in a structured way, which encourages researchers to explore more empirical studies

in the domain of defect management such as defect prediction models using the textual data, rather than typically used categorical data.

- We proposed and refined the Bugzilla defect report form for better capturing usability defects. The form is designed as a guided wizard to support multi-levels of user experience. The usability defect attributes are prompted based on the technical skills and background of the defect reporters. This will contribute towards creating a simple and improved platform with which to report usability defects.

1.8 Thesis Organization

The rest of this thesis is organized as follows:

Chapter 2 describes related work in software defect reporting from both the software engineering and usability engineering perspectives. This chapter also reports our review of relevant literature investigating usability defects in OSS development context.

Chapter 3 presents our SLR of usability defect reporting from both usability engineering and software engineering studies. We employed a process of conducting the SLR as described in the study by Kitchenham et al. [30] and Petersen et al. [31]. Results of the review were discussed in three categories: reporting usability defect information, analysing usability defect data and key challenges. In addition a number of recommendations to improve usability defect reporting and management in software engineering are discussed.

Addressed research question: RQ1- To what extent has usability defect reporting been considered in existing usability and software engineering research?

Chapter 4 describes a triangulated research study to identify information needs when reporting usability defects. The main sources of information include an online survey of software development practitioners and usability defect reports collected from open source defect repositories. For the multi part online surveys, this chapter partially describes the findings. The part of the surveys reported in this chapter focuses mainly on what kind of information is used during reporting and fixing usability defects. The findings from the online survey and defect repositories mining were compared. We analyzed the collected data from different sources to reveal commonalities and dissimilarities between what is claimed by the practitioners with what is actually written in usability defect reports.

Addressed research question: RQ2 - What usability defect information do software developers and reporters emphasize in current usability defect reporting?

Chapter 5 describes the second part of the survey reported in the Chapter 4. This part of the survey focuses on factors influencing usability defect reporting. We collected opinions of software development practitioners about five different factors that they think might influence the quality of usability defect reporting - role of the reporter, knowledge and experience of reporters, use of automation tools, and influence of defect discovery methods.

Addressed research question: RQ3 - What factors influence the description of usability defects?

Chapter 6 presents the revised OSUDC taxonomy for open source software projects. The OSUDC was adapted from UPT and Usability-ODC framework. The OSUDC augments an existing usability classification framework with the introduction of *failure qualifier* and *user difficulty* components. This chapter also presents the results of the online evaluation to collect feedback on the new OSUDC.

Addressed research question: RQ4 - How should open source usability defect reports be classified so that they can be effectively reported?

In **Chapter 7** a set of defect report forms were designed based on the results of the previous studies. It details the four criteria used in designing the forms, and the evaluation strategy to assess the quality of information captured using the proposed forms.

Addressed research question: RQ5- How can open source usability defects be most effectively reported?

Chapter 8 summarizes the findings of the series of research studies conducted as part of this thesis. In addition, the lessons learned, limitations and future research directions are outlined.

2 Related Work

Knowing the importance of quality defect reports in determining how quickly a defect will be fixed has attracted many researchers into exploring how to improve defect reporting tools and to consider what makes a good defect report [32]. Indeed, the latter research area is more critical as a good defect report structure and content can support defect reporters in producing more detailed and comprehensive defect descriptions that are useful for software developers to understand and fix the problem accurately [9], [20]. In this chapter, we discuss related work that has investigated defect report content in order to improve their quality. We reviewed studies spanning open and closed source as well as distributed software development, software engineering, and usability engineering disciplines. A focused treatment of just usability defects is presented in Chapter 3. In this chapter, we present prior work and contextualize them within the context of the work presented in this thesis.

2.1 Software Defect Reporting in General

Software defect reporting is an essential function within most software projects. A good defect reporting tool is required for recording, reporting, and tracking defects with a supporting process for improving project quality. Within the context of industry and academia, there have been several studies that focus on software defects. Their contributions are classified into four types according to their motivation: 1) determining useful defect information, 2) understanding defect report content, 3) determining defect characteristics, and 4) improving defect reporting tools.

Determining useful defect information in software development – Research in this area investigated the specific information that is useful for software developers. We found two studies that surveyed software developers and reporters in order to investigate the most helpful information when reporting and fixing software defects.

Zimmermann et al. [20] surveyed 466 software developers and reporters from Apache, Eclipse and Mozilla projects to find what makes a good defect report. They found that the most helpful information for fixing software defects in OSS projects are steps to reproduce, stack traces, test cases, screenshots, actual outcome, and expected outcome. Additionally, they discovered that the information provided by the defect reporters is contrary to the information needed by the software developers to fix software defects.

Building upon Zimmermann and his team's study, Laukkanen et al. [21] replicated the Zimmermann et al. survey in six industrial software organizations. In addition to investigating useful information for fixing software defects, they extended their survey to investigate missing or incorrect defect information, and find out which defect information could be collected automatically. They confirmed that steps to reproduce and actual outcome are the most important information in defect reports. However, they found that many defect reports lack this technical information and this information is difficult to collect automatically.

Understanding defect report content – Research in this area has examined defect report content to reveal the current patterns used when describing software defects. Davies and Roper [33] examined 1600 defect reports from Eclipse, Firefox, Apache, and Facebook API to understand what information users provided in open source defect reports, how frequently this information is provided, and how the information provided affects the defect outcome. They focused on ten defect attributes – steps to reproduce, stack traces, test cases, actual outcome, screenshots, expected output, code examples, summary, version, and error reports. They found that most of the included information in defect reports is actual outcome, followed by expected outcome, and steps to reproduce. Nevertheless, their observations on these defect reports found that many of them are incomplete and do not contain information expected by the software developers.

In different studies, Bhattacharya et al. [29] analyzed Android defect reports and only studied steps to reproduce, output details, additional information, and description length. Similar to [33], they found Android-based app defect reports often contain steps to reproduce and explanation of the difference between actual and expected outcomes. In fact, they reported that a good quality defect report is one that has long textual descriptions of the problems.

Determining defect characteristics – Research in this area examined real defect report data to understand certain defect characteristics. We found that many studies in this area are interested investigating performance and security defects. For example, Zaman et al. [34] studied defect reports from the Firefox project to find out the differences in time to fix, developer experience, and defect fix complexity between security and performance defects. In contrast, Nistor et al. [35] and Zaman et al. [36], focused on performance and non-performance defects of various open source defect reports. Nistor et al. studied the performance defects of Eclipse JDT, Eclipse SWT, and Mozilla projects across three dimensions - risk of introducing new functional defects, defect-fix time and difficulty, and defect discovery methods and reporting. Zaman et al. on the other hand, studied Mozilla Firefox and Google Chrome performance defects to understand the impact on the stakeholder, context of the defect, the defect fix-time, and defect fix validation.

Other studies examined and characterized defect report content for software defects in general. Tan et al. [37] manually examined 2,060 defects in the Linux Kernel, Mozilla, and Apache projects. They investigated the defect characteristics in three dimensions - root cause, impacts, and components, and measured the association between dimensions. They found semantic defects are the dominant root cause, while the majority of security defects cause severe impacts. Lal et al. [38] investigated the properties and features of regression, security, crash, performance, usability, polish, and cleanup defects in the Google Chromium Browser. They compared the characteristics of different defect types in terms of defect fix time, number of stars, comments, discriminatory and frequent word for each class, entropy across reporters, entropy across component, opening and closing trend, continuity, and debugging. In the context of usability defects, they found mean time to repair and release date (MTTR) of usability defects is fairly high as compared to other types of defects, and milestone change for usability defects is the highest.

Several studies have investigated linguistic aspects of defect reports. Ko et al. [39] studied the nouns, verbs, adverbs, and adjectives in defect report titles. They found 95% of the noun phrases referred to visible software entities, physical devises, or user actions. To solicit more structured defect report titles, they suggested that defect titles should include descriptions about software entity or behavior, quality attribute, the problem, execution context, and types of defects. In different work, Chilana et al. [40] developed a classification scheme for capturing the different types of expectation violations. They started by analyzing a sample of 50 defect report titles and descriptions and produced seven classification codes – runtime logic, standards, reporter expectations, community expectations, genre conventions and prior behavior. They tested this classification model using 1000 defect reports from the Mozilla project. Their findings show that reporter expectation is the most common violation expectation in defect reports, but are less likely to get fixed. In the context of usability defects, they argued that the way defect descriptions are phrased affected the defect resolution status.

Improving defect reporting tools – Research in this area looked at ways to improve defect reporting tools. According to Zimmermann and Breu [18], current defect reporting tools have too few features to help reporters to provide important information needed by developers. Their study proposed four concepts that should be employed in defect reporting tools – (1) tool-centric to automatically collect information, (2) information –centric to provide real-time feedback in the quality of information provided, (3) process-centric to focus on administration of activities related to defect fixing, and (4) user-centric to educate and remind both reporters and developers on what information to provide and how to collect it. As a proof of concept, they developed a prototype, but no studies on the usefulness of these concepts in the prototype were presented.

Other studies focused on improving the readability of discussion threads in the comment section of defect reports. Dit and Marcus [41] proposed a system to recommend to developers a set of comments associated with their own comment, so that they can keep discussions coherent and easy to understand. Lotufo [42], as part of his thesis, proposed two complementary features to improve the readability of defect reports. The first feature is an automated approach for creating defect report summaries that can help developers to select portions of information that they want to focus on. The second feature is a nested comment to allow users to post contextual comments specific for each of the different diagnostics or solution posts. This feature eliminates the problems of mixed information that hinders users' ability to locate and understand problem discussions introduced in the current linear sequence of comments. Moran et al. [43] developed FUSION to assist defect reporters' auto-complete reproduction steps in defect reports for mobile apps. Their findings from experimental studies of 14 Android apps defects showed that FUSION is effective in facilitating defect reporting and reproduction tasks as compared to the unstructured natural language defect descriptions.

2.2 Usability Defect Reporting in the Context of Open Source Communities

In recent years, usability of open source software has become an important topic for investigation. Several studies have shown interest in investigating usability practices and have identified some challenges discovered in open source software development. For example, Terry et al. [44] investigated the perceptions and practices of usability in the OSS community. Based on interviews of 27 individuals involved in various OSS projects, they found that the OSS communities demonstrated a fairly good understanding of usability concepts. Common practices for communicating usability defects are through Internet relay chat (IRC), mailing lists, forums, and defect reports. Their findings show empirical evidence that rich, high quality, positive feedback from users helps in improving software design. In fact, the lack of relevant domain expertise in the developed software makes software developers rely on users feedback. Raza et al. [45] specifically investigated the use of online forums in addressing usability-related issues. They studied 1753 OSS projects collected from sourceforge.net. Their study provided empirical evidence that contribution and support of open source community is significant and active, and voluntary contribution in mailing lists helps to identify and fix usability defects.

However, communicating usability issues in OSS development is not an easy matter. Past research has investigated some of the challenges. For example, Zhao et al. [6] conducted a usability evaluation case study of Ganttproject to understand the usability improvement required in OSS projects. They found that the defect reporting mechanism of OSS projects is not appropriate for reporting usability defects. In fact, they observed that most contributing users are amongst the most experienced ones, in which their request for usability improvement does not reflect the need of non-experienced users. One key implication of their findings towards improving OSS usability is that the defect reporting of OSS projects should consider a new way of promoting contribution from non-active or typical users. Similarly, Cetin et al. [7] has identified the lack of a suitable usability defect reporting tools as a main obstacle to encourage HCI experts and end users to report usability issues. Similar findings were also reported in [6]. Nichols and Twidale [22] reported that the use of textual description in current defect reporting tools is not suitable to describe graphical issues, and some usability issues have a dynamic aspect that could not be explained using a single screenshot. This is because the explanation about solution proposals is more relevantly explained using graphical means such as HTML mockups, ASCII art, or drawing toolkits. Another issue raised by Nichols and Twidale is regarding threaded discussions in open usability defect reports. The linear sequence of comments in current defect reporting tools makes it difficult to manage discussion elements, and this issue was addressed in [41], [46]. Their study brought up two important concerns for attracting more participants to submit usability defects. First, the defect reporting tools must be easy to use, where users can report with less effort and the tools do not contain too many technical aspects; and secondly, the defect reporting tools should address privacy concerns on the automatically collected data.

Raza et al. [47] empirically investigated some of the key factors to improve OSS usability. They hypothesized that understanding users' requirement, seeking usability experts' opinion, incremental design approach, conducting usability testing, and knowledge of user centered design methods could improve usability in OSS. Among the five key factors, they found usability experts' opinion did not play a significant role in improving OSS usability.

2.2.1 Empirical Studies Focusing on Usability Defect Reporting

We identified three types of studies investigating usability defect reporting: 1) determining useful defect information in software developments, 2) improving defect report comprehension, and 3) improving defect reporting tools.

Determining useful defect information in software developments- Research in this area specifically investigated useful information for fixing usability defects. In contrast to [20], [21], Hornbaek and Frokjaer [9] used developers' judgment to assess the quality of defect report descriptions. Their findings show that developers expect usability defect descriptions that are clear, contain solution proposals, and justify what posed a problem and why it was a problem.

Improving defect report comprehension – Research in this area concentrated on the different approaches for improving the quality of usability defect report content. In reporting the results of usability evaluation, there have been a number of description formats described in the literature, which vary to different types of usability evaluation methods and end up being documentation heavy. For example, Theofanos and Quesenbery [2] redefined the form and content of formative test reports based on the outcome of two usability professionals workshops. However, their test reports contain too much information and some of them are not pertinent to non-technical users. 88 information elements, grouped into 15 categories such as business and test goals, methodology, tasks and scenarios, results and recommendations, screenshots and videos are difficult and complicated for a single reporter to provide.

As an alternative to the heavy-documentation approach, which is often compiled into written reports and delivered as a PDF or Word document, several studies have developed tools to record usability defects. Howarth et al. [48] developed the Data Collection, Analysis, and Reporting Tool (DCART) for collecting usability data from lab-based usability evaluations. In contrast to [2], DCART contains slightly less textual information to be filled in by usability evaluators - report title, problem description, problematic user interface object, designer's knowledge and thoughts, and solutions. The use of a form-based approach in DACRT provides additional support for novice evaluators to keep appraised of what important data should be provided. Other tools such as Usability Reporting Manager (URM) [49] offer a structured reporting format for capturing the results of usability evaluations. To speed up the reporting process, usability evaluators only need to enter results of formative evaluation into URM and results are exported into UsabML format. Usability defects in UsabML format can be directly imported into defect reporting tools connected to a source code repository, or can be exported into other formats such as HTML, XSL, and PDF.

Capra [11] and Dumas et al. [16] on the other hand, focus on guiding reporters to provide meaningful usability defect descriptions. Capra developed six guidelines for describing usability defect descriptions based on surveys of usability practitioners – be clear and precise, describe the cause of the problem, describe observed user actions, support with data, describe the impact, and describe a solution. These guidelines were tested in a comparison study between usability practitioners and graduate students. Dumas et al., based on his experience in usability testing, outlined four pieces of generic advice for communicating usability defects effectively – emphasize the positive, express your annoyance tactfully, avoid usability jargon, and be as specific as you can. However, these guidelines did not specify what to include in the descriptions.

Recent work by Simões [24], however, represents an important step towards building lightweight documentation for describing usability defects. Simões, in her thesis, explored the needs of designers in open source projects, and designed a new defect reporting template to support end users reporting usability defects. They conducted an interview with four designers who worked on open source projects and performed content analysis of 547 usability-related defect reports. Based on these findings and adoption of the semiotic engineering concept, they designed an open text form, in which defect reporters have to answer several questions depending on the four labels they chose to characterize their problem. Their contribution shows a positive solution for eliciting the information needed by OSS designers. However, the use of an unstructured open textual form still produced incomplete, ambiguous, and irrelevant information. This was because not all questions were relevant for different types of problems, and such open-ended types of questions may produce non-informative descriptions.

Improving defect reporting tools – Research in this area explored the opportunity to include HCI and usability principles in current defect reporting tools. Faaborg & Schwartz [50] introduced usability heuristics concepts into the Bugzilla repositories. They suggested that each usability defect should be tagged with the specific violated heuristics. In this way, software developers can monitor the different types of issues, and learn about the heuristics. However, this idea is still in the working stage and has not yet been implemented or evaluated.

2.3 Summary

Writing a good defect report is critical for ensuring quick resolution of defects. In empirical research, usability defects are discussed in two main disciplines – software engineering studies and usability studies. In software engineering studies, research on software defect reporting to date has focused especially on determining useful defect information, discovering defect characteristics, and improving defect reporting tools for software defects in general. Research on usability defects in usability studies, on the other hand, more focused on the improvement of usability evaluation methods to effectively discover usability defects. While it is recognized that usability defects are different to many other kinds of software defects, little has been done to understand differences in the way the defects are reported. In order to understand in more detail about similarities, commonalities, and differences in the way usability defects are reported in these two disciplines, we conducted systematic literature review as discussed in Chapter 3.

3 State of the Art in Usability Defect Reporting

This chapter describes the current *state of the art* of research in usability defect reporting. The study was conducted to address our first thesis research question *"RQ1 - To what extent has usability defect reporting been considered in existing usability and software engineering research?"*. We carried out a Systematic Literature Review (SLR) to study the different approaches to describing usability defects in software development and usability engineering. Both studies in software engineering and the usability engineering literature were reviewed and the results are classified into three dimensions: 1) reporting usability defect information - which is related to research on reporting the usability defect; 2) analysing usability defect data - which is related to researching the use of defect data; and 3) key challenges – which refers to issues arising in usability defect reporting and management. In addition, some key areas for future research to improve usability defect reporting and management in software engineering are outlined. The following subsections explain in detail the SLR process.

3.1 Methodology

In order to conduct the SLR, we used the guidelines of Kitchenham et al. [30] and Petersen et al. [31]. An SLR is defined as a process to identify, assess, and interpret available research studies with the purpose of answering specific research questions and providing scientific summary of evidence in a particular area [51], [52]. Our review process consisted of three stages.

In the first stage, we defined a set of research questions and prepared a review protocol. This review protocol assisted supervision of the researchers conducting the review and guided the researchers in the data collection. It specifies the research questions, the inclusion and exclusion criteria, the assessment strategy for study quality, the detailed data that need to be extracted from each of the selected studies, and the strategy to perform a database search. Once the research supervisors approved the review protocol, we conducted a pilot study. This pilot study aimed to ensure the research questions were appropriate to the context of the study and researchable. The outcomes from the pilot study are used to refine the protocol.

In the second stage, we identified relevant databases and sources, and developed several search terms corresponding to the database. In the first screening, we only read the title and abstract on the papers. Any study related to the research topic and within our research boundaries were selected for the second screening. During the second screening, the entire paper was read and we applied the inclusion and exclusion criteria to choose the final primary studies to be included in the review. Then we re-read the papers to extract the data according to the extraction form that we had developed earlier. We used quality assessment criteria to measure the study strength and lack of bias. In the third stage, we analysed and synthesized the results for reporting. The following subsections discuss in details the SLR process used, as illustrated in Figure 3.1.

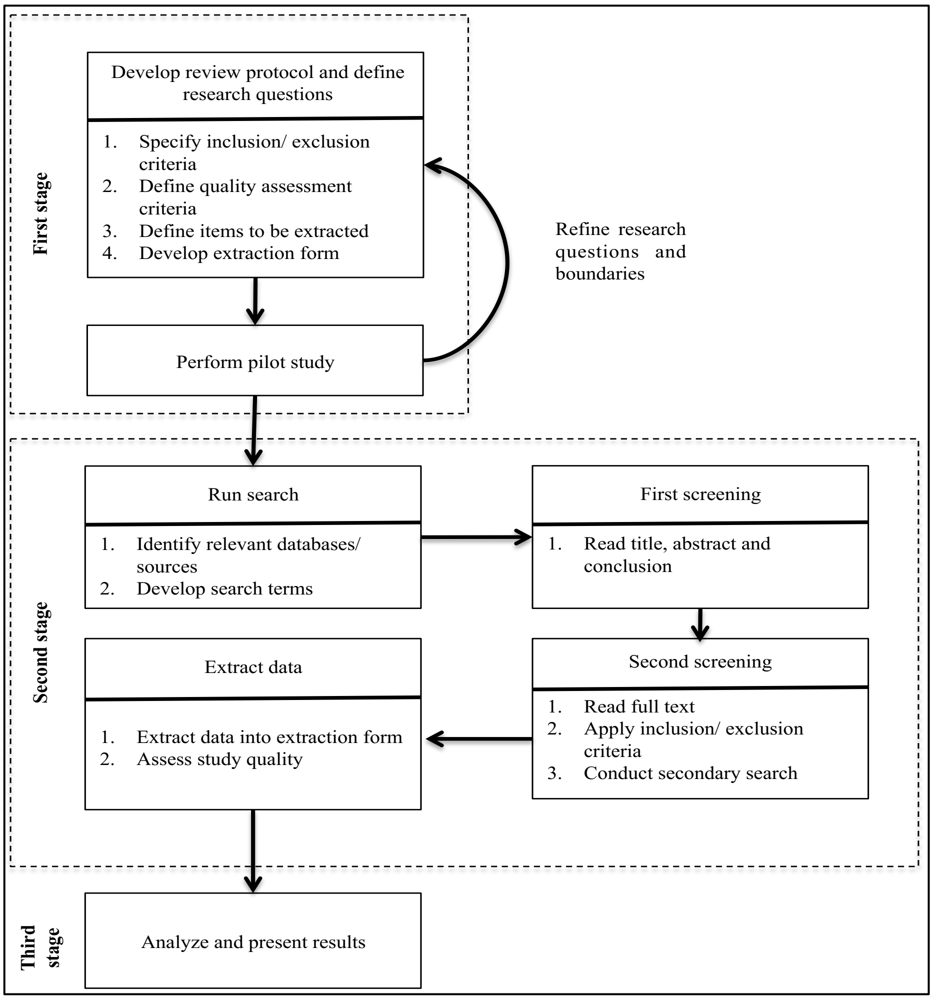

First stage
Develop review protocol and define research questions
1. Specify inclusion/ exclusion criteria
2. Define quality assessment criteria
3. Define items to be extracted
4. Develop extraction form
Perform pilot study
Refine research questions and boundaries
Second stage
Run search
1. Identify relevant databases/ sources
2. Develop search terms
First screening
1. Read title, abstract and conclusion
Second screening
1. Read full text
2. Apply inclusion/ exclusion criteria
3. Conduct secondary search
Extract data
1. Extract data into extraction form
2. Assess study quality
Third stage
Analyze and present results

Figure 3.1. Systematic literature process adapted from [30], [31], [51], [52].

3.1.1 Research Questions

The overarching aim of this SLR was to understand *"To what extent is usability defect reporting considered in existing usability and software engineering research?"*

In usability engineering, usability defects are often found through usability evaluation methods. These usability defects are normally described in written evaluation reports. On the other hand, usability defects that are found during system testing or reported by end users are reported in defect repositories, such as Bugzilla, Google Chromium and JIRA. These usability defects have the same underlying root cause but were found in different testing stages and were reported by a different mechanism. This motivated us to review both the usability and software engineering literature to understand how usability defects are reported in practice. Therefore, the above high-level research question was further divided into the following sub-questions. These research questions are structured based on PICOC criteria suggested by Kitchenham et al. [51] as given in Table 3.1:

1. How are usability defects communicated in the usability and software engineering literature?
    a. What mechanisms are used to report and track usability defects?
    b. What defect information and formats are used for reporting usability defects?
    c. Are there any guidelines available to assist the reporting process?
2. Is there any evidence that usability defects have been studied from the use of data in defect reports?
3. What are the identified challenges of usability defect reporting in the usability and software engineering field?

The first question searched the usability, HCI, and software engineering literature to identify research that focuses specifically on usability defect reporting. These were then analysed and classified into topics of studies as suggested by McInerney [53]. The second question identifies studies that analysed data from usability defect reports or defect repositories. While the third research question was aiming to reveal challenges in reporting usability defects from the perspective of usability and software engineering.

Table 3.1. Summary of PICOC.

Population	Usability defects
Intervention	Defect reporting
Comparison	None
Outcome	Not concentrated on results
Context	Usability engineering and software engineering

3.1.2 Data Sources

Five electronic database resources were primarily used to search usability defect reporting. These included: IEEE Explore, ACM Digital Library, ScienceDirect, Scopus, and Google Scholar. These electronic database selections were based on the recommendations in [1] and [54]. To facilitate the search process, an advanced search option was used that allowed multiple keyword searches. Title and abstract data fields were primarily used to retrieve relevant journal and conference proceeding papers. In this research, we only reviewed papers published from the year 2000 onwards. This limitation was set because we identified a few studies that were reported prior to 2000 and were extended in other studies, which were included in our review.

3.1.3 Search Strings

To ensure a thorough search in both usability and software engineering literature, a set of search strings was created for each research question. The search strings were formulated based on:

- Major terms from the research questions
- Relevant terms extracted from relevant papers, journals, and books
- Synonyms, alternative terms, and related concepts of research questions
- Boolean AND and OR to link all the terms

Three different search strings were derived and executed on different electronic databases. As the literature search progressed, search terms were refined, discarded, and added. Any changes to the search strings were rerun on the selected electronic databases to ensure all relevant papers were retrieved. These strings are listed in Table 3.2.

Table 3.2. Systematic literature review search strings.

Literature	String	Search String	RQ	Purpose
Usability engineering	1	((Usability defect* OR " usability bug* OR usability problem* OR usability issue*) AND (approach OR technique OR methodology OR procedure OR mechanism OR plan OR pattern OR tool OR track* OR manag*)	1	To identify mechanisms used to report and track usability defects and discover existing defect description formats and guidelines that relate to usability defects
Software engineering	2	("Defect reporting" OR "bug reporting" OR "error reporting" OR "fault reporting" OR "defect reports" OR "bug reports" OR "error reports" OR "fault reports" OR "crash reports" OR "defect description" OR "bug description" OR "error description" OR "fault description") AND ("usability" OR "user interface" OR "GUI"	2	To review the literature for usability defect reporting in the software engineering area
	3	("Defect reporting" or "bug reporting" or "problem reporting" or "issue reporting" or "defect tracking" or "bug tracking" or "issue tracking" or "problem tracking" or "bug repository" or "defect repository" or "issue repository" or "problem repository" or "bug tracker" or "defect tracker" or "issue tracker" or "problem tracker" or "bug repositories" or "defect repositories" or "issue repositories" or "problem repositories") AND ("usability" OR "user interface" OR "GUI")	3	To review any issues that are relevant to usability defects and challenges of existing defect repositories in handling usability defects

3.1.4 Study Selection

The primary search resulted in 609 studies. This set was then filtered based on title and abstract analysis, which reduced the total to 191. The significant difference from the first and second filtration was partly due to duplication and irrelevant context of study. For instance, the search on the term "usability defect" often returned studies that belonged to medical, engineering or telecommunication topics, which were out of our research context.

We then conducted a secondary search using a *reference chaining* technique. The *reference chaining* is commonly used in other SLRs [55]–[57] as supportive search approach to find any relevant studies that were not found during the primary search. This resulted in 52 new studies being included in the second filtration process.

A total of 243 studies were then analysed by reading the full paper text. At this stage, inclusion and exclusion criteria, as shown in Table 3.3, were applied to evaluate papers. Since this study was surveying a blend of software engineering, HCI and usability defect reporting literature, a narrow inclusion criteria was used. We defined our inclusion criteria to be specific to each research question [58], while the exclusion criteria were common to all research questions. Reasons to include a paper were: 1) that it belongs to the area of defect reporting in general, and usability defect reporting in particular, and 2) that the defect reports originated from a defect tracking system and usability evaluation reports, and not from other means of reporting usability defects.

As discussed in the introduction and background sections, while a range of means of detecting usability defects exist, we were interested in how they are described, reported, and tracked by software development teams, and hence papers that focus on these aspects. This review only considered papers published from January 2000 to March 2016. Finally, 57 studies were included in this review. See Appendix A for the list of included studies.

Table 3.3. Inclusion and exclusion criteria.

Inclusion criteria	**Exclusion criteria**
• Studies that focus on usability defect description/ format/ report/ guideline • Studies that focus on analysing/ using usability defect information • Studies about a tool or mechanism to report usability defects • Empirical studies on usability defects	• Exclude if the paper is on SLR or systematic mapping • Studies not in English • Short papers, posters, introduction to special issues, tutorials, and mini-tracks • Defect reporting studies not focussed on usability defects • Usability defects studies not focussed on defect reporting

3.1.5 Quality Assessment of Selected Studies

All selected papers were assessed for their quality. Each of these papers was also classified as either a software engineering or a usability study, respectively. Papers were evaluated using two sets of checklists that were formulated to measure the research credibility and validity. Each question was rated as: 1 implies “Completely describe”, 0.5 implies “Exists but does not completely describe” and 0 implies “Does not exist”. The total quality score for each paper was computed by summing up all the scores. This ranged between 0 (very poor) and 5 (very good). The checklist used is shown in Table 3.4. Table 3.5 shows the quality scores for all the primary studies included in the final review. Most achieved above average quality: 13 studies (22.8%) and 30 studies (52.6%) were deemed very good and good quality, respectively.

Table 3.4. Quality assessment questions.

Common questions	
1. Are the aims of the research clearly articulated? 2. Have other authors cited the study? 3. Does the study report credible finding with supported data/ evidence?	
Usability engineering	**Software engineering**
4. Is the study’s focus on usability defect description? 5. Was there an in-depth description of communicating usability defects?	4. Is the study’s focus on defect reporting in software development? 5. Does the paper study usability defects?

Table 3.5. Quality scores of the 57 studies included in the final review.

Quality scale	**Very poor (<1)**	**Poor (1-2)**	**Fair (2.5-3)**	**Good (3.5 - 4)**	**Very good (>4)**	**Total**
Number of studies	0	0	14	30	13	57
Percentage	0%	0%	24.6%	52.6%	22.8%	100%

Table 3.6. Common data items extracted from all papers.

Data items	**Description**
Identifier	Unique identification number
Bibliographic	Title, author, year
Type of article	Journal/ conference/ book chapter/ technical report/ theses
Study aim	The aims, goals or objectives of the primary study
Research methodology	Case study, survey, experiment, interview, observation, questionnaire, lesson learned
Data analysis	Qualitative, quantitative, or mixed
Study findings	Results and conclusions from the primary studies

3.1.6 Data Extraction

We created a data extraction form to extract detailed contents for each study. There were two categories of data extracted for each paper. First, common data such as bibliographic references, type of study, aim, research methodology and data analysis. Second, the specific data that answered each research question. Table 3.6 and Table 3.7 show the data that was extracted for both categories.

All extracted data was put into shared spreadsheets that were reviewed by a review team consisting of the main researcher and her supervisors. The main researcher was responsible for reading and extracting the data. In order to validate the extraction correctness, the research supervisors independently rated a random sample of papers according to the inclusion and exclusion criteria. All discrepancies on the data extracted were discussed between review team members with the aim of reaching a consensus. The reliability of the findings of this review was accomplished by considering only the quality score of relevant studies that are greater than 2.5 (50% of the percentage score) [57]. We did not measure inter-rater reliability since our review aimed for generalizability of the findings, in particular, to clearly describe how conclusions have been derived from the data instead of comparing agreements of the same codes or themes [56], [59].

Table 3.7. Specific data items extracted from all papers.

Search focus	Data item	Description
Quality of defect descriptions	Assessment method	Method used to conduct assessment (i.e. experiment, survey, case study)
	Participant	Participants involved in assessment (i.e. student, usability expert, software developer)
	Assessment criteria	Criteria used to assess the quality of defect description (i.e. clarity, impact, cause)
	Approach	Description of the assessment process
	Outcome	Results of assessment
	Recommendation	Suggestions for future work in related assessment results
Usability defect description content and format	Format	Format used to report usability defects
	Approach	Description of the format usage and characteristics of the format
	Content	List of attributes or information used in the usability description
	Limitation	Limitations of the format
	Benefits	Benefits of the format
	Evidence	The empirical evidence regarding the benefits of using a specified format to improve the quality of defect description
Reporting mechanism	Medium	Medium used to report and track usability defect reports (i.e. form-based reporting and end-user reporting)
	Characteristics	Description of the reporting mechanism
	Benefits	Benefits of the reporting mechanism
	Limitations	Limitations of the reporting mechanism
	Evidence	The empirical evidence regarding the benefits of the reporting mechanism to improve reporting process
Reporting guideline	Aim	The purpose of the guideline
	Guideline	Description of the guideline
	Benefits	Benefits of the guideline
	Limitations	Limitations of the guideline
	Evidence	The empirical evidence regarding the benefits of using the guideline to improve the quality of usability defect reports
	Recommendation	Suggestions for future work
Analyzing defect information	Focus	The purpose of the study
	Attributes used	The data used for analysis purposes
	Approach	Description of the study that related to the research question
	Benefits	The benefits of the study
	Limitations	Limitation of the study
	Evidence	The empirical evidence regarding the benefits of the study
	Recommendation	Suggestions for future work related to analysis results
Issue in defect reporting	Challenges	The problems in defect repositories related to usability defects
	Recommendation	Suggestions for future work related to improving usability defect reporting

3.2 Results

3.2.1 Classification Scheme

A classification scheme was developed to organize the retrieved studies on usability defect reporting. As shown in Figure 3.2, the classification scheme was structured to map onto our research questions. We categorized the studies using the process defined by Petersen et al. We started the classification process by analysing the title, keywords, abstract, and conclusions. We then compiled the keywords and phrases to build a high-level set of categories for classifying the papers. Finally, we grouped the phrases, research objectives and research findings of the papers in each category into a coherent set of themes.

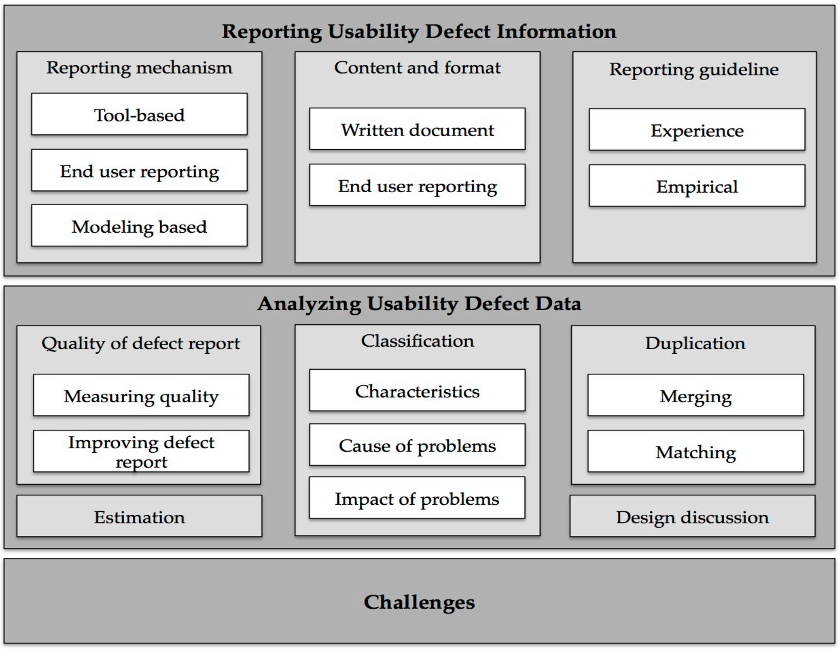


Figure 3.2. Classification scheme.

The classification scheme is composed of three main categories; 1) reporting usability defect information - which is related to research on reporting usability defects; 2) analysing usability defect data - which is related to researching the use of defect data; and 3) challenges – which refer to issues identified in usability defect reporting and management. Table 3.8 summarizes the distribution of the studies per topic. Studies that addressed more than one topic were classified repeatedly in each topic. For example, Nørgaard et al. [60] investigated mechanisms of usability defect reporting and challenges for each mechanism, and their study is counted in both topics.

Table 3.8. Summaries of research areas and topics.

Research areas	Description	Topic	Studies	Total
Reporting mechanism	Investigate how usability defects are collected and reported	• Tool-based reporting	P7, P10, P15, P17, P34, P41, P45, P49, P51, P55	13
		• End-user reporting	P18, P30	
		• Modeling-based support	P28	
Content and format	Investigate what attributes are used to describe usability defects	• Written document format	P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P24, P25, P29, P30, P32, P33, P35, P36, P37, P43, P47, P53	26
		• Learning-oriented format	P37	
Reporting guideline	Studies that assist reporter in reporting usability defects	• Experience	P2, P52	4
		• Empirical	P5, P37	
Defect report quality	Studies that analyzed defect reports for quality assessment	• Measuring usability defect reports quality	P1, P3, P4, P5, P7, P8, P35, P39, P44	11
		• Improving usability defect reports	P3, P7, P27, P35, P44, P48	
Classification	Studies that analyzed usability defect data to understand the characteristics of usability defects	• Usability defect characteristics	P22, P50	5
		• Cause of usability defects	P42	
		• Impact of software defects	P21, P46	
Duplication	Studies that use usability defect data for identifying similar usability problems	• Matching	P12, P13, P56	4
		• Merging	P36	
Estimation	Studies that use defect data to estimate defect discovery rate		P19	1
Design discussion	Studies that examined the structure and content of defect report discussion	• Addressing and resolving usability defects	P21	3
		• Structure and content of design discussion	P26	
		• Online forum	P31	
Challenges	Studies that discussed the issues of reporting usability defects	• Developer mindset	P1, P9, P37, P53, P54	22
		• Subjective bias	P1, P23, P240, P54	
		• Evaluator effect	P1, P5, P7, P8, P14, P20, P39, P53	
		• Defect discovery methods	P3, P20	
		• Complexity management	P26, P29, P40, P53	
		• Lack of appropriate channel for reporting usability defects	P6, P10, P20, P23, P38, P40, P57	
		• Lack of guideline for specific usability defect information	P29, P36, P54,	

In the following section, we present our answers to the review research questions based on analysis of the included studies. Each study is identified as P*m,* where *m* represents the study's number (see Appendix A for the list of studies used in this systematic review).

3.2.2 Usability Defect Reporting Mechanism

Three key types of reporting mechanisms were identified in the usability and software engineering literature. To effectively capture usability defect data, each reporting mechanism uses a variety of input designs, such as auto-generated data, predefined data, free-text form, and online help. Auto-generated data such as *tester name, timestamp*, and *problematic user interface* can be automatically recorded when the test is run and the report is submitted. In contrast, predefined data contains a variety of categorical data that is dependent on the input from the reporter. During a defect report submission, the reporter will select some values, such as *severity, type of defect*, and *heuristics* used. Some of these values can be changed over the defect life cycle, such as *severity*. For a free-text form, the reporter is allowed to write any description – comments, feedback, complaints, feelings or disappointment, steps to reproduce, expected and actual result – regarding the problems. In summary, the description of three reporting mechanisms can be described as follows:

Tool-based reporting is the easiest way to record and generate data as compared to a paper-based approach [61]. Tool-based reporting allows data to be collected instantly, and recorded data can be measured quantitatively, analysed for trends and used to generate feedback for quality improvement. A well-designed tool will assist users to provide sufficient data, thus, in turn, reduce missing data issues. Several tools in the usability evaluation field were developed to assist in usability defect reporting. Some key examples are outlined below.

- *Data Collection, Analysis and Reporting Tool (DCART)* [P7] uses auto-generated data and free-text form input design. The defect form was designed for collecting and organizing usability defect data in lab-based usability evaluation using a Usability Problem (UP) instance concept. Each occurrence of a UP found by multiple evaluators or multiple times by one evaluator is considered as the same UP. However, given multiple instances of UPs, evaluators must manually review and combine them to determine the main UP experienced by the users.
- *Web tool* [P10] uses tooltips, predefined data and free-form text input and was designed to record usability defects found during heuristic evaluation only. By having the tooltips and examples of usability problems, the evaluators get help on attributes and better guidance to assign severity and heuristics used to find problems. However, a non-integrated reporting tool with the software under test may trouble some users in switching between these two systems and users may bias certain values. In contrast, *Usability Reporting Manager* [P17] uses

predefined data and free-text form input. Using a web interface, reporters can enter, manage, and export data in into a defect tracking system connected to a source code repository.

- *Merging duplicate problem descriptions* [P41] helps evaluators to record usability problems into a database using different usability evaluation methods, to search the database for similar problems, exchange datasets, and to perform a meta-analysis of the datasets.
- *Usability Problem Inspector (UPI)* [P15] uses auto-generated data, predefined data and free-text form input incorporating usability action framework (UAF) content. UPI has two modes; task-based and free-based exploration. Using the task based-approach, evaluators are presented with a series of questions from the UAF structure. When a problem is identified, the evaluator is presented with a defect report form and the inspection path is automatically recorded. However, for free-exploration mode, no task information is recorded. *DESTINE* [P34] uses predefined data input. The tool is limited to evaluate the ergonomic quality of websites and it can support two types of user profile; expert and designer.

In the software engineering field, defect tracking systems are commonly used to record and track software defects, including usability defects. Our review only found four tools that explicitly assisted in usability defect reporting.

- *GUI monitoring and automated replaying* [P45] uses a generic non-intrusive GUI usage monitoring mechanism that can be integrated into existing applications. The monitoring of usage can produce actual usage traces that can be included in the defect reports and used as an input for replaying purposes. The traces are triggered by user interactions like mouse clicks or key presses.
- *GUI editor tool support* [P49] was developed on the Eclipse platform to support exploratory graphical user interface testing. The tool uses Eclipse logging to record uncaught exceptions during execution of a test and a cheat sheet viewer for evaluators to describe the observed failures. The test results are available in the form of a results file and can be automatically exported into the defect repositories.
- *Timeline tool* [P51] was developed to visualize monitored interaction traces and application events preceding failures. Using the tool, software developers may analyse the traces to derive steps to reproduce by manually replaying the monitored user interactions.

- FUSION [P55] was developed to produce more reproducible defect reports than traditional defect tracking systems. Using the event-driven paradigm of Android application, the tool aids the reporter in constructing the steps needed to reproduce a defect by making auto-completion suggestions based on the potential GUI actions, such as click (tap), long click (touch), type, and swipe.

End-user reporting tools collect information in much simpler forms to address users' frustration and complaints, and the users report defects as part of their day-to-day activities. We identified two approaches of designing end-user reporting.

- *One-bit-feedback* [P18] uses auto-generated data and a free-text form input. It is a background process that monitors certain system characteristics and packs them into an incident report whenever the user clicks on the screen button or punches the hardware button. The reports are stored locally on the user's system. Usability defect data is collected using auto-generated data and user is given the opportunity to provide comments and feedback through a free-text form. Using this approach, defect incident is automatically recorded and requires less data entry.
- *Two-mouse-click* [P30] uses auto-generated data, predefined data and free-text form input design. The prototype was developed to allow report submission with minimal user click and supplements user comments with objective program state information. The program only collects information relating to the user's interaction. No sensitive information is sent.

Modelling-based reporting provides a standard description with more structured data. The reporter uses a modelling language with defined notation to represent information. For example, ErgoPNets [P28] uses a formalism that combines Petri Nets and ergonomic criteria to describe ergonomic problems and their recommendations. The method uses icons, graphical representation and text to describe problems. In this way, the complex usability defect descriptions can be unified into a single model.

3.2.3 Usability Defect Reports Content and Format

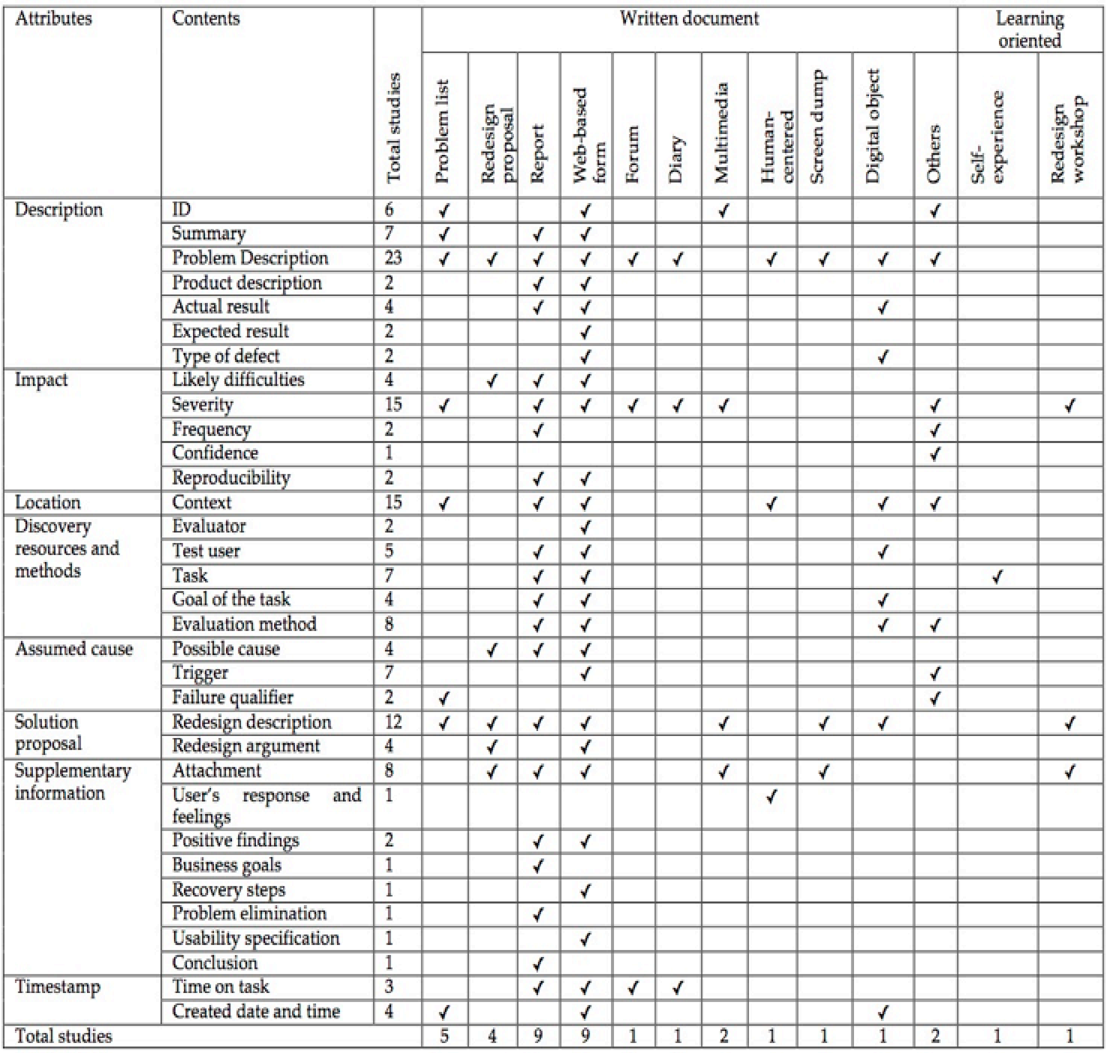
13 usability defect description formats were identified from the selected studies. Eleven out of the 13 formats are presented in written documents, while the other two are learning-oriented formats. These

formats are associated with a list of attributes for communication and report keeping. Altogether, 33 attributes are identified across 13 formats by a total of 26 studies. As shown in Table 3.9, we classified these attributes into eight groups based on the objective of defect description content objective, and we summarize all the formats and attributes in Table 3.10.

Table 3.9. Categories of usability defect attributes.

Group	**Attribute/ Contents**	**Description**
Description	Identifier	Unique defect identification number
	Summary	A headline summarizing the problem [P35]
	Problem description	Concise description of the usability problem
	Product description	Description of the product and the intended users of the product [P11]
	Actual result	Description of errors made [P32]
	Expected result	Describe what evaluator expected the system to do [P6]
	Type of defect	Details of the problematic subject (i.e. number, background0 [P32]
Impact	Likely difficulties	The anticipated difficulties the user will encounter as a consequence of the problem [P16]
	Severity	Indicates what effects the usability problem had on the user [P33]
	Frequency	Number of users/ experts that experienced/predicted a usability problem [P33]
	Confidence	Indicate how confident evaluator believed that the usability problem identified was true
	Reproducibility	Ability to reproduce the problem and make it happen again [P6]
Location	Context	Describe in what part of the user interface the user was when the usability problem occurred [P33] Specific interface component (i.e. logon component, calendar component) [P32] The specification of the environment used for the evaluation, including the location and any hardware (computers, monitors, cameras) [P17]
Discovery resources and methods	Evaluator	Author of the usability problem description (i.e. a meta data that automatically captured the evaluator’s name)
	Test user	Number of users who participated in the evaluation and the criteria by which they were selected [P11] Description of the intended user [P32]
	Task	Specific tasks that the participants were asked to perform during the evaluation [P11]
	Goal of the task	The goal of the reported test [P11]
	Evaluation method	Usability evaluation method [P33]
Assumed causes	Possible cause	Describe the cause(s) of the problem based on the evaluator’s judgment [P16]
	Trigger	Describe what a user is doing when she/he discovers usability problems (task scenario, heuristics) [P33] [P47] Criteria used to justify the usability problems [P36]
	Failure qualifier	Describes how the user/expert experienced a usability problem (i.e. missing, incongruent mental model, irrelevant, wrong, better way, overlooked) [P33]
Solution proposal	Redesign description	A description of how to remedy the problem [P35]
	Redesign argument	Justification of the redesign proposal [P37]
Supplementary information	Attachment	Logfiles, screenshot, questionnaire [P9]
	User’s response and feelings	Narrative description with strong positive or negative connotations, metaphors or stories (i.e. video of users struggling with an application) [P37]
	Positive findings	Include positive comments on the usability of the site [P2]
	Business goals	Justification why business goals were jeopardized by the problem [P35]
	Recovery steps	Description of how the user recovers from the usability problem [P6]
	Problem elimination	Justification of why any problem discovered should warrant elimination [P16]
	Usability specification	Usability requirement under test
	Conclusion	Summary of the report
Timestamp	Time on task	Completion rate (mean time on task [P33]
	Created date and time	A meta data when the problems are reported

Table 3.10. Summary of usability defect attributes used in 13 formats. *Problem description* was the most used attribute across the thirteen formats.



Attributes	Contents	Total studies	Written document											Learning oriented	
			Problem list	Redesign proposal	Report	Web-based form	Forum	Diary	Multimedia	Human-centered	Screen dump	Digital object	Others	Self-experience	Redesign workshop
Description	ID	6	✓			✓			✓				✓		
	Summary	7	✓		✓	✓									
	Problem Description	23	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		
	Product description	2			✓	✓									
	Actual result	4			✓	✓						✓			
	Expected result	2				✓									
	Type of defect	2				✓						✓			
Impact	Likely difficulties	4		✓	✓	✓									
	Severity	15	✓		✓	✓	✓	✓	✓				✓		✓
	Frequency	2			✓								✓		
	Confidence	1											✓		
	Reproducibility	2			✓	✓									
Location	Context	15	✓		✓	✓				✓		✓	✓		
Discovery resources and methods	Evaluator	2				✓									
	Test user	5			✓	✓						✓			
	Task	7			✓	✓								✓	
	Goal of the task	4			✓	✓						✓			
	Evaluation method	8			✓	✓						✓	✓		
Assumed cause	Possible cause	4		✓	✓	✓									
	Trigger	7				✓							✓		
	Failure qualifier	2	✓										✓		
Solution proposal	Redesign description	12	✓	✓	✓	✓			✓		✓	✓			✓
	Redesign argument	4		✓		✓									
Supplementary information	Attachment	8		✓	✓	✓			✓		✓				✓
	User's response and feelings	1								✓					
	Positive findings	2			✓	✓									
	Business goals	1			✓										
	Recovery steps	1				✓									
	Problem elimination	1			✓										
	Usability specification	1				✓									
	Conclusion	1			✓										
Timestamp	Time on task	3			✓	✓	✓	✓							
	Created date and time	4	✓			✓						✓			
Total studies			5	4	9	9	1	1	2	1	1	1	2	1	1

Note that the attributes checked for each format does not mean that all these attributes are present in the format at any one time. Rather, it is a compilation of several studies that mention the use of certain attributes for a particular format. The most common format used to report usability evaluation findings were *web-based form* and *report*. The use of variety input design techniques such as auto-generated data [P7, P15, P17, P18, P30], pre-defined data selection [P10, P15, 17, P30], free-form text [P6, P7, P15, P17, P18, P30, P47] and question-based [P6, P47] can produce more structured and consistent defect information. Common attributes collected in the web-based form are *problem description*, *location* of the problem in user interface, specific *task* where the problem was observed, what *triggers* the problems, and a *severity* rating. There are several key features of web-based reporting that make it easy-to use with little manual effort. Examples include features for reminding users about key information to report [P7], online help and tooltips for quick reference [P10], support for data transformation into different formats [P17], and automatically recorded system generated information [P15]. However, users are exposed to erroneous data entry due to the cognitive load and biased use of default data.

In contrast to web-based forms, conventional *reports* contain unstructured content and a large amount of information. The report generally gives a detailed description of usability evaluation methods that have been conducted so that the content of the report will not only be able to justify the situation of the problems encountered but to present a good argument to management for requesting resources allocation [P9]. Other attributes commonly reported in conventional reports are *problem description, severity rating*, and *attachment*. The *Problem list*, on the other hand, provides lightweight documentation. Even though the content is briefer and lacks contextual information, it is useful to support ongoing discussions [P53], and helps to prioritize tasks during the problem merging process [P37]. In this way, complex problems could be described as multi-faceted without going into a detailed report [P13]. This format usually requires a *problem description* and *severity* rating.

The *redesign proposal* is more focused on a problem solution. It gives ideas on complex problems by providing concrete recommendations and arguments. The recommendations are usually supplied with drawings or code fragments [P53]. While software developers may prefer redesign proposals, it is difficult to write useful recommendations for major changes, especially problems that involve business and technical constraints [P29].

The other nine formats are less commonly reported - these include forum and diary [P6], multimedia [P6, P37], human centred story [P37], screen dump [P37], digital objects [P32], self-experience [P37], and redesign workshop [P37]. Even though these formats do not use a variety of attributes, *multimedia* and *redesign workshop* format, for example, provide persuasive and well-balanced defect description [P37]. Two studies were categorized as “other” format as they do not clearly state the format used but did mention attributes used to extract usability defect data [P33, P36].

In terms of content, we found that *problem description, severity, context,* and *redesign description* were the four attributes most commonly used to describe usability defects across formats. Attributes that rarely present in usability defect descriptions are *product description, expected result, type of defect, frequency, confidence, reproducibility, evaluator, failure qualifier, user’s response and feelings,* *positive findings, business goals, recovery steps, problem elimination, usability specification,* and *conclusion*. These attributes are often captured in report and web-based forms.

23 studies primarily used *problem description* to report usability defects across ten formats, except multimedia, self-experience and redesign workshop. However, it is uncertain whether the *problem description* is mutually exclusive – that is the attribute has only one value. This is because *problem description* has a very vague definition in which the reporter has a probability of mixing it with other attributes such as *possible cause, type of defect,* and *user's response and feelings* [P7]. Sometimes, *problem description* is very brief such as in the problem list format. To support this issue, a *redesign description* can additionally persuade the relevance of usability defects. We found twelve studies that addressed detailed redesign proposals, but only four studies supplied an in-depth justification for why the proposed solution is necessary.

Fifteen studies emphasized the *severity* rating in eight formats. However, there is no standard definition used to indicate severity assessment. Some studies have used seriousness [P3, P13], category [P15] and impact [P33] to describe the same meaning. Additionally, there are several severity schemes used for the rating purpose such as 1) minor, serious, critical [P5], 2) major or minor [P8], and 3) severe, moderate and minor [P10]. In terms of software *context*, fifteen studies mentioned the problematic location of elements in the user interface. This information can be either automatically collected [P15, P18, P30] or manually specified by the reporters [P6, P7, P10, P13, P14, P17, P33, P37]. Among the 13 formats, only redesign proposal, report, web-based form, multimedia, screen dump, redesign workshop provided *attachments*. The attachment can be log files [P8], core dumps [P9], screen image [P12, P18], webcam picture [P18] and video clip [P37].

3.2.4 Usability Defect Reporting Guidelines

The review uncovered four guidelines that suggest the way that usability defects should be reported, as shown in Table 3.11. Two studies provide experience-based guidelines that originated from practical lessons and usability experts' point of view [P2, P52]. According to Dumas et al., the way the usability defect is communicated to developers influences the acceptance of the usability defects. Instead of complaining about the negative aspect of the software product, the usability description should also address the positive findings in a clear and precise form.

Another two guidelines were constructed through empirical studies [P5, P37]. Among the four guidelines, the Capra et al. [P5] guideline is the most rigorous and complements Dumas's guideline. Capra's guideline was developed based on a survey of usability practitioners. This guideline is widely used as a criterion to evaluate the quality of a usability description [48], [62], [63]. Besides that, the guideline may be used in training usability evaluators and as a checklist when writing a usability defect description. Meanwhile, the Nørgaard et al. [P37] guideline is based on Toulmin's model of argumentation and Aristotle's three modes of persuasion.

Table 3.11. Guidelines for writing usability defect reports.

Studies	**Guidelines**
Dumas et al. [P2]	Emphasize positive Express your annoyance tactfully Avoid usability jargon Be as specific as you can
Capra [P5]	Describe a solution to the problem Be clear and precise while avoiding wordiness and jargon Describe the cause of the problem Support your findings with data Help the reader sympathize with the user's problem Describe the impact and severity of the problem Describe observed user actions Be professional and scientific in your description Consider politics and diplomacy Describe your methodology and background
Nørgaard et al. [P37]	Provide evidences for the observed problems Describes the underlying assumptions that must be agreed upon before the claim can be accepted Provide argument's purpose or position – the difficulty arises from the problems Provide insightful remarks and conclusions about the system being evaluated Use log data or statistical data from a user test as backing for a usability problem Show videos of users struggling with an application or even letting the developers experience the problem themselves
Avnon et al. [P52]	Log one defect for each problem Clearly document each defect Include the visuals – observed problems and the intended design Include violated usability interface guideline Include prototype to visualized complex interactions Prioritize the problems

3.2.1 Analyzing Usability Defects

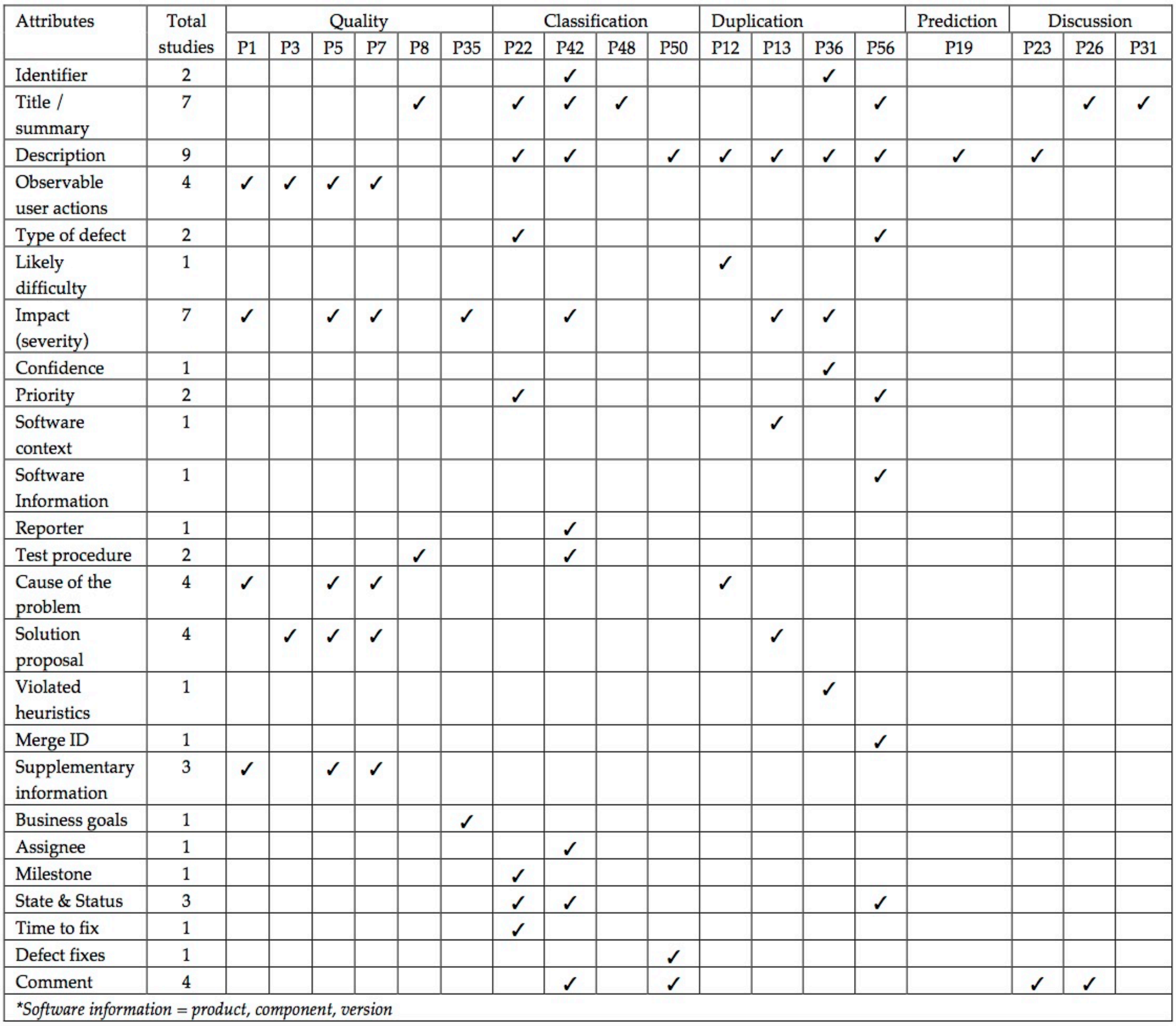
There is a body of research focusing on usability defects for understanding and improving the defect management lifecycle. Our systematic literature review identified five areas that specifically investigate usability defects – quality of usability defect reports, classification of usability defects, duplicate defect report analysis, estimation of usability defects, and discussion of usability defect reports. In terms of attributes that are commonly used in empirical research (refer Table 3.12), problem *description*, *impact*, and *title/ summary* are most widely used. Attributes rarely used by researchers are *type of defect, likely difficulty, confidence, priority, software context, reporter, violated heuristic, business goals, assignee, milestone, time to fix*, and *defect fixes*. Research on classification and defect duplication favourably used *title/summary* and *description*, while research on defect report quality often used *observable user actions, impact*, *assumed cause*, and *supplementary information*. However, two studies did not report defect attributes used as they employed other metrics such as ISO/IEC 9126 quality model [P21] and IBM quality measurement model [P46].

3.2.1.1 Quality of usability defect reports

We identified two key topics in this research area. First, studies that measure the quality of usability defect reports. In general, defect report produced by an expert evaluator had better quality than the defect reports produced by a non-expert evaluator. In most studies, non-expert evaluators are recruited among students, while expert evaluators are from industrial practitioners.

Based on the eight studies, we identified numerous criteria used to measure the quality of usability defect reports (see Table 3.13). We classified these criteria into three categories: *report content, software quality model*, and *general* categories. One study did not mention any assessment criteria that were used [P39]. *Report content* was used by six studies to measure the quality of usability defect reports [P1, P3, P5, P7, P8, P35]. Among the defect attributes, observable user action, impact, supplementary information, assumed cause and solution proposal were the most assessed information. Only one study measured the quality of the usability defect report content using test procedure descriptions, executive summary and report layout [P8] and business goals [P35]. Five studies revealed that non-expert usability evaluators have difficulty in describing certain usability defect information, particularly the impact, solution, supplementary information, assumed cause, and recovery steps [P1, P5, P8, P39].

Table 3.12. List of attributes used across five research areas. *Problem description, title,* and *severity* were the most used attributes.



Attributes	Total studies	Quality						Classification				Duplication				Prediction	Discussion		
		P1	P3	P5	P7	P8	P35	P22	P42	P48	P50	P12	P13	P36	P56	P19	P23	P26	P31
Identifier	2								✓					✓					
Title / summary	7					✓		✓	✓	✓					✓			✓	✓
Description	9							✓	✓		✓	✓	✓	✓	✓	✓	✓		
Observable user actions	4	✓	✓	✓	✓														
Type of defect	2							✓							✓				
Likely difficulty	1											✓							
Impact (severity)	7	✓		✓	✓		✓		✓				✓	✓					
Confidence	1													✓					
Priority	2							✓							✓				
Software context	1												✓						
Software Information	1														✓				
Reporter	1								✓										
Test procedure	2					✓			✓										
Cause of the problem	4	✓		✓	✓							✓							
Solution proposal	4		✓	✓	✓								✓						
Violated heuristics	1													✓					
Merge ID	1														✓				
Supplementary information	3	✓		✓	✓														
Business goals	1						✓												
Assignee	1								✓										
Milestone	1							✓											
State & Status	3							✓	✓						✓				
Time to fix	1							✓											
Defect fixes	1										✓								
Comment	4								✓		✓						✓	✓	

**Software information = product, component, version*

Table 3.13. Categories of assessment criteria to measure quality of usability defect reports.

Category	Assessment criteria	Rank*	Study (s)
Report content	Impact	2	P1, P5, P7, P35
	Supplementary information	3	P1, P5, P7
	Assumed cause	3	P1, P5, P7
	Observable user actions	2	P1, P3, P5, P7
	Solution proposal	3	P3, P5, P7
	Test procedure description	5	P8
	Executive summary	5	P8
	Business goals	5	P35
	Report layout	5	P8
Software quality model	Clarity	1	P1, P3, P5, P7, P8, P35, P44
	Persistent	5	P3
	Justified	5	P3
	Persuasive	5	P44
	Usefulness	4	P35, P44
	Data quality	5	P8
General	Expert judgement	5	P4, P7

(*) The ranking showed the assessment criteria used the most in measuring quality of usability defect reports (in ascending order)

With regards to the *software quality* criteria, seven studies used clarity attributes to assess if the usability defects were described precisely, meaningfully, and contained unambiguous explanation [P1, P3, P5, P7, P8, P35, P44]. We also observed that even non-expert evaluators failed to fulfil all report content criteria, but they could describe some information clearly [P1], and provide positive findings for the evaluation [P39]. However, these studies did not indicate which information non-expert evaluators could explain precisely. Other studies uniquely defined their quality attributes such as persistent [P3], justified [P3], persuasive [P44], and usefulness [P35, P44].

The *general* category comprised of expert judgment. Studies that relied upon expert judgment measured quality using criteria such as how similar problems were identified and use of appropriate claims to justify the problems. In order to minimize judges' bias, measures of association, bias, and distribution of the judgement were calculated. In our review, two studies employed professional judge ratings [P4, P7].

The second key topic is emphasizing the ways to improve usability defect descriptions. There are several aspects of possible improvement. To improve defect report content and structure, Hornbaek and Frøkjær [P3] recommend four elements for good usability defect reports: 1) include solution proposal, 2) justify why something is a problem by referring to the behavioural consequences of a problem, 3) present descriptions of problems that are complex and persistent for users, and 4) make the problem description long enough. In a different work [P35], they proposed the use of business goals in justifying usability defects, as the information would give higher utility and impact to the company. Furthermore, they found that business goals help focus the evaluation. Ko et al. [P48] suggested that the defect report title should consists of software behaviour, relevant quality attribute, problem, execution context, and if the report is a defect or feature request.

In terms of usability defect report format, software developers highly prefer a multimedia presentation, screen dump, and redesign proposal for presenting usability defects because they provide ideas on the problem context [P44]. To improve a usability defect reporting tool, Faaborg and Schwartz [P27] proposed the adaptation of usability heuristics when labelling usability defects. In fact, a usability-shared vocabulary (such as consistency, jargon, feedback) can be useful in describing the cause and impact of user interface problems. However, this approach is highly dependent on the clarity of each heuristic's definition and use of good examples, as users of defect reporting tools may have limited HCI knowledge. To effectively capture usability defects, Howarth et al. [P7] proposed the usability problem instance approach to record usability defects. Using this approach, they found out that expert judgment provided higher ratings for describing the cause of the problem and solution proposal description. Hornbaek and Frøkjær [P3] found that usability evaluation methods influence the level of detail of defect description. For instance, problems identified with the metaphor of human thinking are more justified compared to problems found with testing aloud.

3.2.1.2 Classification of Usability Defects

In existing defect repositories, defects are classified as either a defect (blocker, critical, major, minor, normal) or an enhancement. However, this labelling scheme does not have sufficient knowledge for understanding the properties and features of various types of usability defects. This is evidenced by a number of studies available in the literature [P21, P22, P42, P46, P50]. We group the research into different goals below.

- *Understanding usability defect characteristics* - Lal and Sureka [P22] investigated the differences, similarities, and correlation between terms and usability defect types. They found that terms present in defect report titles and description are related to usability defect type. For instance, usability defects frequently use terms such as “window”, “user”, “zoom”, “menu”, and “click”. In relation to usability defects, they discovered that 1) usability defects are the largest contributor to regression defects, 2) the median mean time to repair (MMTR) value for usability defects is fairly high compared to other defect types (cleanup, crash, polish, performance, regression, and security), and 3) usability defects are the second highest of duplicated defect reports. Xia et al. [P50] studied the relationship between types of defects and severity. They discovered that most user interface and usability defects are assigned as block and critical severity.
- *Understanding the cause of usability defects* - Li et al. [P42] developed a classification model for classifying defects to root cause, impact and software component. They found that graphical software is threatened by graphical user interface (GUI) defects that are mostly caused by semantic errors, such as missing features and wrong functionality.
- *Understanding the impact of software defects*– Vetro et al. [P21] conducted an experiment to classify software defects according to ISO/IEC 9216 quality model (functionality, reliability, usability, efficiency, maintainability and portability). They found functionality and usability

were the most dominant impacted quality attributes. Kreyss et al. [P46] used the IBM quality measurement to categorize defect report distribution. Across the nine quality attributes (capability, usability, performance, reliability, installability, maintainability, documentation, serviceability, and overall usability) was ranked as the second highest problematic quality attribute. The results from the study gave an overview of where improvements should be focused.

3.2.1.3 Duplicate Defect Report Analysis

Many previous studies have reported how duplicate defect reports of any sort may slow down the defect fixing process as more resources and time are needed to identify and close duplicate defects [64]. However, duplicate defects should not be ignored because they may contain additional information that may be useful to resolve defects [20]. With regards to the latter concern, we identified four studies that addressed a way specific usability detects are detected and handled. In the usability engineering literature, duplicate defect report classification is referred to as matching and merging, and does not appear to be an area of active research.

- *Matching* – a process to detect duplicate problems. Vermeerena et al. [P12] analysed the usability problem's similarity based on the situation in which the problem occurred, the user's observable behaviour at the time the difficulty occurred and how the user thought, felt or understood certain tasks. Hornbæk and Frøkjær [P13] studied four matching techniques (similar changes, practical prioritization, a model of Lavery et al. (1997) and the User Action Framework). Their experiment showed that similar changes produced more single problems than the other techniques, and practical prioritization reaches highest level of agreement among novice evaluators. Hindle et al. [P56] used different contextual features – architecture words, non-functional requirement words, LDA topic words, and random English words - to improve the accuracy of defect report deduplication. Their experiments demonstrated that the effectiveness of domain-specific context could improve the quality of duplicate defect detection.
- *Merging* – a process to consolidate similar problems. When similar problems are identified, they must be linked to the primary report the current duplicate refers. To address this process, Law and Hvannberg [P36] used a manual merging process, where evaluators recorded every

change made to the usability problems in their own consolidated list. The results of their study found that the merging process is influenced by the evaluator effect, in which the merging rate and severity increased when evaluators performed merging process in a group. However, confidence level, which is influenced by personal experience, does not fluctuate with the merging process.

3.2.1.4 Estimation of Usability Defects

We only found one study that used usability defect data to determine defect discovery rates. Using Good-Turing discounting with a normalization procedure, Lewis [P19] revealed that higher levels of description produce a higher estimate of discovery rate.

3.2.1.5 Design Discussion of Usability Defects

We found only three studies in the software engineering literature that focused on correction discussion and had goals concerning user interfaces and interaction design.

- *Addressing and resolving usability defects* – Twidale and Nichols [P21] identified two topics commonly discussed by users: 1) debate about the validity of usability problems; and 2) critiques and refinement of candidate solutions. They also expressed concerns about usability defect solutions that may introduce ripple effects.
- *Understanding structure and content of design discussion* – Ko and Chilana [P26] observed trends in online design discussions including establishing scope, proposing ideas, identifying design dimension, defending claims with rationale, moderating process, and making decisions. However, the temporal presentation of discussion comments was inadequate to support proposals and critiques among a broad range of users.
- *Supporting online forums* – Raza et al. [P31] discovered that the open source community works in a collaborative environment to identify and find possible solutions to usability defects. The number of active mailing lists and messages posted on online forums indicated significant and active support from open source community.

3.2.2 Issues in Defect Reporting

Addressing the identified challenges of existing usability defect reporting processes and tools serves as a basis for any improvement in software defect management practices. We identified these reported challenges from both software engineering and usability engineering studies. From the software engineering perspective, these challenges include difficulties faced by reporters to report, track and manage usability defects in existing defect tracking systems. Most challenges identified in the usability engineering field are related to human factors and usability evaluation methods, while challenges in software engineering field are due to limitation of existing defect repositories. Key reported challenges we found from the previous studies are summarized below.

3.2.2.1 Developer mindset

One of the prominent dilemmas among evaluators is when their usability defects reported get a lot less attention than they think they deserve from software developers. This situation seems to happen when software developers cannot understand the problems, especially when they do not participate in the evaluation or witness how users struggle to accomplish certain tasks [P1, P9, P37]. In some cases, software developers do not always agree with the higher severity ratings of usability defects given by reporters. In fact, software developers usually assess severity somewhat differently from reporters, and usability defects often end up with low severity rating and lower priority than functional defects [P37, P53]. Therefore, comparing usability defects in the context of functional defects is impractical as usability defects can be overlooked [P54].

3.2.2.2 Subjective bias

Evaluating usability aspects of a system is highly subjective to an individual and thus the reporter [P26, P40]. That is, one might see one aspect of an interface as problematic, but others may not. It is thus difficult to persuade software developers or designers that the usability defects raised are indeed real defects, that they require the same attention as functional defects, and need fixing. In fact, an agreement/disagreement between severity ratings is also seen as an effect of subjective bias by software developers or designers evaluating their own designs [P1, P23]. This has raised questions as to whether usability defects should be reported into a shared defect database or usability defects should have their own database [P54].

3.2.2.3 Evaluator effect

Our review observed that the way usability defects are described is influenced by skill and experience levels of the evaluator. From the usability engineering literature, most studies reported that expert usability evaluators are better at identifying and describing usability problems than the software developers or novice evaluators [P1, P5, P7, P8, P14]. It should be noted that inexperienced usability evaluators might feel that not all problems should be reported, and when they find a problem they do not know what information should be reported. This challenge has led to incomplete usability defect descriptions, in which a report usually does not contain possible causes of the problems, recovery steps, possible solutions, and clear reasons why something is a problem [P20, P39, P53].

3.2.2.4 Defect discovery methods

The completeness of usability defect descriptions also depends on the defect discovery methods used. [P3, P20] reported that certain methods, such as metaphors of human thinking, are more likely to have more information to justify a problem found compared to think aloud method. In other words, selection of appropriate testing techniques may help evaluators to identify usability problems effectively and collect necessary usability defect information to report. However, in open source projects where often no formal usability testing is conducted, there is still a lack of mechanisms to discover and report usability defects, especially those encountered by typical or non-experienced users [P20].

3.2.2.5 Complexity management

The process of managing usability defects is a largely human task, especially when discussing design solutions in defect repositories. There are two aspects of complexity in managing usability defects we found from literature. First, the linear temporal discussion structure may not be sufficient to enable users to keep track of all the discussion elements, such as elaboration, confirmation, allocation of works, proposed fix, and revision [P40]. This makes it difficult for users to compare and critique a correction proposal, as they have to read through the entire comments. One way to minimize this challenge is to use nested comments [10], [42], [65] so that the critiques in the design discussion can be more explicit. Second, the changes to interface design might be risky, as any changes may have impacts on the other components of the system [P40], cause confusion to existing users [P26], and may involve major changes to business and technical constraints [P29, P53]. In this case, some usability defects are difficult to explain and proposing useful and usable recommendations may be hard.

3.2.2.6 Lack of appropriate channels for reporting usability defects

Existing defect repositories, such as Bugzila, Trac and JIRA, were designed as text-centric mediums for functional defect reporting. This causes some usability defects that relate to user's feelings, emotions and "struggling" with an interface to be difficult to explain textually [P20, P40, P57]. To overcome this limitation, defect repositories, such as Bugzilla could have a mechanism to easily and interactively record, upload, show, maintain, and comment user submitted videos, images and voice [P10, P38, P57]. Furthermore, some defect repositories that were developed by and for software developers have caused usability defect reporters to fill in considerable amount of information, much of them not relevant for usability defects [P6, P38, P57]. Considering these challenges, several studies have suggested a mechanism to support non-expert users in terms of automated collection contextual metadata and cognitive information [P20, P23] and less user registration [P40].

3.2.2.7 Lack of specific guidelines for usability defect information reporting

Although generic defect report templates and evaluation reports are available, most of them do not clearly define specific information that should be reported for usability defects in general and different kinds of usability defects [P29, P36, P54, P57]. For example, in assigning usability defect severity, there are no standard guidelines and rules available. According to [P36], users usually use their personal experiences as a benchmark to judge problem severity. Similarly, a lack of guidelines and exemplary recommendations make the quality of fix recommendations highly varied [P29].

3.2.3 Key Recommendations

In this section, we draw from our review findings and the findings of other studies and surveys to provide a set of key recommendations for further research in usability defect reporting. These provide a road map for further usability defect reporting research and while many are complimentary, we order them roughly in our suggested priority order to address.

3.2.3.1 Recommendation R1 - Prioritize usability defect attributes by their level of importance for software engineers

There are many separate usability defect attributes that we have identified from usability engineering studies (33 attributes). Many of them do not appear to be important for understanding, replicating or correcting the usability defect from a software engineering perspective. Since a key aim in our research was to simplify and improve the defect reporting process, we have to identify which of the attributes have the greatest influence on defect fixing process. We could focus on capturing the ones that will have the greatest impact in convincing software developers of a problem and assisting them in prioritizing, diagnosing cause, and correcting. Related to this, we found little work on how to best prioritize usability defect reports to provide best value to end users i.e. fix those most seriously impacting usability first. As above, this requires better ways to characterise usability defects, classify, determine severity, and convey this to software project managers and developers.

To advance these important research and practice outcomes a detailed survey and interviews with a large number of software engineers is needed to determine critical attributes for them. Additionally, understanding better the difference between usability defect reporter and consumer perspectives is essential. Improved usability attribute terminology and understanding in terms of impact on usability defect description and diagnosis is also needed. Mining existing defect repositories to understand what attributes seem to lead to improved correction may also assist this.

We partially addressed this recommendation in Chapter 5.

3.2.3.2 Recommendation R2 – Provision of key usability-related defect attributes

Following on from R1, we observed that many of the usability defect description formats in use do not define separate attributes to indicate specific key information about a usability defect. This results in many software developers with little experience reporting "usability" issues finding difficulties in understanding the reported issue. This means that software developers do not always agree or understand the usability defects actually reported, even if reported at all. As a result, usability defects get less attention or are sometimes even closed off as not valid.

One way to overcome this issue is to define and capture usability defect attributes at a fine-grained level, which can reveal more detailed issues with usability characteristics, such as heuristics, defect category, location and impact. Additionally, by introducing dedicated fields/ attributes to address likely interaction difficulties, the end user's feeling, and how they see an interface as problematic – so that the usability "struggles" exemplified in the usability defect reported can be better understood and appreciated by software developers using the usability defect report. This information can be used by project managers and software developers for defect management purposes, as well as providing researchers with richer information to conduct empirical analysis of usability defect cause, impact, tracking, and resolution.

In order to identify critical usability defect attributes to report, research needs to be carried out to determine both: (1) what reporters are reporting and think they should be reporting, and what developers require in order to fix usability defects; and (2) what usability defect attributes actually impact defect understanding and correcting. This could be done via surveys and interviews of reporters and developers, to get opinions of attributes required, and mining of existing defect repositories, to understand what is being reported and its impact on resolution.

To minimize unnecessary or irrelevant attributes for a particular usability defect, usability defect reporting forms could be adjusted using for example, a contextualised question-based design so that a reporter can select specific attributes that are relevant to them.

We addressed this this recommendation in Chapter 5 and Chapter 7.

3.2.3.3 Recommendation R3– Provide reporters customised usability defect report forms

The static reporting template offered by most functional requirements-oriented software defect repositories is generally universal. These do not consider the influence of the different types of reporters, different kind and use of diverse usability evaluation methods, and the phase of development where the usability defects are found. Almost all research shows that all defect types are reported using the same generic defect reporting template. In some cases the information requested on the form is simply not relevant and some is beyond the reporter's knowledge [7]. Most are text only and do not support other forms of input collection, or make it difficult to capture and attach.

A number of enhancements to existing reporting tools have been suggested in the literature [P45, P49, P51, P55], or can be deduced from the related usability defect reporting issues discussed above. We think that using a guided reporting method where reporters are assisted with predefined attributes for input selection, online help and question/wizard-based interaction may greatly improve capture and quality of usability defect reports [66], [67]. In this way, even if the reporter has less knowledge about usability, they can still be guided to capture reasonable quality defect reports. As a result, the recorded data will be more structured, fine-grained and uniform for usability defect report management. Users should be prompted/allowed to capture relevant attributes based on types of usability defects, the reporter's profile (e.g. non-technical user, technical user, usability experts, and etc), and usability defect report attributes be prioritized based on the types of defects and relevancy.

Additionally, usability defect report forms should be simple. Simplicity – the art of minimizing the amount of requested attributes in a usability defect report – is a necessary quality focus, by including only what software developers need rather than what reporters think – to make it easier for software developers to understand, replicate and fix the problems [P55]. Giving a reporter a simple set of explicitly usability-focused defect reporting forms for different kinds of defects could encourage them to report more usability defects with better outcomes, rather than imposing on them many complex, irrelevant attributes.

Another issue in usability defect reporting is that usability engineering tools and techniques are quite distinct from software engineering defect repository reporting and management tools and software engineering unit testing methods. This can cause repeated defect reporting when transferring usability defect information found during formal usability evaluations to project defect repositories, a waste of time, and possibility of information loss. Having a standard format that can be shared between the usability and software engineering communities would add value. However, further research to empirically study the impact of using separate and shared defect repositories would suggest a better usability defect reporting approach.

A further area for future research is to investigate the key factors influencing quality usability defect reporting, from the perspective of non-technical reporters. Using this knowledge, how can next generation usability defect reporting tools be better designed to leverage HCI knowledge, domain knowledge and end user knowledge?

We addressed this recommendation in Chapter 4 and 7.

3.2.3.4 Recommendation R4 - Develop an improved taxonomy for classifying usability defects

There are currently many usability defect taxonomies, classifications and attributes of usability defects identified in HCI literature and software engineering literature. We found many studies identifying that many usability defect reports lack sufficient attributes for classifying usability defects. A key obstacle of using existing usability defect report data is the widespread use of unstructured textual features in most current defect tracking systems. Lack of usability knowledge or different usability knowledge among reporters has produced reports that use a wide range of non-standard usability terms that complicates usability defect classification and identification. In addition, existing defect report attributes do not capture usability related information that can be directly used to filter usability defects. We observed only two studies [P30, P32] that specified the types of defects to describe usability defects across the 13 formats we identified.

There are several reasons for classifying usability defects: 1) to better identify and disclose the probable causes of the defect; 2) to highlight the impact of usability defects on the intended user task outcome; 3) to treat usability defect priorities the same way as for other defects; and 4) to quantitatively track usability defects, defect impact and defect resolution over time.

We also observed a great deal of inconsistency in the terms used in usability defect reports for specifying the same usability defect across the 13 formats found. For instance, a "severity" attribute was used in most of the formats to denote the importance of the defects to be fixed [68]. However, other than severity, some studies used impact [P12, P16], seriousness [P13, P53] and category [P5] to refer to severity. This variation of terms for one usability attribute can also be found in use of "minor, major, enhancement" for a defect's severity, while others used "severe, critical", which resulted in inconsistent data which was not comparable. Many other usability attributes are used inconsistently in terms of both name and value. This leads to inconsistent reports even within the same project which are hard to read, understand, track, and prioritize.

To solve these issues, the HCI and software engineering communities need to develop a more comprehensive and agreed usability defect taxonomy. Much of this work has been established in terms of HCI usability evaluation terminology and attributes, but has been inconsistently applied or not applied at all in software engineering practice around usability defect reporting. Along with comprehensive, agreed usability defect taxonomy, an agreed set of names and meanings for usability defect attributes are needed.

We addressed this recommendation in Chapter 6.

3.2.3.5 Recommendation R5 – Provide good contextualized guidelines for well-written usability defect reports

This study identified some research that defined guidelines for characterizing how usability defects should be reported. However, these lack content-related criteria that would assist reporters in collecting important and useful information for describing the defect and correcting the defect [P2, P5, P37, P52]. For instance, a good usability defect report should describe the issue precisely, but often the information that really needs to be reported is not explained clearly or even not captured at all. Inexperienced reporters in particular may think that their reports are complete, but they may actually be providing irrelevant or inadequate information.

These issues require further research into what influences the fixing of usability defects. This might include mining defect repositories for evidence of useful attributes and reports, and surveying and interviewing both reporters and developers. The findings from these kinds of studies could be used to produce better contextual guidelines that assist both reporters and software developers. Another related area for both HCI and software engineering research is studying the "evaluator effect" in terms of how it impacts the usability defect reporting. A related concept we call the "reader effect" – how software developers read, interpret and action usability defect reports – appears to be an as yet unstudied area, that with better knowledge also may improve defect reporting.

3.2.3.6 Recommendation R6 – Develop more automation in usability defect reporting

Much current usability defect reporting in software teams is still highly text-based and manually captured. Apart from better information capture for usability defect reports, as discussed above, more automated data capture and richer kinds of information capture are needed. Many usability engineering tools provide both of these e.g. instrumenting applications to capture traces and user interaction, recording richer user interaction and mapping to user task, and capture of video, audio, screenshots, diverse interaction (touch, sketch, gesture, accelerometer, as well as keyboard and mouse). However, most software engineering defect tracking tools make capture of this highly manual, uni-format (usually free format text), or make adding and manipulating attachments difficult (or impossible). There may be entirely novel approaches to make usability defect reporting possible combining HCI usability engineering methods and tools with software defect reporting and management repositories.

Where possible, supporting automated capture of usability-related defect issues would enhance the reporting process, but also the replication, solution discussion, and correction processes. Such data collection should include structured, contextualized reporting forms as above, but also event traces, interaction traces, screenshots, audio and video, a variety of interaction styles, especially for mobile applications, and enable software developers to view this in context with the usability defect report attributes captured. Attachments such as audio, video and interaction recordings should be interactively manipulable as in some HCI-oriented usability assessment tools.

3.3 Threats to Validity

Even though this systematic review was performed according to a well accepted process [31], [51], we cannot guarantee that we have covered all studies in this area. Each systematic literature review process described in section 3 was exposed to some threats. We describe the threats associated with each process and the mitigation strategies used for this review.

Data source and search strategy. This review is limited to studies that were published from the year 2000 onwards. Thus, it neglects studies that were published before the year 2000. We were aware that a few studies on usability defect reporting were published in 1997 [69] and 1999 [12], but these studies were extended in other studies [70], [71], which were included in our review. Other than that, we cannot guarantee the selection of the search strings covers all terms used in both software engineering and the usability-engineering field. In this case, we tried to derive a different set of search strings for different fields of study and these are adjusted accordingly to each search engine (as described in section 3.2.2). Additionally, we included a *reference chaining* search as a secondary search to minimize this threat.

Study selection. The selection of studies was performed by the main researcher, which may have resulted in missing studies. However, the other supervisors provided detailed feedback during the review process and monitored the systematic literature review protocol execution closely. We have used clear inclusion and exclusion criteria to reduce selection bias.

Data extraction and synthesis. We found that some studies do not have clear details about the format used for reporting usability defects. In this case, we had to make assumptions on the basis of our judgment. Therefore, there is a possibility that some of the extracted results are partially inaccurate. In order to mitigate this, the three supervisors randomly picked several studies, refined and verified the extracted data. The main researcher then rechecked the earlier data extraction. Overall agreement was very high between the supervisors in terms of classification of studies and agreement on extracted data.

3.4 Summary

This chapter describes the SLR process we carried out. We performed a comprehensive literature search on five reputable online databases using multiple search strings and a two-phase screening of papers. As a result, 57 papers were selected. We divided the papers into three main categories; 1) reporting usability defect information - which is related to research on reporting the usability defect, 2) analysing usability defect data - which is related to researching the use of defect data, and 3) challenges – which refer to issues identified in current approaches to usability defect reporting and management.

In usability engineering and HCI studies, evidence showed that various diverse mechanisms are used to capture and record usability defects. This is supported by numerous defect report content and formats to present the information. However, most of these mechanisms and formats were used in isolation. That is, each mechanism and format was designed to the specific usability evaluation method and does not integrate with the central defect database. Furthermore, existing guidelines to assist reporters in writing a good usability defect description lack guidance for collecting usability defect data.

In the software engineering discipline, usability defect reporting has been less frequently investigated. Existing studies that investigated usability defect reporting have focused especially on addressing the limitations of open source defect repositories to support usability defects. While most challenges in the usability engineering field are related to human factors and usability evaluation methods, challenges in software engineering field are due to limitation of existing defect repositories to capture usability-related information.

Overall, the results from the SLR showed that usability defect reporting processes suffered from mixed data, inconsistent terms and values of usability defect data, and insufficient attributes to classify usability defects. Although mailing lists and online forums have become an alternative interaction hub for users to discuss usability defects, especially in OSS development communities, the linear sequence of communication makes it hard to extract the contextual information for developers to fix the problems.

These limitations and challenges motivated this research to investigate further what constitutes the ideal content of a usability defect report. In particular, we are interested in identifying important usability-related attributes, terminologies that best describe usability defect, and critical usability defect attributes that need to capture. We addressed these investigations in Chapter 4.

4 Information Needs for Reporting Usability Defects

Reporting usability defects can be a challenging task, especially in convincing software developers that a usability defect reported is indeed a real defect. Specifically, the subjective nature of usability defects that cause confusion for some people require stronger evidence to describe and report the problem. However, research to date in software defect reporting has not investigated the capturing of different information based on defect types, such as usability defects. This lack of empirical data on information needs for different types of usability defect reporting impedes research on finding what information is best to describe a usability defect. While previous studies have identified steps to reproduce, actual output, and expected output as an important information to fix software defects in general, however, these studies do not consider what information should be reported, and how the information should be presented in the context of specific types of defects.

To fill this gap and to find answers for the second thesis research question*"RQ2 - What usability defect information do software developers and reporters emphasize in current usability defect reporting?"*, we designed a research study to identify what reporters currently provide when describing usability defects, what information software developers need to fix usability defects, and how usability defects are actually described in real software development projects. This chapter presents the details of this study.

4.1 Methodology

To answer *"RQ2 What usability defect information do software developers and reporters emphasize in current usability defect reporting?",* we designed two studies to identify the types of information needed to describe usability defects. In the first study, we surveyed software development practitioners in both open source communities and industrial software organizations about their usability defect reporting practices to better understand information needs to address usability defect reporting issues. The second study involved an analysis of usability related defects reported in the Bugzilla software defect repositories of Mozilla Thunderbird, Firefox for Android and Eclipse Platform projects, respectively.

Since we used more than one method (survey and defect report mining) to collect data on the same topic, we chose a triangulation method to answer RQ2. The use of a triangulation method gives an understanding of how usability defects are described from different perspectives, and reveals some commonalities and dissimilarities between what is claimed by the practitioners with what is written in their reports. Table 4.1 described the different methods and strategies used in this research along with the rationale of selecting the method and strategy.

Table 4.1. Description of research methods and strategies.

Method	**Strategy**	**Rationale**
Surveying software development practitioners	Questionnaire	Conducting online surveys of practitioners is a useful way of gathering insight into how participants deal with usability defects. In this method, we are expecting to collect detailed information on what reporters provide when reporting usability defects and what information is expected by the software developers to fix usability defects. Unlike face-to-face survey interviews, online survey caters to privacy needs of the participants in order to provide open and honest feedback, and gather meaningful opinions.
Analyzing open source usability defect reports	Observation	Open source defect repositories offer significant source of empirical data. Analyzing the text data of defect report could provide confluence evidence of how usability defects are described in real software projects development. In addition, the use of document analysis method is useful to support and strengthen research topic, especially in contextualizing research topics within its subject or field, and provide supplementary data that participants in the survey have forgotten [72].

4.1.1 Online Survey of Software Development Practitioners

This survey was divided into two themes investigating “important usability defect attributes” and “factors that influence usability defect report quality”. This chapter only reports the results relating to the first theme about the information needed for reporting usability defects. The next chapter reports the finding on the second theme on factors influencing usability defect reporting.

4.1.1.1 Research Questions

The main objective of the first theme of the survey was to identify:

1. What information do reporters use to describe usability defects?
2. What information do developers consider useful for fixing usability defects?

This investigation set out to provide a comprehensive view on the day-to-day practices when dealing with usability defects and pinpointing challenges. Through this study, researchers can find characteristics, open issues, and understand the nature of describing usability defects, which can be valuable for improving defect reporting processes and tools. Software development practitioners, in turn, will also find technical references for reporting specific types of defects.

4.1.1.2 Survey Design

The survey was developed using the Opinio tool. We conducted a self-administered survey, as this kind of survey approach offers greater flexibility to participants. Participants can answer the survey at their convenience without intervention of the researchers collecting the data. The survey was open from June until November 2015.

4.1.1.3 Development of Survey Instruments

We did not find any research on usability defect reporting in the software engineering literature. However, our review of software defect reporting reveals a number of studies that investigate the key information to fix software defects in general [20], [21], [29], [33], [73]. As our survey is in the context of software engineering research, we reused some defect attributes relevant to usability defects studied in [20], [21] – steps to reproduce, actual and expected results, software context, and screenshots. Furthermore we added video, audio, assumed cause, UI event trace, proposed solution and usability principle to the list of usability defect information. This information is useful when describing usability defects, especially in addressing usability concerns, indicating the problematic user interface component, and describing particular specifications of the environment used for evaluation.

4.1.1.4 Questionnaire Design

The survey had a total of 50 questions split into seven sections. Around 14% of the questions on investigating usability defect attributes were derived from [20], [21]. Questions on the influential factors of defect reporting practices, like knowledge, experience, tools and methods were based on [74]. Other questions were formulated based on our literature review.

The survey questionnaire included both closed and open questions. Most of the closed questions used a Likert scale with five possible responses ("Never", "Rarely", "Sometimes", "Often", and "Always"). The questionnaires consisted of two versions: one for usability defects fixer (developers) and one for reporters. The sections are:

1) *Background information*: We collected general information about the respondents including gender, age ranges, employment information, and role in dealing with usability defects.

2) *Training/ certification in Human-Computer Interaction*: We asked both reporters and developers if they attended any HCI and/ or usability training and how useful the training/certification was.

3) *Discovering usability defects*: We asked the reporters about their experience in software testing and methods they used to discover usability defects. The respondents were also asked if they agreed (on a Likert scale) that the amount of information available for reporting usability defects varies according to how defects are discovered. Some of the findings are discussed in Chapter 5.

4) *Reporting usability defects:* We asked what information reporters usually provide, evidence they used to support their claim, and how usability defects are presented. This section also asked reporters to rank top five most difficult attributes to provide.

5) *Fixing usability defects*: We asked what information do software developers usually use when fixing usability defects and ranked the top five most importance attributes. The software developers were also asked to indicate the problematic attributes that they have experienced and their opinions on the quality of defect reports produced by different types of reporters.

6) *Defect reporting and automation tool*: We asked both reporters and software developers on their experience of using defect reporting and automation tools. Questions focused on tools used and their effectiveness to capture and manage usability defects. Other questions aimed to get opinions on the influence of experience, and knowledge in designing new defect reporting form. Some of the findings are discussed in Chapter 5.

7) *Knowledge and experience in usability defect reporting*: We asked both reporters and software developers about their view of experience and knowledge in usability defect reporting. The questions asked about the influence of level of experience, and whether different types of knowledge (usability/ software engineering, domain and technical) can affect the level of detail of defect reports. The findings of this section are discussed in Chapter 5.

See Appendix B for complete survey questions.

4.1.1.5 Evaluation of Survey Instruments

This survey was piloted with Swinburne Software Innovation Lab (SSIL) software engineers and fifteen software developers were recruited during a developer conference (DDD Melbourne 2014).

Based on the written and verbal comments, and the pattern of responses received, the survey instruments were refined.

This survey study was approved on behalf of Swinburne's Human Research Committee (SUHREC) by a delegated SUHREC subcommittee (SHESC2) (Approval number: SHR Project 2014/231). See Appendix F.

4.1.1.6 Selection of Participants

Since our target sector is open source development, which is an increasingly broad group of people ranging from professionals to end users, we replicated this environment by surveying software development practitioners with varying experience levels and roles (including developers, testers, and managers), and non-IT related professionals. We made assumptions that these practitioners and users have similar characteristics (albeit working in a different environment with different resources) to those users in an open source context. We used a survey of practitioners to collect their current practices, challenges, and perspectives of reporting and handling usability defects.

The respondents were recruited from both open source and industrial communities. For open source respondents, we advertised the survey through community forums, such as Eclipse Community forums. While industrial respondents were invited through Facebook, LinkedIn, Software Testing Club[1] and researchers' industrial contacts. Participation was voluntary and participants were allowed to discontinue participation at any time during the research activity. The consent to participate in the survey was implied by the return of the anonymous questionnaire. However, a precise response rate cannot be determined, as the total number of the participants who received the invitation is unknown.

4.1.1.7 Data Analysis

This survey collected both qualitative and quantitative data. For quantitative data, we used descriptive statistics, while qualitative data was analyzed using exploratory analysis [75]. We began by reading the respondents' comments, looking for keywords, trends and themes. Next, the results of the analyses were used as supportive evidence for the quantitative results. Finally, we generated hypotheses for further study. The responses to the qualitative questions are discussed only briefly in this paper.

[1] http://www.softwaretestingclub.com/forum

4.1.2 Software Defect Repositories Mining

In addition to the survey of software development practitioners, we examined a subset of developers-tagged usability defects reported in Bugzilla defect repositories of the Mozilla Thunderbird, Firefox for Android, and Eclipse Platform projects. The motivation of this analysis was to examine how usability defects are described in actual defect repositories, as compared to what is claimed by the participants through the online surveys. Despite identifying the presence of certain defect information when describing usability defects, we also studied the characteristics of usability defects in our dataset. The findings of this analysis were used to validate the findings from our online surveys using triangulation method.

4.1.2.1 Research Questions

In order to investigate the extent usability defects are described by open source communities, and reveal common usability defect characteristics, we addressed the following six research questions, as listed in Table 4.2.

Table 4.2. Software defect repositories research questions.

Research Questions		Rationale
RQ1	What information is commonly provided in open source usability defect reports?	To compare the actual usability defect report content with what software development practitioners claimed in the online survey.
RQ2	How if at all, is a proposed solution to the usability defect described?	
RQ3	Are usability defects described differently from performance-related defects?	
RQ4	What are the dominant types of usability defects (e.g., interface and interaction) in open source projects?	To investigate if the existing usability classification model can be used for open source projects, and whether the existing models need to be revised.
RQ5	What are the impacts of usability defects and what types of usability defects have a severe impact?	To investigate the cause and impact aspect of reported defects. This information is useful as a ground basis of the open source usability defect taxonomy.
RQ6	On what basis, do usability defect reporters justify that the user difficulty that they experience is an issue?	

4.1.2.2 Defect Sources

We performed an investigation of usability defects gathered from the Bugzilla defect repository of the Mozilla Thunderbird, Firefox for Android, and Eclipse Platform projects. Our choice of these projects was based on the following factors:

- These projects represent a variety of different uses and environments;

- These projects have significant GUI that use windows, icons and menus, and interaction with the GUI can be done using a mouse, keyboard or touch screen;
- These projects use standard defect-reporting templates provided by the Bugzilla defect repository and this maintains the consistency when comparing the type of information presented when reporting defects;
- These projects make use of keywords to label and classify defect type reducing selection bias;
- These projects are amongst the most successful user-facing applications that engage various levels of user participation with different levels of knowledge and technical experience.

Across the three projects, only 23,373 defect reports are available to download in CVS format. However, we only studied 377 FIXED defect reports tagged with predefined Bugzilla usability keywords as listed in Table 4.3. These usability-related defects were representative of usability defects in OSS projects from 2001 to 2016. The reason we chose to use FIXED developer-tagged usability defect reports is to reduce selection bias, as the software developers had already completed the resolution process and have reached agreement on the actual types of defects reported and corrected. Furthermore, by analyzing FIXED defect reports rather than the UN-FIXED defect reports, this can be useful to identify some patterns of information that are important for usability defects to be accepted and fixed.

Table 4.3. Open source usability defect reports studied.

Project	Total	Other resolution	Resolved/ Verified						
			Fixed	Duplicate	Incomplete	Invalid	Wontfix	Worksforme	Expired
Mozilla Thunderbird	384	185	**88**	64	4	9	16	17	1
Firefox for Android	292	62	**101**	59	3	11	36	20	0
Eclipse Platform	530	78	**188**	46	-	68	103	47	-
Total	1206	325	**377**	169	7	88	155	84	1

Other resolution – New, unconfirmed, assigned, and reopened

Usability-related – ue, uiwanted, useless-UI, ux-affordance, ux-consistency, ux-control, ux-discovery, ux-efficiency, ux-error-prevention, ux-error-recovery, ux-implementation, ux-interruption, ux-jargon, ux-minimalism, ux-mode-error, ux-natural-mapping, ux-tone, ux-trust, ux-undo, ux-userfeddback, ux-visual-hierarchy

We extracted sample defect reports for each project in four common steps: 1) filter defect reports that were resolved as FIXED; 2) specify columns/ attributes that we wish to appear in the defect list. In this study, we add *Keywords, Opened, Reporter, Number of Comments* and *Last Resolved* attributes; 3) extract and save the data in CSV format; 4) filter the usability and performance defect reports. Since the usability defects downloaded for all projects only constitute a small percentage of all reported defects we chose to analyze all of them in this work.

4.1.2.3 Analyzing Defect Report Content

Our analysis of usability defect report content only focused on the initial reporting of a defect, not investigating the subsequent discussion about the problem and its possible solution in the comments sections. We used the defect report *title, description* and *attachment* fields as our main source of investigation. We defined eight metrics based on Capra's guidelines as modified in [48] to assess the presence of certain information when describing usability defects. Since defect reports presented in open source defect repositories are in unstructured plain text, we were unable to automatically assess the presence of these metrics. Even though the Bugzilla defect report template can be customized to label some of these metrics, such as "Steps to reproduce", "Expected Output" and "Actual Output", many reporters do not explicitly describe these metrics.

As such, we manually analyzed 377 usability defect reports to identify whether the criteria listed in Table 4.4 were presented in the defect report. The extraction and analysis processes were mainly performed by main researcher and validated by her supervisors. The presence of *steps to reproduce, impact, software context, expected output*, *actual output, assumed cause, solution proposal,* and *supplementary information* were set as 1 implies "information exist", and 0 implies "information does not exist". Since *impact, assumed cause* and *solution proposal* do not have separate fields, we measured the presence of this information based on the following criteria:

1. *Assumed cause* – criteria used to justify the usability problem identified is true. For example the defect report number, in which reporter feels the current issue was likely due to the previous fixed issue.
2. *Impact* – user difficulty, number of reproducibility, high numbers of users encountered the same problem, and severity.
3. *Solution proposal* – justification of the proposed solution or fragmental/ modification of affected code/ patch description on how to fix the problem.

The procedure we used to analyze the defect reports consisted of going through each report twice. The first reading focused on understanding the context of usability problems and identifying the main interface or interaction problems described by the reporter. The second reading was to highlight the keywords and snippets of the defect description describing the problem types, impact, and failure qualifier based on the previous classification. We used a card-sorting technique to group impact information into several groups that have similar ground of user difficulty, while problem types and failure qualifier were reorganized according to UPT and ODC, respectively. The detailed analysis of usability defect categories, user difficulty and failure qualifier are discussed in Chapter 6.

Table 4.4. Criteria used to check the presence of usability information in defect reports based on Capra's guideline.

Quality criteria	**Related Capra's Guidelines**
1. Does the defect report describe details steps to reproduce the defect? [29] (*Steps to reproduce*)	Describe observed user actions
2. Does the defect report indicate the effect of the problems on the user? (*Impact*)	Describe the impact and severity of the problem
3. Does the defect report describe the problematic part of the user interface? (*Software context*)	Describe the impact and severity of the problem
4. Does the defect report contain details of expected output and actual output? [29] (*Actual and expected output*)	Describe observed user actions
5. Does the defect report contain criteria used to justify the usability problem identified is true? (*Assumed cause*)	Describe the cause of the problem
6. Does the defect report contain design ideas? (*Solution proposal)*	Describe a solution to the problem
7. Does the defect report contain support information as evidence to the problem? (*Supplementary information*)	Support your findings with data

4.1.2.4 Data Analysis

We report results with descriptive statistics. We used Chi-square test of independence to find the significance relationship between: 1) the types of projects and defects, and the presence of seven usability defect attributes, and 2) the presence of impact information and defect severity. In addition, we used point-biserial correlation to determine whether there is association between length of defect description, measured in words, and the presence of usability defect information (which has two categories – "present" and "not present").

4.2 Results

4.2.1 Online Survey of Software Development Practitioners

The data was collected during June – November 2015. A total of 294 respondents attempted the survey. However, only 147 responses were included in this analysis. The remaining 50% of responses were excluded due to no response beyond the first parts of the questionnaire. One possible explanation of the high percentage of invalid responses is due to the *out of scope* problems, where the respondents are not in the target population. For example, the software development practitioners who do not have experience in dealing with usability defects would be not interested or they may find the questions are not relevant and return blank questionnaires.

4.2.1.1 Respondents Background

The majority of the respondents were male (65.3%), with 34.0% female participants, and 0.7% of participants who did not indicate their gender. About 85% of the respondents were between 25 - 44 years of age. As shown in Table 4.5, majority of the respondents are software developers (40.9%) followed by software testers (14.8%) and project managers (10.0%). In terms of years of experiences, 63.8% of respondents had one to five years of work experience in their current position, while 25.3% had more than five years. Among the respondents, only 13.7% had experience on open source projects.

Table 4.5. Distribution of respondents across professional position and year of experience.

Professional Position	**Less than 1 year**	**Between 1 and 3 years**	**Between 3 and 5 years**	**More than 5 years**
Software developer	4.1%	21.8%	7.5%	7.5%
Software tester	2.0%	5.4%	3.3%	4.1%
Quality assurance engineer	0%	0%	2.0%	0.7%
Customer consultant/ support	0%	0.7%	1.4%	0.7%
System engineer	0%	0.7%	0%	0.7%
Test manager	0.7%	0%	0.7%	0.7%
Project manager	0.7%	3.3%	3.3%	2.7%
Usability engineer	0%	0%	0.7%	0.7%
User interface designer	0.7%	0.7%	0.7%	0.7%
Other	2.7%	6.8%	4.8%	6.8%

Table 4.6. Distribution of participants' knowledge of HCI.

Role in dealing with usability defects	**HCI related training**		**Total**
	Yes	**No**	
Reporting usability defects	17	65	55.8%
Fixing usability defects	8	57	44.2%

Table 4.7. Responses on "usefulness" of usability-related training for reporting usability defects.

Very useful	44.0%
Somewhat useful	40.0%
Neither useful or not useful	4.0%
Not very useful	12.0%
Complete waste of time	0%

Table 4.8. Years of experience in software testing.

No experience	25.6%
Less than 1 year	6.1%
Between 1 and 3 years	22.0%
Between 3 and 5 years	12.2%
More than 5 years	34.1%

Table 4.6 shows respondents' knowledge of HCI. The vast majority of respondents had not received any usability-related training. However, for those who had acquired the related training, 84% believed the training was useful for understanding and reporting usability defects (see Table 4.7). Our survey findings also show that most respondents had experience reporting usability defects (55.8%), while 44.2% has experience fixing them. Most of them have more than 5 years of experience in software testing (34.1%) (see Table 4.8).

Table 4.9 summarizes the defect discovery methods used by the respondents to discover usability defects. Note that respondents could select more than one option. More than 60% of respondents indicated that they found usability defects when performing exploratory testing, functional testing and while using a product. A smaller proportion of respondents indicated that they discovered usability defects through alpha/beta testing (26.8%). From the free-text explanation, some respondents explicitly mentioned other methods including focus groups, GUI testing, performance testing, heuristics evaluation and automated testing.

Table 4.9. Defect discovery methods.

Exploratory testing	62.2%
Functional testing	63.4%
Usability testing	58.5%
Beta/ alpha testing	26.8%
Complaints/ reports from customers	53.7%
Using the product	62.2%
Other	7.3%

4.2.1.2 Medium to report software defects

As shown in Table 4.10, nearly half of our respondents used written reports (50%), verbal meetings (53.7%) and defect reporting tools (53.7%) as a medium for their defect reporting (Q25). Only a few respondents used edited video for reporting purposes. For those who have used a defect-reporting tool, we asked respondents to mention their tool (Q36).

As listed in Table 4.11, the most commonly used defect reporting tools reported by our respondents were JIRA, Bugzilla and Redmine. Mantis, HP Quality Center, Trello, IBM Rational Team Concert, HP Application Lifecycle Management and Visual Studio TFS were listed multiple times. For JIRA, Bugzilla and Redmine users - 90% of them agreed to some extent that the tool offers sufficient flexibility to capture and manage usability defects (Q37), but free-text feedback revealed considerable negative satisfaction (Q38). The following are representative: "Most of the defect reporting tools do not have exhaustive options for usability defects" and "JIRA more customized by client but no specific customizations done for usability".

Some respondents nominated specific recommendations for usability defect reporting tool improvements (Q50). For example, they argued that video evidence could reduce the amount of time to reproduce and describe the issues, especially when working with offshore development teams. One respondent also suggested a questionnaire feature.

Table 4.10. Medium for report usability defects – respondents mostly used defect-reporting tools and discussed through verbal meeting.

Medium of reporting	Never	Rarely	Sometimes	Often	Always	Not answer
Traditional written report	13.4%	11.0%	12.2%	22.0%	28.0%	13.4%
Verbally in a meeting with designers/ developers	3.7%	6.1%	23.2%	42.7%	11.0%	13.4%
Edited videos	31.7%	31.7%	13.4%	8.5%	1.2%	13.4%
Entry in defect reporting tool	7.3%	6.1%	19.5%	15.9%	37.8%	13.4%

Table 4.11. Defect reporting tools used by respondents.

Defect reporting tool	Reporter	Developer	Defect reporting tool	Reporter	Developer
Bugzilla	15	11	SourceForge	1	0
JIRA	2	5	Mantis	1	0
Github	1	1	Target Process	0	1
Redmine	1	1	Clearcase	0	1
Pivotal Track	0	1	FogBugz	0	1
Web Helpdesk	1	0	Unknown	2	14
Trac	1	0			

4.2.1.3 Information Provided by Defect Reporters When Reporting Usability Defects

Question 13 asked respondents to indicate a frequency of supplying attributes listed in Table 4.12 when reporting usability defects. Based on "Often" and "Always" rating, the reporters mostly provide *title/ summary (81.8%), actual output (79.5%)* and *expected output (76.8%). steps to reproduce (74.4%)*, *software context (70.7%)* and *software information (70.7%).* In order to understand the content of each selected attribute, for those who selected option other than "Never", we further asked questions about the elements and supportive materials that they used. For these subsequent questions, multiple answers were allowed and respondents could describe other details in a free text section (Question 14 – Question 24). Table 4.13 summarizes the responses.

Title/summary is a headline summarizing the problem. When crafting a title, the majority of the reporters explained the situation that was happening at the time the problem occurred (70.7%). Some of them (58.8%) preferred to copy and paste an error message. Less than half of the reporters provided product details (such as build, version and operating system) (47.6%) and clarified what the quality issues that were affected (43.9%). As expected, good quality issues were created mostly by reporters with at least five years of experience.

Expected output describes reporters' expectations on how the software should respond. A majority of reporters used their knowledge and experience to interpret the intended results (72%). Some reporters mentioned usability requirements (59.8%) and usability design guidelines (40.2%) that were used to justify their expectations.

Actual output describes the actual results that differ from the reporter's expectation or violating specification. As indicated in Table 4.13, the actual output was usually described based on the effect on a user's performance (65.9%). In this case, the justification on what was wrong and why it was wrong (53.7%) were supplied. Others explained the user's behavior by outlining the issues (50%). To support the claimed issues, reporters preferred to attach annotated screenshots (64.6%) followed by error messages (57.3%).

Software context addresses the location of the problem in the interface. As expected, the majority of the reporters mentioned the name of the defective interface, such as screen title (76.8%) and problematic user interface object (70.7%). Some respondents (58.5%) described user's task to indicate the context of usage. In terms of specifying the components affected, only 41.5% of reporters could supply the information. This information was often conveyed in annotated screenshots (82.9%) followed by textual description (64.6%) – as shown in Table 4.14.

Severity indicates the level of effect the usability defects had on the user. In order to rate the severity level, most reporters considered the impact of the issue (74.4%). Others examined the frequency of issues occurred during usage (61%) and business impact (48.8%).

Table 4.12. Defect attributes used to report usability defects – title, actual and expected output are the most provided attributes by reporters.

Items	Never	Rarely	Sometimes	Often	Always
Title/ Summary	1.2%	2.4%	14.6%	25.6%	56.2%
Assumed cause	3.7%	6.1%	23.2%	23.2%	43.8%
Software context	1.2%	3.7%	24.4%	24.4%	46.3%
Solution proposal	9.8%	12.2%	32.9%	26.8%	18.3%
Actual output	1.2%	3.7%	15.9%	20.7%	58.5%
Expected output	1.2%	7.4%	14.6%	26.8%	50.0%
Steps to reproduce	2.4%	6.1%	17.1%	31.7%	42.7%
Severity	1.2%	18.3%	20.7%	22.0%	37.8%
Software information (product, component, version)	0%	9.8%	19.5%	29.3%	41.4%
Test environment (Operating system, hardware, browser)	0%	9.8%	26.8%	29.3%	34.1%

Table 4.13. Detailed explanation included for each attribute.

Items	%
Title/ Summary	
Explanation on the situation that was happening at the time the problem occurred	70.7%
Build or version of the software or operating system on which the problem occurred	47.6%
An error message that come up	58.5%
Quality attributes affected	43.9%
Other – Not mentioned	4.9%
Assumed cause	
Heuristics that are violated	26.8%
Design fault	51.2%
My knowledge of performing and understanding a task or object interface	61.0%
Other – consequence to the user	3.7%
Software context	
Location of the problem in the interface, such as screen title	76.8%
The problematic user interface object, such as button, menu and dialogue box	70.7%
User's task	58.5%
System components that are affected	41.5%
Other – Not mentioned	2.4%
Solution proposal	
Alternate solutions	37.8%
Advantages and disadvantages for alternatives solutions	28.0%
Usability design principles and/ or previous research	26.8%
My knowledge to interpret how design is supposed to work	45.1%
The expected behavior (defects)	46.3%
The behavior desired (enhancements)	56.1%
Other – as suggested by project manager	2.4%
Actual output	
The effect on the user's performance	65.9%
The user's behavior following the issue	50.0%
What was wrong and why is it wrong	53.7%
Other – user experience	1.2%
Expected output	
Usability requirements	59.8%
Knowledge/ experience to interpret the expected result	72.0%
Usability design guideline	40.2%
Other – Not mentioned	3.7%
Steps to reproduce	
User's navigation flow through the system	72.0%
Record the steps	30.5%
Other - logs	3.7%
Severity	
Impact of the issue	74.4%
Business effects, such as costs and time loss	48.8%
Frequency of the issue occurs during usage	61.0%
Other – Not mentioned	2.4%

Assumed cause describes the cause(s) of the problem based on the reporter's judgment. Our findings indicated that the cause of the problem is normally justified based on the reporter's knowledge (61.0%). About half of reporters (51.2%) could explain a design fault, such as how the interaction architecture may contribute to the problem. Nearly a quarter (26.8%) of the reporters pointed out violated usability heuristics. Other information, as described in the free text, specified the consequence to the user (3.7%). To support this justification, a screenshot with accompanying text is the most widely used material other than annotated screenshot and textual description (see Table 4.14). Other materials used by reporters included UI event trace (36.6%) and videos with captions (17.1%).

Solution Proposal describes a recommendation to remedy the usability defects. 56.1% (see Table 4.13) of reporters addressed the proposal solutions as a way to improve the desired software behavior, rather than to correct a defective software behavior (46.3%). Reporters mentioned that these recommendations originated from their knowledge (45.1%). To support these recommendations, only a few reporters provided alternate solutions (37.8%), advantages and disadvantages for alternative solutions (28.0%), and usability design principles (26.8%). Additionally, nearly half of the respondents (48.8%) preferred to use a textual form to explain the recommendations. Some of them supported these recommendations with an annotated screenshot (39%), and simple sketches (28%). While 25% of reporters prefer to demonstrate their recommendations through oral presentations.

In Question 27, we asked respondents to rank the top five attributes in order of difficulty from 1 to 5 where 1 is the most difficult and 5 is least difficult. Among the attributes considered to be most difficult to provide, respondents ranked *assumed cause, usability principle, video recording, UI event trace* and *title*.

Table 4.14. Materials used to support usability defect description – the most prevalent material was screenshots with annotations.

Material	Assumed cause	Software context	Solution proposal	Actual output
UI event trace	36.6%	-	-	-
Screenshot and accompanying text	75.6%	1.2%	-	-
Screenshot with annotations	62.2%	82.9%	39.0%	64.6%
Textual description	61.0%	64.6%	48.8%	-
Video with captions	17.1%	18.3%	-	-
Fix patch	-	-	12.2%	-
Digital mockups	-	-	11.0%	-
Simple sketches	-	-	28.0%	-
ASCII art	-	-	-	-
Graphical elements or code	-	-	-	-
Error messages	-	-	-	57.3%
Oral presentation	-	-	25.6%	-
Other – audio video	2.4%	1.2%	3.7%	1.2%

4.2.1.4 Information Needed by Software Developers when Fixing Usability Defects

As shown in Table 4.15, question Q28, Q29 and Q30 asked software developers to rate a frequency of using textual information and supplementary information respectively when fixing usability defects. Based on the “Often” and “Always” rating, *assumed cause* (83.1%,), *steps to reproduce* (81.6%), *software context* (78.5%), *expected output* (78.5%) and *actual output (73.8%)* were the most widely used textual information. The three least used textual information were: *title/ summary (50.8%)*, *hardware (55.4%)* and *severity (56.9%).*

About half the developers seldom used title/summary. One possible explanation can be that title/summary contains limited information for understanding the likely difficulty faced by users. Meanwhile, severity may only be useful for prioritizing defects to be fixed but it does not provide input to solve the problem. Supplementary information which was most frequently used for fixing usability defects were: *screenshots (83.1%)*, *UI event trace (56.9%)* and *patch (41.5%)* (see Table 4.15). Even graphical elements such as *ASCII art (9.3%)* and *digital mockups (23.1%)* provided a rich source of proposed fix, but software developers rarely used them.

Table 4.15. Frequency of attributes used to fix usability defects – the most useful attribute was assumed cause.

Items	**Never**	**Rarely**	**Sometimes**	**Often**	**Always**
Textual information					
Title/ Summary	3.1%	13.8%	32.3%	15.4%	35.4%
Assumed cause	3.1%	0%	13.8%	24.6%	58.5%
Software context	4.6%	4.6%	12.3%	27.7%	50.8%
Solution proposal	3.1%	10.8%	26.2%	46.1%	13.8%
Actual output	3.1%	1.5%	21.5%	32.3%	41.5%
Expected output	3.1%	4.6%	13.8%	33.8%	44.7%
Steps to reproduce	1.5%	3.1%	13.8%	27.7%	53.9%
Severity	3.1%	7.8%	32.3%	21.5%	35.4%
Product	7.8%	6.2%	20.0%	36.9%	29.2%
Component	1.5%	10.8%	26.2%	38.5%	23.1%
Version	0%	6.2%	29.2%	23.1%	41.5%
Hardware	0%	18.5%	26.2%	32.3%	23.1%
Operating system	0%	12.3%	29.2%	27.7%	30.8%
Supplementary information					
Usability principle/ violated heuristic	9.2%	15.4%	43.1%	20.0%	12.3%
Video recording	16.9%	15.4%	30.8%	24.6%	12.3%
Audio recording	21.5%	35.4%	26.2%	10.8%	6.2%
UI event trace	6.2%	0%	36.9%	35.4%	21.5%
Screenshots	1.5%	3.1%	12.3%	29.2%	53.9%
Fix patch	3.1%	13.8%	41.5%	21.5%	20.0%
Digital mockups	15.4%	24.6%	36.9%	10.8%	12.3%
ASCII art	30.8%	33.8%	26.2%	7.8%	1.5%

For the importance of information (Question 31), *assumed cause* clearly stands out. This is followed by *screenshot, steps to reproduce, expected output, software context* and *solution proposal*. Similar to the findings from [20], [21], in order to fix usability defects, mandatory fields such as hardware, product, component, and severity were thought to be little value. This does not mean the information is not useful; rather they might be used in a different context for different purposes.

In Question 32, we asked software developers to select problematic attributes supplied by reporters. Note that software developers were able to select more than one option. As shown in Table 4.16, among the problems experienced, *unclear assumed cause* (76.9%) and *insufficient information in steps to reproduce* (72.3%) was the most commonly encountered. Other common problems include *unclear software context* (46.2%) and *screenshots* (46.2%). Apart from lacking technical information, the software developers also received *vague comment* (52.3%), *unstructured text* (55.4%) and *duplicate defect reports* (50.8%).

Table 4.16. Problems with usability defect reports – unclear assumed cause and insufficient information in steps to reproduce were the most common encountered problems experienced by software developers.

Problems	Attributes	Frequency	Problems	Attributes	Frequency
You were given unclear	Title/ summary	38.5%	There was insufficient information in	Steps to reproduce	72.3%
	Assumed cause	76.9%		UI event trace	38.5%
	Software context	46.2%	The reporter used	Bad grammar	27.7%
	Usability principle/ heuristic violated	21.5%		Unstructured text/ format	55.4%
	Solution proposal	29.2%		Vague comment	52.3%
	Screenshots	46.2%		Too long text	32.3%
	Audio recording	20.0%		Non technical language	35.4%
	Video recording	16.9%		Usability jargon/ term	20.0%
You were given incorrect	Component	16.9%	Other	Duplicate	50.8%
	Actual output	38.5%		Spam	23.1%
	Expected output	44.6%		Viruses/ worms	15.4%

4.2.2 Software Defect Repositories Mining

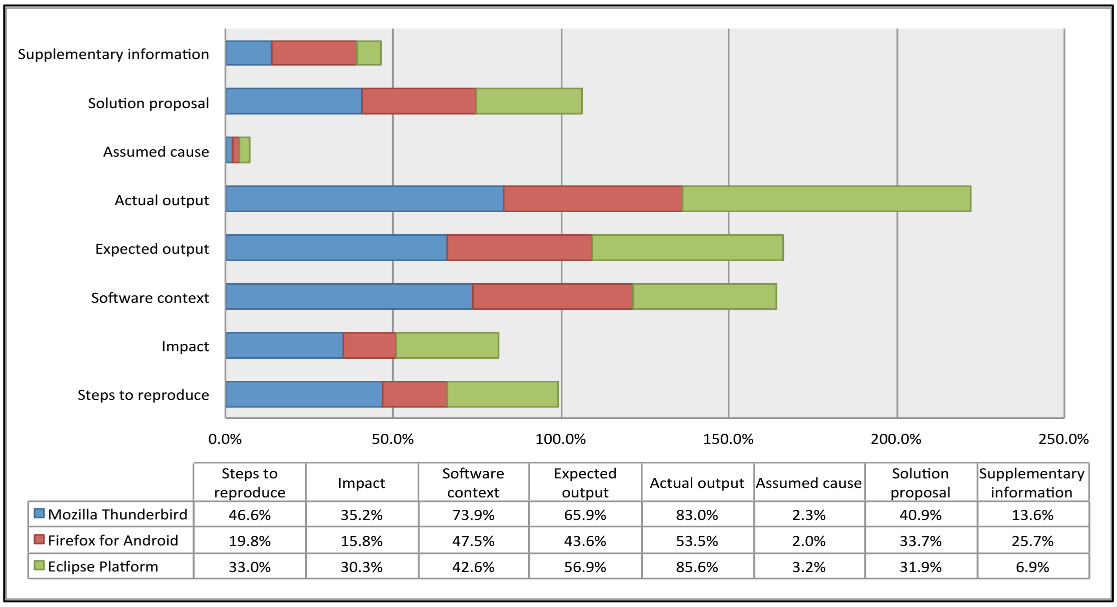
4.2.2.1 Usability Defect Report Contents

In this section, we describe our study of the information patterns of usability defect reports in open source projects. The information patterns that we are interested in including are the following types of information: (1) steps to reproduce the problem, (2) user difficulty that could affect human task and emotional reaction, (3) context, or settings, in which the problem occurs, (4) actual and expected output, (5) assumption of or known cause of the problem, (6) solution proposed to solve the problem, and (7) supplementary information. We answer the following study research questions: *RQ1: What information is commonly provided in open source usability defect reports?* when the defect is initially reported. In addition, we investigate *RQ2: How if at all, is a proposed solution to the usability problem described?*

4.2.2.1.1 Content Distribution

Figure 4.1 shows that the most widely reported attributes in usability defect reports in the open source projects studied are *actual output, expected output* and *software context*. Attributes rarely presented in the studied usability defect descriptions are *assumed cause* and *supplementary information*. The *assumed cause* could only be found in less than 5% of defect reports. While *steps to reproduce* is considered as the most important attribute in defect reports [20], [21], our study found that only between 19% - 46% of defect reports contain this information, depending on project.

Similar to previous findings investigating open source defect reporting in general [33], we also found the presence of information for usability defects varies between projects. Referring to Table 4.17, the p-value for the Chi-square tests for each attribute except *assumed cause*, which is less than 0.05, suggesting that there is indeed a relationship between projects and the presence of certain attributes for usability defects. As shown in Figure 4.1 it is apparent that certain attributes are common in some projects. From this data, we can draw several observations. First, the *actual output* is found in more than 80% of Mozilla Thunderbird and Eclipse Platform. Surprisingly, only about 50% of Firefox for Android usability defects contain actual outcome even though the defect reporting tool specifically prompts its users for this information.



	Steps to reproduce	Impact	Software context	Expected output	Actual output	Assumed cause	Solution proposal	Supplementary information
🟦 Mozilla Thunderbird	46.6%	35.2%	73.9%	65.9%	83.0%	2.3%	40.9%	13.6%
🟥 Firefox for Android	19.8%	15.8%	47.5%	43.6%	53.5%	2.0%	33.7%	25.7%
🟩 Eclipse Platform	33.0%	30.3%	42.6%	56.9%	85.6%	3.2%	31.9%	6.9%

Figure 4.1. Distribution of usability defects in which each attribute was presented by project.

Table 4.17. Chi-Square test results to examine the influence of project and the presence of defect attributes.

Defect attributes	**ρ value**	**χ^2 (df=2, n=377)**
Steps to reproduce	**0.000**	16.67
Impact	**0.006**	10.16
Software context	**0.000**	24.96
Expected output	**0.003**	11.51
Actual output	**0.000**	32.78
Assumed cause	0.934	0.137
Solution proposal	**0.024**	7.45
Supplementary information	**0.000**	26.68

Table 4.18. Types of supplementary information provided in usability defect reports.

Supplementary information	**Project**		
	Mozilla Thunderbird (n=88)	***Firefox for Android (n=101)***	***Eclipse Platform (n=188)***
Annotated screenshots	1.1%	0.0%	0.0%
Error report	1.1%	0.0%	0.5%
External URL link	**6.8%**	**10.0%**	1.1%
Problem occurrence	0.0%	1.0%	1.6%
Screenshots	0.0%	**13.9%**	1.1%
Stack traces	0.0%	0.0%	1.1%
Task workaround	2.3%	0.0%	0.0%
Usability guidelines	0.0%	0.0%	0.5%
Violated UX-consistency	2.3%	0.0%	0.0%

Second, while Mozilla Thunderbird and Eclipse Platform reporters mainly used textual descriptions to describe usability defects, Firefox for Android more often provided supplementary information. As we can see from Table 4.18, screenshots are the most favorable supplementary material used in Firefox for Android.

Third, across the three projects, Firefox for Android usability defects contained very little information on *steps to reproduce* (19%), *impact* (15.8%), and *expected output* (43%). Fourth, given the nature of the open source communities, in which the users have a varying level of technical knowledge, defect reports in all projects contained virtually no *assumed cause*. This attribute, however, is not necessarily appropriate for all types of usability defects. For example, a usability defect that is not related to technical deficiency is quite difficult to explain even by technical users. The limited knowledge and experience of open source communities around usability and HCI aspects is one of the barriers for reporters to technically relate the usability issues and the violated usability principles and HCI guidelines. In the context of understanding the root cause of usability defects, perhaps *assumed cause* could be explained in the form of rationale as a ground for a specific usability issue.

In addition, we also examined the presence of steps to reproduce, impact, software context, expected output, actual output, assumed cause, solution proposal and supplementary information on different length of defect description. Generally, the amount of information present in a usability defect report increases with the increase in the length of the defect description. As shown in Table 4.19, the values in column three to eight represent the percentage in which the steps to reproduce, impact, software context, expected output, actual output, assumed cause, solution proposal, and supplementary information is present in the defect descriptions for a set of defect reports within the same range of length. From the 377 usability defects we studied, only 74 defect reports contain more than 100 words. Within this length, more than 50% of defect descriptions contain steps to reproduce, impact, software context, expected output, and actual output. Interestingly, usability defect reports written with more than 400 words contain almost all the information. However, the presence of steps to reproduce, assumed cause, solution proposal and supplementary information on the length of defect description was variable.

Table 4.19. The presence of *steps to reproduce, impact, software context, expected output, actual output, assumed cause, solution proposal,* and *supplementary information* on different lengths of defect description.

Length of description (words)	**#Defect Reports**	**Information presence in usability defect description (%)**							
		Steps to reproduce	***Impact***	***Software context***	***Expected output***	***Actual output***	***Assumed cause***	***Solution proposal***	***Supplementary information***
0-20	26	0.00	0.00	26.92	34.62	19.23	3.85	0.00	19.23
21-40	74	13.51	9.46	36.49	48.65	67.57	0.00	9.46	14.86
41- 60	71	36.62	14.08	42.25	60.56	77.46	0.00	9.86	7.04
61-80	62	37.10	27.42	54.84	75.81	79.03	1.61	6.45	11.29
81-100	38	31.58	34.21	63.16	57.89	89.47	5.26	5.26	18.42
101-120	18	44.44	55.56	72.22	61.11	88.89	5.56	11.11	11.11
121-140	21	38.10	66.67	57.14	66.67	90.48	0.00	9.52	9.52
141- 160	11	45.45	45.45	63.64	81.82	81.82	9.09	27.27	0.00
161-180	16	56.25	43.75	56.25	75.00	100.00	0.00	43.75	18.75
181-200	9	44.44	22.22	55.56	77.78	88.89	0.00	44.44	0.00
201-220	8	25.00	62.50	50.00	75.00	87.50	0.00	12.50	12.50
221-240	7	85.71	71.43	85.71	85.71	100.00	0.00	28.57	14.29
241- 260	4	25.00	75.00	100.00	75.00	100.00	25.00	25.00	25.00
261-280	3	33.33	33.33	66.67	66.67	100.00	33.33	33.33	0.00
281-300	2	50.00	100.00	100.00	50.00	50.00	0.00	50.00	0.00
301-320	1	100.00	100.00	100.00	100.00	100.00	0.00	0.00	100.00
321-340	1	0.00	0.00	100.00	100.00	100.00	0.00	100.00	0.00
341- 360	1	0.00	100.00	100.00	100.00	100.00	0.00	100.00	0.00
361-380	0	-	-	-	-	-	-	-	-
381-400	1	100.00	0.00	0.00	0.00	100.00	100.00	100.00	100.00
401-420	1	100.00	100.00	100.00	100.00	100.00	0.00	100.00	0.00
> 420	2	100.00	50.00	100.00	100.00	100.00	0.00	100.00	100.00

4.2.2.1.2 Proposing Solutions and Ideas

For answering RQ2, we observed about one third of the defect descriptions contain *solution proposals*. A close inspection of usability defect data as in Table 4.20 shows that about 30% - 40% of solution proposals were generally described in words. The other means of describing solution proposals were in a form of ASCII art, graphical elements or code, and UI-digital mockups, that only account less than 2% of defects in all three projects.

Table 4.20. The presentation of solution proposals – textual description is the most common medium to present solution proposal in the three open source projects.

Solution proposals presentation	**Project**		
	Mozilla Thunderbird (n=88)	***Firefox for Android (n=101)***	***Eclipse Platform (n=188)***
Textual descriptions	39.8%	29.7%	30.3%
ASCII arts	1.1%	1.0%	0.0%
Graphical elements or code that could be immediately implemented	0.0%	2.0%	0.5%
UI digital mockups	0.0%	1.0%	0.5%

4.2.2.1.3 Differences Between Usability and Performance-related Defect Report Content

In this study, we were also interested to investigate how the different types of defects affect the presence of information. In answering *RQ3: Are usability defects described differently from performance-related defects?*, we chose performance-related defects as a benchmark to compare these differences. We considered the defect attributes steps to reproduce, impact, software context, expected output, actual output, assumed cause, solution proposal, and supplementary information as our dependent variables (nominal data) and defect types as the independent variable (nominal data). We analyzed the defect descriptions and rated the presence or absence of information as 0= information is present, and 1 = information is not present.

As shown in Table 4.21, at a significant level p=0.05, we found relationships between *software context* and *expected output*, and defect types in all the three projects. However, no relationships were found between *actual output* and *solution proposal*, and defect types in the three projects. At the same significance level, the relationship between *assumed cause* and *supplementary information* and defect types was only significant in the Eclipse Platform and Firefox for Android projects, while a relationship between *steps to reproduce* and defect types was only observed in Eclipse Platform. For *impact* and *defect types*, a significant relationship was observed on Mozilla Thunderbird and Eclipse Platform.

Figure 4.2 reveals the trends of information presented in usability and performance defects. As shown, *actual output* is found in the vast majority of usability and performance defect reports for all three projects. On average, more than 70% of usability and performance defects contain explanation about what happened or what they saw while using the product. While *expected output*, *software context, impact* and *solution proposal* is more common to be found in usability defect than performance defect reports, *assumed cause* least reported in usability defects (only being found in less than 5% of usability defect reports).

Table 4.21 Chi-square test result to examine the influence of defect types and the presence of defect attributes.

Attributes	**Mozilla Thunderbird**			**Firefox for Android**			**Eclipse Platform**		
	ρ	***χ^2 (df=1, n=121)***	***Φ***	***ρ***	***χ^2 (df=1, n=177)***	***Φ***	***ρ***	***χ^2 (df=1, n=800)***	***Φ***
Steps to reproduce	0.06	3.69	0.18	0.13	2.30	0.11	**0.01**	**7.54**	**0.10**
Impact	**0.03**	**4.54**	**0.19**	0.18	1.84	0.10	**0.00**	**65.65**	**0.29**
Software context	**0.00**	**12.45**	**0.32**	**0.03**	**4.57**	**0.16**	**0.00**	**3.52**	**0.07**
Expected output	**0.01**	**6.85**	**0.23**	**0.00**	**12.25**	**0.26**	**0.00**	**58.21**	**0.27**
Actual output	0.21	1.58	0.11	0.21	1.59	0.10	0.75	0.10	0.01
Assumed cause	0.38	0.76	0.08	**0.00**	**14.26**	**0.28**	**0.00**	**32.95**	**0.20**
Solution proposal	0.88	0.02	0.01	0.77	0.09	0.02	**0.05**	**3.97**	**0.07**
Supplementary information	0.83	0.05	0.02	**0.00**	**8.06**	**0.21**	**0.00**	**11.59**	**0.12**

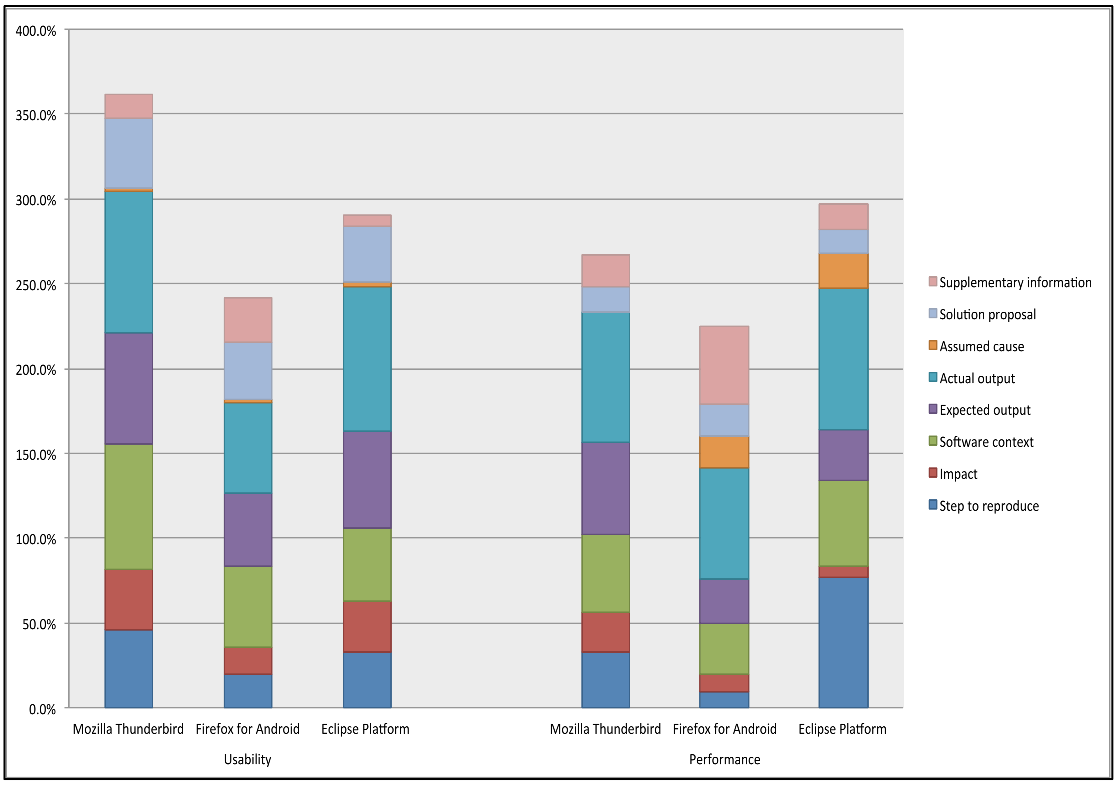


Figure 4.2. Distribution of usability defect attributes between usability and performance defects.

From the project perspectives, regardless of defect types, Mozilla Thunderbird reports contain more information than Firefox for Android and Eclipse Platform. Mozilla Thunderbird is responsible for a significant proportion of the usability defects with *steps to reproduce, impact, software context, expected output, actual output, assumed cause*, and *solution proposal*. While *supplementary information* is mostly attached to performance defects, this information is rarely presented in Mozilla Thunderbird usability defects.

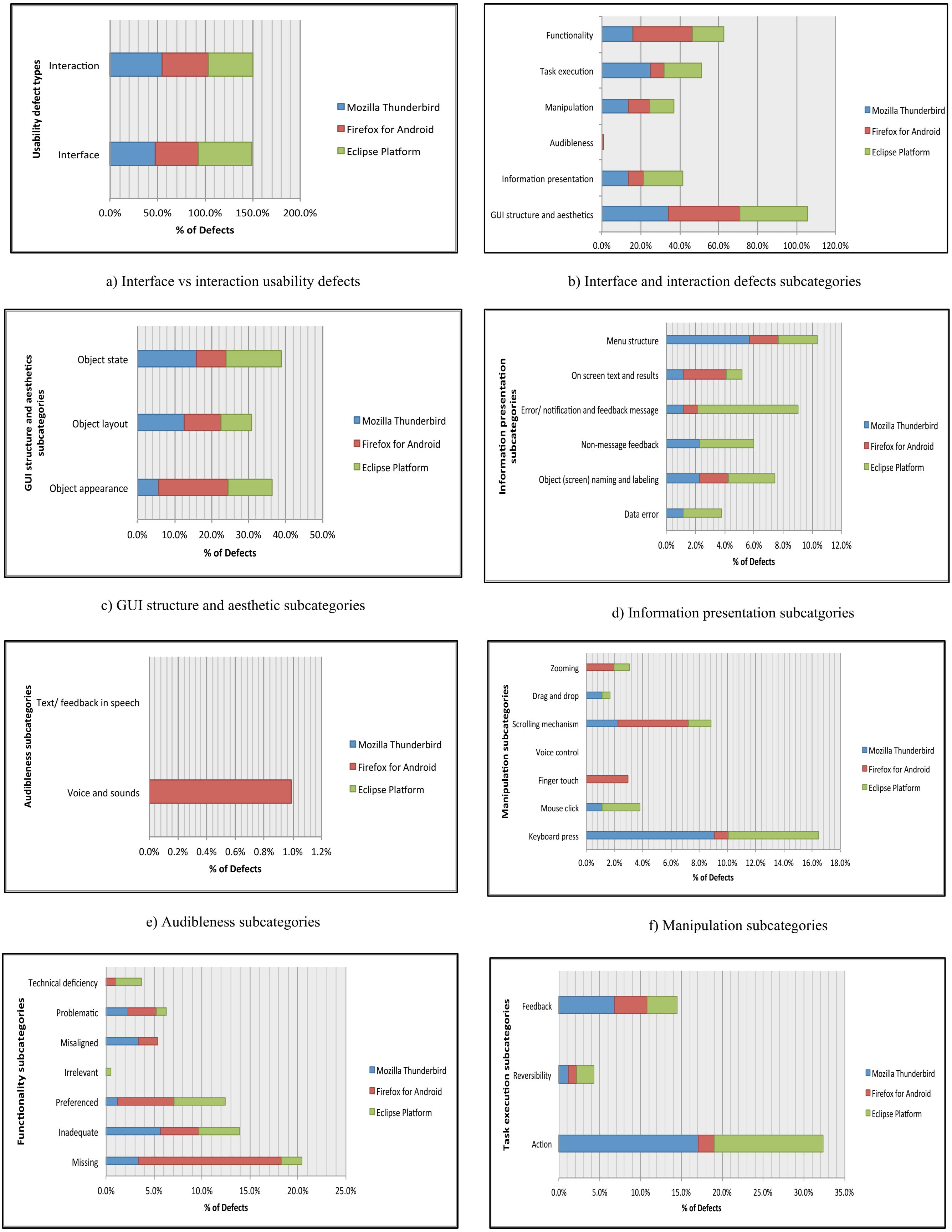
There are perhaps surprisingly few *steps to reproduce*, and *impact* is found in Firefox for Android usability and performance defect reports (found in less than 20% of defect reports). Probably, instead of using steps to reproduce and impact, Firefox for Android contributors much prefer to provide *supplementary information*, and as can be seen this attribute is more common in Firefox for Android than Mozilla Thunderbird and Eclipse Platform. In Eclipse Platform, while *steps to reproduce* is mostly found in performance defects, *expected output* is more common to usability defects. Also, less than 30% of Firefox for Android and Eclipse Platform performance defects described expected output.

In summary, we found Mozilla Thunderbird and Firefox for Android usability defects contain more information than performance defects. The most common attributes used to describe usability and performance defects across projects are *actual output.* Usability defects were favorably described using *impact, expected output*, and *solution proposal*, while performance defects more often used *supplementary information*, and *assumed cause*.

4.2.2.2 Types of Usability Defects

In this section, we describe the distribution of different types of usability defect categories, including interface and interaction usability defects. We answer *RQ4: What are the dominant types of usability defects (e.g., interface or interaction) in open source projects?*. In addition, we study the subcategories of interface and interaction usability defects, as suggested in [12], [76].

Figure 4.3 summarizes the distribution of usability defects within different categories and subcategories. Based on using the UPT [12] and fault GUI model [76], we classified interface usability defects into GUI structure and aesthetics, information presentation and audibleness, while interaction usability defects are divided into manipulation, task execution and functionality. From Figure 4.3, we can draw the following findings and implications.



a) Interface vs interaction usability defects

b) Interface and interaction defects subcategories

c) GUI structure and aesthetic subcategories

d) Information presentation subcatgories

e) Audibleness subcategories

f) Manipulation subcategories

g) Functionality subcategories

h) Task execution subcatgories

Figure 4.3. Distribution of usability defects with categories and subcategories.

a) Both interface and interaction usability defects are common in all three open source projects. However, as can be seen in Figure 4.3, GUI structure and aesthetics are the most dominant types of usability defects, 34.1% in Mozilla Thunderbird, 36.6% in Firefox for Android, and 35.1% in the Eclipse Platform. To this end, the object state problem is responsible for a significant proportion of the GUI structure and aesthetics categories in Mozilla Thunderbird and Eclipse Platform (Figure 4.3c). Object state issues account for 15.9% and 14.9% of the sample usability defects in Mozilla Thunderbird and Eclipse Platform, respectively. Firefox for Android, on the other hand, contains a much smaller percentage of object state problems (7.9%) but has the largest percentage of object appearance problems, accounting for 18.8%. One possible reason is due to the nature of the application, in which Firefox for Android is an application developed for mobile devices that have several limitations such as small screen, single window, touchscreen, and screen orientation support. For example, the small screen size may limit the content displayed and organization of information that substantially affects the overall look and feel of the application.

b) For the information presentation and audibleness category, all of the three projects only contain a small percentage of defects, accounting for less than 7% of the usability defects reported. Interestingly, in the 377 usability defects that we analyzed, none of the reported problems were related to audibleness. Among the information presentation problems, error and notification message and menu structure are the highest problems reported for Eclipse Platform and Mozilla Thunderbird, respectively. One possible reason for the large fraction of error message problems may be that the Eclipse Platform is targeting technical users, and therefore the use of syntax error codes or abbreviations in error message such as *"an error of type 2 has occurred"* should not be a problem. However, as everyone can contribute to open source projects, inexplicit and technical messages may sometimes not be understandable by users with limited "technical-development" knowledge. Perhaps, no matter what kind of projects and intended users, error message should include explicit, polite, precise, constructive advice written in human-readable language so that all users can understand it. Surprisingly, non-message feedback and data error defects are virtually non exist in Firefox for Android, while these two problems only occur less than 4% of defects in Mozilla Thunderbird and Eclipse platform.

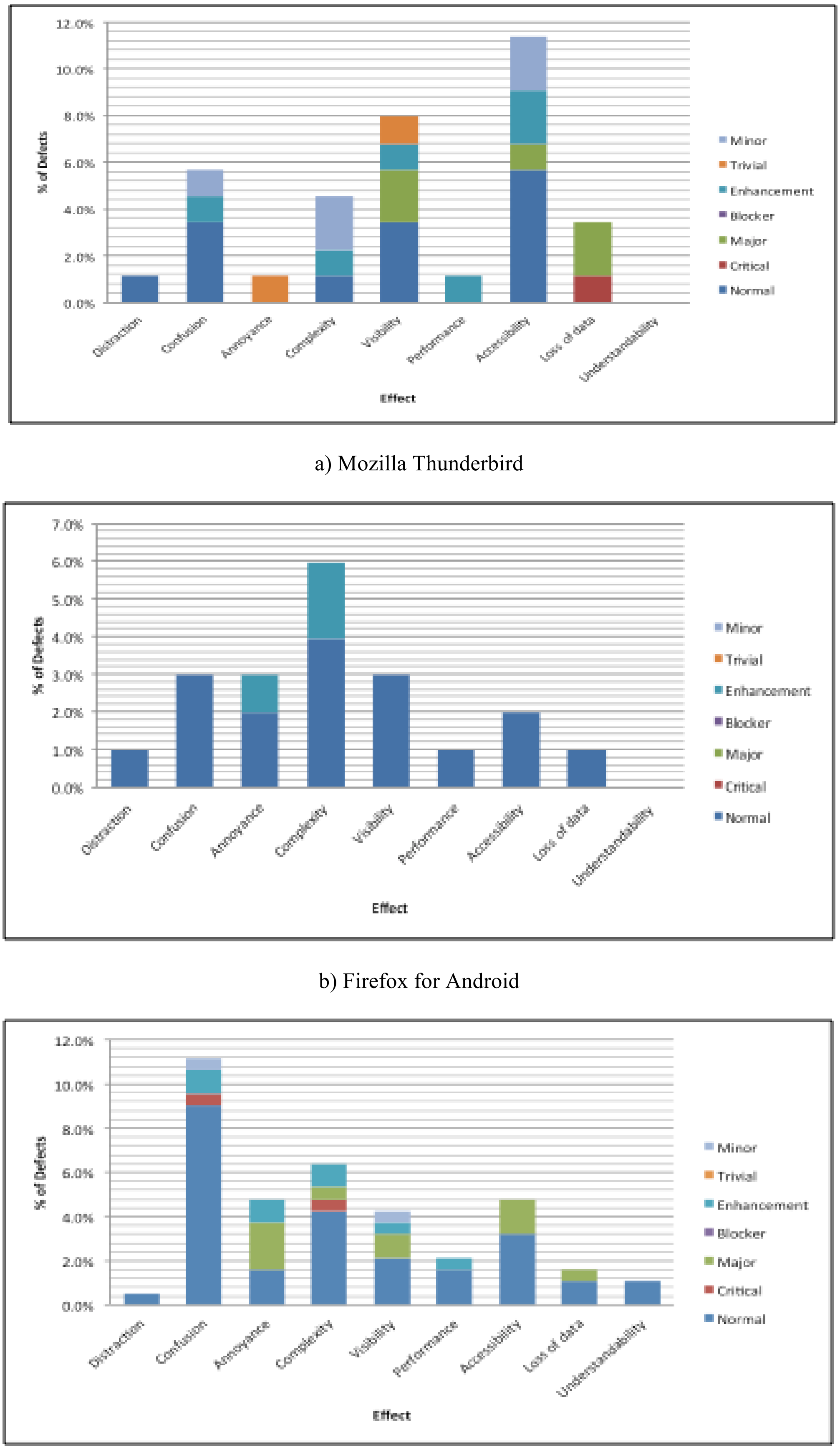
c) As shown in Figure 4.3b, issues related to functionality are the dominant interaction problem and our study results indicate that Firefox for Android has the highest missing functionality, accounting for 14.9% of defects. As Mozilla Thunderbird is a more stable project than Firefox for Android and Eclipse Platform, technical deficiency is very low in comparison.

d) For the task execution subcategories, incorrect task action has the largest proportion in Mozilla Thunderbird (17%), followed by the Eclipse Platform (13.3%). In all three projects, less than 2% of usability defects had reversibility problems.

e) As shown in Figure 4.3f, keyboard press accounts for the largest manipulation problems in Mozilla Thunderbird (9.1%) and the Eclipse Platform (6.4%). Across the three projects a voice control problem does not exist. This finding suggests that the voice control category is not pertinent to the current usability problem taxonomy and could be eliminated. Unsurprisingly given the nature of the application, Firefox for Android usability defects contained virtually no mouse click and drag and drop problems, and very few keyboard problems (1.0%). As Firefox for Android is an application developed for mobile devices, finger touch is the relevant category in Firefox for Android. Also, likely due to the fact that direct manipulation using mobile gestures is a much more sensitive interaction, scrolling is a more common manipulation problem in Firefox for Android than with Mozilla Thunderbird and Eclipse Platform.

4.2.2.3 The Impact of Usability Defects

In this section, we describe the distribution of effect and the correlation between effects and defect severity. We answer *RQ5: What are the effects of usability defects and what types of usability defects have a severe effect?* In our analysis, we identified two fundamental types of impact: the (1) impact on *human emotion,* to which the defect reporter described their feelings when they were experiencing difficulties using the software due to the usability defect, and the (2) impact on *task performance,* that prevented the reporter from completing task execution. Overall, when describing a particular users difficulty, reporters tended to support a single impact description at a time (rather than discuss both impact on emotion and task performance) and reporters provided little evidence for their impact claims. Figure 4.4 summarizes the distribution of different impacts with the corresponding defect severity.



a) Mozilla Thunderbird

b) Firefox for Android

c) Eclipse Platform

Figure 4.4. Distribution of usability defect impact and severity.

As can be seen, accessibility is the dominant task performance impact in Mozilla Thunderbird, accounting for 11.4% of sampled defects. In terms of impact on human emotion, confusion is the most addressed in all three projects: 5.6% in Mozilla Thunderbird, 3.0% in Firefox for Android, and 11.1% in Eclipse Platform. However, these percentages are still considered low suggesting that the open source usability defect description does not contain sufficient information on describing how the interface-interaction defects cause user difficulty that confuse and mislead users. Perhaps because of this, usability defect reporting tools should specifically prompt their users for this information in a separate field so that users can clearly describe this information.

4.2.2.4 Usability Defect Failure Qualifier

In a formal usability evaluation, a failure qualifier can usually be easily identified by observing how users experience a usability problem during software testing. However, in the context of open source projects, where usability defects are normally reported after "black box" usage without a proper usability evaluation method, it is quite difficult to identify the qualifier of the problems. This section answers *RQ6: On what basis, do usability defect reporters justify that the user difficulty that they experience is an issue*? This information is helpful for software developers to understand the nature of the problem. In our study, we used qualifier attributes of the ODC [77] and revised some of the original definitions to suit the context of our analysis, as shown in Table 6.2 in Chapter 6.

As shown in Table 4.22, the most common failure qualifiers across the three projects are incongruent mental model (27.3% in Mozilla Thunderbird, 20.8% in Firefox for Android, and 15.4% in Eclipse Platform), better way (25.0% in Mozilla Thunderbird, 10.9% in Firefox for Android, and 22.9% in Eclipse Platform) and wrong (23.9% in Mozilla Thunderbird, 9.9% in Firefox for Android, and 19.1% in Eclipse Platform). With regards to incongruent mental model, it was common for reporters to compare two different things that have some similar characteristics in order to justify a desire for consistency in design. It was also common for reporters to use their previous experiences to illustrate some point of view on certain issues:

> *With the menu changes, and the removal of the numbered perspective items, we need another way of switching between perspectives. We should add a perspective switcher****, similar to the editor and view switchers*** ... [Consistency]

With recent E4 builds I often see unjustified flickering in the workbench window. ***I haven't seen this before****.* [Previous experience]

Only a small fraction of usability defect reports contain missing (3.4% in Mozilla Thunderbird, 1.0% in Firefox for Android, and 1.1% in Eclipse Platform), overlooked (3.4% in Mozilla Thunderbird, 1.9% in Firefox for Android, and 1.1% in Eclipse Platform) and irrelevant (5.7% in Mozilla Thunderbird, 5.9% in Firefox for Android, and 3.2% in Eclipse Platform) to support the claim of the usability issues being reported. Overall, we observed that these failure qualifiers are primarily used to support a claim and defend solution proposals and ideas.

Table 4.22. Distribution of failure qualifier across projects.

Failure Qualifier	**Project**		
	Mozilla Thunderbird (n=88)	***Firefox for Android (n=101)***	***Eclipse Platform (n=188)***
Wrong	23.9%	9.9%	19.1%
Missing	3.4%	1.0%	1.1%
Irrelevant	5.7%	5.9%	3.2%
Better way	25.0%	10.9%	22.9%
Overlooked	3.4%	1.9%	1.1%
Incongruent mental model	27.3%	20.8%	15.4%
None	11.4%	49.5%	37.2%

4.3 Threats to Validity

There are several threats that can impact the validity of our findings.

4.3.1 Construct validity

Construct validity is concerned with the appropriateness of the metrics we used to study. For example, to investigate the presence of certain defect attributes in usability defect descriptions, we have chosen important usability attributes from human computer interaction studies [11]. One possible threat might be the reliability of the attributes to be significant for comparing performance-related defects with usability defects. We minimized this threat by using common defect attributes such as *steps to reproduce, actual output* and *expected output*, and introducing general attributes like *supplementary information*.

4.3.2 Internal validity

In conducting an online survey with anonymous participants whose professional background is unknown, misunderstanding of the survey context by respondents is a main threat to internal validity. Our goal is to focus on usability defect reporting instead of general software defect reporting. Since nearly half of the respondents have at least three years of experience in software testing, there is a possibility that the respondents have answered the questionnaire based on their general defect reporting knowledge and experience. We addressed this threat by (a) giving three different types of usability defect examples at the beginning of the survey, (b) highlighting the ***usability defects*** (bold and italic) keyword for every question, so the respondents were always aware of the survey context.

For software defect repositories mining, the information we extracted for this research is highly dependent on the capabilities of the main researcher to understand and analyze. For each defect in the sample, the main researcher manually read the textual description. This process required reading the bundle of text, interpreting the problems, classifying the information into any of the defined attributes and assigning a score if the information is present. Since this qualitative analysis leads to the subjective evaluation, incorrect interpretation, classification and scoring may potentially affect our results. To minimize this threat the research supervisors cross-checked a selection of classifications. We also built a checklist that consists of guidelines, so that both main researcher and supervisors are consistent when classifying and giving score for the information presented in the defect reports.

4.3.3 External Validity

External validity is concerned with the generality of our findings. In the survey outcome, one possible external threat to the validity of the survey outcome is the representativeness of the participants. While the participants were recruited from a range of software practitioners, there is the possibility that the software developers and testers responding have not used formal defect reporting processes and tools. Therefore, there is the possibility of response bias when providing answers and feedback.

In the software defect repositories mining, our study was limited to open source projects and restricted to Bugzilla defect repository. The collection of defect information for these open source projects may not guarantee that the same information is present in the other OSS projects as well. Other types of projects may have different defect report structure and different types of defect attributes. This is a common threat of this kind of research. To reduce selection bias and for comparison purposes, we selected two projects from the same defect repository while the other project is from a different defect repository. We examined three open source projects, with projects spanning diverse product functionality and environment; this small sample cannot be regarded as representative. This is because the pattern of information could be different when more samples are included in the analysis and different OSS projects are used. However, the findings of this research is an early indicator that existing defect information is not useful to explain usability issues, and hence need to be further investigated and confirmed with larger sample data with various OSS project backgrounds.

4.4 Analysis and Discussion

4.4.1 Reporting Usability Defects

Our qualitative observations on open source usability defect reports confirmed the online survey results that *actual output*, *expected output* and *software context* are the most supplied information when reporting usability defects. We believe that these findings are biased in a way the defect form is designed. In Bugzilla, for example, by default the defect form contains title/ summary, software information (i.e. product, component and version), steps to reproduce, actual results, expected results and attachment. Therefore, the use of this generic defect reporting form will produce mostly the same content structure for all types of defects. However, it is surprising that *steps to reproduce* is less reported in Mozilla Thunderbird, Firefox for Android and Eclipse Platform even software practitioners claimed that they always provide this information while reporting usability defects. Similar to [33], our findings suggest that *steps to reproduce* is not necessarily appropriate for all defect reports, as we found *steps to reproduce* is often used to explain issues for performance-related defects than for usability defects.

Even though most of the reporters can produce a relatively complete defect description, usability-related information is rarely included. For the three OSS projects studied, we found less than 2% of usability defect reports contain descriptions of violated usability heuristics. In fact, only about 27% of our online survey respondents claimed that they used violated usability heuristics to augment the cause of the problem. In terms of describing proposed solutions and expected results, barely a quarter of the reporters included usability design principles to justify their idea.

A closer inspection of usability defect reports showed that open source communities favorably discussed their ideas using narrative text rather than graphical structure such as ASCII arts and UI-digital mockups. The low amount of this information present in defect reports suggests that existing generic defect report templates do not emphasize the theoretical background of usability. This could be a disadvantage as usability defects may be seem insignificant in the absence of usability and HCI principles as a rationale to justify the problems and defending proposed solutions. Possibly due to the limited usability and HCI knowledge among open source communities, this information is somewhat difficult to provide. Perhaps, a defect report form should be designed as a wizard-style guided-answering form that consists of necessary information for different types of defects. A good example is the question-based structure form proposed by Simões [24]. The form contains of six questions to report HCI issues. However, the uses of text form to collect HCI information could be a burden for the reporter, as they may not know what to explain. To overcome this limitation, a pre-defined set of usability attributes for input selection can be explored in future work.

Many respondents considered *assumed cause, usability principles, video recording, UI event trace* and *title/ summary* to be the most difficult items to provide. Possibly, the varying level of technical knowledge of open source communities is one of the reasons why the defect reports in all projects studied contained virtually no *assumed cause.* Since UI event traces, for instance, are not readily available to end users, reporters may need to take extra steps. In these circumstances, reporters with deficient programming skills may only report problems in a GUI rather than a sequence of events that can be mapped into user tasks. This corresponds with Wang et al. [78]'s examination of open defect reports, which showed that most of the defect reports that contain technical information are often submitted by highly technical reporters. Moreover, a lack of automated tools and limited data types supported for recording and attachments in existing defect reporting tools make it challenging to include video and audio files [17].

As shown in Table 4.10, many reporters mentioned verbal discussions with developers as a common approach to communicate usability defects. In our opinion, as Andre et al. [3] addressed, verbal communication alone is not sufficient to resolve usability defects without a written description. This is because every software defect needs to record formal logs as an evidence to diagnose the problems. In this case, the use of a formal reporting tool, such as defect reporting tools were considered particularly useful to track and manage defects in a timely manner. However, as our survey results indicated, defect descriptions suffer from insufficient information for problem correction. One possibility is due to the text-centric approach used by the defect tracking system.

The unstructured text may contain a mix of data such as assumed cause, proposed solution and impact. This results in unorganized data and ambiguities that make it difficult to understand the whole issue as compared to data stored in fielded form. Furthermore, it is important to consider the subjective nature of the usability defects that may not be easy to explain textually [22]. Therefore, additional information in the form of attachable files may be required to complement this deficiency. We have identified four categories of attachable files: (1) screenshots, (2) videos, (3) graphical elements such as ASCII art, and (4) UI event traces / error messages. In this survey, we found that reporters tend to use screenshots, rather than videos, UI event traces and error messages when highlighting the cause of the problem, software context and actual output. Whereas, to propose a design solution, reporters often used textual descriptions as compared to graphical elements (i.e. digital mockups, simple sketches and ASCII art). Possibly, the low rate of use of non-textual media is due to the fact that good drawing tools may not be readily available [7] and producing graphical representations using a toolkit may require extra skill and time to learn and use [10].

4.4.2 Fixing Usability Defects

The most widely used textual information mentioned by software developers are *assumed cause, steps to reproduce, software context*, *expected output* and *actual output*. Similar to previous studies [20], [21] that focused on general defects, *steps to reproduce, expected output* and *actual output* were also commonly to be used by developers when fixing usability defects. In particular, it indicates that *steps to reproduce*, *expected output* and *actual output* are fundamental pieces of information for understanding all types of defects. In contrast to previous studies [20], [21], we added *assumed cause*, *usability principles* and *solution proposal* to the list of defect information to reflect relevant attributes for fixing usability defects. Out of these items, *assumed cause* was selected as the most useful and important attribute for fixing usability defects. Hence, our research extends the knowledge from the previous research.

Since our study is solely focused on usability defects that are primarily dealing with graphical elements, *screenshot* was rated as the most widely used supplementary information other than video, ASCII art, digital mockups and patch. When looking at the low frequency use of video, ASCII art, digital mockups and patch, we can explain that this is probably due to the reporters rarely supplying such information to justify the problems and propose their suggestions. Therefore, in the absence of this information, it is not surprising that software developers rarely use this information when fixing usability defects.

In [20], [21], software context was not considered useful, but was rated the third most widely used attribute in our research context. We think that the difference in these results comes from the different definition we introduced for software context. In our study, we referred to the software context as the specific location of problem in the interface where the problem was observed, such as button, menu and dialogue box. In contrast, [20], [21] defined software context as the operating environment where the problem occurred, such as web application.

Several problems in the way usability defects are described were identified. *Unclear assumed cause* and *insufficient information in steps to reproduce* was very strongly identified as problematic defect attributes. This is not surprising because, as noted above, the reporters found it indeed difficult to clearly explain the cause of the problem. In fact, the inability of reporters to supply UI event traces in defect reports will limit the essential information regarding user behavior and task sequences with respect to the application’s user interface. The absence of this technical information, such as the time to complete a certain task, number of erroneous action sequences, and usage of certain functions, may cause software developers to misinterpret the usability problems [79]. Another problem to consider is *“vagueness”*, a similar issue raised by Dumas et al. [16]. In our study, about half of the software developers claimed that they received vague comments. These either did not have precise problem descriptions or the solutions suggested were too general. According to Dumas et al., when describing usability problems or giving fix suggestions, reporters should be as specific as they can, and not let software developers use self- judgment. For instance, instead of suggesting “use a color with better contrast to the background and increase the font clarity” one could precisely suggest color contrast theory - black text on a brown background, for example.

4.4.3 Mismatch Between Information Provided by Defect Reporters and Information Needed by Software Developers

We compared the responses obtained from reporters and software developers to find out whether reporters provide sufficient information for a software developer to fix usability defects. In Table 4.23, attributes are ranked based on the mean of response to the questions *" Q13- Do you use the following items when describing usability defects*" and "*Q28 - Do you use the following items when fixing usability defects*" that range from 1 (Never) to Always (5). To discover the level of agreement between what reporters provide and what software developers need, we measured the absolute value of differences between reporters' and developers' mean. The lowest difference indicates more agreement and vice versa. As shown in Table 4.24, reporters and developers are in agreement on severity, software context, and expected result. However, more disagreements were observed. The most notable ones are *title/ summary* and *AC*.

Table 4.23. Rank of attributes – while software developers claimed *assumed cause* was the most needed information, reporters mostly provided *title*.

Rank (based on mean)	**Reporter**	**Developer**
1	Title/ summary	Assumed cause
2	Actual output	Steps to reproduce
3	Expected output	Software context
4	Software context	Expected output
5	Steps to reproduce	Actual output
6	Software information	Software information
7	Assumed cause	Severity
8	Test environment	Test environment
9	Severity	Title/ summary
10	Solution proposal	Solution proposal

Table 4.24. Agreement level between what reporters provide and what software developers need.

Item	**Mean ($\bar{X}$)**		Differences of mean
	Reporter	**Developer**	
Severity	3.77	3.78	0.01
Software context	4.11	4.15	0.04
Expected output	4.17	4.12	0.05
Test environment	3.88	3.68	0.20
Software information	4.02	3.82	0.20
Steps to reproduce	4.06	4.29	0.23
Actual output	4.32	4.08	0.24
Solution proposal	3.32	3.57	0.25
Assumed cause	3.98	4.35	0.37
Title/ summary	4.33	3.66	0.67

*Differences of mean = | $\bar{X}_{reporter}$ - $\bar{X}_{developer}$ |

While our study and [21] identified *title/summary* as the least problematic information, *title/summary* is not really needed by software developers to fix usability defects, as it was ranked as the second lowest. On the contrary, the *assumed cause* that is expected to be present in usability defect descriptions is seldom provided by reporters. In summary, our experiments suggest that reporters do not provide information that is frequently used by software developers.

4.5 Summary

The purpose of the research described in this chapter was to identify the most important usability information for reporting and correcting usability defects. We conducted a survey amongst open source software communities and industrial practitioners to understand the most valuable information in reporting and fixing usability defects. We extended previous studies [20], [21] that focused on software defects in general. We added *assumed cause, software context* and *proposed solution* in context of usability-related defect information. Our study extends the previous findings on software defect reporting. We discovered that developers need additional defect information when fixing usability defects. We found that *assumed cause* is the most useful, but seldom supplied by reporters. Our software defect repositories mining results confirm that *actual output*, *expected output* and *steps to reproduce* are also substantially important for software developers. These findings give us good ground to design a new defect report form for reporting usability defects, which we discussed in Chapter 7.

According to reporters, they usually provided *title/ summary, steps to reproduce, actual output, and expected output*. While usability-related information: *assumed cause, video recording, UI event trace* and *usability principle* are the most difficult to provide. However, our software defect repositories mining results revealed that *actual output, expected output* and *software context* are the most supplied information for usability defects. When we compared the responses from software developers and reporters, we found that the information most expected by the software developers was the least provided by reporters. The most significant ones were found to be *title/ summary* and *assumed cause*. Our statistical analysis shows a mismatch between what reporters reported and software developers claim that they need to fix usability defects. Our results showed that unclear *assumed cause* and insufficient information in *steps to reproduce* were most commonly experienced by software developers. This reaffirms with evidence of an anecdotal expectation that the cause of the problem is difficult to provide. Other problems include vague comments, unstructured text and duplication of reported usability defects.

To effectively report usability defects, there are several factors that may be influential – with the key area being skilled people supported by appropriate tools and methods. Through the same survey used in this chapter, we separately explained these factors in Chapter 5.

5 Factors Influencing Usability Defect Reporting

Previous studies have reported that the overall quality of defect reports is far from satisfactory. Common issues include incomplete information, lack of clarification and focus, mixed data, and mismatch information in defect descriptions. To obtain a quick defect resolution, a well-written defect report is important. While the majority of software defect reporting research has been devoted to the enhancement and improvement of new techniques and tools for capturing software defects, to the best of our knowledge, there is no study investigating the influence of different factors on the quality of defect descriptions.

To address this gap, we investigated our third thesis research question: *"RQ3 - What factors influence the description of usability defects?"* This research question is the second theme of the online survey we discussed in Chapter 4. This study aimed to collect opinions of developers and reporters on the factors influencing usability defect reports quality, including role of the reporter, reporters' experience and knowledge, defect discovery methods and usefulness of automation tools. The views are obtained from software development practitioners from industry who had experience in dealing with usability defects.

In this chapter, we present the results of the second part of our multi part survey that was discussed in Chapter 4. This second part of the survey examined what factors might influence the description of usability defects.

5.1 Methodology

5.1.1 Development of Survey Instruments

Before designing this survey, we reviewed relevant literature to avoid duplicate research, and where similar research existed, we learned and adopted questions and experimental design from these. Since we did not find any research into factors related to usability defect reporting in the literature, the relevant research investigating the influence of different factors on software testing practices in general was helpful. For example [74], [80], [81], explicitly investigate the importance of tester's knowledge in performing exploratory software testing. Kettunen et al. [82] also reported domain knowledge as the most emphasized area of testers' expertise and point out that the role of technical knowledge is particularly important in the agile development context. In a survey on the effect of experience, Kanij et al. [83] identified expertise in the problem domain and knowledge of specific testing techniques were significant factors influencing the performance of software testers. Beer et al. [84] and Poon et al. [85] reported the impact of the testers' experience on improving testing strategies - test case design, regression testing, and test automation. In the context of usability studies, F∅lstad [86] reported that usability defects found by work-domain experts were classified as more severe and received higher priority by developers than the usability experts. Therefore, in this research we speculate that the *role of reporter, reporters' knowledge and experience*, *defect discovery methods* and usefulness of *automation tools* might have a significant impact on reporting usability defects as well.

5.1.2 Research Questions

The main objective of this second theme of the survey was to investigate the influences of reporters' experience and knowledge, defect discovery methods and usefulness of automation tools on usability defect reporting practices. The key research questions to answer is listed in Table 5.1:

Table 5.1. The second theme of online survey research questions.

Research Questions		**Rationale**
RQ1	Is there a significant difference in the way that usability defects is described by the participants who have received usability/ HCI training and by those who have not?	Knowledge on usability or HCI can help participants to explain the usability issues better.
RQ2	Is there a significant difference in the way that usability defects is described between participants who have experiences in usability testing and those who have not?	Participants who have experience in usability testing may know better what kind of information is useful to report usability issues, and what kind of information is needed by software developers.
RQ3	Is there a significant difference in the way usability defects is described between the group of participants who used automated tools and those who did not?	Automation tool could speed up the defect reporting process, especially in collecting technical information and evidences.
RQ4	Is there a significant difference in the way usability defects are described between participants who conduct usability testing and those who do not?	Different testing approach will collect different information. For example, during system testing, the icon on the user interface may be reported as inappropriate. However, if usability testing is conducted, user's difficulties and how user is struggling to understand the icon can be understand.

5.1.3 Survey Design

We used self-administered online survey using Opinio survey tool. The respondents could answer this survey at their convenience. The survey is available at http://bit.ly/UsabilityDefectsReportingSurvey and can be found in Appendix B.

5.1.4 Questions Related to Factors Under Studied

In this section, we summarize the questions relating to the factors studied. The detailed structure of the survey can be found in Section 4.1.1.4. Each factor consists of several questions as listed below. Most of the questions used a Likert scale with five levels of agreements (“Completely disagree”, “Somewhat disagree”, “Neither disagree or agree”, “Somewhat agree”, and “Completely agree”).

1. *The role of the reporter*: This factor had two closed questions. Q33 listed five statements to investigate the way technical and non-technical users report usability defects. The software developers were required to rate the frequency of these statements may be true when they assess a usability defect. Q45 asked both software developers and reporters whether they think the level of detail of usability defects may vary between reporters.
2. *Knowledge*: This factor had four closed questions. Q7 and Q8 asked respondents to indicate their participation in any usability-related training/ certification and rate the usefulness of the training respectively. Q41 asked respondents whether they think defect report form should be customized according to reporters’ knowledge. Q46 listed seven knowledge factors that may affect the effectiveness of writing a good usability defect description. Respondents were required to indicate their level of agreement as to whether these were important.
3. *Experience*: This factor had four closed questions that asked respondents’ opinions on experience in usability testing (Q26 and Q34), and general software testing (Q48). Q40 asked respondents whether they think defect report form should be reflected to reporters’ level of experience.
4. *Automation tools*: This factor had four questions. Q42 asked if reporters have experience in using any automation tools to capture usability defect information. In the accompanying open question (Q43) reporters were requested to name the tools. Q44 asked reporters’ opinions on the limitation of the manual process to capture usability defect information. Q39 asked whether respondents think user experience (UX) of using defect reporting tools may influence the quality of defect reports.
5. *Defect discovery methods*: This factor had two closed questions. Q11 listed fives statements on the availability of usability defect information for different defect discovery methods, and respondents were required to indicate their agreement on these statements. Q12 asked whether

respondents think that defect report from should be customized according to how usability defects are discovered.

5.1.5 Evaluation of Survey Instruments

The survey was piloted with software engineers from the Swinburne Software Innovation Lab (SSIL) and software developers recruited at a developers conference (described in Chapter 4). Based on the verbal comments and the pattern of responses received, the survey instruments were refined.

5.1.6 Selection of Participants

The same selection criteria as detailed in Chapter 4 was used.

5.1.7 Data Analysis

Participants indicated their agreement about influences of usability defect reporting using five scale scores questions measuring the role of reporter, importance of knowledge and experience, influence of defect discovery methods, and usefulness of tools (1 implies “strongly disagree”, 2 implies “somewhat disagree”, 3 implies “neither disagree nor agree”, 4 implies “somewhat agree” and 5 implies “strongly agree”). Since each influence consists of several questions, composite scale scores were used for analysis by taking a mean of all the questions for each scale.

Chi-Square tests were used for statistical analysis. Since multiple Chi-Square were being performed simultaneously, we used the Bonferroni correction to avoid influence of spurious positives. In this case, we lowered the *P* value by dividing $P = 0.05$ by the number of tests to get the Bonferroni critical value. For example, to measure the association of usability defect attributes and usability/ HCI knowledge for reporters (attributes tested = 10) and software developers (attributes tested = 16) a test would have $P<0.05/ 10 = 0.005$ and $P<0.05/16 = 3.125\text{x}10^{-3}$ to be significant.

5.2 Results

5.2.1 Demographic Information

The details of respondents’ background were reported in Chapter 4.

5.2.2 Factors Influencing Usability Defect Reporting

We postulated five factors (“role of reporter”, “knowledge of specific usability/ software engineering”, “experience in software/usability testing defect reporting”, “automation tools”, and “defect discovery methods”) that we believed might influence the way usability defects are described. Each factor has several questions, where participants indicated their level of agreement that these were influential. The responses to these factors are reported in the following sections.

5.2.2.1 Role of the Reporter

We obtained developer feedback on the influence of role of reporter when they assessed and dealt with usability defects (Q33). As shown in Figure 5.1, more than 50% of software developers agreed that technical users, such as usability experts and developers, provide more informative usability defect information, which include proposed solutions. Reponses to Q45 suggest significant variability in level of detail between reporters (see Table 5.2).

Table 5.2. Responses to “the level of detail of usability defect reports varies greatly from reporter to reporter” (Q45).

Completely disagree	0.7%
Somewhat disagree	4.1%
Neither disagree or agree	18.4%
Somewhat agree	29.9%
Completely agree	29.9%
No response	0.7%

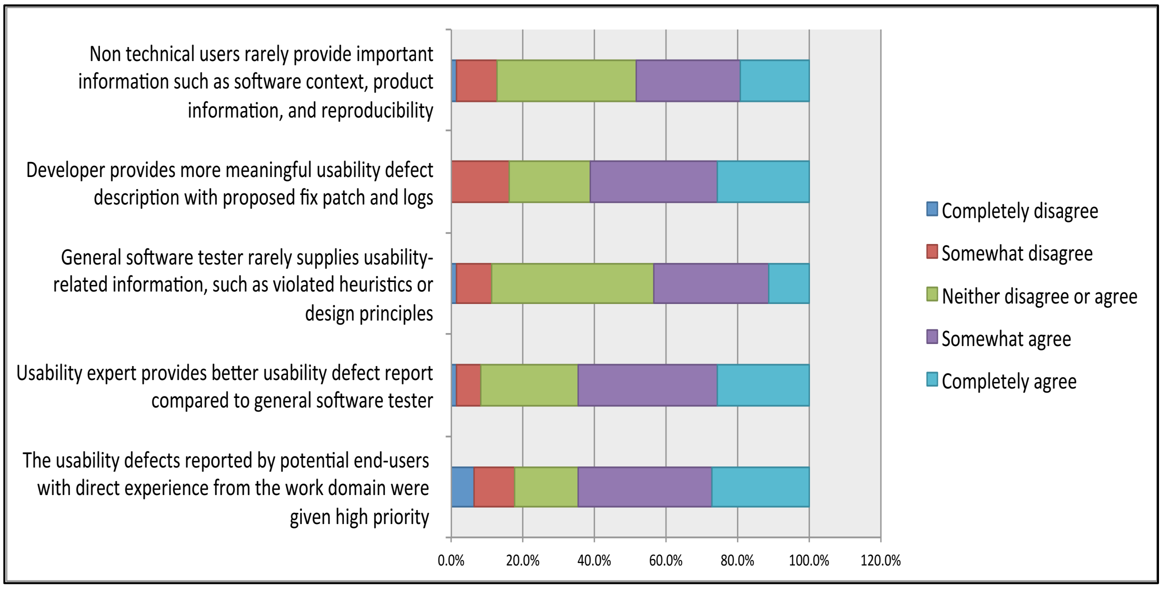


Figure 5.1. Role of reporter in reporting usability defects (Q33).

5.2.2.2 Knowledge of Specific Usability/ Software Engineering

We listed seven knowledge factors ("practical knowledge", "user's perspective thinking", "users' mental model", "usability principles", "domain expertise", "technical knowledge", and "defect reporting") that we postulated might influence the ability of respondents to write good usability defect reports. More than 50% of respondents agreed that these factors were influential (see Figure 5.2). The level of agreement was highest for "user's perspective thinking" and "practical knowledge".

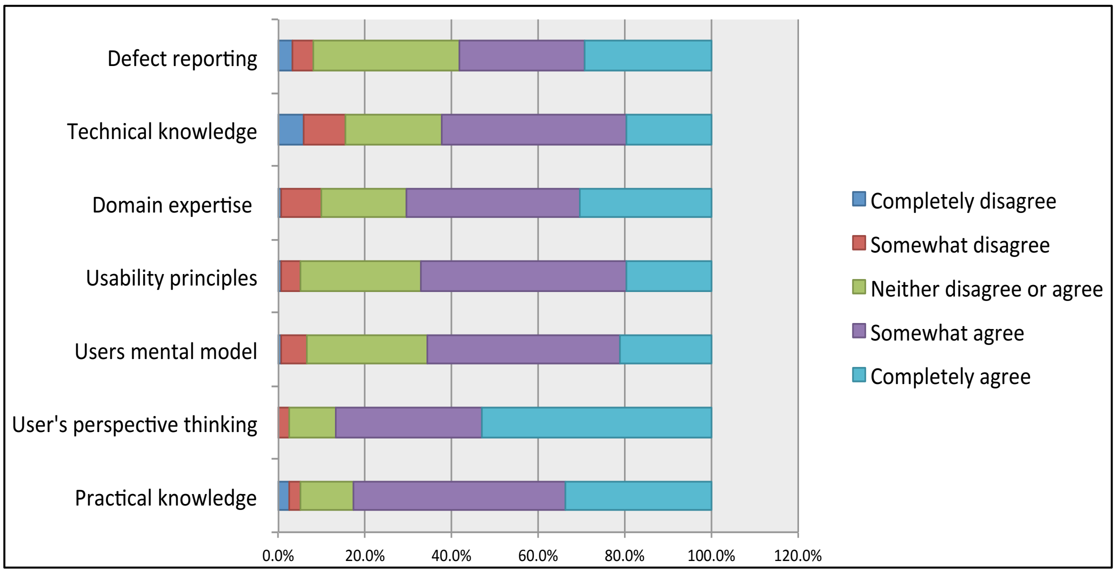


Figure 5.2. Responses on "knowledge factors" in reporting usability defects (Q46).

In the related question (Q41), we obtained respondents opinions on the value of a "custom defect report form" – whether a defect reporting tool should provide a custom form depending on reporter's knowledge (non-technical, technical, HCI expert). While 51% agreed that a defect report form should map to the reporter's knowledge, 11.5% disagreed and 21.1% neither disagree nor agree (see Table 5.5).

Only a minority (about 17%) indicated that they had received usability-related training/ certification (see Table 5.3). Of those, about 12% of them have experience in reporting usability defects and the remaining 5% fixing them. For those who had acquired the related training, the majority of them found the training was useful (see Table 5.3).

Table 5.3. Usability-related training received by the participants (Q7).

Role in dealing with usability defects	**Participation in usability-related training**	
	Yes	**No**
Reporting usability defects	17	65
Fixing usability defects	8	57

Table 5.4. “Usefulness” of usability-related training received (Q8).

Very useful	44%
Somewhat useful	40%
Neither useful or not useful	4%
Not very useful	12%
Complete waste of time	0%

Table 5.5. Responses to question “Do you think your defect reporting tool should provide custom forms for reporting defects depending on reporter’s knowledge?” (Q41).

Completely disagree	5.4%
Somewhat disagree	6.1%
Neither disagree or agree	21.1%
Somewhat agree	33.3%
Completely agree	17.7%
No response	16.3%

Table 5.6. Association between usability/ HCI knowledge and frequency of usability defect attributes supplied.

Usability defect attributes		**Software developers**		**Reporters**	
		χ2 (df=2, N=65)	**p-value**	**χ2 (df=2, N=82)**	**p-value**
Title/ summary		1.741	0.419	3.692	0.158
Assumed cause		0.832	0.660	6.820	0.033
Software context		2.589	0.274	5.366	0.068
Actual output		0.884	0.643	6.623	0.036
Expected output		1.541	0.463	0.703	0.703
Steps to reproduce		0.518	0.772	5.732	0.057
Severity		1.783	0.410	13.725	0.001
Solution	Solution proposal	1.520	0.468	2.857	0.240
	Fix patch	0.333	0.847		
	Digital mockups	2.737	0.255		
	ASCII art	0.928	0.629		
SoftInfo	Product	1.332	0.514	4.773	0.092
	Component	5.208	0.074		
	Version	1.435	0.488		
TestEnv	Hardware	0.194	0.907	5.814	0.055
	Operating System	1.824	0.402		

To answer RQ1 of the first theme of the survey, a Chi-Square test was performed to examine the relationship between participants who received usability (or related) training and usability defect attributes. As can be seen in Table 5.6, three of the usability attributes are significant (severity, assumed cause, actual output) for reporters. As we tested 10 usability defect attributes, we expected one or two attributes to show a significant result purely by chance. By applying the Bonferroni correction, at significant level 0.005 ($P < 0.05/10 = 0.005$), only the test for severity (χ2, (2, N=82) = 13.725, $P < 0.005$) is significant. However, at significant level P < 0.05/16 = 3.125 x 10-3 (the number of usability attributes tested are 16 for software developers), there is no significant relationship between usability knowledge and usability defect attributes for the software developers.

5.2.2.3 Experience in Software/ Usability Testing

A majority of respondents agreed that experience in software testing and usability testing has a good influence on the quality of usability defect report (~ 71% agreed in Q26). This view is supported by many software developers (~58.4%), that reporters are much better at proposing redesign solutions when they have usability experience (Q34). Furthermore, around 60% agreed, with minimal disagreement (less than 6%), that length of experience has a significant influence (Q48). In addition, nearly 62% of respondents agreed that prior experience in usability defect reporting helps write better reports (Q49). Since no open-ended questions were asked around these factors, it is difficult to further clarify these findings.

We asked respondents whether they agreed that a defect reporting tool should provide a "custom defect report form" – that is defect report form should reflect the reporter's experience, and their level of experience in software/ usability testing (Q40). Our findings show considerably more respondents choosing "somewhat agree (31.3%)" than "completely agree (23.8%)", and only a small proportion disagree (12.9%).

To answer RQ2 of the first theme of the survey, the survey included questions about whether reporters had experience in software testing (no experience, less than 1 year, between 1 and 3 years, between 3 and 5 years or more than 5 years). The "less than 1 year", "between 1 and 3 years", "between 3 and 5 years" and "more than 5 years" scores were converted to a score of 1 to indicate "yes", and "no experience" scores were converted to a score of 0 to indicate "no".

A Chi-Square test at significant P=0.05 was performed to examine the relationship between reporters with software testing experience and usability defect description attributes: there were seven significant attributes (see Table 5.7). However, by applying Bonferroni critical value, at significant level P=0.005, only relations between software testing experience and title/summary (χ2, (2, N=82) = 15.216, $p < 0.005$), context (χ2, (2, N=82) = 24.696, $p < 0.005$), expected results (χ2, (2, N=82) = 11.187, $p < 0.005$), and severity (χ2, (2, N=82) = 13.508, $p < 0.005$) were significantly observed.

Table 5.7. Association between software testing experience and frequency of usability defect attributes supplied.

Usability defect attributes	**$\chi 2$ (df=2, N=82)**	***P*-value**
Title/ summary	15.216	0.000
Assumed Cause	4.963	0.084
Software context	24.696	0.000
Solution proposal	2.300	0.317
Actual output	8.005	0.018
Expected output	11.187	0.004
Steps to reproduce	9.058	0.011
Severity	13.508	0.001
SoftInfo	6.828	0.033
TestEnv	6.527	0.038

5.2.2.4 Influence of Defect Reporting and Automation Tools

In response to Q25 and Q35, more than 50% of the reporters and 38.7% of software developers indicated that they use defect reporting tools to manage usability defects. Common tools (Q36) were JIRA, Bugzilla and Redmine. Mantis, Trello, IBM Rational Team Concert, and Visual Studio TFS were also listed multiple times. For JIRA, Bugzilla and Redmine users - 90% agreed to some extent that the defect reporting tool offers sufficient flexibility to capture and manage usability defects (Q37), but free-text feedback revealed considerable negative satisfaction (Q38). The following are representative: “Most of the defect reporting tool do not have exhaustive options for usability defects” and “JIRA more customized by client but no specific customizations done for usability”.

In response to Q42, only 16% of reporters used tools to capture additional usability defect data, while 61% had never used any additional tools. For those who used automated tools, we asked them to name the tool, using an open-ended question (Q43). Among the tools mentioned were CrazyEgg, Google Analytics, Microsoft Snipping tool, QuickTest Professional (QTP), WinRunner, Snagit, LICEcap, Screencast, Jing and Selenium. We asked reporters to indicate whether they agreed that the manual process used to capture usability defect information is a cause of erroneous or incomplete defect reports (Q44). Our findings reported a mixed response, with roughly 20% of the reporters agreeing to some extent, but 22% disagreed and 35.4% neither agreed nor disagreed. As no open-ended question was provided, it is difficult to justify the satisfaction and feedback responses associated with those tools. It is notable that a high proportion of respondents either indicated a dissatisfied view, or did not answer the question at all (23.2%).

In a related question, we asked respondents' opinion if the user experience of using defect reporting tools might influence the quality of defect reports (Q39). Our findings indicate that nearly 56% believed that user experience is one of the key considerations when reporting usability defects.

To answer RQ3 of the first theme of the survey, a Chi-Square test was performed to examine the relation between reporters that used automated tools to capture usability defects and usability defect description attributes. At Bonferroni correction significant level P=0.005, a significant association exists between used of automated tools and context ($\chi2$, (2, N=63) = 11.443, P< 0.005). Table 5.8 summarizes the statistical results.

Table 5.8. Association between the usage of automated tools and frequency of usability defect attributes supplied.

Usability defect attributes	**$\chi2$ (df=2, N=63)**	***P*-value**
Title/ summary	5.418	0.067
Assumed cause	1.079	0.583
Software context	11.443	0.003
Solution proposal	4.435	0.109
Actual output	4.199	0.123
Expected output	3.311	0.191
Steps to reproduce	3.482	0.175
Severity	1.920	0.383
SoftInfo	0.536	0.765
TestEnv	5.054	0.080

5.2.2.5 Influence of Defect Discovery Methods

The vast majority of respondents agreed that the amount of information available for reporting usability defects varied according to how the defects were discovered. In this context, defects found during usability testing revealed more information as compared to other types of testing – approximately 87% of respondents agreed that using usability testing information about user's knowledge, likely difficulties, actual task scenario and realistic redesign solutions can be obtained from the actual user, and approximately 73% agreed other usability-related information such as violated heuristics or design principles, and user response and feelings can be collected (see Figure 5.3).

In an accompanying question (Q12), we asked respondents whether a defect reporting tool should provide "custom defect report forms" (that is, a defect report form should be designed to map to different types of defects) – the majority of the respondents (~63.9%) agreed and only 7.5% disagreed with this idea.

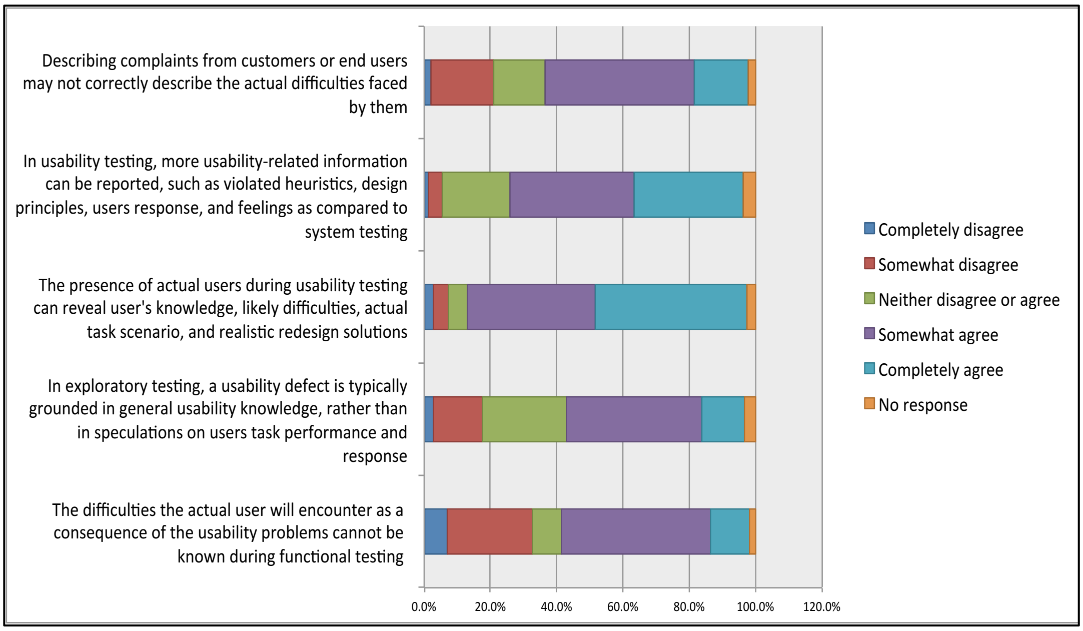


Figure 5.3. “Influence of defect discovery methods” in reporting usability defects (Q11).

Table 5.9. Association between usability testing and frequency of usability defect attributes supplied.

Usability defect attributes	**χ2 (df=2, N=82)**	***P*-value**
Title/ summary	2.045	0.360
Assumed cause	14.530	0.001
Software context	4.361	0.113
Solution proposal	1.146	0.564
Actual output	6.066	0.048
Expected output	3.245	0.197
Steps to reproduce	6.020	0.049
Severity	5.274	0.072
SoftInfo	4.455	0.108
TestEnv	0.270	0.874

To answer RQ4 of the first theme of the survey, the survey asked questions about how reporters discover usability defects (i.e., exploratory testing, functional testing, usability testing, beta/ alpha testing, complaints/ reports from customers, using the product, and other). Participants were allowed to choose multiple answers. Any response that consists of a usability testing option were converted to “usability testing” response options, while other responses that did not have usability testing option were converted to “not using usability” response options. A Chi- Square test was performed to examine the relation between reporters that conducted usability testing and usability defect description attributes. Table 5.9 shows that at significant level 0.05 three attributes were significant, however by applying Bonferroni correction a test would have to have P<0.005 to be significant. Therefore, only the cause is significant (χ2, (2, *N*=82) = 14.530, p < 0.005).

5.3 Threats to Validity

We have considered three types of possible threats that can affect the validity of the survey results, which we discuss below.

5.3.1 Internal Validity

The main threat to this study is a misunderstanding of the survey context by the respondents. Our goal was to focus on usability defect reporting instead of general software defect reporting. Respondents may have answered the questionnaire based on their general defect reporting knowledge and experience. We addressed this threat by (a) giving three different types of usability defect examples at the beginning of the survey, and (b) highlighting the ***usability defects*** (bold and italic) keyword for every question, so the respondents were always aware of the survey context.

5.3.2 External Validity

One possible external threat to the validity of the survey outcome is the representativeness of the respondents. Although 147 valid responses to the survey is a good start, the population size is relatively small to examine the research questions and answer them in appropriate manner. There could have been some level of confirmation bias among the participants as not all of them were directly working in HCI projects, reporting usability defects, or even using formal defect reporting processes and tools. For example, only a small number of people used automation tools (16%) to report usability defects. As such, we cannot claim our findings about the question on automation tools to be conclusive. Instead, our survey may be a plausible start for more detailed empirical studies. Furthermore, the use of a self-rating technique to identify how participants discovered usability defects did not really reveal the participant's experience, approach and skill in usability defect discovery.

5.3.3 Construct Validity

One concern is regarding incorrect measures, i.e. not precisely measuring respondents' practices in reporting usability defects. To mitigate this concern, we reused previous surveys and added questions from both usability and software engineering fields. Another possible threat is that our respondent recommendation does not entirely reflect the true reality of defect reporting practice. Since our survey is anonymous, some responses we received stated that they have never used defect reporting tools. In fact, some comments were not at all meaningful.

5.4 Discussion

In this section, we present an overall discussion of our findings regarding the five factors that we postulated to influence the quality of usability defect reporting. This study has presented initial findings on how respondents see the listed five factors influencing the descriptions of usability defects.

5.4.1 The Role of the Reporter

In earlier research, it was recognized that the evaluator or software tester plays a significant role in the outcome of software testing [14], [87]. Since communicating defect information is part of software testing activities, we argue that the presence of evaluator effect will also influence the quality of defect information submitted.

In this study, most of the respondents agreed that the reporters' role matters a great deal in getting a usability defect fixed. There are possibly two aspects in OSS development that support this factor. The first aspect lies on the different *user capabilities* to supply meaningful information. In the context of OSS development, defect reporters can vary between three groups [4], [88]; 1) software developers with high technical background and in-depth knowledge of the software, 2) user support who may posses expertise in using the software, but may have very limited knowledge and technical details of the software, and 3) end users who only used the software as a "black box". Among these three groups of users, responses from our survey revealed that technical users often provided more informative usability defect information, such as redesign solutions, which is important for software developers to correct software defects [9]. However, the solutions from these technical users may pose a threat to less-technical users. For example, the subjective nature of usability means that some technical users (with no usability background) may be able to personally justify the usability problems. However, the problem might not be an issue for the wider user base, or in fact, could lead to "fixes" or "shortcuts" or "simplifications" that negatively affect less-technical users.

The second aspect is related to *important and trusted reporter*. As Zimmermann et al. [20] claimed, "well known reporters usually get more consideration than unknown reporters, assuming the reporter has a pretty good history in bug reporting". This suggests that when software developers have put their trust in a reporter, they are likely to give them a higher priority. We also found that usability defects reported by customers typically get reviewed and fixed faster. Possibly, defects raised by customers have higher business value and fixing their defect reports is a key driver in gaining users' loyalty.

Based on this finding, we suggest that one criterion for effective defect report writing is to customize the defect report content according to the reporter background. For example, if the reporter is an end user with limited access to the software, information such as configuration of the application, hardware information, product information, event traces, and error logs should be automatically collected rather than manually asking them to fill in the form. This is because such technical information is very important for software developers to diagnose the problems but they were the most problematic information as reported in [21].

5.4.2 The Role of Experience and Knowledge in Usability Defect Reporting

Experience and knowledge have been studied in the context of software engineering, especially in software testing [80], [84], [85], [89]. Often, experienced software testers could find more software defects. However, in the context of reporting a defect, previous studies have reported that only a small group of experienced reporters could produce a good quality of report because of their limited understanding of what constituted a defect [4]. This suggests that while software testing experience might be useful for test execution and evaluation, it does not guarantee a production of high quality defect reports. For this reason, we studied three categories of experience that may influence the effectiveness of writing usability defect descriptions – experience in usability testing, experience in general software testing, and experience in reporting usability defects. In the opinion of our respondents, they strongly believed that these experiences were helpful to provide more organized and detailed usability defect descriptions. Since our survey does not quantitatively measure to what extent the quality of the usability defect report is due to the influence of these experiences, it would be unwise to over-interpret their response. However, this finding raises a question on how testing skills and expertise would be beneficial to an individual's usability defect report writing effectiveness.

Concerning the influence of defect reporter's knowledge, "users' perspective" and "practical knowledge" were the most agreed type of knowledge that should be acquired by defect reporters in producing well-written usability defect descriptions. In the context of reporting usability defects, we argue that understanding the user's perspective and practical knowledge are interrelated knowledge elements. While understanding the user's perspective could be much easier to gain, practical knowledge, on the other hand, may require direct observation and participation in activities [84] and be increased by the involvement of professional experience [85]. In this case, we see that to be a good defect reporter, despite understanding the problems from what the user feels, the defect reporter also needs to have a deep system knowledge of the features, and in most cases, based on personal experience as a user of the system [74]. As Itkonen and Lassenius [74] reported, a good understanding of usage procedures and context allows defect reporters to identify actual usability defects by reflecting on the system's behavior with realistic usage tasks and context rather than based on personal speculation.

As well as practical and domain knowledge, some of our respondents agreed that understanding usability principles and knowledge are desirable. However, only 17% of respondents had received usability-related training. For respondents with related training, they seem to understand usability defects, and tend to provide severity ratings. Despite the value of this type of training it does not seem to be common. We assume usability-related training is normally conducted in the context of learning about user experience, building skills on UX best practices, and designing user interface[2]. The training seems to provide insufficient focus on writing good defect reports.

Knowledge on defect reporting tools was also considered important in writing good usability defect reports. As Ko and Chilana [4] reported, lack of knowledge of defect reporting tool and process may influence the quality of defect reports. We argue that current defect report forms offered by most defect reporting tools are too generic to support different types of defects that may need different kinds of information reported. A range of enhancements to existing defect reporting tools is suggested in [43], [90]–[92] but none of them are specifically designed to consider the reporters' knowledge and experience. Novice defect reporters in particular struggle to prepare good usability defect reports [93]. Perhaps, defect report forms could be customized to reflect a reporter's profile (i.e., non- technical user, usability expert, customer, etc).

[2] https://www.nngroup.com/training/

5.4.3 Use of Automation Tools

Many usability engineering tools offer automated data capture and richer kinds of information capture e.g., instrumenting applications to capture traces and user interaction, recording richer user interaction and mapping to user task, and capture of video, audio, screenshots, diverse interaction (touch, sketch, gesture, accelerometer, as well as keyboard and mouse). As shown in Section 5.2.2.4, however, not many reporters used automation tools to capture usability defect information. Among the tools mentioned by our respondents to record screenshots, video and audio are Snagit, Screencast, Jing, LICEcap, and Snipping tool. Since these tools are not tightly integrated into defect reporting workflow, there is friction in creating useful hypermedia attachments, especially in adding and manipulating attachments. Perhaps, by combining HCI usability engineering methods and tools with software defect reporting and management repositories will be of greater value.

Concerning usability defects, existing research on usability defect descriptions has shown the value of capturing additional usability context. Instead of knowing which particular interface element causes confusion, it is useful to know how and why users get confused [22]. Therefore, in addition to screenshots of the problematic interfaces, it might be useful to automatically record facial expression, body language, gestures, and tone of voice of the users conducting usability tests. Perhaps these kinds of recording tools could be integrated with defect reporting tools to reduce the defect reporting workload.

Furthermore, software engineering research on collecting logs and traces [90], [91] and automatically capturing steps to reproduce [43], [92] does not seem to have a big contribution in the practical usability testing context. Possibly, this preference for commercially available software may be attributed to the fact that tools developed by research community tend to be concept demonstrators with ad-hoc support.

5.4.4 Defect Information Obtained Through Usability Testing

In our survey, about 60% of reporters used usability testing to discover usability defects. The majority believed that usability information such as user's knowledge, likely difficulties, actual task scenario, realistic redesign solutions, violated heuristics, and user's feelings are more easily obtained via usability tests. In addition, respondents also agreed that reporters who conducted usability tests are able to explain possible causes better other than traditional testing. Possibly, when reporters observed users performing certain tasks, they gain direct feedback from the users. Quantitative data such as users' performance (i.e., time on task and error rates) also can be collected.

However, in OSS development, since no formal usability evaluations are conducted such information heavily relies on the ability of the users who experienced the problems to precisely describe them. As stated earlier, the diverse backgrounds of OSS communities make it difficult for the users to identify and provide valuable information. Therefore, we suggest that additional information such as impact, assumed caused, and violated heuristics should be explicitly prompted to the reporters. In this way, reporters will be guided to supply meaningful information even if they are not "usability-savvy".

5.5 Summary

Our survey responses indicate that, according to software development practitioners, quality of usability defect descriptions does strongly depend on the experience and knowledge of reporters, use of automation tools, and defect discovery methods. However, our finding is preliminary and needs to be extended. Researchers could explore the effect of implicit usability testing activities, and the type of knowledge and experience applied when reporting usability defects.

The responses indicate that the experience and knowledge of defect reporters has a great influence on the information presented in the usability defect descriptions. Reporters who have usability and/ or HCI knowledge and have software testing experience were likely to provide severity ratings, title/ summary, context, and expected results when describing usability defects. We also found that reporters who used automated tools were likely to capture software context better than those who used manual processes. Also, reporters who discover usability defects using usability testing were likely to provide possible causes other than those who used other testing methods. We consider such influence in designing our new usability defect report form as discussed in Chapter 7.

6 A New Open Source Usability Defect Classification Taxonomy

In open source software usability defect reporting, little research to date has focused on understanding just what different categories usability defects belong to. Existing defect repositories, such as Bugzilla, have used keyword functionality to label usability-related defects. For instance, a defect can be labeled as *uiwanted, useless-UI, ux-affordance, uc-consistency* and *ux-efficiency.* However, such a high-level classification does not assist developers to identify the underlying flaws or problems. In fact, lack of descriptions, examples and limited usability terms make it difficult for non-human computer interaction evaluators to assign such labels for certain usability defects [50]. Moreover, understanding software engineering classification models, such as ODC and RCA, requires users to understand usability problems from the context of software development and the perspective of software developers. Since usability defects are subjective in nature, characterizing usability defects using predefined attributes does not really reveal the users' task difficulties [94]. These limitations encouraged us to revise the existing usability defect classification models. There are several reasons for categorizing usability defects:

1) to more clearly disclose the probable causes of the defect;
2) to highlight the impact of usability defects on the task outcome;
3) to treat usability defect priority the same as the other defects; and
4) to quantitatively track usability defects over time.

This chapter answers our fourth thesis research question *"RQ4 - How should open source usability defect reports be classified so that they can be effectively reported?"* Based on our analysis of open source usability defect reports, we revised the existing usability defect classification model [12], [76], [95] to incorporate software engineering and usability engineering needs. We also aimed to collect feedback on a new proposed open source usability defect classification model by requesting software development practitioners to classify a sample of usability defects.

6.1 Existing Usability Defect Classification Schemes

In the classification schemes based on a usability engineering perspective, usability defects are often categorized according to a single perspective. Earlier efforts to classify usability defects were done by

Nielsen and Molich [96]. They developed nine heuristics to assist usability evaluators to assess the usability of a user interface. Since Nielsen's heuristics only offer a high-level classification, there are some usability problems that cannot be mapped to any of these heuristics [12].

To overcome this limitation, Keenan et al. [12] developed the UPT that classifies usability defects into artifact and task components. The artifact component consists of visualness, language, and manipulation categories, while the task component consists of task mapping and task facilitation categories. Each category is composed of multi-level subcategories. For example, language consists of two-level sub-categories; the first level consists of naming/ labeling, and other wording, while in the second level, other wording if further categorized into feedback messages, error messages, other system message, on screen text, and user-requested information/ results. The depth of classification along the components and categories may result in one of three outcomes; full classification, partial classification, and null classification.

This approach to classification, however, relies on a high quality defect description, which as our earlier chapters demonstrate, are rarely present in open source usability defect reports. Our observations are that many open source usability defect reports have defect descriptions that contain a lack of contextual information, particularly on the user-task. As a result, when using UPT to classify usability defects, we have to make many assumptions and a self-judgment about the task performed by the users that lead to the problems. We believe UPT is useful for usability evaluators to assess the usability defects during usability evaluation with the presence of users, but not to classify defects by just reviewing the usability defect description.

Andre et al. [3] have expanded the UPT to include other usability engineering support methods and tools. By adapting and extending Norman's [97] theory of action model, they developed UAF that used different interaction styles. For example, the high-level planning and translation phase contains all cognitive actions for users to understand the user work goals, task and intentions, and how to perform them with physical actions. The physical action phase is about executing tasks by manipulating user interface objects, while the assessment phase includes user feedback and the user's ability to assess the effectiveness of physical actions outcome. Even if the UAF was viewed as a reliable classification scheme that supports dissimilarity of defect descriptions for the same underlying design flaw, the complexity in determining which phase of the interaction the problem occurred is a real challenge for novice evaluators.

Concerning the limitation of single perspective classification schemes, many researchers have started to explore usability defect classification models from multiple perspectives. One of the most prominent approaches is the cause-effect model. For example, Khajouei et al. [98] argued that the lack of information on the effects of usability defects in UAF have caused a long discussion to reshape the way developers think about usability. They augmented the UAF to include Nielsen's severity classification and the potential impact of usability defects on the task outcome, in order to provide necessary information for software developers to understand and prioritize problems.

Vilbergsdottir et al. [99] have developed a Classification of Usability Problem (CUP) framework that consists of two-way feedback; Pre-CUP that describes how usability defects are found, and Post-CUP that describes how usability defects are fixed. In Pre-CUP usability evaluators use nine attributes (Defect identifier, frequency, trigger, context, description, defect removal activity, impact, failure qualifier and expected phase) to describe usability defects in details. Once the usability defects have been fixed, the developers record four attributes (actual phase, types of fault removed, cause and error prevention technique) in Post-CUP.

Geng et al. [95] claimed that CUP can capture important usability defect information and provide feedback for usability improvement, but that CUP could not be used to analyze the effect on users and task performance. Considering the importance of the cause – effect relationship, they have customized the ODC and UPT. This usability-ODC framework consists of three causal attributes (artifact, task, and trigger) and four effects attributes (learning, performing, perception, and severity).

In relation to open source defect reporting, we found that some of these attributes are not relevant for open source communities that have a very low involvement in usability experts. For example, technical information about defect removal activity, failure qualifier, expected phase, and frequency are difficult to obtain, particularly for users with limited human-computer interaction knowledge. In fact, without formal usability evaluation in open source projects, the trigger attribute as suggested in the usability-ODC framework is not possible to justify. Additionally, the use of pre-defined values for some of the attributes may introduce selection bias and users are likely to select incorrect values.

Other usability problem classifications use a combination of models to support practical use of classification in different software development context [94]. This model-based framework consists of three perspectives, in which each perspective is facilitated by the use of models: artifact-users-tasks-organization-situation model for Context of Use, abstraction hierarchy model for Design Knowledge and function-behavior-structure model for Design Activities- in which the usability problem needs to be analyzed through the collective consideration of the three models. The Context of Use perspective is to understand the cause of the problem, either related to design factors (violated user interface design guidelines) or non-design factors (user preferences). If a usability problem is judged as "design factors", it should be further analyzed from the Design Knowledge and Design Activities perspectives. Such a reference framework would allow usability evaluators to develop a specific classification scheme for a particular context. However, a lesser degree of involvement of usability evaluators in open source projects makes it quite impossible to adopt such a comprehensive framework. In fact, contributors who participated voluntarily in open source projects prefer to work more on the main functionality of a certain application rather than focusing on user-centric design [7].

6.2 Rationale for Revising Existing Usability Defect Classification

Generally, software problems are often classified as a *defect* when software is not behaving to its expected purpose, or *enhancement* when a feature is desired but not present. From a software engineering perspective, cause-effect classifications provide a deeper understanding of a software problem. To the best of our knowledge, only one usability cause-effect classification currently exists. Geng's classification [95], in our view, is not appropriate to classify open source usability defects with limited information. The trigger component in the causal attributes can be limited because in the absence of formal usability evaluation in OSS development, it is quite impossible to identify the usability evaluation methods that trigger the usability defects.

Even if usability evaluations were to be conducted, OSS projects would still lack the effective mechanism to formally conduct the evaluations, for two reasons. First, many of the volunteers who contribute to OSS development are developers, who might have limited knowledge and skills required for usability evaluation. Second, in order to formally conduct usability evaluations, extra commitment from contributors is necessary, but volunteers may not be able to spend more time on this. Considering these limitations, we revised Geng's classification [95] to better suit an OSS environment. The following paragraph summarizes the rationale for our revision.

Defect category - In software development, quantitative measurements such as the amount of memory used, the time to load application or response time is very crucial and often gets immediate attention from software developers, as opposed to subjective usability issues that cannot be scientifically quantified and measured. To address this issue, common open source defect repositories such as Bugzilla have implemented *keyword* functionality to address usability heuristics terms, such as consistency, jargon, and feedback so that the concept of user interface and the underlying implementation could be described effectively. Each usability issue is tagged with the specific usability heuristic being violated. In this way, software developers with limited usability and interface design knowledge could learn about the heuristics and understand how the same types of defects could be resolved.

However, current UX principles being used by Bugzilla's keyword functionality are too broad [100]. Software developers may be able to know the violated usability principles but they might not be able to relate the common characteristics for the set of problems to reveal the root cause and resolution. For example, using the term POSTDATA in a dialog is ux-jargon. This high-level classification is unable to facilitate the identification of specific problems on specific software context and provide quick summaries of the problematic interface components or defective task execution. Thus, by adopting a usability classification scheme such as UPT, software developers will be enabled to analyze usability defects more easily. Managers will be enabled to monitor the trends, patterns, and occurrence of certain types of usability defects for tracking improvements.

Effect – Previous studies reported that usability defects are treated at a lower priority compared to functional defects [23]. In the existing ODC classification, severity is used to measure the degree of the defect impact. However, due to unclear usability category definitions, many usability defects end up with low severity ratings [23]. From our analysis of open source defect reports discussed in Chapter 4, we think unclear and missing descriptions about user difficulty caused by the usability defect is one of the reasons why software developers do not prioritize the importance of fixing many reported usability defects. The fact that only a small fraction of usability defect reports contain *impact* information reveals the lack of contextual information to convey information to software developers about the user difficulty and how it impacts user emotion from the perspective of usability engineering. However, the use of only textual descriptions to capture user difficulty could be a disadvantage as users are likely to provide lengthy explanations that may be unhelpful to many software developers. One way to reduce this limitation is to create a set of predefined impact attributes so that the impact can be objectively measured. For example, we can use rating scale to measure emotion, while task difficulty could be selected from a predefined set of attributes.

Causal – Since no formal usability evaluation is usually conducted in OSS projects, usability problem triggers cannot be identified. In OSS projects, usability defects are most often reported from online user feedback facilities and results of developer black box testing. Considering this limitation, instead of looking at trigger attributes we study the failure qualifier of the problem. This information could help software developers to understand the reason why a user considers the problem as a valid usability defect.

6.3 Our New Revised Open Source Usability Defect Classification Taxonomy

We began to construct our new usability classification model by reviewing usability defect classification models in the literature, in particular UPT [12], ODC [77], and usability-ODC framework [95]. While we wanted our usability defect classification to be in line with software engineering principles, we also wanted to develop a model that is simple and easy to use by users (the users can be, for example, software developers, testers, customer support, or end users) with limited usability knowledge.

In contrast to the previous classifications [12], [95], our usability defect classification model is designed to address usability defects reported in open source defects without a formal usability evaluation method being used. Thus, observable data such as time-on-task, number and types of help sought, frequency of expressed frustration cannot be obtained from the reporters' self-reporting defects. In fact, in open source usability defect reports, we could not identify testing techniques employed by the users that triggered the usability defect. For these reasons, we revised the previous classification models to only consider information normally available in the open source defect reports.

We adapted the original ODC framework to better understand usability defect causes and effects, and integrated it with usability practices. Figure 6.1 illustrates the high-level cause-effect usability defect classification model. We collected 377 usability defects from Mozilla Thunderbird, Firefox for Android, and Eclipse Platform project. We began by randomly reading a few samples of defect reports in detail, trying to understand the defect content structure, and to construct a list of potential metrics to be used for ODC attributes (cause and effect attributes). Most of the defect descriptions that we read contained significant information about defect reproduction, actual results (i.e., what is wrong with the usability defects), expected results (i.e., what should be fixed) and user difficulty expressions.

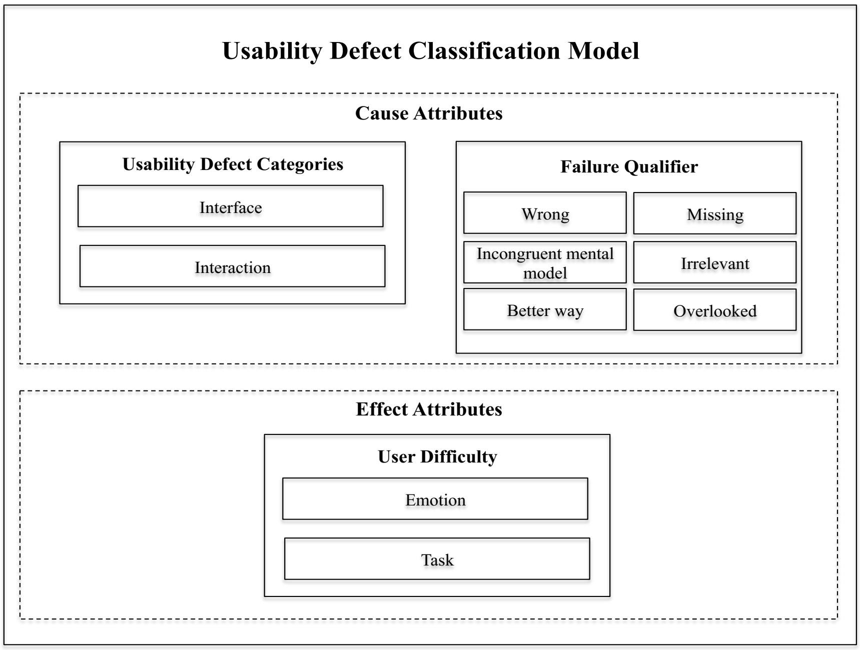


Figure 6.1. Our revised usability cause- effect defect classification model [95].

Based on this common information, we included *usability defect categories* and *failure qualifier* as ODC-cause attributes and *user difficulty* as ODC-effect attribute to our first draft of classification model. The rationale of using failure qualifier and user difficulty is to help developers understand the validity of the problem and help them to prioritize defect fix accordingly.

We started to classify one project at a time. We used user-written statement criteria to signal *usability defect categories*, *failure qualifier,* and *user difficulty* attributes. To analyze the defect descriptions into the three attributes, we conducted a card sort. Card sorting is a method to generate information grouping of specific data items [101]. In our case, we applied both closed and open card sorts. In a closed card sort a set of pre-defined categories were used to organize and map the defect descriptions into usability defect categories [12], [95], and failure qualifier attributes [102]. While in an open card sort no predefined groups were used, instead the groups were emerged and evolved to identify common themes for user difficulty attributes.

Once we had identified more global definitions of categories and subcategories, we then proceeded to more rigorously classify the other two projects, and iteratively refined these categories and their definitions. Finally, when the model was established, the other three supervisors read half of the sample independently, applying the final classification model, and consulted the category definitions, and terminology until a consensus was reached. We assessed agreement informally throughout.

The remainder of this section provides an overview of the three attributes, including the changes and addition to the original UPT.

6.3.1 Cause Attribute

This attribute is derived from ODC. In the original ODC, cause attributes are measured using defect types and trigger. *Defect type* gives information about type of defects uncovered by different testing techniques, while *trigger* is a condition that allows a defect to be discovered. However, in our research we used defect types to group usability defects that share common characteristics, and trigger was used to understand the underlying usability design flaws. The detailed description of defect types and trigger are given below.

Usability Defect Categories

To classify usability defect categories, we used closed card sort. We began by classifying the written usability defect description to a set of predefined usability defect categories as in [12], [95]. Since we experienced some difficulties – lack of specificity and insufficient definition when using [12], [95], and lack of information in defect reports during our preliminary analysis, we revised the original UPT by only categorizing the defect types into two categories – *interface* related defects and *interaction* related defects [76]. Interface defects refer to defects affecting the structure and behavior of graphical user interface (GUI) aspects that affect the overall look and feel of the application. Interaction defects refer to defects affecting the interaction process when a user interacts with a GUI. To reflect these two categories, we reconstructed the original UPT's primary and subcategories as follow:

- The original UPT category "visualness" is replaced by "GUI structure and aesthetics" as the primary category, the "language" category is removed and all the subcategories are assigned to new category "information presentation", and the "manipulation" category is moved to interaction defects.
- We extracted three primary categories of interface defects - GUI structure and aesthetics, information presentation, and audibleness. GUI structure and aesthetics are about feel and display given by interface, such as colors harmonious, object affordance and layout

coherence. We only retained two subcategories of the original UPT (object appearance and object layout), replaced “object movement” subcategories with “object (screen) state”, and moved two subcategories “presentation of information/ results” and “non-message feedback” to “information presentation” primary category.

- The primary category “information presentation” is about information relevancy and credibility of data, feedback message, on screen text, and results presented in the user interface. It is divided hierarchically into six subcategories. We adapted “data presentation” subcategories in [76], reused “non-message feedback”, “error, notification and feedback message” in UPT, and added two subcategories “on screen text” and “menu structure”.
- The primary category “audibleness” was adapted from [95] to accommodate the audio, speech and voice capability. We replaced the subcategories “prompt” with “audio cues”.
- For interaction defects, we extracted three primary categories – manipulation, task execution, and functionality. In contrast to the original UPT, the primary category “task mapping” and “task facilitation” are refined.
- Manipulation is concerned with the user’s ability to understand and manipulate user interface objects [12]. We adapted four subcategories as in ODC-usability framework – keyboard press, mouse click, finger touch, and voice control. We added three subcategories; scrolling mechanism, drag and drop, and zooming.
- Task execution focuses on the outcome of certain tasks. We adapted three subcategories as in the fault model [76] – action, reversibility, and feedback. Referring to the original UPT, we considered the subcategories “interaction”, “navigation”, and “task/function automation” as “action” subcategories. The subcategory “alternatives” was replaced by “reversibility”, and the two subcategories “user error tolerance” and “keeping the user on track” were combined in “system task feedback” subcategory.
- Functionality refers to any problem due to the capabilities provided by the product. We reused functionality definitions from [103].

The resulting model is illustrated in detail in Figure 6.2, and a definition of each category is listed in Table 6.1. The detail categories, subcategories, and example defects can be found in Appendix C.

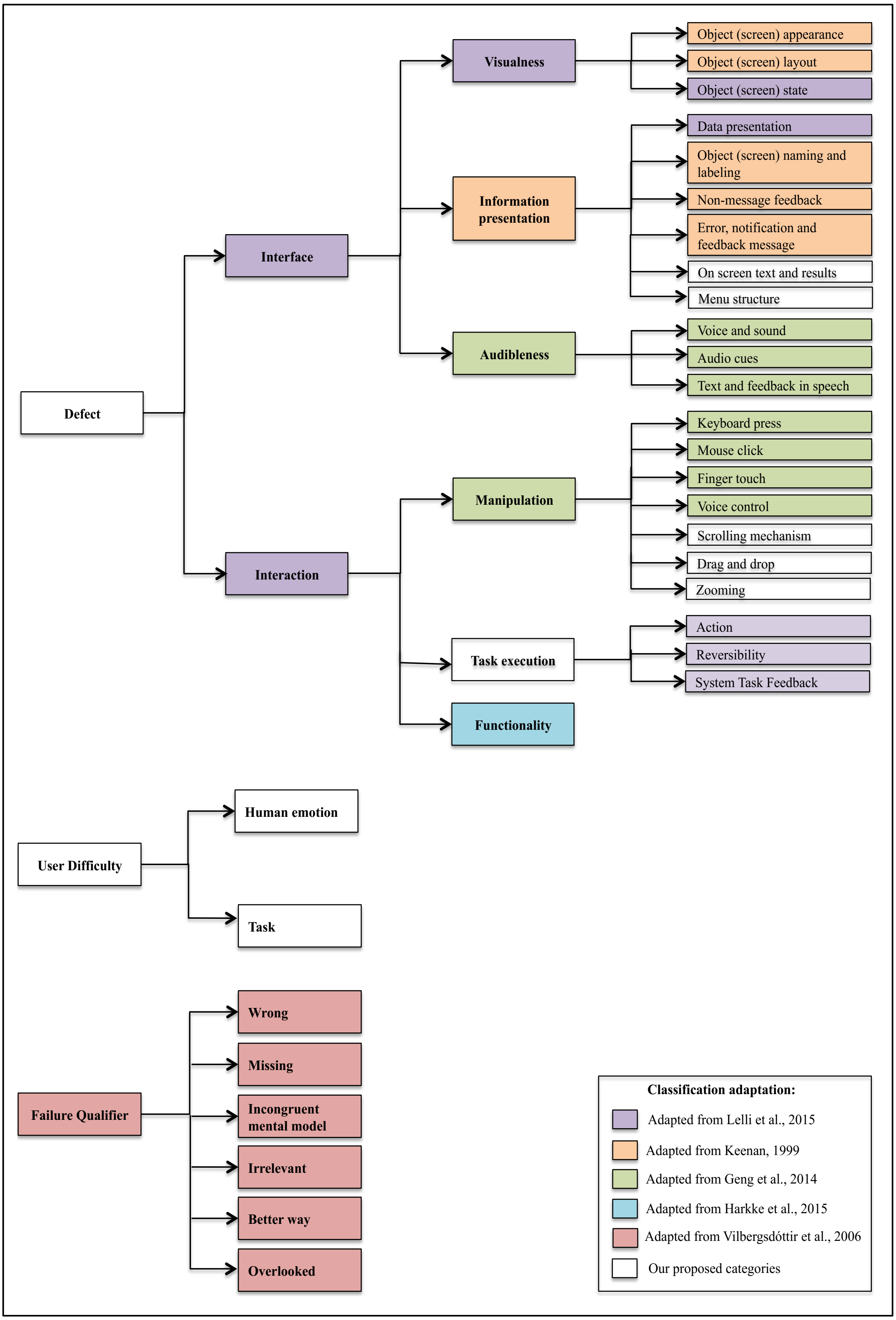


Figure 6.2. Hierarchical structure of defect types, effect and failure qualifier. The colors indicate the different sources we adapted in our classification model.

Table 6.1. Definition of key defect categories.

Defect	Definition
Interface	Any unpleasant graphical user interface aspects that affect the overall look and feel of the application.
Visualness	Any difficulty encountered by the user when they view objects (icon, menu item, scroll bar, button, favicon), symbols, and images present in (or missing) the user interface.
Object (screen) appearance	Refers to how individual objects look, sound, or appear to other senses. These problems involve object affordance such as the use of colors, size, and animation.
Object (screen) layout	Refers to layout coherence and how user interface objects are laid out on the screen. These problems involve spatial organization, such as the use of balance and symmetry, the alignment and spacing of elements, the grouping of related elements, the placement of screen objects, and consistent use of the GUI elements across applications.
Object (screen) state	Any difficulty encountered by the user when they cannot recognize or are unclear about the effect of object state change, including the change to its behavior and appearance.
Information presentation	Any difficulty encountered by the user when they view, read, and interpret the information or data presented in the user interface.
Data presentation	Refers to how data is presented, structured, and controlled.
Object (screen) naming and labeling	Any difficulty in language such as words/ terminology used as names on objects (such as buttons, title bars, field labels) and screens [12]. These problems also include inconsistencies of naming and labeling standard.
Non-message feedback	Any difficulty that is due to distracting, annoying, and confusing feedback [12], and insignificant use of visual cues that appears while using user interface.
Error, notification and feedback message	Any difficulty in language such as words used in phrases and sentences in error, notification, and feedback message. These problems also include the ability of users to understand and interpret the meaning of information presented in the message.
On screen text and results	Refers to completeness, accuracy and credibility of information in on screen text/ instructions, online help and tutorials, and results of user queries.
Menu structure	Refers to organization of menu, and grouping of related options.
Audibleness	Any unpleasant audio like sound management and sound alerts.
Voice and sound	Refers to any problem related to audio cues at the interface like giving distracting, disturbing, and annoying sounds, or missing sound alerts when the message comes to the screen.
Text and feedback in speech	Refers to any problem when there is difficulty in understanding speech signs and translating these to text.
Interaction	Any difficulty encountered by the user when they interact with the application
Manipulation	Any defects that occur when the user has trouble with some aspects of manipulating objects on the user interface
Keyboard press	Any difficulty encountered by the users when they use keyboard to interact with the user interface. These problems include the problematic use of access keys as a shortcut to issue menu commands.
Mouse click	Any difficulty that involves the user's ability to use mouse and its buttons to directly manipulate objects (speed of cursor tracking triple clicking, depressing multiple mouse buttons simultaneously) [12]
Finger touch	Any difficulty encountered by the users when they touch areas of the screen to move the pointer, press button, and manipulate image.
Voice control	Any difficulty encountered by the users when they use voice signals to activate user interface or invoke certain tasks.
Scrolling mechanism	Any difficulty encountered by the users when they use vertical scrollbars to move data up and down, and horizontal scrollbars to move the data left and right within the view.
Drag and drop	Any difficulty encountered by the users when they select and drag an object, and drop it into another location in the interface.
Zooming	Any difficulty to change gradual image scaling operation.
Task execution	Any defects encountered by the users that inhibit a user from completing an intended action.
Action	Any defects that occur as a result of executing a task.
Reversibility	Refers to the ability of the application to allow user to explore the interface and make mistake, and roll back the action such as multi-level undo operation, and the ability to cancel long-running actions.
System Task Feedback	Refers to the ability of the application to always keep users informed about what is going on for every user event, such as prompt a warning, status, and error message.
Functionality	Any problem that is due to the facilities provided by the product to user.

Trigger

Another aspect of cause attributes is to help software developers understand the root causes that trigger dissatisfaction of the software product from the viewpoint of the users. Since in OSS projects, usability is not formally evaluated with the presence of actual users, it is quite difficult to explain to software developers why certain aspects of the software product become an issue for some users. With this in mind, we believe an effective classification model is the one that may explain why users are experiencing problems. In the ODC model [77], trigger measures the nature of testing being conducted in order to highlight the kind of testing techniques necessary to discover defects. However, in open source projects, such information is not available. Therefore, when analyzing trigger attributes from defect description we used *failure qualifier* from ODC as a set of predefined metric.

In the usability evaluation context, the failure qualifier attribute is used to capture more information about usability defects through verbal communication with the test participant or recording user test session [99]. For instance, the usability evaluator can ask the test participants why they did not notice the presence of menu or some elements on the user interface (missing). On the contrary, we interpret one failure qualifier just based on a statement written in the defect report. As such, we have refined the definitions of the failure qualifier attributes to suit our case as listed in Table 6.2. We assume that this ODC failure qualifier would be a good ground on which to base users' justification on how they discover the usability defects.

Table 6.2. Failure Qualifier – sample phrases from usability defect reports [99].

Qualifier	**Definition**	**Representative quotes from sample**
Wrong	When the reporter notices that something has gone wrong while performing a task or some elements on the user interface are violating usability principles and standards.	1. The New project wizard has an icon based on the closed project icon, which is **not** how it would appear to the user. 2. On reload, the lock icon **should** immediately disappear when the old document has finally gone away, not when reload has just been tapped upon
Missing	When the reporter fails to find something in the user interface that he/ she expected to be present, or the results of performing certain task did not meet his/ her expectations.	1. When initiating a WebRTC call, Firefox for Android currently **doesn't** pop up a permission request to use the camera/microphone. We **need this** to pref on.
Irrelevant	When the user interface contains information objects, steps to accomplish task or functionality that do not contribute to system services and are unnecessary	1. This is **needless** functionality and annoying….) 2. It is **pointles**s to show the button that lists all your tabs when you only have one open. In fact, it really is pointless to show them unless you have more tabs than your Window can hold.
Better way	When the reporter suggests that something in the user interface could have been done differently, or suggests a different way of doing a certain task	1. It's **nice** to change the dialog resizable and scrollable in the tabs' contents for temporarily (The Account Settings is so). 2. I **would prefer** that if someone wants to re-populate the dialog text from selected text, they simply type ^F again.

Table 6.2. Failure Qualifier – Sample phrases from usability defect reports [103] (Continue)

Overlooked	When the reporter overlooks an entity in the user interface, or does not know how to perform a certain task	1. It happened to me a couple of times that I **thought** that I closed all editors by mistake 2. **Didn't know** how to change it back or that it's even possible
Incongruent mental model	When the user interface is unclear because it does not match the reporter's mental model, previous experiences, or they notice inconsistencies with other similar applications	1. I **haven't found** a official DL link for Royale but the one I now used looks most "official". 2. **I expected to** be able to enter my username and my password as usual, not having the keyboard overlapping text input fields.

6.3.2 Effect Attribute

In many defect classification models [77], [95], [102] severity rating is commonly used as a metric to measure the potential effect of usability defects on the intended user. Since usability defects severity tend to be unfairly treated by software developers [23], we argue that defect severity is not a reliable metric for analyzing usability defects for software quality improvement.

In Geng's classification [95], the effect attribute is studied from three components – problem in learning, problem in performing given tasks, and user perception. Each of these components constitutes more specific values such as that problem in learning has three values - learnability, memorability, and retention over time, and performing has two values - effectiveness and efficiency. These values are measured by examining time, effort, success rate and level of happiness showed by users when performing assigned tasks during usability testing. Since usability defects in OSS projects are not directly observable from usability test data, such metrics cannot be used in our model. In fact, in software development we recognize that the impact the defect has on the user and likelihood of occurrence is important to prioritize defect fixes, but such information is rarely present in defect reports, or if it is reported, the information is not clear. For this reason, we examined the effect of usability defects as the user difficulty, in terms of *human emotion* and *task performance.*

In this research, we examined statements and phrases of defect reports to decide which statements constitute impact on human emotion and impact on task performance. Using open card sort, the main researcher reviewed statements and generated labels of the emotion and task difficulty, then merged and sorted the lists into meaningful descriptive labels, and finally, created a set of codes. From our analysis, we interpreted impact of task performance based of the software quality attributes, such as accessibility, understandability, noticeability, loss of data, complexity, and visibility. For human emotion such as confusion, frustration, and annoyance, our interpretation was based on the terms such as “distract”, “annoy”, “frustrate”. If such terms are not presented in the defect reports, we analyzed the phrases such as [104]:

- “I am confused about …”
- “I’m not sure …”
- “I don’t know …”
- “I can’t figure out …”
- “I am having a problem …”
- “I assume …”
- “How do I …”

We started the examination with the Mozilla Thunderbird dataset. Once the Mozilla Thunderbird dataset had been classified, the process was repeated with Firefox for Android and Eclipse Platform datasets. If there were inconsistencies, the draft codes were modified and refined again. Finally, the main researcher and supervisors reviewed and discussed the appropriateness of the resulting codes and definitions. Overall, when reporting a particular usability defect, reporters tended to address a single difficulty at a time, and reporters provided little evidence to substantiate their difficulties claim. Table 6.3 and Table 6.4 define each of these user difficulty attributes, and lists sample phrases from open source defect reports.

Table 6.3. Effect on human emotion and quotes for each. Bold indicates emotion that affected human emotion.

Emotion	**Definition**	**Representative quotes from sample**
Distraction	Anything that draws a user’s attention away from their current focus or desired focus of the user interface or doing a certain task	1. The user is confronted with too many coolitems and their "grab bars". It is **distracting**, and it is unlikely that this granularity of repositioning is required. 2. The lock animation that I'm currently seeing is **distracting**. It makes me take a second look at the screen to find out why something has just moved.
Confusion	The feeling of being unclear about the software function	1. This leaves the novice user in an **unexpected state**. 2. I find the navigation arrows in the toolbar **confusing**.
Annoyance	Frustration or hardship induced by using the user interface or software functionality that leads to irritation, frustration and anger	1. I find it **frustrating to navigate** to the file when I know the name and just want it opened. 2. Some people (me) find vibrate on every single click to be quite a **nuisance**.

Table 6.4. Effect on task performance and quotes for each. Bold indicates software qualities that affected task performance.

Task	**Definition**	**Representative quotes from sample**
Complexity	The difficulties about understanding and using the software product and its components, in which the user has to perform irrelevant actions or needs to perform extra steps to accomplish a task	1. I was able to copy the file on to my windows machine, edit it using Thunderbird on that machine, and then re-copy it to the EeePC, but that **should not be necessary**. 2. It generally **takes about 2-4 taps to figure out** where the checkbox tap target is at …
Visibility	The poor capability of user interface or product components to keep users informed about what is going on, through appropriate feedback, obvious prompts and cues within reasonable time	1. The cvs and resource icons are **hard to distinguish** as well. 2. … active editor (tab) **hard to detect** (see screenshot)
Performance	The effect on task execution such as peed efficiency, availability, accuracy, throughput, response time, recovery time, resource usage	1. Clicking on the 'Plug-in Details' or 'Configuration Details' buttons in the About dialog are **time-consuming** operations when the product in question has a large number of plug-ins 2. When a file type is selected the dialog is frozen for a **very long time** and no busy cursor is shown.
Accessibility	The difficulties the user has to access, use and benefit from certain functions in the user interface. The degree to which a product, device, service, or environment is available to as many people as possible	1. Therefore, **I can't do any searches** 2. … thus a new user will **not be able to click** anything on that page at all.
Loss of data	An unexpected error made by the user when performing a task, in which information is destroyed by failures, or neglect in storage, transmission, or processing.	1. No thanks for making me **lose everything**, my tabs, bookmarks etc by adding an extra search app, which I do not need. 2. This will cause **data loss** and perceived instability in the IDE
Understandability	The difficulties about understanding the user interface metaphors and product functionality	1. If you close all the views in the perspective, it remains open, but looks very bare, and **it's not clear what the user can do next**. 2. … The new user **has no idea** what a perspective is.

6.4 Evaluation

We wanted to evaluate our new usability defect classification taxonomy with reporters and developers. We used a web-based survey as a tool to evaluate our proposed revision of usability defect classification. The following subsections describe evaluator selection, problem selection, and protocol in conducting our evaluation.

6.4.1 Evaluator Selection

Our evaluators were recruited from the researchers' industrial contacts. The evaluators had varying levels of experience in industry and academic software development environments. The evaluators do not necessarily require HCI expertise, since the aim of the OSUDC taxonomy evaluation targets the understandability of the model rather than the utility of the classification. Participation was voluntary and evaluators were allowed to discontinue participation at any time during the research activity. The consent to participate in the survey was implied by the return of the anonymous questionnaire. However, a precise response rate cannot be determined, as the total number of the evaluators who received the invitation is unknown.

We obtained approval from the Swinburne University of Technology Human Research Ethics Committee (SUHREC) prior conducting this survey (Approval number: SHR Project 2016/325). The details instruments used for the evaluation can be found in Appendix F.

6.4.2 Problem Selection

We randomly selected ten usability defects from the 377 usability defects that had been examined during the analysis phase prior to building the revised open source usability defect classification.

6.4.3 Protocol

We evaluated the classification model using web-based survey, Opinio tool. We conducted self-administered evaluation survey, as this kind of survey approach offers greater flexibility to evaluators. Evaluators can participate at locations and times of their choosing. The survey was opened from February until May 2017. Each evaluator was given the following material:

- OSUDC taxonomy document – to understand how the taxonomy works, sample problem classifications, and glossary of terms.
- Link to the survey – the survey has three sections. In the first section, evaluators are required to fill out a small survey questionnaire about personal background. The pre-questionnaire contains a total of six questions. In the second section, evaluators are given ten usability defects to be analyzed according to the OSUDC taxonomy. Each usability defect contains a total of four to six questions depending on a evaluator's answer (s). In the third section, the evaluator was asked to give feedback based on their experience of using the proposed taxonomy.
- Consent Information Statement – to indicate evaluator consent to participate in the study.

The evaluators were not monitored and were allowed to classify the problems in any order, revisiting any problems they wished. There was no time limit imposed on the evaluators. See Appendix D for complete evaluation questions.

6.5 Results

In our survey, we only focused on the inter-rater agreement between evaluators when classifying usability defects. Since our target users are from OSS communities that have varying level of knowledge in HCI and usability-related matter, this research is aims to produce a simple taxonomy that could be understood by both technical and non-technical users when they first read the OSUDC documents. Based on our small-scale evaluation strategy, the findings discussed in the next section established principles of classifying usability in OSS projects and subsequently portrayed an early design element of the required OSS defect reports.

6.5.1 Evaluator Demographic Information

A total of twelve evaluators from 26 to 55 years of age, participated in the evaluation of OSUDC taxonomy. The majority of the evaluators were female (66.7%). As shown in Table 6.5, most of the evaluators are academic researchers and software developers, accounting for 41.7% to 50%. Almost 92% of evaluators had not received any training or certification related to usability evaluation/ HCI/ UX. As Table 6.6 indicates, the majority of evaluators had limited familiarity in handling usability defects. In terms of experience in using a defect classification scheme, RCA was the most commonly used classification scheme among the evaluators (see Table 6.7).

Table 6.5. Demographic information of the evaluators.

Job responsibility	Evaluators
Academic researcher	50.0%
Software developer with experience in both user interface and software development	41.7%
End user with HCI/ UX/ usability knowledge	8.3%

Table 6.6. Evaluators' familiarity with usability defects.

Extremely familiar	0.0%
Moderately familiar	8.3%
Somewhat familiar	41.7%
Slightly familiar	41.7%
Not at all familiar	8.3%

Table 6.7. Experience in using defect classification scheme.

Orthogonal Defect Classification (ODC)	8.3%
Root Cause Analysis (RCA)	25.0%
Hewlett Packard Defect Classification Scheme (HP-DCS)	0.0%
Classification of Usability Problems (CUP)	0.0%
Usability Problem Taxonomy (UPT)	0.0%
Usability Action Framework (UAF)	16.7%
Other	8.3%

6.5.2 Reliability Analysis

To measure the reliability of evaluators' agreement to classify usability defect reports using our revised usability problem taxonomy, we used Fleiss' kappa as reference [105]. The Fleiss' kappa is an extension of Cohen's kappa to measure inter-rater agreement between three or more evaluators. We used the Real Statistics Data Analysis Tool[3] installed in an Excel spreadsheet to calculate the Fleiss' kappa values.

For the classification of the defect category component, kappa was computed at the primary category level only. Since the number of observations within each primary category was varied, analysis at the subcategory level would have invalidated the kappa values. The Fleiss' kappa results for each OSUDC component are reported in Table 6.8. According to [106], Cohen suggested the Kappa result could be interpreted as follows: values <= 0 indicating no agreement and 0.01-0.20 as none to slight, 0.21-0.40 as fair, 0.41 – 0.60 as moderate, 0.61-0.80 as substantial, and 0.81 – 1.00 as almost perfect agreement.

As we can see in Table 6.8, the evaluators' agreement for the *defect category* component is the highest, and it is the lowest for the *failure qualifier* component. Possibly, too many values defined for failure qualifier and task difficulty have influenced our results, as addressed in [102]. Since we only measured the agreement of *defect types* at the primary category, which has only three possible nominal values (interface, interaction, and both), it is much easier for the evaluators to understand and learn the *defect types* component rather than the eight and seven nominal values of the *task difficulty* and *failure qualifier*, respectively.

To assess the level of agreement between evaluators classifying defect types at the subcategory level, percent agreement was used instead of kappa statistics. The percent agreement for each usability defect report studied was computed using cell matrix computation [107]. As shown in

[3] http://www.real-statistics.com/reliability/fleiss-kappa/

Table 6.9, the twelve evaluators were labeled with E1 until E12, and the six usability defect subcategories were labeled as G - GUI structure and aesthetics, I – information presentation, A – audibleness, M – manipulation, T – task execution, and F- functionality. The cells in the matrix is set to 1 if the evaluator assesses the usability defect is belongs to that particular defect types; otherwise it is set to 0. Then, the percent agreement for each usability defect report was computed by calculating the percentage agreement for each row and average the rows. In Table 6.9, it can be seen 7 out of 10 usability defect reports received high inter-rater reliability of more than 80%.

Table 6.8. Reliability measures for 12 raters in open source usability problem taxonomy evaluation.

Reliability measure	**Defect category**	**User difficulty**	**Failure qualifier**
Fleiss' kappa	0.240	0.130	0.114

Table 6.9. Percent agreement across twelve evaluators classifying defect types component.

Report	**Defect subcategories**	**Evaluators**												**% Agreement**
		E1	***E2***	***E3***	***E4***	***E5***	***E6***	***E7***	***E8***	***E9***	***E10***	***E11***	***E12***	
1	G	1	1	1	1	1	1	1	1	0	1	1	1	0.92
	I	0	0	0	0	0	0	0	0	1	0	0	0	0.92
	A	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	M	1	0	0	0	0	0	0	0	1	0	0	0	0.83
	T	0	0	0	0	0	0	0	1	0	1	0	0	0.83
	F	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	Inter-rater Reliability													0.92
2	G	1	1	1	0	0	0	0	1	1	0	0	1	0.50
	I	0	0	0	0	1	0	0	0	0	1	1	0	0.75
	A	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	M	0	0	0	1	0	1	0	0	0	1	1	1	0.58
	T	0	0	0	0	0	0	1	0	1	0	0	0	0.83
	F	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	Inter-rater Reliability													0.78
3	G	1	0	1	1	0	0	0	0	0	0	0	1	0.67
	I	0	0	0	0	1	1	1	1	1	1	0	0	0.50
	A	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	M	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	T	0	1	0	0	0	1	1	0	0	0	0	0	0.75
	F	0	0	0	1	0	0	0	0	0	1	1	0	0.75
	Inter-rater Reliability													0.78
4	G	0	0	0	0	0	1	0	0	1	0	0	1	0.75
	I	0	0	0	1	1	0	0	0	0	1	0	0	0.75
	A	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	M	1	1	0	0	1	0	0	1	0	0	1	0	0.58
	T	0	0	1	1	0	0	1	0	1	0	0	0	0.67
	F	0	0	0	0	0	0	0	0	0	1	0	0	0.92
	Inter-rater Reliability													0.78
5	G	1	1	1	1	0	0	1	1	1	1	0	1	0.75
	I	0	0	0	0	1	0	0	0	0	0	1	0	0.83
	A	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	M	0	0	0	0	0	1	0	0	0	0	0	0	0.92
	T	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	F	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	Inter-rater Reliability													0.92

Table 6.9. Percent agreement across twelve evaluators classifying defect types component (Continue)

Report	Defect	Evaluators												%
	subcategories	*E1*	*E2*	*E3*	*E4*	*E5*	*E6*	*E7*	*E8*	*E9*	*E10*	*E11*	*E12*	Agreement
6	G	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	I	0	0	0	0	1	1	0	0	1	1	0	1	0.50
	A	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	M	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	T	1	1	1	1	0	0	1	1	0	0	1	0	0.58
	F	0	0	0	0	0	0	0	0	0	0	0	0	0.92
	Inter-rater Reliability													0.83
7	G	1	1	0	1	1	1	1	0	0	1	1	1	0.75
	I	0	0	0	0	0	0	0	1	1	0	0	0	0.83
	A	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	M	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	T	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	F	0	0	0	0	0	1	0	0	0	0	0	0	0.92
	Inter-rater Reliability													0.92
8	G	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	I	0	0	1	0	1	1	1	0	1	1	0	0	0.50
	A	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	M	0	0	0	0	1	0	0	0	0	0	0	0	0.92
	T	1	1	1	1	0	1	1	1	0	0	0	1	0.67
	F	0	0	0	0	0	0	0	0	0	0	1	0	0.92
	Inter-rater Reliability													0.84
9	G	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	I	0	0	0	1	1	0	0	0	0	1	0	0	0.75
	A	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	M	0	0	0	0	0	0	1	0	0	0	0	0	0.92
	T	1	1	0	1	0	1	0	0	1	1	1	1	0.67
	F	0	0	1	0	0	0	0	1	0	0	0	0	0.67
	Inter-rater Reliability													0.84
10	G	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	I	0	0	0	0	1	0	0	0	0	1	0	0	0.83
	A	0	0	0	0	0	0	0	0	0	0	0	0	1.00
	M	0	1	0	1	0	0	0	1	0	0	1	1	0.58
	T	1	0	1	0	0	0	1	0	1	0	0	0	0.67
	F	0	0	0	0	0	1	0	0	0	1	0	0	0.83
	Inter-rater Reliability													0.82

Notes:

G – GUI structure and aesthetics, I – Information presentation, A – Audibleness, M – Manipulation, T – Task execution, F - Functionality

According to [107], percent agreement less than 61% can be seen as problematic, and many texts recommend 80% agreement as the minimum acceptable inter-rater agreement. Using this standard, we found that most evaluators were in disagreement when classifying usability defects into the *task execution* subcategory. Possibly, the ambiguity around definition and unclear separation of manipulation, task execution, and functionality subcategories caused them to struggle to classify a number of usability defects, as addressed by a few evaluators in post-questionnaires:

> *"I wasn't too sure about the distinction between Task Execution - Action and Functionality but it didn't come up in the evaluation."*

> *"Some of the items appear to overlap; 'Any difficulty encountered by the user when they interact with the application or while performing a task.' - The interaction with the application could be a mouse click. Why do we have both a generic 'difficulty' and a specific action? The definitions could be clarified by exclusions: 'items like.. are listed under ..."*

> *"Some of the terms have to recheck the meaning and the taxonomy can be justified using more than one term."*

Table 6.10. Kappa for the attributes of task difficulty and failure qualifier.

OSUDC component	Category	Kappa
Task difficulty	Complexity	0.030
	Visibility	**0.297**
	Performance	0.109
	Accessibility	0.061
	Loss of data	-0.008
	Understandability	**0.215**
	None	0.003
	Other	-0.017
Failure qualifier	Missing	0.144
	Wrong	-0.016
	Incongruent mental model	0.020
	Irrelevant	0.050
	Overlooked	0.026
	Better way	**0.307**
	None	0.037

As shown in Table 6.10, in ratings for the *task difficulty* and *failure qualifier* component, the agreement among evaluators is slightly poor. As can be seen, there is fair level of agreement for visibility, understandability, and better way category ($\kappa > 0.3$). The non-significant result obtained for this classification is likely due to the nature of open source defect report descriptions that were vague, incomplete, and contained irrelevant information [12]. Some of our evaluators also addressed this concern:

> *"The identification of human emotion was a bit hard and more clarity in this area would be helpful. E.g. how do you identify annoyance from simple text? Not all the statements have 'emotion' words such as 'I find it a distraction' etc"*

> *"Some of the defects are rather 'vague' and it is not easy to fully classify them..."*

Another explanation on the low level of agreement in *task difficulty* component could be due to the limited choices of *task difficulty* values provided in our OSUDC taxonomy. The introduction of these difficulties – *complexity, visibility, performance, loss of data,* and *understandability* is highly influenced by our sampling approach: these difficulties could be varied between different usability defect context and the projects we studied.

With regard to the *failure qualifier* component, the value “better way” was the most common value used by the evaluators. Unsurprisingly, given the nature of usability defects that constitute subjective users satisfactions and opinions, we observed many defect reporters preferred to use words or phrases such as “it’s nice to...”, “I would prefer...”, or “I suggest to...” to express their expectations. Obviously, this kind of phrase is much easier to classify as compared to other qualifiers that could be misinterpreted. For example, consider the snippet of Bug#70513 of Eclipse Platform below. If we read the first sentence, one may consider that the reporter found something went wrong when he closed the parent window. However, the following sentence shows that the action result is not wrong, but his concerns were that some people might find it useful if some other options were presented.

> *“…… Seems like* ***closing the parent window doesn't close the lightweight windows*** ^{wrong} *opened by it. Maybe somebody finds that useful,* ***but you have to put at least some kind of closing option to the lightweight window*** $^{Better\ way}$ *”* [Bug#70513 of Eclipse Platform]

6.5.3 Feedback on Revised OSUDC

This section presents and discusses the results from the post-evaluation questionnaire filled out by the evaluators at the end of the survey. The post-evaluation questionnaire had four questions consisting of one closed question and three open-ended questions.

The closed question was measured on a 5-point Likert scale using the satisfaction-based statements as follows:

- Learnability – the degree to which an evaluator is satisfied that the revised OSUDC is easy to be learned with no training or demonstration;
- Easy to use – the degree to which an evaluator is satisfied that the revised OSUDC is simple, user friendly, and flexible to be used;
- Completeness – the degree to which an evaluator is satisfied that the revised OSUDC contains all required categories and components to be able to classify usability defects;
- Clarity – the degree to which an evaluator is satisfied that the definitions and examples of revised OSUDC are clearly written so that it is easily understandable;

- Practicality – the degree to which an evaluator is satisfied that the revised OSUDC is convenient to use;

The responses are depicted in Figure 6.3. Based on the "Strongly satisfied" and "satisfied" rating, we see that more than 50% of the evaluators were satisfied with the revised OSUDC. One evaluator did not fully complete feedback on the revised OSUDC.

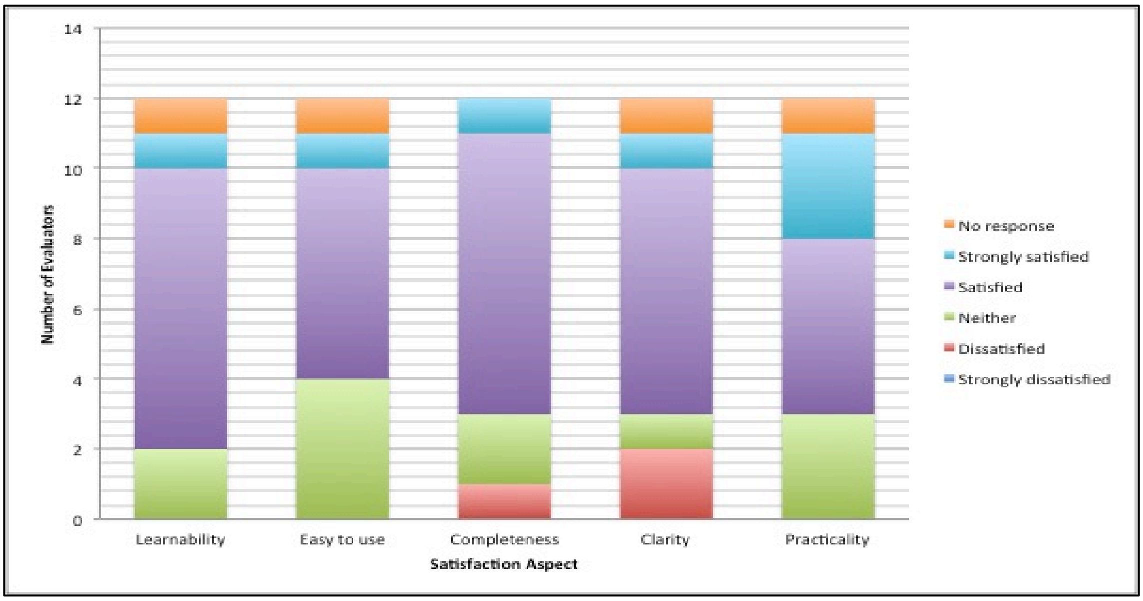


Figure 6.3. Responses on the five satisfaction aspect.

Among the five satisfaction aspects, only one evaluator was dissatisfied with the completeness of OSUDC, and two evaluators were dissatisfied on the clarity of OSUDC. These evaluators expressed their dissatisfaction comments in the accompanying open-ended questions. Unfortunately, these comments were too vague for us to consider for future improvement. For example, the evaluator who was not satisfied with the completeness of OSUDC only stated the following comment:

"It sometimes feels 'incomplete'..."

Concerning the ease-to-use aspect, four evaluators rated neutral in satisfaction. Possibly, the use of Opinio tool as a medium of evaluation can have a negative effect on the evaluator's experience. Switching back and forth between the Opinio tool and the OSUDC reference document can decrease the sense of annoyance because the numerous defect categories are unintuitive, making the understanding and selection of appropriate category can be difficult. If using a self-developed tool, a pop up window could be used to display the definitions and examples when a category is selected. This could possibly increase the flexibility of using the OSUDC. One evaluator addressed this concern as follows:

> *“Rather than opening the guidelines in a different tab, it could list them at the side (and appear all the time) for ease of reference.”*

Overall, the evaluators’ feedback provides a positive indicator on the effectiveness of OSUDC in classifying usability defects and have important implications for further research. While the evaluators conveyed a great understanding on the OSUDC, we are uncertain if different types of users will have the same understanding. For example, this new classification would be useful for helping experienced evaluators analyzing usability defects but may not be helpful for less experienced evaluators, because the information required in this new classification may not be relevant to them. Thus, we may need a comparison study with a larger number of respondents so that we can confirm the effectiveness of OSUDC on all users.

> *“Easy to understand the components and the descriptions.”*
>
> *“It uses normal terms and easy to understand”*
>
> *“It has a wide range of classifications and the definitions of each classification are clear”*
>
> *“Differentiating between interface problem and interaction problem with interface is a good feature”*
>
> *“Seems to be complete. I think it is beneficial to have single taxonomy for that captures usability defect, failure qualifier and user difficulty”*

6.6 Threats to Validity

We have considered two types of possible threats that can affect the validity of our revised OSUDC, which we discuss below.

6.6.1 External Validity

One possible threat to the external validity of this evaluation survey is generalization of the findings. There are three factors to consider. First, the small number of participations in this survey did not represent the varying level of expertise of software development practitioners. We found that six out of twelve evaluators are academic researchers with low familiarity of usability defects. Second, since our research design strategy included recruiting evaluators through researcher's industrial contacts, there could have been evaluators who volunteered to take part in this survey with a specific purpose (e.g personal reasons), which may influence how they responded to the survey. Third, the evaluators have limited usability knowledge, as we found 11 out of 12 evaluators had not received any usability-related training. This might affect their understanding on some of the usability-technical terms used in the OSUDC definitions. However, the sample classification we provided in the reference document may have helped the evaluators to understand the classification process.

Whilst these factors will inevitably result in selection bias, it is therefore difficult for us to argue that the results we obtained can be generalized to the wider population, especially in the context of OSS development.

6.6.2 Internal Validity

In our study we analyzed 377 usability defect reports from Bugzilla for Mozilla Thunderbird, Firefox for Android, and Eclipse Platform projects that were tagged with usability-related keywords. We did not consider defect reports that were not tagged with usability-related keywords although in our observation they were related to usability issues. We expect that our findings also apply to other OSS projects, even this limitation may not be fully representative of other OSS projects. However, the categories identified in OSUDC cover vast spectrum of categories from both usability (adaptation of UPT) and software (adaptation of ODC) engineering domains. We can hence expect that our findings are generalizable and reflective of the OSS projects.

6.7 Discussion

The level of agreement between the twelve evaluators was overall best for the *defect types* component, while agreement on *task difficulty* and *failure qualifier* component is slightly poor. We considered three possible factors that affected the non-significant results obtained for this evaluation process. First, as pointed out by Keenan's [12], poor quality of defect descriptions could potentially affect classification results. From our observation of 377 usability defect reports being studied, most of these reports are composed of simple text. While [20], [29] suggested that a high quality defect report should contain long textual descriptions, our findings show that the median length of usability defect descriptions studied only have 65 words. With regard to *task difficulty*, our results in Section 4.2.2.1 of Chapter 4 indicated that within the 65 words length of description, less than 30% of defect reports explained the impact of the problems on human emotion and task. Therefore, the lack of contextual information in the usability defect descriptions possibly makes it difficult for evaluators to interpret and classify *task difficulty* and *failure qualifier* components.

The second factor that produces insignificant evaluation results was likely due to the absence of training and a demo prior to conducting the evaluation. Previous studies have demonstrated the necessity of initial training to increase user's familiarity and understanding of certain tools, aspects, and concepts [12], [102], [62] We also acknowledge the need for brief training, especially for novice usability evaluators, to help them better identify and rate usability defects. However, training was intentionally not provided in the experimental design of this research. This is because we wanted to know if the material and documentation of the proposed taxonomy is complete and understandable to guide users in classifying usability defects without training. In our future work (discussed in Section 8.4.2), we will ensure that the evaluators receive more training in the use of the OSUDC and we will be more selective in recruiting evaluators that have sufficient knowledge of the software, domain, and usability-related context.

The third factor may be caused by the effect of novice usability evaluators. As described in Section 6.5.1, almost all evaluators in this evaluation were novice usability evaluators. More than 90% of them never received any usability-related training, and less than 25% of them have used ODC, RCA and UAF. The lack of knowledge in usability/ HCI terms and concepts is one of the obstacles for the evaluators to produce more accurate analysis.

The feedback we obtained from the post-questionnaires provides a good insight into the needs of non-technical users when analyzing and understanding usability defects. Especially in OSS projects development, where usability experts are not always available, the classification scheme to be introduced must be simple to cater the needs of open source communities that are not "usability-savvy". To address the abundance of technical words and make a clear OSUDC attribute definition, for example, we could supply some snippets from existing usability defect descriptions. In this way, we could minimize the risk of misunderstanding the OSUDC attributes that lead to incorrect classification. Furthermore, the results of our evaluation reveal potential deficiencies in the current open source defect report content as it relates to usability defects. For the purpose of practical usability defect reporting in conjunction with the proposed OSUDC, we recommend four characteristics for capturing usability defects:

1. State the type of usability problem encountered
2. Justify the impact of the usability defects on user and task, possibly by relating to human emotion and software quality attributes. Perhaps the human emotions could use scale rating so that it could be objectively quantified.
3. State how the problem is identified
4. Use predefined attributes with accompanying open text; so that non-technical reporters can have ideas what information should be included, and further explanation can be supplied in open text input.

Overall, we found that the majority of the evaluators considered our revised OSUDC as appropriate for OSS development. The classification components and categories in the revised OSUDC were also considered generally sufficient. Based on the evaluators' feedback, we refined our revised OSUDC as summarized in Table 6.11. The modified version of the revised OSUDC was reflected in the proposed prototype discussed in Chapter 7.

Table 6.11. Feedback on revised OSUDC and corresponding modifications.

Feedback	**Modification to be carried out**	**Response to Feedback**
Justify the taxonomy using more than one term	No modification	The comment is too general. The evaluator did not point out which component or term needed to be justified using multiple terms. We leave the OSUDC taxonomy as it is until the formal industry evaluation is conducted.
Too many classification terms	No modification	We agree that the revised OSUDC contains a wide range of categories. Under each component there is a list of categories that represent situations that may occur. Such classifications would be useful for separating usability defects to one and only one category that will uniquely describe each defect. The use of simple categories will result in overlapping defects for different situations.
Overlap definitions in for Interaction	Refined definition	One evaluator addressed that one value of the Failure Qualifier component "*'Any difficulty encountered by the user when they interact with the application or while performing a task.' is overlapped.* The phrases "interact with application" addresses a generic interaction context, while "performing a task" is specific action. We revised the definition as it appears in Table 6.1.
Definitions of human emotion should be clarified in the context of defect reports that have simple text	No modification	We agree that human emotion is difficult to be identified from simple text. To address this concern, we plan to conduct a linguistic analysis to study how defect reporters describe their emotions when experiencing usability defects, similar to [39]. The findings can later be used to develop glossary of terms as a reference for defect reporters and software developers.

Table 6.12. Feedback on revised OSUDC and corresponding modifications. (Continue)

Feedback	Modification to be carried out	Response to Feedback
Separate the items 'structure' and 'aesthetics'?	Change category name	Suggested by one evaluator. The word "structure" to represent the interface design layout may cause confusion to people with limited usability or HCI background. We changed the "GUI structure and aesthetics" category to "Visualness" as it reflected in Figure 6.2 and Table 6.1.
Unclear distinction between Task Execution – Action and Functionality	Refined definition	We revised the functionality definitions as shown in Table 6.1.

6.8 Summary

This study presented an OSUDC to classify and analyze usability defects. In an absence of formal usability evaluation in OSS development and limited information available in usability defect descriptions, we revised the existing defect classification schemes to accommodate these limitations. We integrated the traditional UPT from usability engineering practices with the original ODC. In our revised OSUDC, we introduced cause-effect classification model that contains three main classification attributes, namely (1) usability defect categories, (2) failure qualifier, and (3) user difficulty.

The revised OSUDC was evaluated through an online survey. Overall, we obtained good feedback on our revised OSUDC and we refined this based on the feedback. Although we received a small number of evaluators and the majority of them had very limited usability knowledge, their feedback was important for us to understand the needs of those people who are not "usability-technically" knowledgeable to classify usability defects. We found some categories and values of the revised OSUDC were unclear and overlap to our evaluators and so we refined the definitions of those categories to make it more understandable. Nevertheless, vagueness, incompletion, and lack of contextual information in the usability defect description are identified as reasons for evaluators being unable to use OSUDC effectively. We believe the new refined OSUDC can help software developers to better understand usability defects and prioritize them accordingly. For the researchers, the revised OSUDC will be helpful when investigating the trend of usability defect types and understanding the root cause of usability defect problems. However, further deployment and evaluation of the OSUDC by open source users is required to verify this.

The revised OSUDC was used in designing our new usability defect report forms. We explained the utilization of this OSUDC in Chapter 7.

7 Improving Usability Defect Reporting in Open Source Project Development

In common software development practice, software defects are normally reported, tracked, and managed using centralized defect tracking tools. While defect reporting tools are used to collect information from reporters, current defect report forms such as in Bugzilla are too simple to report all types of defects well, particularly usability defects. Moreover, the brief form does not help open source communities that have varying levels of technical knowledge to create informative usability defect reports, thus making it challenging to get an idea on what kind of information should be reported and what kind of information is important to software developers. This generic approach to capturing all information about the defects also limits the empirical research of defect data. Researchers are not able to process the data directly, as they have to extract and partition the information to be able to use it for further research. These limitations motivated us to find an answer for the fifth thesis research question *"RQ5 - How can open source usability defects be most effectively reported?"*

Based on our review of the literature and analysis of different requirements gathered from our survey of practitioners and open source software defect repositories mining, we designed a guided defect report form for reporting usability defects. We tried to maximize the alignment between the information needed by the OSS developers and the information that could be provided by OSS users. We also aimed to assess the quality of usability defect descriptions generated using our proposed defect report form.

7.1 Design

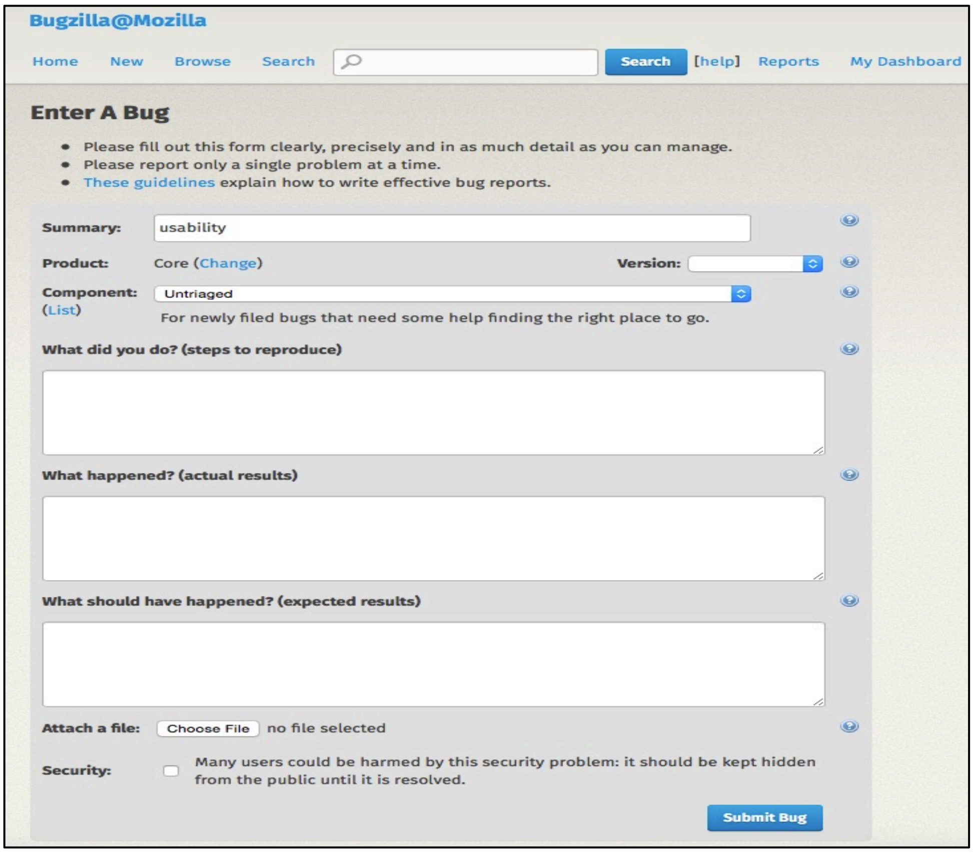
The form was designed to be like existing open source defect reporting form structures. This was to both make the approach more feasible to adopt into practice but also to enable it to be built on top of existing defect reporting tools. Common open source defect repositories, such as Bugzilla typically use unstructured textual forms to report all software defects. As discussed previously, this kind of defect report form is not helpful for different types of users, with varying technical knowledge. Specifically, we are motivated by the knowledge that OSS software developers often receive incomplete and uninformative defect information to fix usability defects [12]. Based on the analysis of the findings from the three different studies discussed in Chapter 3 until Chapter 6, we argue it is essential to anticipate usability and software engineering principles to optimize usability defect reporting processes.

We used the Bugzilla defect report form layout as a point of reference, a well-known open source defect reporting tool. As far as possible, we tried to maintain our proposed usability defect forms as similar as the original Bugzilla defect form layout to maximize users' familiarity while minimizing their confusion. Our strategy for designing the usability defect report forms consisted of the following four criteria, based on the findings of the three studies that we have conducted earlier. Table 7.1 summarizes the main findings that have been extracted as an input to the proposed defect report form.

Table 7.1. Summary of the three studies translated into desirable features of our open source usability defect reports.

Studies	**Main findings**
Systematic Literature Review	Problem *description, severity*, *context,* and *redesign description* were the four attributes most commonly used to describe usability defects
Online survey	• The desirable usability defect report form based on survey responses – simple, reasonable predefined values, and introduce usability keywords, options and descriptors • Reporters often provide *actual output, expected output,* and *steps to reproduce* when describing usability defects • Reporters ranked *assumed cause* as the most difficult information to provide. But this information was considered to be the most helpful information by software developers • The top five most important attributes used by software developers are *assumed cause, screenshots, steps to reproduce, excepted output*, and *software context* • Among the problems experienced, *unclear assumed cause* and *insufficient information in steps to reproduce* was the most commonly encountered. Other common problems include *unclear software context and screenshots*
Software defect repositories mining	*Supplementary information* could improve defect resolution time

Mutually exclusive defect description – the attributes only capture one value. As shown in Figure 7.1 and Figure 7.2, the existing Bugzilla defect report form used in the Mozilla and Eclipse projects is designed as unstructured textual forms. While Bugzilla supports custom fields to capture defect data that is unique to each project and organization, it still uses a plain textual defect description to collect information to reproduce the defect. The vague definition of this general *description* field in Eclipse, for example, is likely to cause the reporter to report mixed information. This is because the reporter may not have a sufficient appreciation of what is supposed to be written in the *description* field. They will write whatever comes to mind and in many cases the information reported is not important or useful to the software developer to use in order to correct the defects.



Bugzilla@Mozilla

Enter A Bug

- Please fill out this form clearly, precisely and in as much detail as you can manage.
- Please report only a single problem at a time.
- These guidelines explain how to write effective bug reports.

Summary: usability

Product: Core (Change) Version:

Component: (List) Untriaged

For newly filed bugs that need some help finding the right place to go.

What did you do? (steps to reproduce)

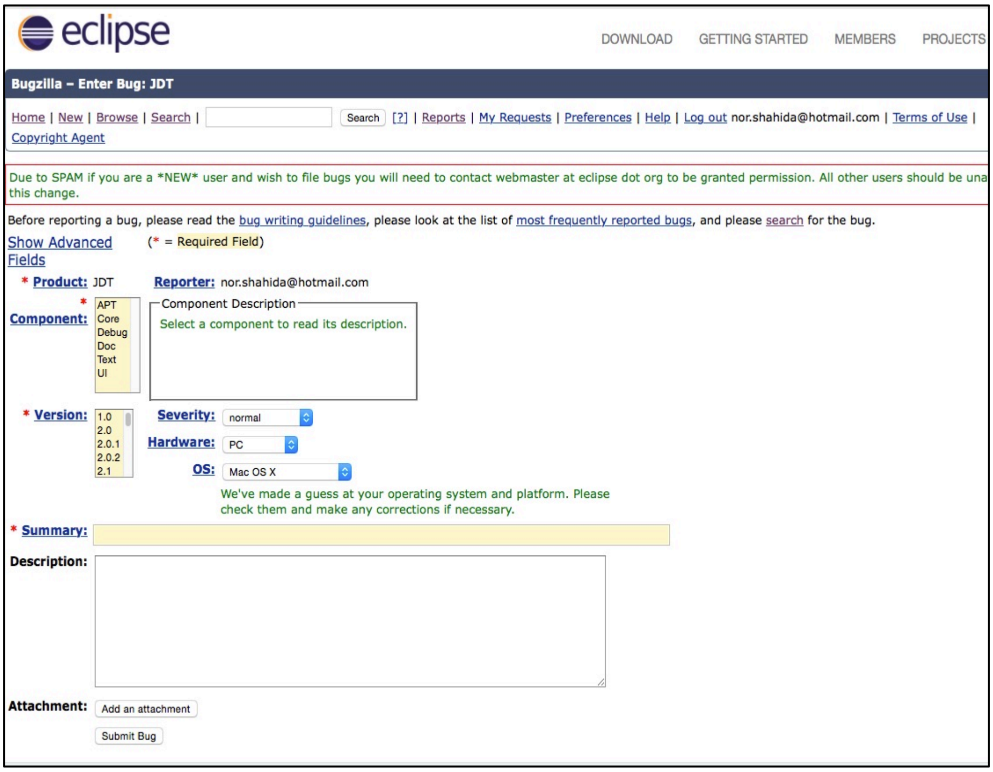
What happened? (actual results)

What should have happened? (expected results)

Attach a file: Choose File no file selected

Security: □ Many users could be harmed by this security problem: it should be kept hidden from the public until it is resolved.

Figure 7.1. Bugzilla defect report form of the Mozilla project.



eclipse
DOWNLOAD
GETTING STARTED
MEMBERS
PROJECTS

Bugzilla – Enter Bug: JDT

Home | New | Browse | Search | Search [?] | Reports | My Requests | Preferences | Help | Log out nor.shahida@hotmail.com | Terms of Use | Copyright Agent

Due to SPAM if you are a *NEW* user and wish to file bugs you will need to contact webmaster at eclipse dot org to be granted permission. All other users should be una this change.

Before reporting a bug, please read the bug writing guidelines, please look at the list of most frequently reported bugs, and please search for the bug.

Show Advanced Fields (* = Required Field)

* Product: JDT Reporter: nor.shahida@hotmail.com

* Component: APT
Core
Debug
Doc
Text
UI

Component Description
Select a component to read its description.

* Version: 1.0
2.0
2.0.1
2.0.2
2.1

Severity: normal

Hardware: PC

OS: Mac OS X

We've made a guess at your operating system and platform. Please check them and make any corrections if necessary.

* Summary:

Description:

Attachment: Add an attachment

Figure 7.2. Bugzilla defect report form of the Eclipse project.

To this end, including reusing existing defect attributes – *software information* (e.g., product, version, component), *steps to reproduce, actual results*, and *expected results,* we added new specific attributes to capture usability defect information: *solution proposal, impact, failure qualifier*, *assumed cause*, and *usability defect category* into the new usability defect report forms. For example, by introducing dedicated attributes to address problem *impact*, we can help software developers to understand and appreciate how users struggle with the usability issues.

Contextualized attributes with multiple instances – a defect attribute is explained with multiple instances. Single attributes in the existing Bugzilla defect report form are considered inadequate for reporting usability defects. For example, while defect report forms in the Mozilla project were explicitly defined as three separate attributes to capture steps to reproduce, actual results, and expected results, often the information that really needed to be reported was not explained clearly or even not reported at all. In fact, some reporters may think that their reports are well explained, but they may provide irrelevant or inadequate information.

For this reason, we propose several usability defect attribute instances to assist reporters in supplying the most useful information in their usability defect reports. Even breaking up an attribute into multiple instances does not always provide a better user experience and produce high quality defect information, but this kind of form design may provide specific hints and examples about the information the user should enter. These attribute instances depend on the context of information to be described. For example, to explain the *impact* of the problem, we introduced three instances – *How does this problem affect your task? Explain your challenges*, and *How annoying is this problem to you?* Table 7.2 lists usability defect attributes and its corresponding instances.

Guided wizard defect report form – this provides defect attributes relevant to the reporter's information needs. From a technical user's standpoint, plain forms are sufficient to report software defects as they know what to write and which information is necessary to report, but not for less technical users. For an optimum usability defect reporting process, we considered a guided wizard form solution to guide novice reporters or less technical users through the reporting process, to hint at what is expected from them at each step, and to present relevant options. Since we introduced more usability attribute instances to collect important information, a plain form is not the best way to collect this kind of data. We designed our usability defect report form as follows:

Table 7.2. List of attributes used in new usability defect forms.

Attribute	**Attribute instances**	**Input types**
General descriptive information	Summary / title	Free form text
	Usability defect type	Predefined data
	Problem summary	Free form text
	Steps to reproduce	Free form text
Actual results	Observation results	Free form text
	Support evidence	Attachment
Impact	User difficulty type	Predefined data
	User difficulty summary	Free form text
	Annoyance level	Predefined data
	Reproducibility	Predefined data
Failure qualifier	Failure qualifier	Predefined data
Expected results	User expectation	Free form text
	Support evidence	Attachment
Solution proposal	Solution proposal	Free form text
	Solution attachment	Attachment
Software context	Product	Predefined data
	Component	Predefined data
	Version	Predefined data
	Operating system	Auto-generated data
	Platform	Auto-generated data

- We broke up the attributes into smaller sections presented one at a time. We used form tabs with a random navigation approach to make it easy for reporters to navigate between data-entry fields. As recommended in interaction design best practices, this approach makes providing input much more manageable, even though it is the same amount of information. This is a significant consideration to reduce the seeming complexity of reports by hiding long lists of attributes. However, there are some attributes that are set as mandatory fields (i.e., *product, version, component*) – these attributes must be filled out before navigating or submitting the form.
- Typically, early data entered influences later data. We used hide/show logic at the attribute and page level, so that pages and attributes themselves may appear or disappear based on the choices the reporter makes. For example, choosing between interface and interaction defect will change defect category options. For these situations, wizard forms create the right pace by preventing unnecessary information and ensuring that the reporter is presented with a guided experience, rather than confusing attributes.
- We set predefined values for certain attributes to keep the content clear and concise. In this way, we can avoid long defect descriptions with uninformative information.

- Provide specific hints about the information the reporter must enter. We used a question-based approach so that users have a better idea on what should be written in the textual form. For example, in explaining the problem, we listed a few questions such as *"what you saw when you manipulated the object?", "how well the user tasks are mapped to the system?"*, or *"how well the system assists task completion?".*
- Defect attributes are different between software developers and users. Due to the limited technical knowledge of many reporters, we used different attributes for understanding the validity of the problem. While software developers were interested in knowing the technical causes of the problem, we believe it is impossible for less technical users to supply such information. Therefore, we used the *failure qualifier* attribute to know how users experienced a usability problem, and *assumed cause* attribute for software developers to point out programming code error or technical explanation.

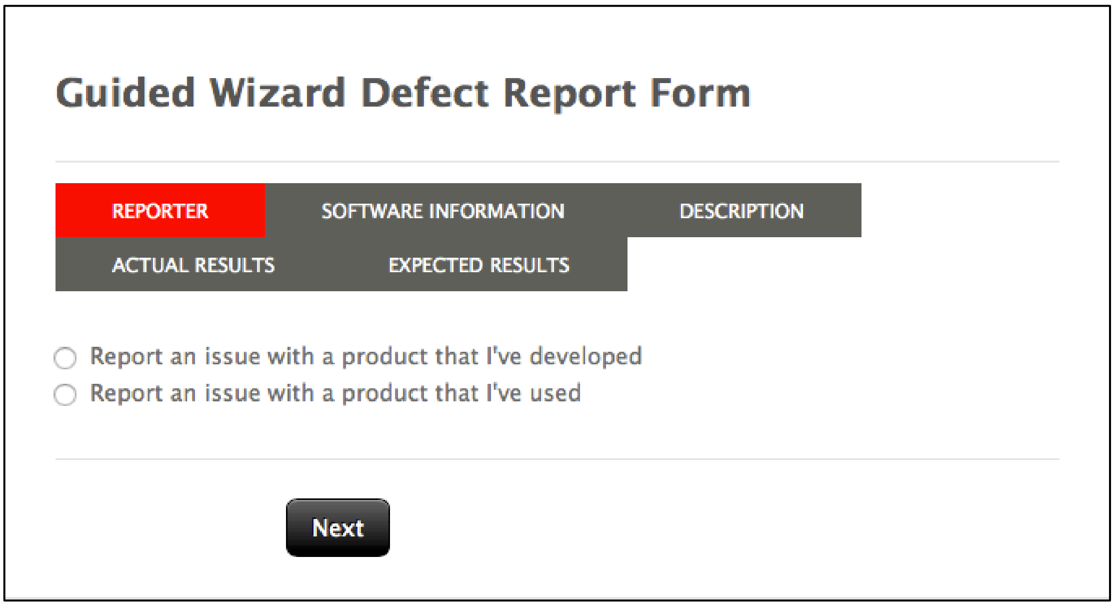
Objective assessment of user difficulty – to measure the ability of impaired tasks (users' feelings and difficulty). An inspection of existing Bugzilla defect reports revealed its limitation for eliciting information about user difficulty, seeming to miss a coherent expression of users' feelings and struggling to accomplish certain tasks. Moreover, the subjective nature of usability defects made some people think the issue was invalid. Reeder and Maxion [104] defined the user difficulty effect as 'when the ability to achieved a goal is impaired'. We designed two user difficulty dimensions for instances when users are experiencing usability problems according to [104]. The subjectivity in determining user difficulty was measured by using scale rating and predefined value.

- *Dimension 1 – Human emotion*. This dimension helps to assess the internal cognitive state such as confusion, frustration, annoyance, and uncertainty. We used a five rating scale to rate this dimension towards the usability problems, which are not addressed by existing defect report forms.
- *Dimension 2 – Task difficulty*. This dimension determines the likely consequence of usability defects that caused each instance of difficulty. We used software quality metric attributes to denote the task difficulty options– complexity, understandability, visibility, performance, and accessibility. These attributes were extracted based on the most difficulties apparent in the open source usability defect reports (discussed in Chapter 4). However, reporters have the use of an *"other"* option if their difficulties do not match any of the options listed.

7.1.1 Usability Form Implementation

The prototype of guided usability defect report forms was designed using Jotform[4], an online application to create custom online forms using intuitive drag-and-drop user interface. We used form tab feature to group a set of usability defect attribute instances. The detail descriptions of the tab and associated attributes are described below.

Reporter Identification: Similar to the Bugzilla defect report form, upon submission of a new defect report a reporter has to select their role. The selection of this role depends on whether the defect reporter finds the defects while using an OSS product, or when they contribute to OSS development. The selection of this role will determine the relevant defect attributes that will be prompted in the next display.


Guided Wizard Defect Report Form

○ Report an issue with a product that I've developed
○ Report an issue with a product that I've used

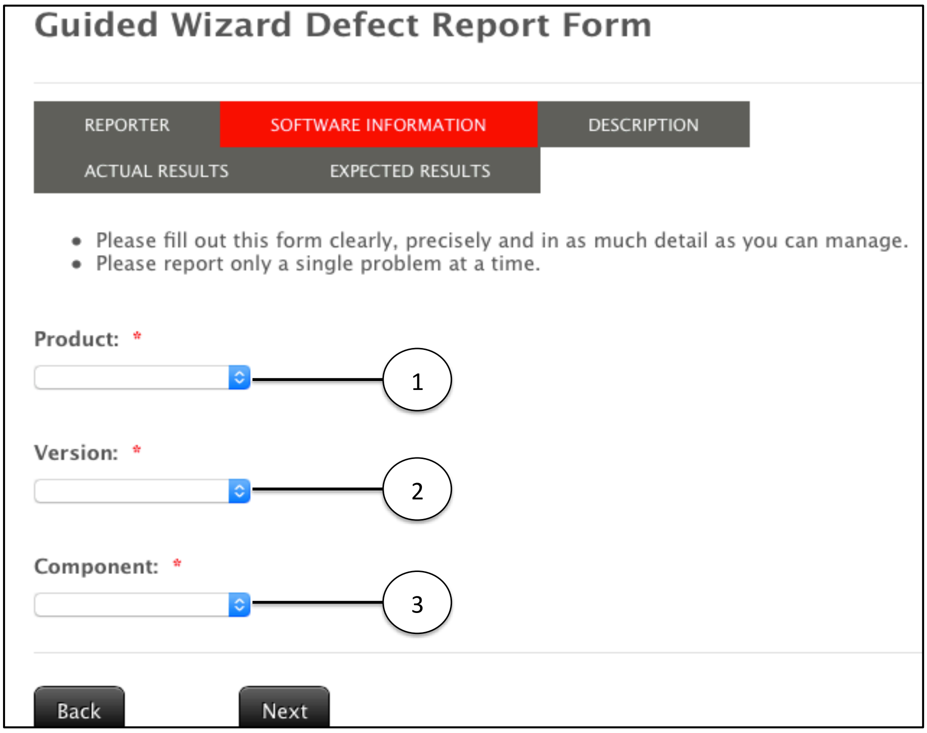
Figure 7.3. Reporter identification.

Software Information: The requested information about the open source software used will vary between software developer and user. In Figure 7.4, when a software developer reports a usability defect, the information about *product, version*, and *component* is compulsory to fill in. On the contrary, for a user, such *component* information is not compulsory (see Figure 7.5). The compulsory information is denoted by a red asterisk (*), and the defect reporter cannot go to the next page if this compulsory information is left blank.

1. Product – The name of the OSS application.

[4] https://www.jotform.com/form-templates/

2. Version – The version of OSS application in which user discovered the usability problems. If user can reproduce the problem in more than one version, select the earliest.
3. Component – The sub-part of the software in which the problem exists.



Guided Wizard Defect Report Form

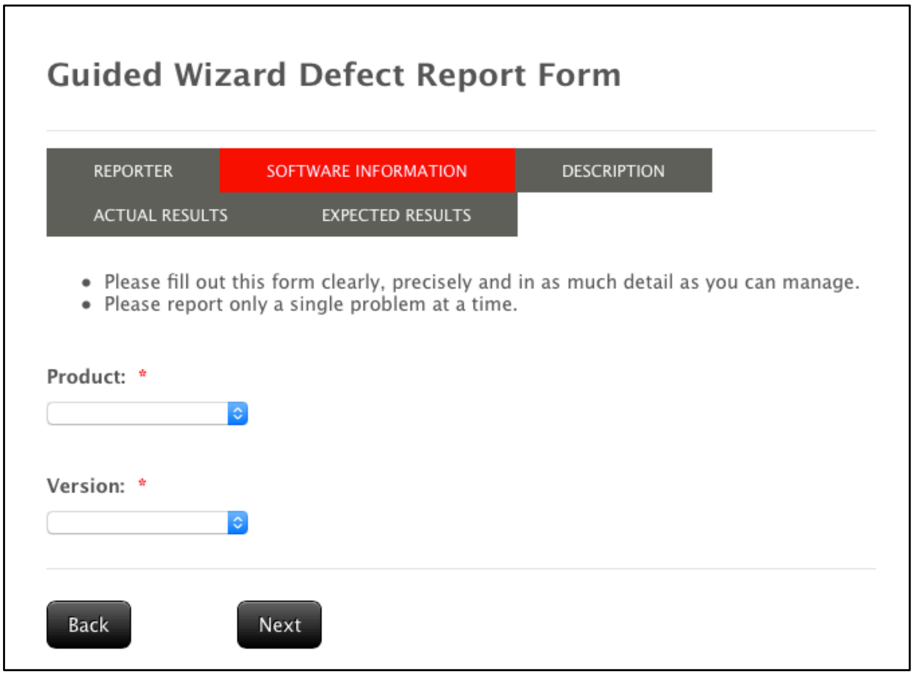
- Please fill out this form clearly, precisely and in as much detail as you can manage.
- Please report only a single problem at a time.

Product: *

Version: *

Component: *

Figure 7.4. Software information tab for software developer.



Guided Wizard Defect Report Form

- Please fill out this form clearly, precisely and in as much detail as you can manage.
- Please report only a single problem at a time.

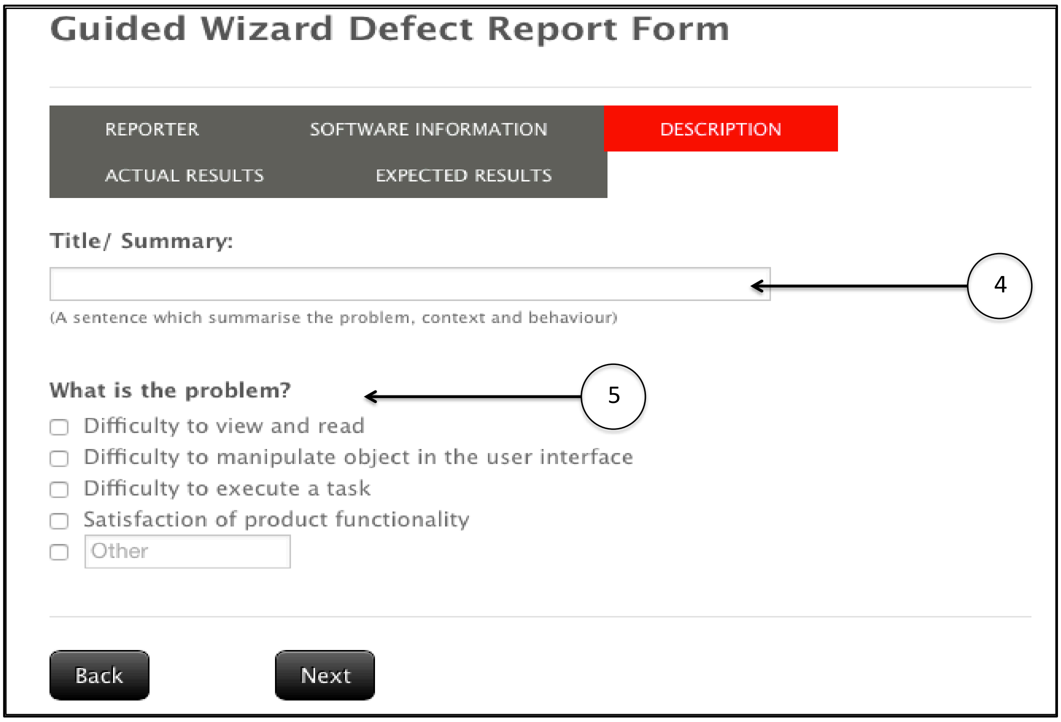
Product: *

Version: *

Figure 7.5. Software information tab for user.

Description: The reporter must describe the details of the problem to be reported. Rather than an unstructured text form as in traditional reporting tools, the information is captured in multiple attribute instances as listed below:

4. Title – A headline summarizing the usability issues. As suggested in [39], the defect report title should consists of descriptions about (1) a software entity or an entity behavior, (2) a relevant quality attribute, (3) the problem, (4) the execution context, and (5) whether the report is a defect or feature request.
5. Usability defect main category – There are four options for specifying the usability defect types based on the revised OSUDC taxonomy. The selection of the main usability defect category will determine the associated usability defect subcategories that will be prompted in (6).
6. Usability defect subcategories – The value of defect subcategories are dependent on the selection of the main usability defect category in (5). Upon selection of usability defect subcategories, examples of defects associated with the selected category will be listed. If the experienced problem is not in the list, the defect reporter may choose "Other" option and describe in detail in the following text input (7).
7. Explanation of the problem – A detailed explanation of the problem.
8. Steps to reproduce – Step by step instructions to allow the usability defect to be reproduced on another machine. It is recommended that the instructions be explicitly given as a numbered sequence of instructions.
9. Failure qualifier –There are six failure qualifier values to understand how usability defect reporters experienced the usability problems. Only one value can be selected at one time.
10. Pop up callouts - A hint on what kind of information should be supplied on the textual text input. The callouts will pop out when a cursor is moved to that particular text box.



Guided Wizard Defect Report Form

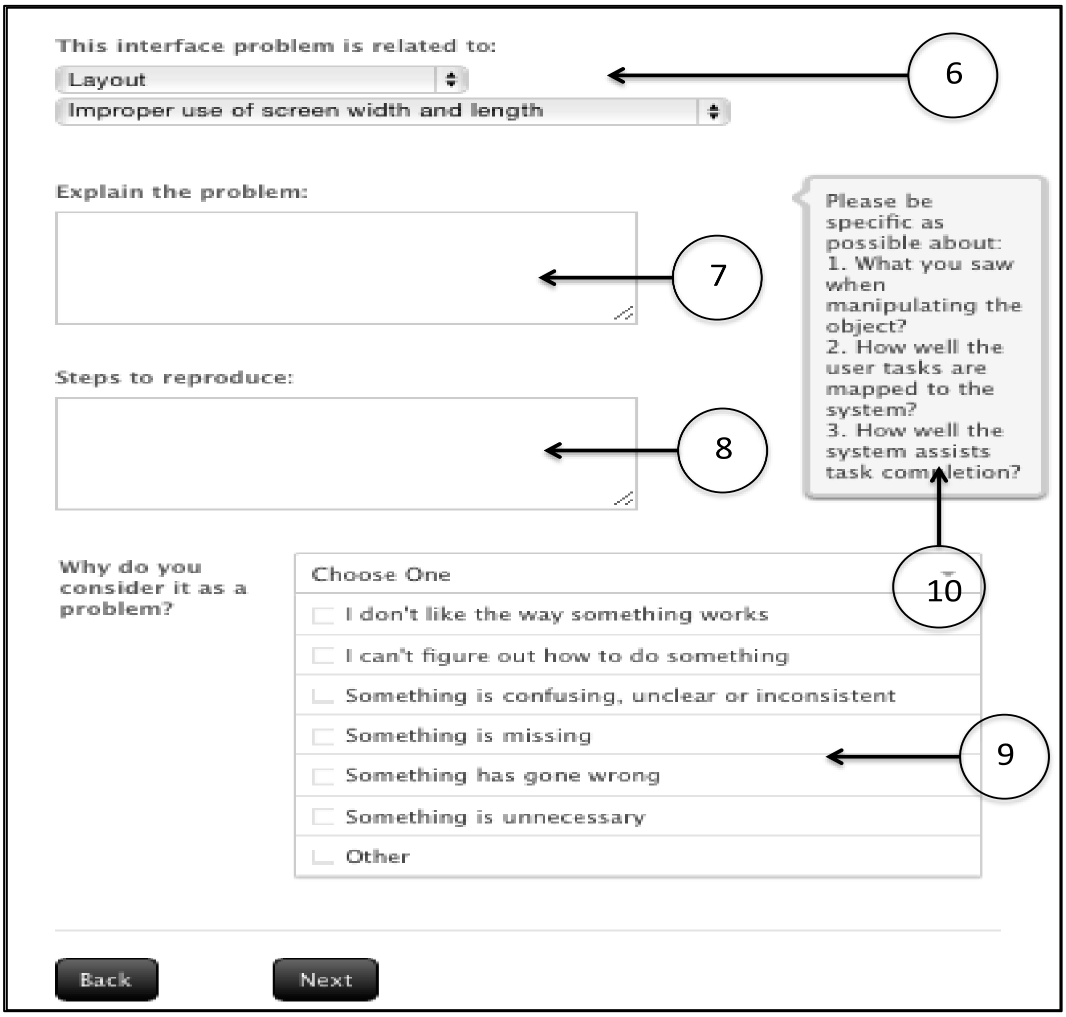
Title/ Summary:

(A sentence which summarise the problem, context and behaviour)

What is the problem?

- □ Difficulty to view and read
- □ Difficulty to manipulate object in the user interface
- □ Difficulty to execute a task
- □ Satisfaction of product functionality
- □ Other

Figure 7.6. Description tab.



This interface problem is related to:

Layout

Improper use of screen width and length

Explain the problem:

Steps to reproduce:

Please be specific as possible about:
1. What you saw when manipulating the object?
2. How well the user tasks are mapped to the system?
3. How well the system assists task completion?

Why do you consider it as a problem?

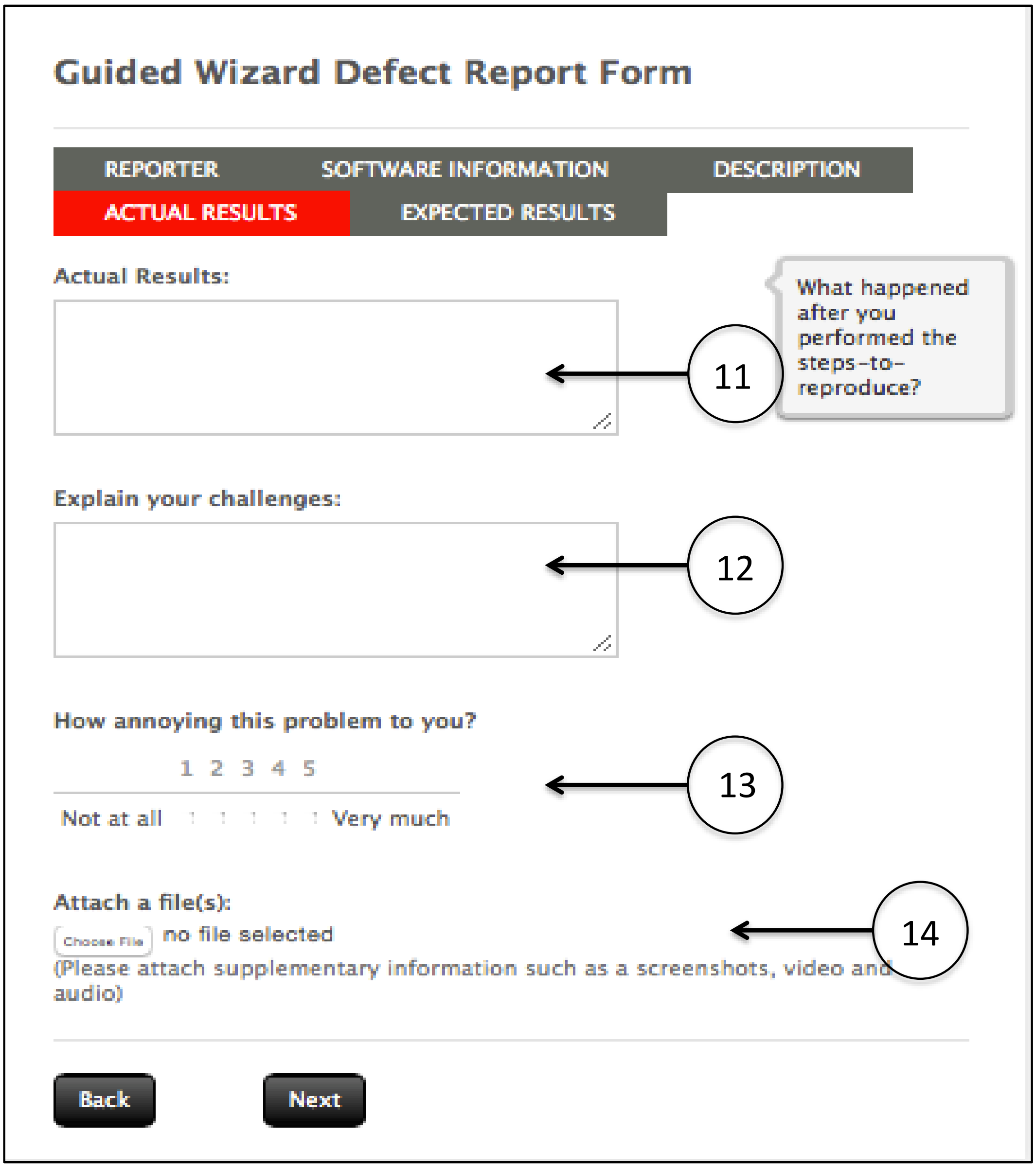
Choose One

- □ I don't like the way something works
- □ I can't figure out how to do something
- □ Something is confusing, unclear or inconsistent
- □ Something is missing
- □ Something has gone wrong
- □ Something is unnecessary
- □ Other

Figure 7.6. Description tab (cont).

Actual Results: Describe what currently happens when the usability defect is present.

11. Actual results – Describe what was wrong, why is it wrong, or, if an error was thrown, what was the error.
12. Task difficulty – The anticipated difficulties/ challenges the user encountered as a consequence of the problem. Additional information about how you did a workaround for the usability defect to continue using the software could be explained here.
13. Mood indicator – A scale to rate the user's annoyance when the problem happens.
14. File attachment – Supplementary material such as image, audio, video, stack traces, or crash log that can help others to reproduce the problem.



Guided Wizard Defect Report Form

Actual Results:

What happened after you performed the steps-to-reproduce?

Explain your challenges:

How annoying this problem to you?

1 2 3 4 5

Not at all ○ ○ ○ ○ ○ Very much

Attach a file(s):

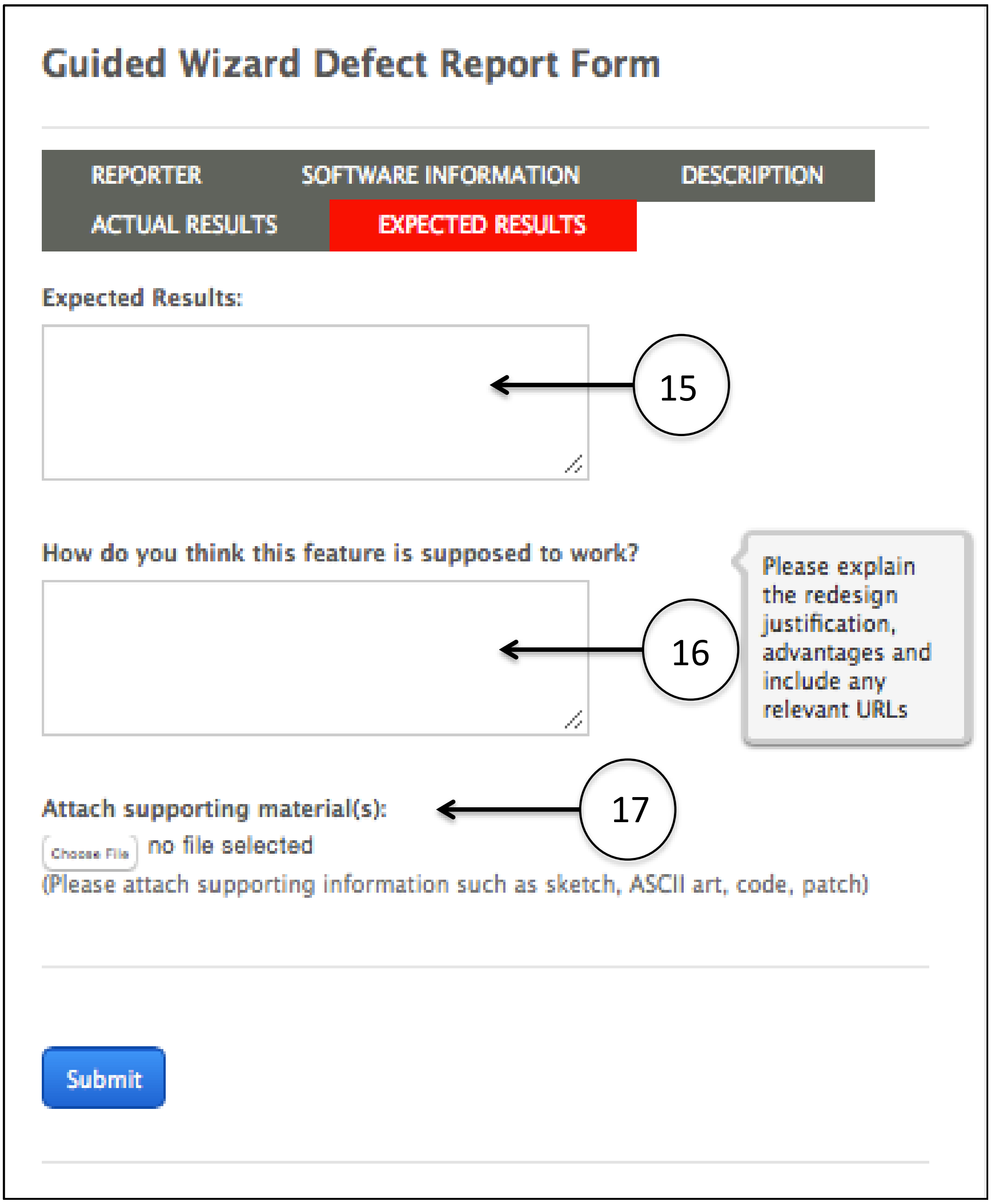
Choose File no file selected

(Please attach supplementary information such as a screenshots, video and audio)

Figure 7.7. Actual results tab.

Expected Results: Describe what should happen if the defect was fixed.

15. Expected results - What behavior you expected when the problem occurred, or what would you like to change the way the software works or to improve some other aspects.
16. Solution proposal – A description of how to remedy the problem and justification of the redesign proposal.
17. File attachment – Supplementary material to support the idea of proposal such as photoshop sketches, ASCII art, screenshots, or code patch.



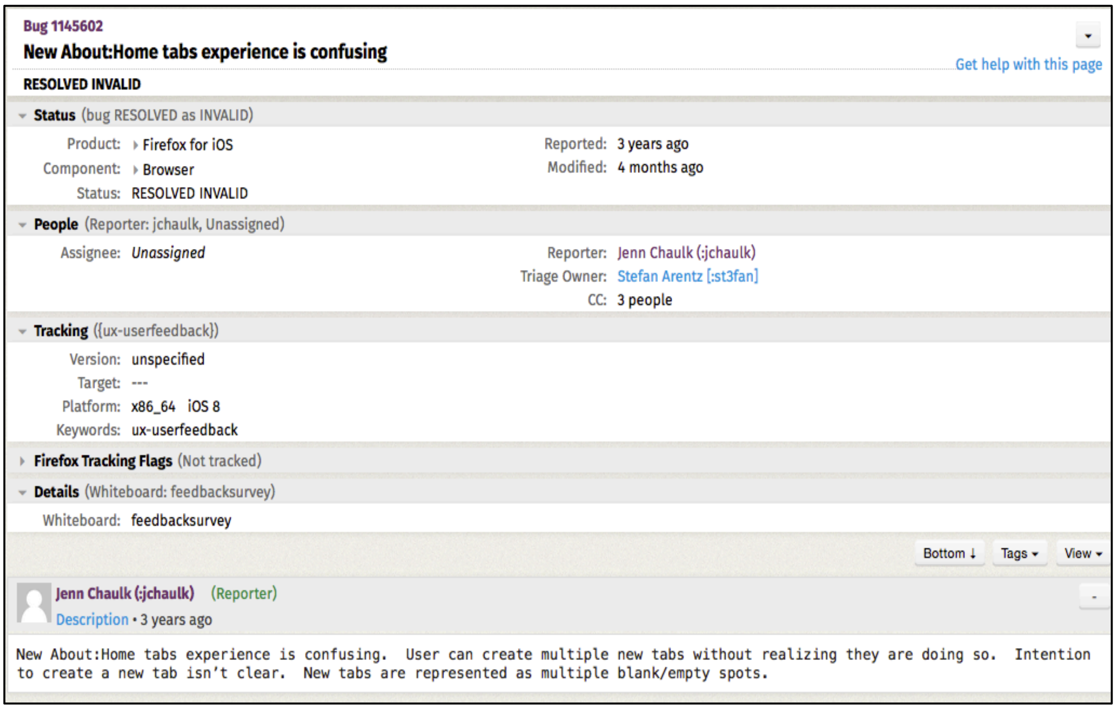
Please explain the redesign justification, advantages and include any relevant URLs

(Please attach supporting information such as sketch, ASCII art, code, patch)

Figure 7.8. Expected results tab.

7.1.2 Example of Usage

We demonstrate our usability defect report form prototype, which captures usability issues related to visualness. We reproduced the Firefox for iOS#1145602 issue in Figure 7.9 on our iPhone and wrote the detailed defect descriptions based on our reproduction steps.



Bug 1145602

New About:Home tabs experience is confusing

RESOLVED INVALID

Status (bug RESOLVED as INVALID)

Product: Firefox for iOS
Component: Browser
Status: RESOLVED INVALID
Reported: 3 years ago
Modified: 4 months ago

People (Reporter: jchaulk, Unassigned)

Assignee: Unassigned
Reporter: Jenn Chaulk (:jchaulk)
Triage Owner: Stefan Arentz [:st3fan]
CC: 3 people

Tracking ([ux-userfeedback])

Version: unspecified
Target: ---
Platform: x86_64 iOS 8
Keywords: ux-userfeedback

Firefox Tracking Flags (Not tracked)

Details (Whiteboard: feedbacksurvey)

Whiteboard: feedbacksurvey

Jenn Chaulk (:jchaulk) (Reporter)
Description • 3 years ago

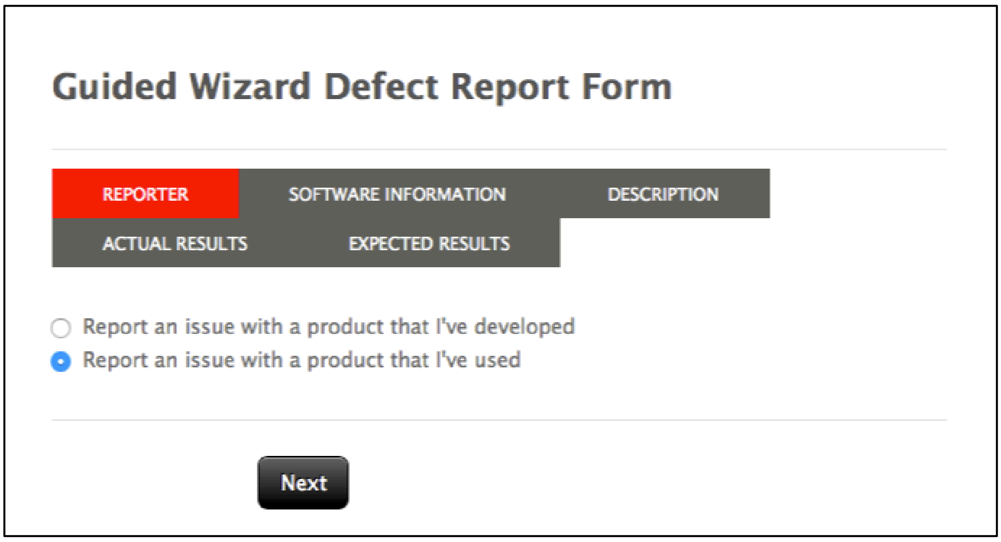
```
New About:Home tabs experience is confusing.  User can create multiple new tabs without realizing they are doing so.  Intention
to create a new tab isn't clear.  New tabs are represented as multiple blank/empty spots.
```

Figure 7.9. Firefox for iOS Report#1145602

This issue was found by OSS contributor, named jchaulk. Jchauk was not satisfied with the About:Home tabs in Firefox browser apps, in which he could not directly find the newly opened tabs. Since jchaulk did not contribute to the Firefox for iOS project, upon submission of a new usability issue, he selected the second option (see Figure 7.10). Then he filled up the product and version information in Software Information tab (see Figure 7.11). In the following Description tab, he wrote the report title and selected a relevant usability issue he had experienced (see Figure 7.12). In this case, the usability issue encountered by jchaulk is related to visualness – the difficulty to view newly opened tab on the user interface, in particular, the visibility behavior of the tabs was problematic. To support this issue, he explained his situation when he experienced the problem and supplied a step by step explanation of how to reproduce the problems. He also indicated that he raised this issue was because he felt that the new implemented tab feature was confusing (see Figure 7.12).

In the Actual Results tab, he highlighted the software context in which the problem occurred, and explained the challenges and difficulties encountered by the problem. He also provided workaround solutions to overcome the problem. Then he indicated the annoyance level of the problem and attached the screenshots to support his explanations (see Figure 7.13).

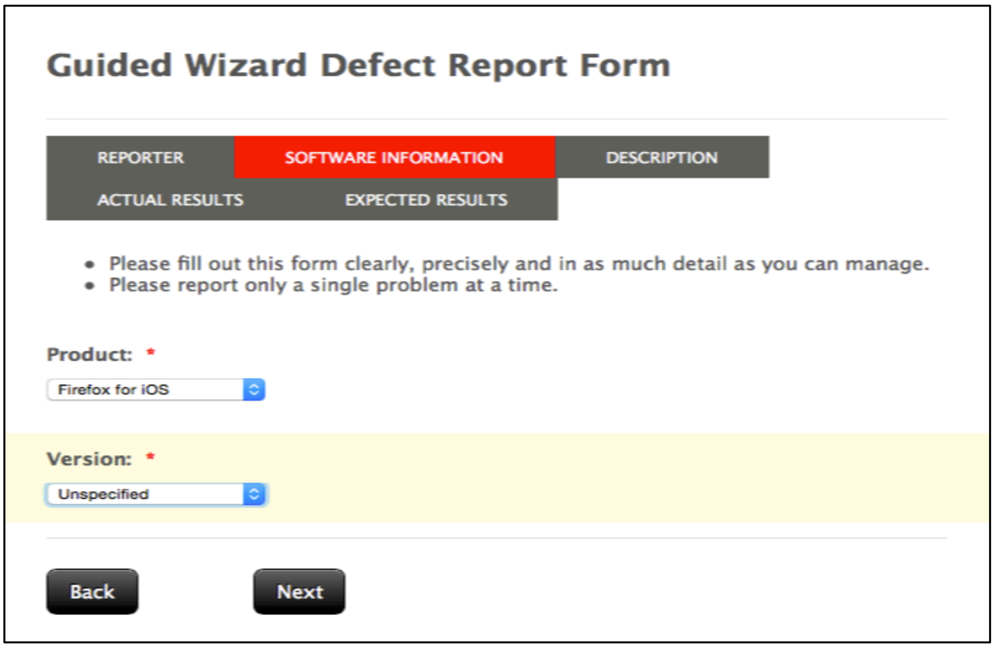
Lastly, in the Expected Result tab, jchaulk expressed his expectations on how the About:Home tab should behave instead. He also provided suggestions to improve the visibility of the newly open tabs (see Figure 7.14).



Guided Wizard Defect Report Form

○ Report an issue with a product that I've developed
● Report an issue with a product that I've used

Figure 7.10. Identification of reporter background to identify necessary information to be prompted.



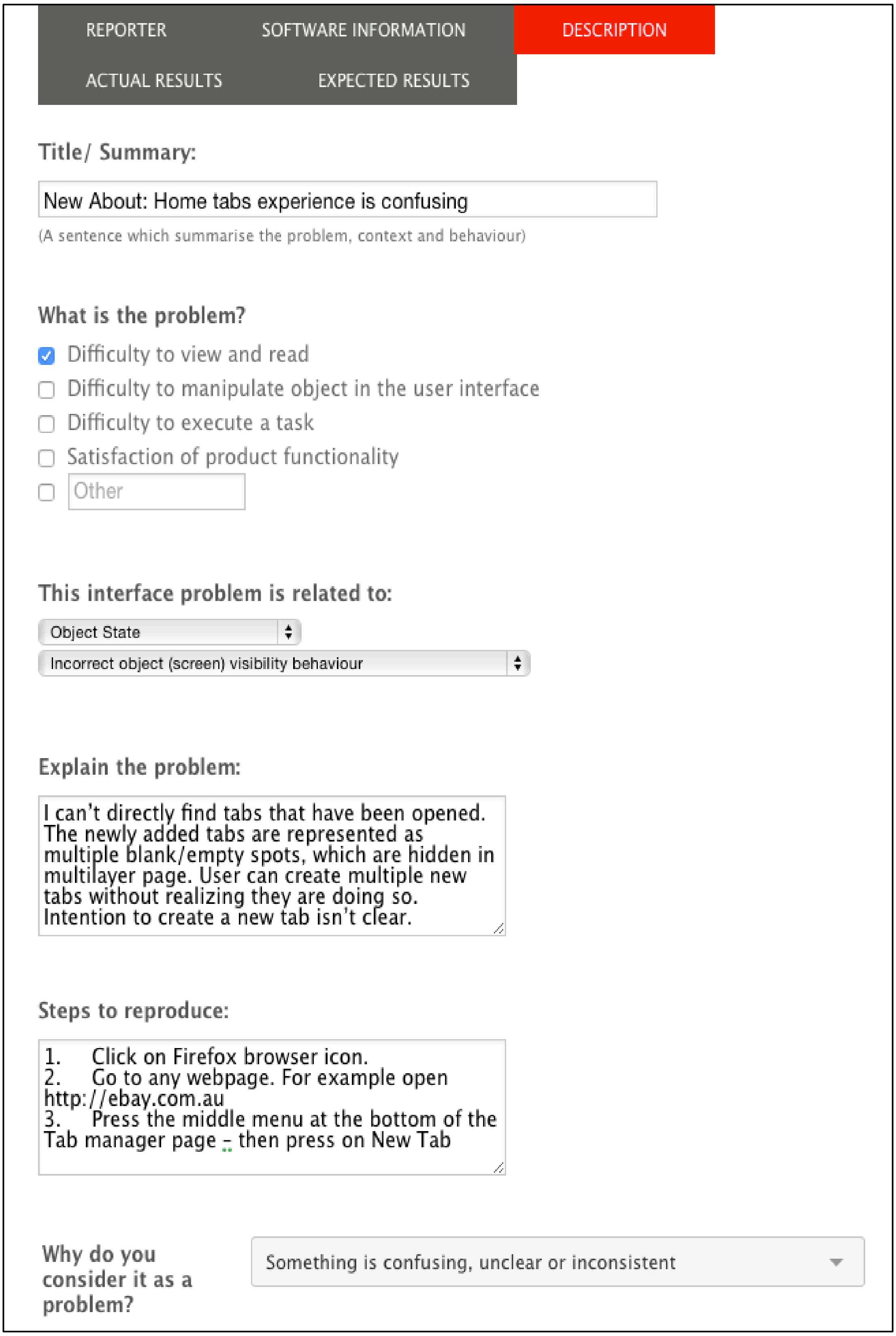
Guided Wizard Defect Report Form

- Please fill out this form clearly, precisely and in as much detail as you can manage.
- Please report only a single problem at a time.

Product: *
Firefox for iOS

Version: *
Unspecified

Figure 7.11. Software Information tab to collect details of the problematic application.



Title/ Summary:

New About: Home tabs experience is confusing

(A sentence which summarise the problem, context and behaviour)

What is the problem?

☑ Difficulty to view and read
□ Difficulty to manipulate object in the user interface
□ Difficulty to execute a task
□ Satisfaction of product functionality
□ Other

This interface problem is related to:

Object State
Incorrect object (screen) visibility behaviour

Explain the problem:

I can't directly find tabs that have been opened. The newly added tabs are represented as multiple blank/empty spots, which are hidden in multilayer page. User can create multiple new tabs without realizing they are doing so. Intention to create a new tab isn't clear.

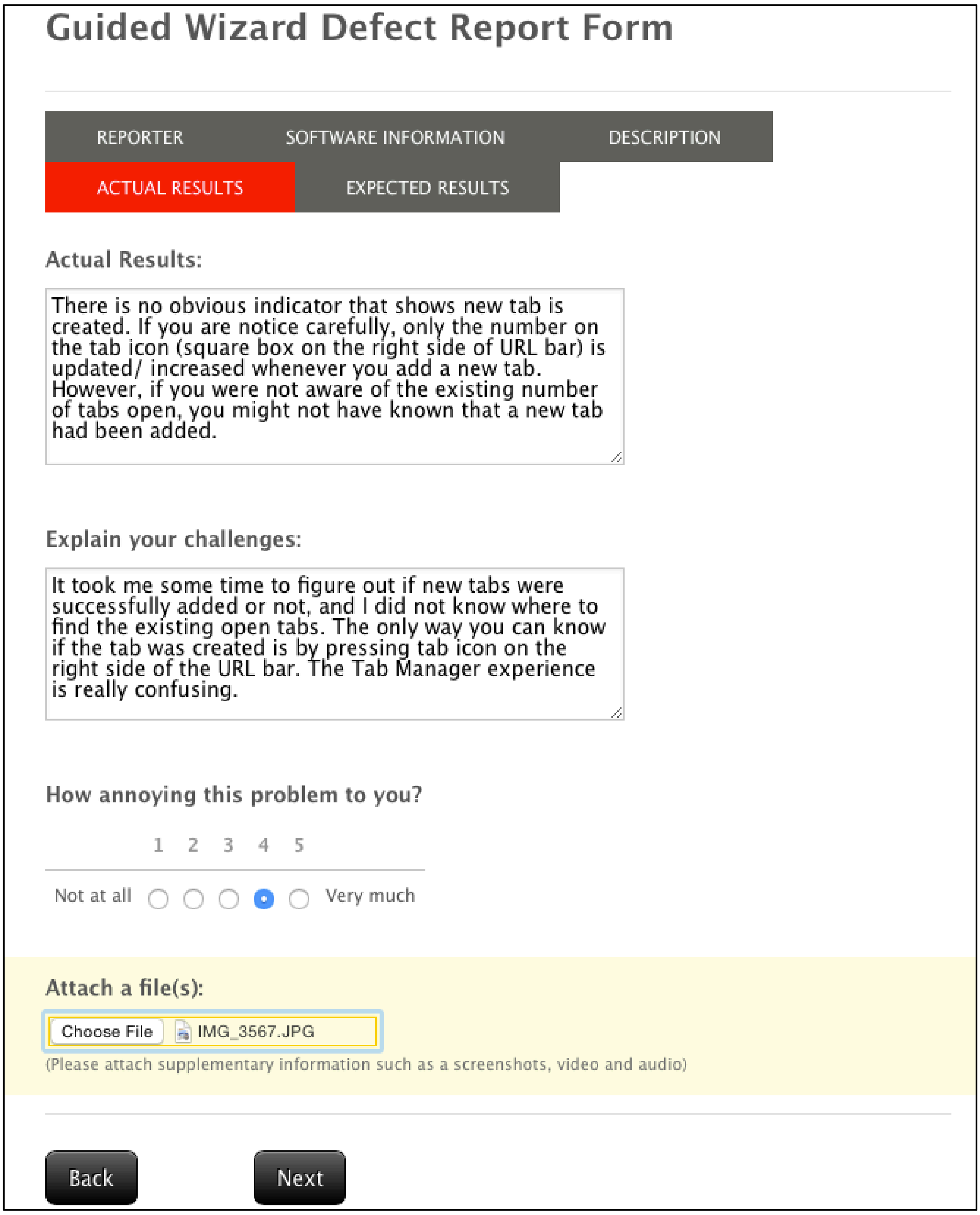
Steps to reproduce:

1. Click on Firefox browser icon.
2. Go to any webpage. For example open http://ebay.com.au
3. Press the middle menu at the bottom of the Tab manager page - then press on New Tab

Why do you consider it as a problem?

Something is confusing, unclear or inconsistent

Figure 7.12. Description tab to collect types of usability defects, detailed explanations, steps to reproduce, and failure qualifier.



Guided Wizard Defect Report Form

Actual Results:

There is no obvious indicator that shows new tab is created. If you are notice carefully, only the number on the tab icon (square box on the right side of URL bar) is updated/ increased whenever you add a new tab. However, if you were not aware of the existing number of tabs open, you might not have known that a new tab had been added.

Explain your challenges:

It took me some time to figure out if new tabs were successfully added or not, and I did not know where to find the existing open tabs. The only way you can know if the tab was created is by pressing tab icon on the right side of the URL bar. The Tab Manager experience is really confusing.

How annoying this problem to you?

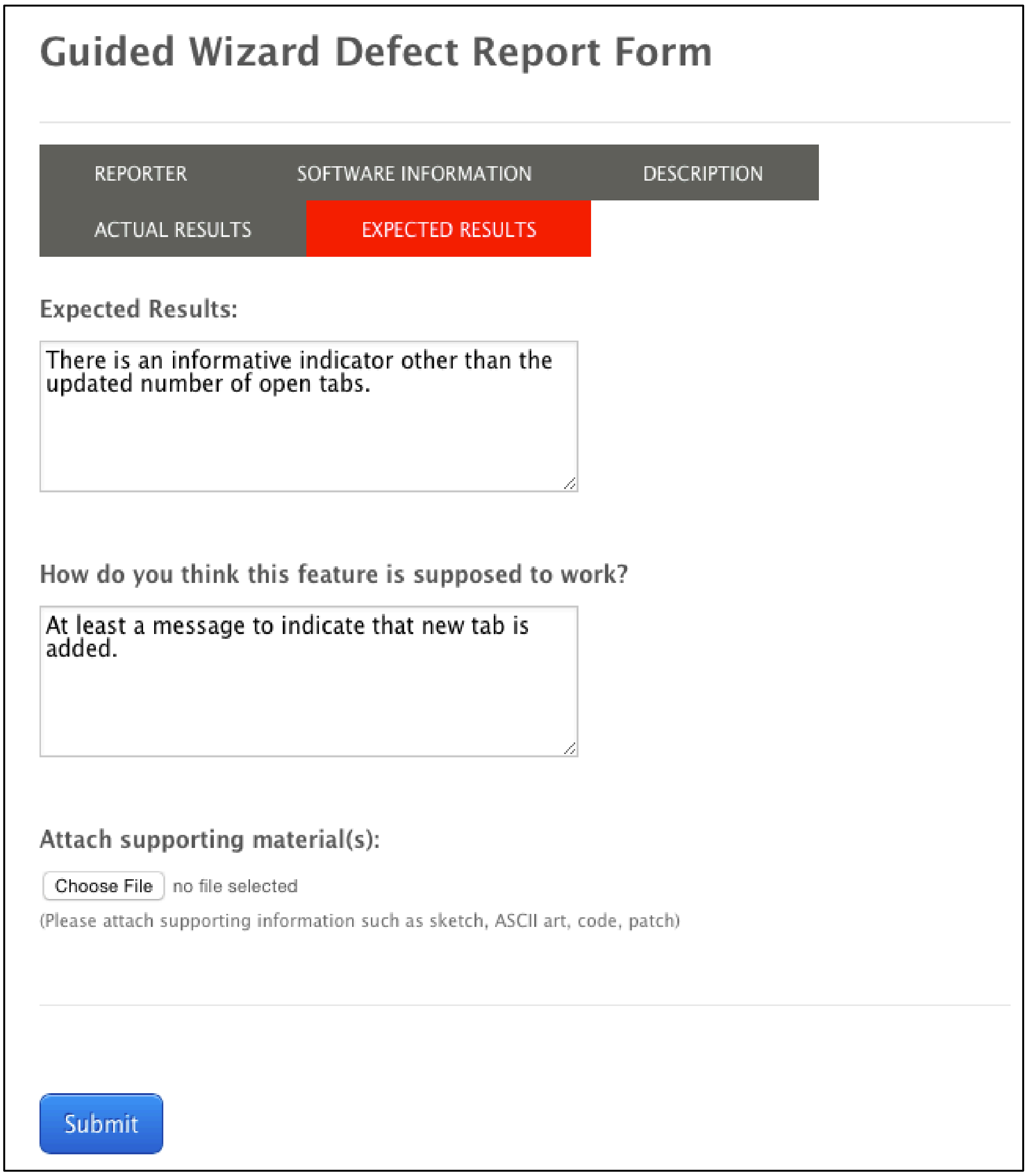
	1	2	3	4	5	
Not at all	○	○	○	●	○	Very much

Attach a file(s):

Choose File IMG_3567.JPG

(Please attach supplementary information such as a screenshots, video and audio)

Figure 7.13. Actual Results tab to collect actual output, task difficulty, emotions, and supplementary information to support justifications.



Guided Wizard Defect Report Form

Expected Results:

There is an informative indicator other than the updated number of open tabs.

How do you think this feature is supposed to work?

At least a message to indicate that new tab is added.

Attach supporting material(s):

no file selected

(Please attach supporting information such as sketch, ASCII art, code, patch)

Figure 7.14. Expected Results tab to collect expectation behavior, solution proposals, and supplementary information to support solution proposals.

7.2 Evaluation

This section describes the study conducted in order to evaluate the quality of usability defect information collected using our proposed forms from the defect report reader's perspective. We used an expert judgment approach to evaluate the presence of specific usability defect information. The form's effectiveness in eliciting the usability defect information is beyond the scope of this research, and therefore the aspect of usability, ease of use, or other aspects of our proposed reports were not evaluated. This aspect will be evaluated in a follow-up experiment as discussed in Section 8.4.4. The following subsections describe participant selection, problem selection, and protocol in conducting our evaluation.

7.2.1 Hypotheses

We hypothesized that the use of proposed usability defect report forms would increase the quality of the usability defect descriptions as rated by expert judges.

7.2.2 Participation Selection

In this study, three very experienced software development experts evaluated the information presented in the Bugzilla defect report form and the proposed form. The evaluators had significant levels of experience in industrial and academic development environments. In particular, all experts have more than 10 years of experience in software development industries. At the outset of the study, two evaluators had substantial experience with Bugzilla, and one had limited exposure to the Bugzilla defect reporting tool. Table 7.3 summarizes the participated expert details.

Table 7.3. Experts who participated.

Evaluator	Academic and Industrial Background
Evaluator 1 (E1)	He has been an academic for approximately 25 years and has worked in the IT industry as a programmer/ analyst and consultant. His research expertise includes software tools and techniques, software architecture, model-driven software engineering, visual languages, software security engineering, service-based and component-based systems, and user interfaces.
Evaluator 2 (E2)	He is a senior IT professional with over 20 years experience. He specializes in software engineering with a special emphasis on component technology and user-centered approaches.
Evaluator 3 (E3)	He is an information technology expert with over 15 years of experience. He specializes in solution architecture, mobile technology, and analysis of large data sets. He has extensive experience on open source projects in relation to software testing, which will be very relevant to this study.

7.2.3 Problem Selection

We studied Firefox, Firefox for Android, and Eclipse Platform projects to understand the contextual content of usability defect reports and the findings were used to design our new usability defect report forms. To explore the generalizability of our usability defect report design for other open source products, we decided to use the Firefox for iOS project. Firefox for iOS is a mobile web browser from Mozilla for the iPhone, iPad, and iPod touch.

We selected ten usability defects from the Firefox for iOS project as case studies. These case studies were then used for our evaluation. The ten usability defects were chosen in the following way:

- The usability problems were selected randomly from the 861 *New* defects (as of 21st March 2017). The decision to use defects with *New* status guaranteed that the defects had not been examined by the software developers, and we have the possibility of reproducing the defects and reporting them in our proposed form.
- The defects are tagged with Bugzilla usability keywords - ue, uiwanted, useless-UI, ux-affordance, ux-consistency, ux-control, ux-discovery, ux-efficiency, ux-error-prevention, ux-error-recovery, ux-implementation, ux-interruption, ux-jargon, ux-minimalism, ux-mode-error, ux-natural-mapping, ux-tone, ux-trust, ux-undo, ux-userfeedback, ux-visual-hierarchy. The rationale for using these developer-tagged keywords was to reduce selection bias, as the software developers already assessed the validity of the defects and accepted the need for fixing.
- The defects are reproducible in our iOS mobile device. This is because we wanted to rewrite the usability defect descriptions using our defect report form and not bias them based on the original descriptions submitted by the reporters.

7.2.4 Development of Case Studies

We chose five usability defect reports from Firefox for iOS projects. We considered two approaches to select the report to reproduce: sampling randomly, or sampling only reports with GUI-related usability defects. We chose the latter, since our goal was to reproduce the issue and rewrite the usability defect descriptions, in which we found GUI-related usability defects are more objective and much easier to reproduce in our iPhone. We read the defect report, understanding the problem context, and reproduced the problem on our own until we found the reported problem. Then, we used our proposed usability defect report forms to write the usability defect descriptions. For the purpose of the evaluation, only plain text is displayed for both the original and reproduced defects according to the current Bugzilla defect report template. The questions prompted in our new forms were not made available on the reproduced defect reports. Figure 7.15 shows the five case studies we reproduced.

Bugzilla Report#1237631

Report Title:
Do not exit private mode on browser re-launch even if there are no private tabs to restore

Description:
I am not satisfied with the existing Private Tab browsing feature. Currently, Firefox browser only opens browser in normal mode, even though the last opened browser was in Private mode.

Steps to Reproduce:

1. Press on Firefox browser icon.
2. Press the middle menu at the bottom of the Tab manager page – then press on New Private Tab.
3. Repeat step 2 to open several private tabs.
4. Press on the Tab button (tiny purple box) on the right side of the URL bar.
5. Press the middle menu at the bottom of the Tab manager page – then press on Close All Tabs.
6. Re-launch Firefox

Actual Results:
Firefox has opened in normal browsing. I am confused because normally I browse using Private mode. Even though this feature design is not affecting my task, it is actually affecting my browsing privacy.

Expected Results:
Firefox will open in private mode, similar to Safari browser.

Bugzilla Report#1231815

Report Title:
Irrelevant steps to add new tab

Description:
I am not satisfied with the steps to open a new Tab. The current steps are irrelevant to speed up the process to add new Tab.

Steps to Reproduce:

1. Click on Firefox browser icon.
2. Go to any webpage. For example open http://ebay.com.au
3. When a webpage is open, open a new tab. Press the Tab button on the right side of the URL bar.
4. Press the "+" button at the bottom of the Tab manager to add a new tab. On the newly open page, open a new page, for example – http://gumtree.com.au

Actual Results:
To open new tab, user requires two steps (step 3 and step 4 above). Even though this issue is not affecting my task, this is really annoying when I have to use only one hand to hold the phone and use several tabs to look for something.

Expected Results:
Long-press the tab button to add new Tab. Using long-press button could provide an easy way to interact with the web browser, especially when the user is moving.

Bugzilla Report#1145602

Report Title:
New About:Home tabs experience is confusing

Description:
I can’t directly find tabs that have been opened. The newly added tabs are represented as multiple blank/empty spots, which are hidden in multilayer page. User can create multiple new tabs without realizing they are doing so. Intention to create a new tab isn’t clear.

Steps to reproduce:

1. Click on Firefox browser icon.
2. Go to any webpage. For example open http://ebay.com.au
3. Press the middle menu at the bottom of the Tab manager page – then press on New Tab

Actual results:
There is no obvious indicator that shows new tab is created. If you are notice carefully, only the number on the tab icon (square box on the right side of URL bar) is updated/ increased whenever you add a new tab. However, if you were not aware of the existing number of tabs open, you might not have known that a new tab had been added. It took me some time to figure out if new tabs were successfully added or not, and I did not know where to find the existing open tabs. The only way you can know if the tab was created is by pressing tab icon on the right side of the URL bar. The Tab Manager experience is really confusing.

Expected results:
There is an informative indicator other than the updated number of open tabs. At least a message to indicate that new tab is added.

Bugzilla Report#1199983

Report Title:
Unable to scroll long page using scroll indicator

Description:
I have difficulty in manipulating object (scroll indicator) in the user interface. When I open a really large page, like a Wikipedia page, it appears that I can only scroll further down with the swipe gesture. The scroll indicator on the right page is not able to grab.

Steps to reproduce:

1. Go to Wikipedia page and search for information, which may have long information.
2. Touch the screen. You will see the scroll indicator on the right side.
3. Try to grab the scroll indicator and scroll down.

Actual results:
You are unable to scroll using the scroll indicator. Instead, you are scrolling using swipe gesture. It took so much time for me to scroll to the middle of the page. I spent literally 5 mins just with the swiping restore to get to the middle of the page, because it seems that I can’t grab the scroll indicator on the right and move it down with my finger to scroll faster like in other iOS apps.

Expected results:
The scroll indicator works as desktop webpage. User can grab the scroll indicator and fast scroll to the up/ bottom of the page.

Bugzilla Report#1146389

Report Title:
Unclear how to access Reader View

Description:
I have difficulty in finding out Reader view icon. The reader View icon is not persistently displayed when user scrolls down the page. In addition, the very small Reader view icon and its color are not intuitive for user to discover.

Steps to reproduce:

1. Click on Firefox browser icon.
2. Go to google search and find some articles – for example search on “software engineering”.
3. Select on one of the search results, and wait for the information page to be loaded.
4. While the information is loading, scroll down the page.
5. Observe the “Reader View” icon and try to access Reader View.

Actual results:
When the information is loading and I scroll down the page, I couldn’t find any sign to access Reader View. The “Reader View” icon is not immediately shown at the end of the URL bar and the icon is not discoverable when you scroll down the page. I have to scroll up to the very top of the page to find Reader View icon. This can be time consuming if I have a very long page. Furthermore, the “Reader View” icon is very small and the grey icon on the black background (in private browsing) is not discoverable enough.

Expected results:
The “Reader View” icon should be triggered while you are in any part of the page. So that user can access Reader View icon instantly without scrolling up to the top of the page.

Figure 7.15. Five Firefox for iOS case studies that were reproduced from the original defect reports.

7.2.5 Protocol

In order to compare the quality of usability defect descriptions produced by current and proposed defect report forms, we reproduced a set of usability defects and rewrote the defect descriptions using our proposed form. For this purpose, we selected five usability defects (as described in section 7.2.4), reproduced the usability defects in her own iPhone device, and wrote the defect description using the proposed guided usability defect report form. Although both original and proposed defect report forms contain specific contextual information (i.e., status, people, tracking, software information), the defect descriptions given to evaluators contained minimal information. We only provided contextual information about reproduction steps, actual and expected results, and explanation of the usability defects. We limited the amount of detail provided to evaluators to ensure the evaluators were not biased with a specific defect report format. The final copy of defect reports that were presented to the evaluators was modified to prevent the identification of specific formats.

We used an independent judgment approach, in which the evaluators performed the evaluation at locations and time (s) of their choosing. Each evaluator was given the following material (also included in Appendix E).

- Five original usability defect reports of Firefox for iOS product
- Five usability defect reports of Firefox for iOS developed as case studies

The evaluators were required to read ten defect reports (five original Firefox for iOS defect reports and five developed case studies) and evaluated the contextual information of each report on four aspects. For each aspect, we evaluated whether the report provided or failed to provide a description containing the aspect under consideration. Problems were evaluated in random order and it was not made known as to which report format was used to record the usability defects. The four aspects we used for the evaluation were adopted from [9], [63], and are described below:

Informative – According to Capra's guidelines [11], informative usability defect descriptions should describe the solution to the problem, the cause of the problem, and the usability issue involved in the problem. Describing this information has been suggested as important to better understand and fix the problem [11],[108]. This information should be supported with screen snapshots, pictures, video and audio, usability design principles and/ or previous research.

Accuracy – Accuracy is measured in terms of how closely the problem can be reproduced by the evaluators. Good defect descriptions should consist of a clear set of instructions that other readers can use to reproduce the defect on their own machine.

Claim and rationale - In the absence of usability specialists to observe and verify usability defects in open source projects, justification about why it was a problem [9] is very important to help software developers understand the nature of the problem. The claim about the problem should justify the failure qualifier criteria that violates user expectations, including missing, incongruent mental model, irrelevant, wrong, better way, and overlooked. When arguing for a particular claim, support for rationale and evidence is valuable in confirming the validity of the problems.

Impact – The defect description should contain something valuable that highlights the priority of defects that need to be fixed. For this purpose, defect reports should describe the impact of the problem on business goals (i.e., costs, time loss), user task, and human emotion [4]. Impact on the user's task explains about interruptions of task performance, unnecessary steps to workaround the problems, or the user struggling with task completion, while human emotion places emphasis on confusion, frustration, annoyance, and uncertainty [104].

For each of these aspects, the evaluators were given eleven questions as listed in Table 7.4. Each evaluation aspect was given a score of 1 if the evaluators thought that the usability defect report "completely described", 0.5 if the usability defect report "partially described", and 0 if the usability defect report "do not describe" the evaluation aspects. The total quality score was calculated by summing up the scores, which ranged between 0 (low quality) and 11 (high quality).

Table 7.4. List of questions in the survey evaluation.

Aspect	**Questions**	**Answer**
Informative	1. Does the defect report offer proposals for solutions? For example, the descriptions provide alternatives and tradeoffs, and supplied rationale for the recommendations [9]. 2. Does the defect report describe the cause of the problem, including a justification of what posed a problem, including system components that are affected or involved? 3. Does the defect report describe the main usability issue involved in the problem? For example, a description about what is wrong with the interaction architecture, interface and user task design.	Yes/ No/ partially
Accuracy	4. Has the defect report explained in detail step by step how to reproduce the problem, including user's navigation flow through the system? 5. Are you able to reproduce the problem on your own device and environment? 6. Were the actual results you observed similar to the one in the defect description?	Yes/ No/ partially
Justified	7. Does the defect report offer a justification for why the reporter thinks that it was a problem?	Yes/ No/ partially
Impact	8. Does the defect report explicitly mention what poses a problem to the user? 9. Does the defect report describe the impact of the problem on business effect, impact on the user's task, and importance of the task? 10. Does the defect report describe reporters' emotion, feeling, or reactions with regards to the issues? 11. Does the defect report mention how often the problem occurred or if other users experienced the same problem?	Yes/ No/ partially

7.2.6 Analysis

We hypothesized that using our proposed usability defect report forms would increase the quality of the usability defect reports as rated by evaluators – in terms of capturing more information. To test this hypothesis, we conducted a two-way 3 (evaluator: evaluator 1, evaluator 2, and evaluator 3) x 2 (type of report: original, and case study) mixed ANOVA with repeated measures on the type of report variable. Additionally, we also measured inter-rater reliability among evaluators. We used SPSS (version 23) to conduct the repeated measures ANOVA analysis, and Microsoft Excel 2011 to calculate the Fleiss Kappa inter-rater reliability.

7.3 Results

7.3.1 Repeated Measures ANOVA

Means are based on individual ratings given by each evaluator, rather than the sums of the three ratings. Evaluators are required to rate five original Firefox for iOS usability defect reports, and five Firefox for iOS usability defect reports that have been rewritten using the proposed content, called as case study.

Prior to conducting repeated measures ANOVA, Mauchly's sphericity test was conducted to measure the variances of the differences between all combinations of related groups. As shown in Table 7.5, the main effect of evaluator does not significantly violate the sphericity assumption because the significance value is greater than 0.05, W= 0.247, X^2 (2) = 4.20, p > 0.05. Therefore, the F-value for the main effect of evaluator (and its interaction with the between-group variable *defect type*) does not need to be corrected for violations of sphericity. Since type of report has only two categories, no significance test is needed, W=1.

Table 7.5. Mauchly's Test of Sphericity[a]

					Epsilon[b]		
Within Subjects Effect	Mauchly's W	Approx. Chi-Square	df	Sig.	Greenhouse-Geisser	Huynh-Feldt	Lower-bound
Evaluator	.247	4.201	2	.122	.570	.648	.500
Report	1.000	.000	0	.	1.000	1.000	1.000
Evaluator * Report	.886	.364	2	.833	.897	1.000	.500

Tests the null hypothesis that the error covariance matrix of the orthonormalized transformed dependent variables is proportional to an identity matrix.

a. Design: Intercept
Within Subjects Design: Evaluator + Report + Evaluator * Report

b. May be used to adjust the degrees of freedom for the averaged tests of significance. Corrected tests are displayed in the Tests of Within-Subjects Effects table.

Table 7.6. Tests of Within-Subjects Effects.

Source		Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Evaluator	**Sphericity Assumed**	**40.617**	**2**	**20.308**	**14.272**	**.002**	**.781**
	Greenhouse-Geisser	40.617	1.141	35.610	14.272	.014	.781
	Huynh-Feldt	40.617	1.295	31.364	14.272	.010	.781
	Lower-bound	40.617	1.000	40.617	14.272	.019	.781
Error(Evaluator)	Sphericity Assumed	11.383	8	1.423			
	Greenhouse-Geisser	11.383	4.562	2.495			
	Huynh-Feldt	11.383	5.180	2.198			
	Lower-bound	11.383	4.000	2.846			
Report	**Sphericity Assumed**	**70.533**	**1**	**70.533**	**59.816**	**.002**	**.937**
	Greenhouse-Geisser	70.533	1.000	70.533	59.816	.002	.937
	Huynh-Feldt	70.533	1.000	70.533	59.816	.002	.937
	Lower-bound	70.533	1.000	70.533	59.816	.002	.937
Error(Report)	Sphericity Assumed	4.717	4	1.179			
	Greenhouse-Geisser	4.717	4.000	1.179			
	Huynh-Feldt	4.717	4.000	1.179			
	Lower-bound	4.717	4.000	1.179			
Evaluator * Report	**Sphericity Assumed**	**40.417**	**2**	**20.208**	**9.194**	**.008**	**.697**
	Greenhouse-Geisser	40.417	1.795	22.520	9.194	.011	.697
	Huynh-Feldt	40.417	2.000	20.208	9.194	.008	.697
	Lower-bound	40.417	1.000	40.417	9.194	.039	.697
Error(Evaluator* Report)	Sphericity Assumed	17.583	8	2.198			
	Greenhouse-Geisser	17.583	7.179	2.449			
	Huynh-Feldt	17.583	8.000	2.198			
	Lower-bound	17.583	4.000	4.396			

Table 7.7. Post-hoc tests – Multiple comparisons for evaluator.

					95% Confidence Interval	
(I) Evaluator	(J) Evaluator	Mean Difference (I-J)	Std. Error	Sig.	Lower Bound	Upper Bound
E1	E2	-1.4500	.66958	.122	-3.1733	.2733
	E3	**-2.8500***	**.66958**	**.001**	**-4.5733**	**-1.1267**
E2	E1	1.4500	.66958	.122	-.2733	3.1733
	E3	-1.4000	.66958	.142	-3.1233	.3233
E3	**E1**	**2.8500***	**.66958**	**.001**	**1.1267**	**4.5733**
	E2	1.4000	.66958	.142	-.3233	3.1233

Based on observed means.
The error term is Mean Square(Error) = 2.242.
*. The mean difference is significant at the

As shown in Table 7.6, there was an evaluator main effect, *F*(2,5) = 14.27, $p < 0.001$, and a report type main effect, *F*(1,5) = 59.82, p<0.01, and there was also an evaluator x report type interaction, *F*(2,5) = 20.21, $p < 0.01$. For evaluator effects, the results suggested that if type of defect report is ignored, some evaluators still rated significantly differently to others.

To find the nature of this effect, we performed post hoc tests. The multiple comparisons for the main effect of evaluator corrected using Bonferroni adjustments are shown in Table 7.7. As can be seen, the significant main effect reflects a significant difference (p<0.01) between E1 and E3 (evaluator 1 and evaluator 3) but not between E1 and E2 (evaluator 1 and evaluator 2) and E2 and E3 (evaluator 2 and evaluator 3). This indicates that the quality scores given by E1 and E3 had an effect on ratings of both original and case study defect report quality.

In terms of report type effects, the results suggested that if all other variables are ignored, case study defect report quality ratings were significantly different as compared to the original defect report. As depicted in Figure 7.16, quality of case study defect reports is rated higher than original defect report, regardless of who the evaluators are. Also, as we can see, the relationship between mean quality scores and evaluator is different for original and case study defect reports. For original defect reports, mean quality scores increase from evaluator 1 to evaluator 3. While the original defect reports show a slightly large variation in mean quality scores for different evaluators, case study defect reports had much less variation. However, mean quality scores rated by evaluator 3 is higher for both original and case study defect reports. For case study defect reports, mean quality scores increased from evaluator 1 to evaluator 2, and decreased from evaluator 2 to evaluator 3. Overall, the mean quality scores for the case study defect reports, M = 8.77, SD = 1.58, were significantly higher than for the original defect reports, M= 5.70, SD = 2.67, which supported our hypothesis.

In summary, even though the quality of usability defect descriptions is significantly influenced by *evaluator effect*, overall assessment showed that usability defect descriptions written using our proposed forms can capture more information than the usability defect descriptions in the three OSS projects studied.

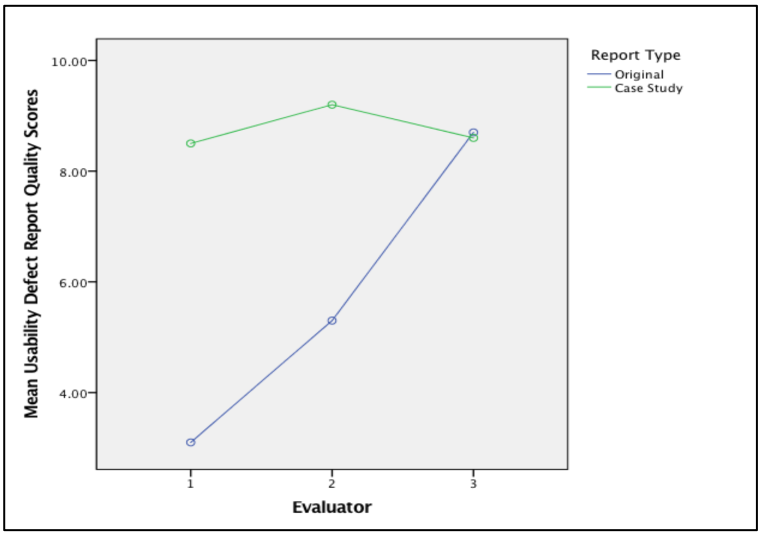


Figure 7.16. Mean usability defect report quality scores as rated by three evaluators.

7.3.2 Reliability Analysis

Based on the previous analysis, since the evaluator has statistically proven to have significant effect on the scores of usability defect report quality, we further investigated the level of agreement among the three evaluators. This is because of the diverse experience among evaluators and the different interpretation of usability defect report content could affect the confidence of the study results [107]. In this study, we used percent agreement to measure the degree of agreement among evaluators, which is called *inter-rater reliability*.

Table 7.8 summarizes the inter-rater reliability results. For each usability defect report, we measured the percent agreement of each evaluation aspect (Q1 – Q11). We calculated the percent agreement for each column and calculated averages of the rows. To understand this procedure, consider Report 1 in Table 7.7. Since in this analysis we only have three evaluators and the scores are limited to three values, to obtain percent agreement we only need to count the number of evaluators that have the same score and divide by the number of evaluators. For Report 1, as an example, the three evaluators scored 0.5 for Q1 and therefore, the percent agreement was computed as 3 divided by 3, which is 1.00 – perfect agreement. For Q5, two evaluators scored 1, and this given the percent agreement of 0.67 for Q2 (2 divided by 3). The inter-rater reliability for Report 1 was computed by summing up the percent agreement of Q1 until Q11 and divided by the number of questions, which is 11. This resulted in an inter-rater reliability of Report 1 of 0.79.

In Table 7.8, it can be seen that, on average, the agreement of case study usability defect report score is higher than the original usability defect report. The highest agreement was observed in case study usability defect Report#2 and Report#9, which exhibits overall inter-rater reliability of 88% each, and the lowest inter-rater reliability of 48% for Report#6. The case study agreement, which is more than 80%, means that only about 20% of the scores rated by the evaluators were erroneous because only one of the evaluators can be corrected when there is disagreement. On the contrary, about 20%-50% of the scores in original defect reports disagreed.

The highest agreement of evaluation aspects was observed in Q6. Two and four of original and case study usability defect reports, respectively had inter-rater reliability of 100%. Interestingly, the three evaluators were in perfect agreement, inter-rater reliability of 100% when scoring Q8 –

justification of impact in the five-case study usability defect reports. The percent agreement statistics can also be of benefit in identifying evaluation aspects (Q1 – Q11) that may be problematic.

Note that Table 7.9 shows that the three evaluators were most in disagreement when scoring Q2 and Q7. The evaluators achieved 0% agreement for Q2 and Q7 in three out of the five original usability defect reports. These evaluation aspects that are related to justification of cause of the problems and failure qualifier may warrant scrutiny to identify the cause of such low agreement in its scoring.

Table 7.8. Percent agreement across three evaluators evaluated the quality of usability defect reports.

	Report	Evaluators	Evaluation Aspects											IRR
			Q1	*Q2*	*Q3*	*Q4*	*Q5*	*Q6*	*Q7*	*Q8*	*Q9*	*Q10*	*Q11*	
Original	**1**	E1	0.5	1	1	0.5	0	0	0	0.5	0	0.5	0	0.79
		E2	0.5	1	1	0.5	1	0.5	0	1	0.5	0.5	0	
		E3	0.5	1	1	1	1	0.5	1	1	0	1	0	
		% Agreement	1.00	1.00	1.00	0.67	0.67	0.67	0.67	0.67	0.67	0.67	1.00	
	4	E1	0.5	0	1	0.5	0	0	0	0.5	0	0	0.5	0.55
		E2	0.5	0.5	1	0.5	0	0	0.5	1	0.5	0	0	
		E3	1	1	1	1	1	1	1	1	0	0	1	
		% Agreement	0.67	0.00	1.00	0.67	0.67	0.67	0.00	0.67	0.67	1.00	0.00	
	6	E1	0	0	0	1	0	0	0	0	0	0	0	0.48
		E2	0.5	0.5	0.5	1	0	0.5	0.5	0.5	0.5	0.5	0	
		E3	1	1	1	1	0	0	1	0.5	0	0	1	
		% Agreement	0.00	0.00	0.00	1.00	1.00	0.67	0.00	0.67	0.67	0.67	0.67	
	8	E1	0.5	0.5	0.5	0.5	0.5	1	0	0	0	0	0	0.61
		E2	0.5	0.5	0.5	1	0.5	0.5	0.5	0.5	0	0	0	
		E3	0.5	1	1	1	1	1	1	1	0	0.5	1	
		% Agreement	1.00	0.67	0.67	0.67	0.67	0.67	0.00	0.00	1.00	0.67	0.67	
	10	E1	0	0	0.5	0.5	1	1	0.5	0.5	0	0	0	0.55
		E2	1	0.5	1	0.5	0.5	0.5	0.5	1	0.5	0	0	
		E3	1	1	1	1	1	1	1	1	1	1	1	
		% Agreement	0.67	0.00	0.67	0.67	0.67	0.67	0.67	0.67	0.00	0.67	0.67	
Case Study	**2**	E1	0.5	1	1	1	1	1	1	1	0.5	1	0	0.88
		E2	0.5	0.5	1	1	1	1	1	1	1	1	0	
		E3	1	1	1	1	1	1	1	1	0.5	1	0.5	
		% Agreement	0.67	0.67	1.00	1.00	1.00	1.00	1.00	1.00	0.67	1.00	0.67	
	3	E1	1	0.5	1	0.5	1	1	0.5	1	0	0.5	0	0.73
		E2	1	0.5	1	1	1	1	1	1	1	1	0	
		E3	1	1	1	0.5	0	0	0	1	0	1	0	
		% Agreement	1.00	0.67	1.00	0.67	0.67	0.67	0.00	1.00	0.67	0.67	1.00	
	5	E1	1	1	1	1	1	1	1	1	0	0.5	1	0.73
		E2	1	0.5	1	1	1	1	0.5	1	0.5	1	0.5	
		E3	1	1	1	1	0	0	1	1	0	0	1	
		% Agreement	1.00	0.67	1.00	1.00	0.67	0.67	0.67	1.00	0.67	0.00	0.67	
	7	E1	0	1	1	1	0	0	0.5	1	0	1	1	0.70
		E2	1	0.5	1	1	1	1	1	1	0.5	0	0.5	
		E3	1	1	1	1	1	1	1	1	1	1	1	
		% Agreement	0.67	0.67	1.00	1.00	0.67	0.67	0.67	1.00	0.00	0.67	0.67	
	9	E1	1	0.5	1	1	1	1	1	1	1	1	1	0.88
		E2	1	0.5	1	1	1	1	1	1	1	1	0.5	
		E3	1	1	0	1	1	1	1	1	0.5	1	1	
		% Agreement	1.00	0.67	0.67	1.00	1.00	1.00	1.00	1.00	0.67	1.00	0.67	

** IRR – Inter Rater Reliability*

Table 7.9. Number of scores that disagree with the majority of evaluators' scores

Is an evaluator an Outlier?	**Report 1**			**Report 2**			**Report 3**			**Report 4**			**Report 5**		
	E1	***E2***	***E3***	***E1***	***E2***	***E3***	***E1***	***E2***	***E3***	***E1***	***E2***	***E3***	***E1***	***E2***	***E3***
#of unlike responses:	3	1	3	0	2	2	1	2	3	1	1	4	0	4	2

Table 7.9. Number of scores that disagree with the majority of evaluators' scores (Continue)

Is an evaluator an Outlier?	**Report 6**			**Report 7**			**Report 8**			**Report 9**			**Report 10**		
	E1	***E2***	***E3***	***E1***	***E2***	***E3***	***E1***	***E2***	***E3***	***E1***	***E2***	***E3***	***E1***	***E2***	***E3***
#of unlike responses:	1	3	1	4	3	0	1	1	5	0	1	3	3	2	4

As the post-hoc test in section 7.3.1 revealed, evaluator E1 and E3 had a significant effect on the quality scores. As Table 7.9 shows, E1 and E3 had an excessive number of outlier scores, in which they frequently gave scores different from the other evaluators. For example, E3 disagreed with the other two evaluators' scores five times when assessing Report#8.

In summary, the high level of agreement between evaluators who assessed the case study usability defect descriptions suggests that our proposed usability defect report forms is able to improve the clarity of contextual information. The low level agreement for original usability defect descriptions is an indicator that the defect information presented in the current open source usability defect reports is unclear, which leads to different interpretation and assessment between evaluators.

7.4 Threats to Validity

7.4.1 External Validity

One possible threat to the external validity is regarding the choice of evaluators and number of evaluators that can affect the outcome of the evaluation. Previous studies have reported that evaluators' background can play a significant roles on the outcome of software testing [14], [87], and we have seen this potentially affect the quality of the assessment defect report as well. We plan to conduct different evaluation methods in future to test the outcome of this study. For example, we plan to replicate the study conducted by Capra [63] with different settings.

7.4.2 Internal Validity

Subject selection bias represents one possible threat to internal validity. In our evaluation, we only used ten Firefox for iOS usability defect reports. OSS projects may have different report content and structure, and the same evaluations on different OSS projects may yield different results. Additionally, the evaluation was performed on a smaller number of case studies (5 self-constructed case studies described in Chapter 7.2.4), which further reduce the generality of our results.

7.4.3 Construct Validity

For the evaluation of our proposed usability defect reporting forms, we developed the instruments ourselves. We have rewritten the original five usability defect descriptions using our proposed forms as case study defect reports. One possible threat might be the reliability of the evaluation. However, instead of copying the original information and adding dummy information to the new form, we reproduced the defects on our machine and wrote the usability defect description using our own experience and interpretation. This reproduction process may affect the amount of information reported as specific information was requested. The original defect reports were used as the guidelines to reproduce the problems. Hence, we consider the instruments as a reliable instrument for evaluating the quality of information between original and case study defect reports.

Furthermore, the appropriateness of the assessment metrics used to rate the quality of information threatens the construct validity. To mitigate this threat, we adopted Capra's well-accepted guidelines [11] as grounds for quality assessment metrics used for evaluation.

7.5 Discussion

The results from ANOVA statistics and reliability analysis supported our hypothesis. The expert judgment approach (discussed in Section 7.3.1 and 7.3.2) has demonstrated an improvement in the quality of information when usability defects are reported using the proposed usability defect report forms. There are two benefits that we identified from the proposed form. As shown in Figure 7.17, usability defect descriptions rewritten using our forms received full score (score =1) for *Q8 – user difficulty*. Other attributes, such as *Q3 - description of usability issues*, *Q4 - steps to reproduce*, *Q5 - reproducibility*, *Q6 actual output, Q7 - failure qualifier*, and *Q10 – human emotions* were also received high scores than those original usability defect reports written on generic report templates. This finding shows that by separating attributes into multiple instances, giving hints and examples could provide competence assistance for non-technical reporters to construct more meaningful usability defect descriptions.

From a defect report readers' perspective, the structured contextualized attributes help software developers, for example, to easily identify the presence of key defect information. From Figure 7.17 below, the high scores given for each evaluation criteria of case study defect reports suggested that the key information could be easily identified when a defect is described in a contextualized context. This is necessarily important in OSS development, where most software developers are "non usability-savvy" and work for limited time to read concise defect description in order to understand the usability issues so that they can directly point the solution to fix the issues.

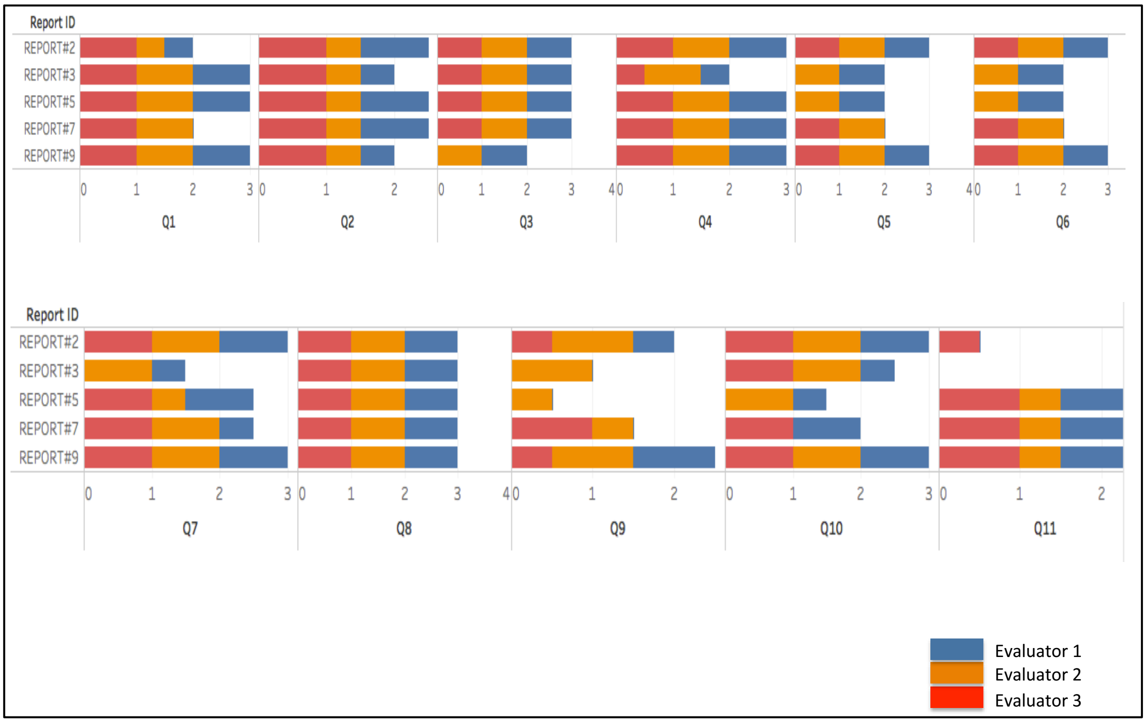


Figure 7.17. Case study defect reports content evaluation scores. On average, the three evaluators in agreement on the presence of *Q2 – assumed cause, Q4 – steps to reproduce, Q7 – failure qualifier,* and *Q8 – user difficulty*

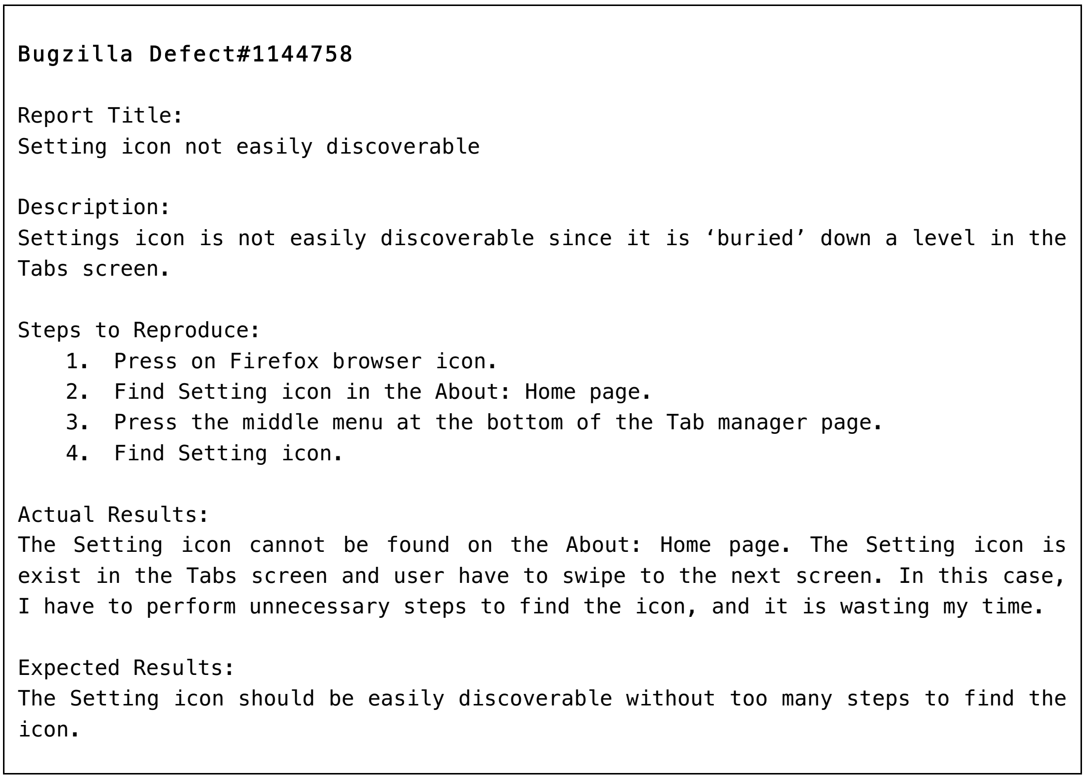
Through our analysis, we also observed that *Q2- assumed cause* and *Q7 – failure qualifier* was difficult to identify in usability defect descriptions and received different interpretation between evaluators. As we discussed in Chapter 4, the cause of the problem is rarely found in the usability defect descriptions. Indeed, information such as stack traces, UI event traces, and error logs, which are useful for software developers to identify the root cause of the problem, are the most difficult type of information to be provided by non-technical reporters.

Many other works have been devoted to capture stack traces, UI event traces, error logs, and monitoring user interactions [90], [91], [109]–[112], [110]–[112], but most of them are complicated for non-technical reporters. Furthermore, given the nature of the OSS projects, where users are involved on a voluntary basis, it would be tedious and time-consuming to manually collect this kind of information. As a solution to this, when designing new defect report forms, attributes that are related to Q2 and Q7 might need to be rephrased or redefined to include other ways of capturing and presenting this information.

In relation to Q7, however, in the five-case study usability defect reports that we developed for evaluation purposes, we demonstrated the presence of *Q7 – failure qualifier* using plain text, instead of using the predefined values (missing, wrong, incongruent mental model, irrelevant, better way, and overlooked) as suggested in our prototype design. Therefore, we could not justify if the predefined values that we suggested in the new usability defect report forms could help the reporters to describe the failure qualifier that triggered them to report the usability defects. Future evaluation on reporting usability defects using our forms could provide suggestions on a suitable input format to capture this information. This is because the correct choice of input format could help to narrow down the information needed by software developers. In some cases, using plain text to express a reporter's feelings and difficulties on usability issues may not be convincing. As such, using a rating scale to rate a user’s difficulties, feelings, and emotions, for example, could be more objective for software developers to assess the priority/ severity of the problem to be fixed. Nevertheless, the use of plain text input cannot be disregarded – especially in allowing reporters to write their concerns about usability, which may not be relevant from a software developer perspective, but can be useful resources for future references.

The results of this research also suggest that the interpretation of usability defect descriptions is subject to an evaluator effect, similar to other research investigating evaluator effect when finding software defects [48], [87], [113]. There are two explanations regarding the evaluator effect in assessing usability defect report quality. First, as different evaluators may understand the problems from different perspectives, the assessment scores tend to vary. If evaluators were more reliable, involving more evaluators would not result in a substantial difference of defect report quality scores because the evaluators would rate roughly the same scores.

Secondly, the evaluators’ personal experiences and their familiarity with the system under test may introduce subjective interpretation [113]. This is because even though the evaluators read the same usability defect descriptions and used the same assessment guidelines, the quality scores were still significantly different. Consider Report#1144758 used in our evaluation study (see Figure 7.18). For this quality assessment, E1 had score of 1, while E2 and E3 had 5 and 6.5 scores, respectively. The obvious difference was observed in rating the three aspects *Q1- solution proposal*, *Q2 – assumed cause*, and *Q3 – description of usability issues*. While E1 considered Report#1144758 did not contain this information, E2 and E3 agreed that this information is present although the level of detail of this information is variedly rated by E2 and E3. Research on evaluator effects could be extended to understand specific skills, competencies, and abilities of reporters to write a good usability defect description.



```
Bugzilla Defect#1144758

Report Title:
Setting icon not easily discoverable

Description:
Settings icon is not easily discoverable since it is ‘buried’ down a level in the
Tabs screen.

Steps to Reproduce:
    1.  Press on Firefox browser icon.
    2.  Find Setting icon in the About: Home page.
    3.  Press the middle menu at the bottom of the Tab manager page.
    4.  Find Setting icon.

Actual Results:
The Setting icon cannot be found on the About: Home page. The Setting icon is
exist in the Tabs screen and user have to swipe to the next screen. In this case,
I have to perform unnecessary steps to find the icon, and it is wasting my time.

Expected Results:
The Setting icon should be easily discoverable without too many steps to find the
icon.
```

Figure 7.18. Report#1144758 of Firefox for iOS

7.6 Summary

Through a series of studies, we have investigated challenges and limitations experienced by reporters in reporting usability defects (Chapter 1), information needs by software developers when fixing usability defects (Chapter 4), and factors influencing usability defect reporting (Chapter 5). Results from these studies were extracted as desirable features of new open source usability defect report forms to facilitate open source communities, especially non-technical reporters to identify, understand, and report usability defects. We then evaluated the proposed usability effect report forms in the context of quality of information using an expert judgment approach. The results from the evaluation suggest that by using multiple attribute instances to capture only one value per attribute could improve the quality of information presented in the usability defect descriptions.

This research represents a first step in investigating a guided wizard reporting form approach for better capturing usability defect descriptions. Further research is necessary to understand if such guided forms could benefit experienced reporters as well as novice reporters, in terms of ease of use, and the nature of the problems reported. Additionally, further research is also necessary to understand if the information captured is in accordance to the information needs of software developers and whether the information is useful for software developers in prioritizing and fixing usability defects.

8 Conclusions

This chapter concludes this thesis by presenting a summary of the work carried out answering the thesis key research questions. It discusses the contributions of this research in the context of academic and industrial practice, reports the lessons that we have learned from our experience, and addresses some general limitations in the design of the usability defect classification taxonomy and new usability defect report form. The chapter ends with discussing directions for possible future work.

8.1 Research Summary and Contributions

A goal of this thesis was to develop a structured usability defect report form by introducing a set of attributes adapted from usability engineering studies, so that it can assist open source communities with limited usability knowledge to better report meaningful usability defect descriptions. The research also addressed the challenges identified in Section 1.2 about the inappropriateness of current defect report forms to report usability defects, and the insufficient attributes and terminology to describe usability defects.

The research was carried out in three stages. The first was a review of the literature on usability defect reporting. Secondly, surveys of software development practitioners and software defect repositories mining were conducted, which identified the desirable features to the new usability defect report forms. Finally, a new usability defect taxonomy and new usability defect report forms were designed and evaluated. A summary of the research, key findings and contributions found at each stage has been provided in the following sections.

8.1.1 A Review of Usability Defect Reporting

The aim of the literature review was to understand the *state of the art* in usability defect reporting from both software engineering and usability engineering studies. The review consisted of three parts. The first part reviewed the mechanisms to track and manage usability defects, format and content, and guidelines to assist usability defect reporting process. The second part investigated the use of usability defect data for improving software defect quality and management. The third part analyzed challenges in existing usability defect reporting processes and tools.

In managing usability defects, we identified three types of mechanisms used to report these defects, namely tool-based reporting, end-user reporting, and modeling-based reporting. Tool-based reporting is the most widely stated mechanism in the literatures due to its features that can facilitate data collection and process feedback instantly. Typically, usability defect reports are presented in written documents with a wide variety of attributes. Among the thirteen written document formats we identified, structured web-based forms are the most discussed type of defect report format in the literature. However, reporting a usability defect using this format can be impacted by information overload, lack of checking of input, and bias due to form content. Conventional reports, on the other hand, offer richer data but often provide heavy-weight documentation. Regardless of the format used, the problem description, severity, context, and redesign description were the top four attributes commonly used to describe usability defects. While a few guidelines were available to assist reporters to better describe usability defects, Capra's guideline is the most common guideline used by researchers to write usability defect descriptions and to measure usability defect report quality.

In previous empirical studies, many have shown significant interest in using data from defect reports to improve software quality. Based on the commonalities of these studies, we identified five areas that studied usability defect data: (1) measuring usability defect report quality, (2) understanding usability defect characteristics, (3) identifying duplicate usability defect report, (4) estimating effort, and (5) discussing usability defect resolution. The most commonly used defect attributes across these research areas were *problem description*, *impact*, and *title/ summary*. Attributes rarely used were *type of defect, likely difficulty, confidence, priority, software context, reporter, violated heuristic, business goals, assignee, milestone, time to fix*, and *defect fixes*. Research on classification and defect duplication favourably used *title/summary* and *description*, while research on defect report quality often used *observable user actions, impact*, *cause of the problem*, and *supplementary information*.

The literature review also revealed some challenges in reporting usability defects. Most of the challenges were due to limited support of current defect reporting tools to report, track, and manage usability defects. A generic defect report form, on the other hand, caused confusion to some defect reporters in reporting the information needed by the software developers. In fact, the differences between defect reporter and software developer points of view have resulted in information mismatch and impacted the prioritization of usability defect severity. Other challenges were related to limited skill and experience of reporters in completely describing usability defects, inappropriate testing techniques to collect necessary usability defect information, and lack of specific guidelines to define specific information that should be reported.

This SLR study contributes in two ways. First, it highlights current practices, key open issues and limitations with respect to usability defect reporting and serves as a guide to future research. Second, evidence from the review helps open source communities to understand which information is important when describing usability defects.

The detailed findings of this SLR - “Reporting Usability Defects: A Systematic Literature Review,” have been published in IEEE Transactions of Software Engineering [114].

8.1.2 Development of New Usability Defect Report Forms

The development of new usability defect report forms was divided into two steps. Step one focused on collecting desirable features of the usability defect forms through web surveys and software defect repositories mining. Step two involved the development of open source usability defect classification taxonomy adapted from UPT. The following subsections summarize the approach taken during each step, present the findings, and outline their contributions.

8.1.2.1 Collecting Inputs for Designing New Usability Defect Report Forms

To assist in developing new usability defect report forms, two studies were conducted. In the first study, we surveyed software development practitioners in both open source communities and industrial software organizations. The aim of the survey was twofold. First, the survey was conducted to identify information frequently supplied by defect reporters to report usability defects, and information needed by defect reporters to fix usability defects.

Our analysis of 147 responses showed a substantial gap between the information provided by defect reporters and the information needed by software developers to fix usability defects. When describing usability defects, defect reporters often supplied *actual output, expected output,* and *steps to reproduce,* while usability-related information such as violated heuristics, design principles, and impact were rarely provided. While software developers considered *cause of the problem* as the most helpful information for them to fix usability defects, defect reporters ranked *cause of the problem* as the most difficult information to provide followed by *usability principle, video recording, UI event trace,* and *title*.

Secondly, the survey also investigated the five factors that we speculated have influence on the quality of usability defect reports – the role of the reports, reporters' knowledge and experience, defect discovery methods, and usefulness of automation tools. We found usability defects reported by customers typically got reviewed and fixed faster. The detailed findings of this survey were published in three separate papers:

- "*Reporting Usability Defects: Limitations of Open Source Defect Repositories and Suggestions for Improvement*" published in the Proceedings of the 24th Australasian Software Engineering Conference (ASWEC 2015) [17];
- *"Reporting Usability Defects – Do Reporters Report What Software Developers Need?"* published in the Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering (EASE 2016) [115];
- *"What Influences Usability Defect Reporting? – A Survey of Software of Development Practitioners"* published in the Proceedings of the 23rd Asia Pacific Software Engineering Conference (APSEC 2016) [116].

The second study focused on software defect repositories mining. In this study, we mined 377 developer-tagged usability defect reports from Mozilla Thunderbird, Firefox for Android, and Eclipse Platform to confirm the findings of previous survey on the information provided by software development practitioners when reporting usability defects.

Our findings demonstrate a mismatch between what software development practitioners claimed to provide when reporting usability defects, and the information that appears in the defect reports. For *steps to reproduce*, for example, while we found this information less reported in Mozilla Thunderbird, Firefox for Android, and Eclipse Platform, software practitioners in our previous survey claimed that they always provided this information while reporting usability defects. A paper describing this work was published:

- *"Analysis of the Textual Content of Mined Open Source Usability Defect Reports"* published in the Proceedings of the 24th Asia Pacific Software Engineering Conference (APSEC 2017) [117].

The findings from these two studies have important implications for future research and practice of OSS development. Through these studies, researchers can find characteristics, open issues, and understand the nature of describing usability defects, which can be valuable for improving defect reporting processes and tools. For practitioners, especially software testers, it provides a general guide about the important defect attributes that should be described when reporting usability defects in order to receive faster defect resolution time.

8.1.3 Revising Usability Defect Classification Model

Based on the review of existing usability defect classification models and our experience of analyzing 377 open source usability defects, we found some limitations in these models. First, the lack of contextual information in open source usability defect descriptions lead to misinterpretation of the problem, in which the defect report readers had made assumptions and self-judgment. Second, the absence of formal usability evaluation methods in OSS development makes it difficult to understand the problematic user tasks, particularly in assessing the defect severity and impact to the users. In fact, it is quite impossible to justify what triggered the problem as suggested in the usability-ODC framework [95]. Third, well-known usability classification models such as UPT [12], UAF [3] and model-based framework [94] are too complex for open source communities with limited usability knowledge. Low involvement of usability experts in OSS development makes it impossible to adopt such a comprehensive model.

Considering these limitations, we revised the UPT [12], GUI fault model [76], and usability-ODC framework [95] to suit the needs of the OSS development environment. The aim of our new OSUDC model is to understand and classify usability defects based on the content of defect reports submitted by open source communities, which do not formally conduct usability evaluations. A total of 377 developer-tagged usability defect descriptions were analyzed from Mozilla Thunderbird, Firefox for Android, and Eclipse Platform projects. The analysis did not impact OSUDC development directly, however it was used to construct the subcategories of the primary categories adapted from [12], [76], [95]. The defect descriptions were read and organized according to the primary categories, and were then re-examined and grouped according to commonalities to form a set of subcategories. Two primary categories and five subcategories were added in our revised OSUDC, as the usability defect descriptions could not be classified according to the original classification. The resulted hierarchical structure of OSUDC contains three components (defect, user difficulty, and failure qualifier), 10 primary categories, and 28 subcategories, respectively.

A web based survey evaluation was conducted to assess the reliability of the revised OSUDC to classify usability defects. Twelve evaluators from industry and academia with varying degrees of experience in software development and usability participated in the study by classifying 10 usability defects that were randomly selected from the group of 377 usability defects. Kappa statistics was used to assess the level of agreement among evaluators. The findings of the OSUDC reliability study showed that the evaluators' agreement for the *defect category* component was the highest, and the lowest was for the *failure qualifier* component.

The primary contribution of our revised OSUDC is to enable software developers and defect reporters to understand, describe, and analyze usability defects. Each category in the revised OSUDC contains explanations and sample defects from real-world OSS projects. Usability defects that are classified using OSUDC could improve open source defect management, particularly to facilitate triaging processes identifying similar solutions for the same problems, suitable resources to fix the problems, and prioritizing defect correction. From the research perspective, this work contributes to the body of knowledge by incorporating a cause-effect relationship, which was not previously considered in the usability engineering studies.

8.1.4 Usability Defect Report Form Evaluation

Our proposed usability defect report forms contain four main criteria based on the findings of literature reviews, online surveys, and software defect repositories mining: 1) each attribute captures only one value, 2) one attribute is explained with multiple instances, 3) attributes are prompted when relevant to the reporter's information needs, and 4) user difficulties are measured using objective scales. These criteria were prototyped using Jotform, based on the current Bugzilla defect report layout.

The practical use of the proposed forms to capture meaningful usability defect descriptions was evaluated using an expert judgment approach. Our evaluation focused on comparing the quality of information between the original usability defect reports and the case study defect reports reproduced on our proposed forms. Using the guided defect report forms, reproduction of usability defects were constructed systematically and the multiple attribute instances were found sufficient to capture important information and giving hint to defect reporter; although not all attributes requested in the forms could be provided. The case studies also provide useful information on the usage of predefined usability defect categories in identifying usability defects that have common characteristics. The results from the evaluation suggest that by using multiple attribute instances to capture only one value per attribute could improve the quality of information presented in the usability defect descriptions.

8.2 Lessons Learned

In this section, we discuss practical challenges, experiences, and selected lessons we learnt in the five research studies that we conducted during this PhD.

8.2.1 Getting Ethics Approval for Conducting Online Surveys

In accordance to Swinburne research integrity, ethics approval must be sought for any research that involved human participants. During this PhD, we have applied two ethics approvals for two different studies. Getting the right strategy to write a good ethics application could speed up the approval process. For example, it is helpful to review a completed application that secured ethical approval from the Swinburne's Human Research Ethics Committee (SUHREC). In our experience, a well-planned execution of the surveys is a key determinant of a successful ethics application. Every method and instrument to be used must be specific, such as in selecting target population, recruitment strategy, location of study, and data privacy. Furthermore, it is important that every part of the document is aligned (e.g., aims, methods, instruments, etc) to make the document coherent.

8.2.2 Recruiting Survey Participants

In the first survey investigating information needs for reporting and fixing usability defects, we used coarse target population strategy to recruit potential participants. We posted requests for participation on the open source community forums, Facebook, LinkedIn, and Software Testing Club website. However, in our experience of using the coarse target population recruitment strategy, it produced a high percentage of invalid responses and therefore should be avoided in the future. In our first and second survey, more than around 50% of responses were excluded for no response beyond the first parts of the questionnaire.

One possible explanation of this problem was due to the *out of scope* factor, where the respondents are not in the target population. For example, the software development practitioners who do not have experience in dealing with usability defects may be reluctant to proceed with the surveys. As a lesson learnt, when investigating usability defect reporting practices by specific professional groups, such as open source communities, we should only use community-specific mailing lists and focus on the UX team, rather than approach the whole OSS community, some of whom might not be familiar with usability.

In the second survey evaluating our new OSUDC taxonomy, instead of focusing on a coarse target population, we used the researcher's referral contacts to recruit participants. We selected participants with a known software development background and sent specific invitation emails to them. In our experience, we found that industrial participation was low as compared to that of academic researchers. One potential reason behind this low participation could be that industrial professionals were too busy with their essential task and participating in the survey would add extra work. On the other hand, we believe academic researchers are more highly motivated to participate in the surveys because of their understanding of the pressure and intensity of other researchers to investigate and publish findings promptly.

During both surveys, we noticed three significant factors as a useful means to gain more participation. The first was sending reminder messages within a certain period of time. In our survey, we only sent a one time reminder because sending too many reminders may annoy some potential participants, thereby making them less likely to respond. The second significant factor was forwarding invitations from researchers' industrial contacts. Even though this method is likely to get participation beyond the target population [118], in our case we only asked our industrial colleagues to extend the invitations to individuals that they felt were suitable for this research context rather than forwarding to the company mailing list. The third useful factor to gain more participation was to offer incentives. Other empirical studies have shown that incentives have a great influence on response rate and quality, especially for a lengthy survey like our surveys [119]–[121]. However, with limited funding, we could only conduct a voluntary survey. In future work, we might consider offering rewards either in the form of raffles, payments, or sharing results to motivate less motivated participants.

8.2.3 Designing Survey Questionnaires

One of the crucial parts of conducting a survey is to get the right questions to measure the constructs that are investigated through the survey. We found the systematic literature review that has been conducted earlier was helpful in constructing simple questions that referred to facts and in choosing the appropriate words for the survey. Apart from that, a pilot study, in our opinion, is a good practice to obtain early feedback before the survey goes live. In the first survey, we obtained feedback and suggestions from fifteen software developers that was very helpful to improve the content and structure of the survey. The identified irrelevant, duplicate, and unclear questions were modified to ensure the consistency of the survey sections.

However, we believe that the lengthy survey is one possible reason for a high invalid participation in both of our surveys. In the first survey, for example, a closer inspection of data showed that about 17% of participants left the survey after answering the first section. Possibly the use of a survey progress bar indicator was not helpful to motivate participants to finish the survey. Since the survey had a total of 50 questions, in the beginning of first section the progress bar indicator certainly still showed significant remaining percentage to be completed, though this percentage changed progressively as not all questions needed to be answered.

In the second survey, even though the questions were straightforward, it required significant effort from participants to read the usability defect descriptions before they could answer the questions. We believe that participants were tending to avoid such a survey that would take too much effort from their daily work, and if they were willing to participate, the validity of their answers to some extent is dubious. One way to deal with this problem is via technical mechanisms to monitor the time taken by the participants to answer each question, or at least the time spent on each page.

8.2.4 Mining Textual Usability Defect Descriptions

Most usability defect reports in OSS defect repositories are semi-structured text. While text mining researchers have developed a wide range of algorithms and tools to deal with natural language texts, the algorithms rely somewhat on the quality of the documents itself, which we have found is often very variable in open source usability defect reports. For this reason, we conducted a fully qualitative analysis of the natural language data to create a full human interpretation. We created a simple coding of the textual data, and attached the code to a piece of the text that can be either single word, or a sentence. In our experience, the best way to create the codes to classify utterances in the defect descriptions is to find anchor studies that had similar research aims, and brainstorm some potential codes that could be relevant to the research context. Since the process of tagging part of text, and grouping of words and phrases requires a global understanding of defect report concepts and structures, we iteratively refined a set of codes that have been defined earlier until these codes established.

8.2.5 Analysis of Data

Our surveys include both quantitative and qualitative analysis. For quantitative analysis, in our experience we found it was useful to clearly define survey research questions, which we then mapped with survey questions and identified potential statistical tests to analyze the collected data. In this way, we could avoid unnecessary questions, and select the most suitable response format for the data analysis. This is because questions with inappropriate response formats may hinder the application of relevant statistical tests, or the response cannot be analyzed and interpreted in a meaningful manner. Qualitative analysis was mainly used as support evidence for the quantitative results.

Further, it is important to seek advice from professional statisticians. In our experience, even though we used correct statistical tests, the use of some statistic values was inappropriate. For example, when the Chi-Square test is performed simultaneously for multiple variables, Bonferroni correction must be used to avoid the influence of a spurious positive. Additionally, it is necessary to explain the statistical findings in simple terms to non-technical readers. Reporting *p*-value, and *F*-value, for example, while convenient for statisticians, is less important when presenting findings to other non-statistician readers. Therefore, it is better to explain easy-to-understand results than statistically valid, yet complex, results.

8.3 Limitations

8.3.1 Diverse Respondents Background

Our investigations on the nature of reporting and fixing usability defects were based on input from respondents that had varying levels of usability background and commitment. For example, software developers and software testers working on closed proprietary software may have better exposure to writing effective usability defect reports than open source contributors who work on a voluntary basis, and may report usability-related defects without formal training [122]. This may have introduced some bias to our findings.

8.3.2 Manual Classification of Usability Defect Information

Even though some computational programs and tools were available to automatically analyze textual data in a much shorter time, we manually read through all the 377 usability defect descriptions to examine the presence of certain pieces of information. We realized that within a limited research timeframe, to understand and develop the text mining algorithms that were suitable to our research context was quite impossible. Indeed, applying analytical algorithms would require a significant effort to understand the usage of each algorithm and the combination of selected techniques for handling text. Furthermore, the text mining process to automatically classify usability defects was not really in the scope of this work.

8.3.3 Limited Evaluation Aspects of the Proposed Usability Defect Report Forms

In this work, we only evaluated the quality of the information aspect produced by the proposed usability defect report forms. Our evaluation compared the level of details of usability defect report content written using our proposed forms with original reports that used default Bugzilla templates. However, outcomes were only based on the judgment of experts that were not directly using this report in their daily work. Other aspects, such as ease of use of the forms to assist software reporters to report useful usability defect descriptions, and the usefulness of the information produced by the proposed forms to help software developers to understand and fix the problems, are still to be investigated further in our future work.

8.4 Future Research

Several possible directions for future investigation have been identified as a consequence of our study. Some of these are to overcome the current limitations of this study, and to integrate with current trends in OSS development and usability studies; these are discussed in the following sections. Sections 8.4.1 to 8.4.4 present the short-term future work, while sections 8.4.5 to 8.4.7 discuss long-term future work.

8.4.1 Replicating the Survey for Different Focus Groups

In software engineering research, replication and reproduction experiments, case studies, and surveys have been recognized for creating and extending scientific knowledge [123]–[126]. Since the main target population of our previous surveys were individuals who have experience in reporting or fixing usability defects, we were interested in replicating and extending the surveys to the organizational level. For example, we could compare the usability defect reporting practices in organizations with usability teams, organizations without usability teams, and OSS development communities. We believe that different organizations may have different strategies and policies to enforce the implementation of HCI activities, which would eventually influence the information needs of usability designers, software development practitioners, and development management. This also raises the following new research question: How do different organizational backgrounds influence usability defect reporting practices?

8.4.2 Additional Reliability Studies to Assess Our OSUDC Taxonomy

An additional reliability study is planned to assess the level of agreement achieved in OSUDC categories, especially on the subcategories level. Since the reliability study reported in Section 6.5.2 mainly involved non-usability expert respondents, a different target population is needed to gain feedback from various respondents with different levels of expertise. This is because our aim of revising existing usability classification [12], [76], [95] was to produce a simple and easy-to-use model that can be used by OSS users with different technical backgrounds. Possible approaches include enlarging the number of respondents with different backgrounds, the number of usability defects, or both.

8.4.3 Automated Text Mining Process

Text mining is a useful method for extracting key trends, such as word usage, or vocabulary in a textual document. Using appropriate analytical algorithms, many text documents can be automatically converted into a structured and numerical format that can reveal more meaningful information. Although this thesis has demonstrated the classification of the usability defect descriptions to a set of predefined categories, such classification processes were carried out manually. A potential drawback of dealing with manual classification is that the analysis and interpretation of defect descriptions is greatly dependent on the research team involved. Therefore, further research is needed to understand different text mining practices areas – possibly in the area of concept extraction to group certain words and phrases into semantically similar groups. By using machine-learning tools like the Weka[5] tool and RapidMiner, it is hoped that the glossary of terms for each category can be generated successfully.

In addition, using the text-mining method, it would be interesting to explore how usability defects were understood in the comments section of reports, which is outside the scope of this study. In this way, clarification on how the usability defects should have been reported, and contrasting this with the way in which it was reported can be made. The results from this kind of study can be used to improve the OSUDC taxonomy proposed. For example a "how not to report defects" model can be developed with the associated terms for specific kind of usability issues.

8.4.4 Additional Evaluation of Our Proposed Usability Defect Report Form

As described in this thesis, the usability defect forms we are proposing have so far only been evaluated in terms of the quality of information, which has been assessed by software engineering experts. Other aspects such as the ease of use of the forms in reporting usability defects, workload imposed on reporters, and the usefulness of the information on software developers, are still open for investigation. In future work, we plan to assess to what extent the proposed forms were successful in eliciting the information needed by software developers. In this study, we would like to compare the usability defect descriptions produced by practitioners and students (represented for non-technical users), and the workload imposed on them when reporting the usability defects. Once the proposed forms are in a stable stage, we also plan to validate our proposed forms by requesting IT organizations to use the forms to report usability defects. The reported defects will then go to the defect management lifecycle, and we will monitor the defect resolution status. In this way, we could confirm the effectiveness of our proposed forms to speed up usability defect resolution.

While feedback on the proposed usability defect report forms was generally positive, this study primarily consulted software engineering experts who were not directly involved with software development. Other stakeholders in the software development process, such as software developers, managers, usability designers, and customer support, may have different feedback about the

[5] http://www.cs.waikato.ac.nz/ml/weka/downloading.html

information that is important to them. Thus, further evaluation may need to be carried out for different stakeholders as well.

8.4.5 Defect Reporter Effect in Software Defect Reporting

From our survey of software development practitioners and review of previous literature, we found that the defect reporters play a significant role in determining which usability defects get fixed. We also found that relationships and trust built between software developers and known-defect reporters influence the defect resolution. This shows some indication that *defect reporter effect* might be influential to the quality of defect report descriptions. Therefore, we plan to investigate the effect of defect reporter expertise, knowledge of the application domain, and familiarity with the defect reporting tools on software defect reporting, which we believe has not been investigated by any other researchers, with more specific research study. In the proposed research study, groups of defect reporters with different backgrounds will be asked to watch a video of representative users performing certain tasks. Then the defect reporters will be required to report each usability defect they discovered in the video using our proposed forms. The quality of defect descriptions of each group will be assessed and compared.

In addition, we will collect the personality profile (big five personality traits) of the defect reporters [127], and will analyze the personality profiles and quality of defect descriptions to find association, if there is any.

8.4.6 Modeling Defect Prediction Using Textual Defect Information

We found that textual content of usability defect descriptions can provide an early insight into the defect resolution time. Compared with other types of defects, usability defects require detailed explanations to clarify the validity of issues. Typically, the nature and complexity of the issues is often described in the defect description. For instance, the likely difficulty, emotion, and feelings can only be expressed in the text-form. However, most of the existing research in predicting defect resolution time is involved with using categorical data such as severity, priority, version, component, and attachment supplied [128]–[132]. This is because textual analysis is more complex to analyze, especially as the number of defect reports grows, more time is required. Therefore, research challenges following from our results are: “How can textual content of defect descriptions automatically be classified into certain components?" and "Which textual components should be used to predict usability defect resolution time?"

8.4.7 Multilingual Patterns of Describing Usability Defects

In global software development, language is challenging in many software development activities, including software defect reporting. Besides the different use of vocabulary to explain technical subject, interpretation and meaning of defect descriptions written in various languages can be different. For example, in Malaysia code switching between Malay and English has become common practice in requirements elicitation [133], and it is also possible when expressing software defects. Motivated by this problem, we plan to investigate the nature of how usability defects are described in non-English based IT companies. Through this kind of research, we could develop a Malay-usability glossary of terms for better understanding of usability defects.

8.5 Final Remarks

This research progressed from the need to have an appropriate defect report form for capturing different types of defects; in this study, we focused on usability defects. The importance of capturing complete and meaningful defect information for fixing software defects was evident in both the software engineering and usability engineering domains. However, most research in usability defect reporting was conducted in usability engineering studies where the main concern was to find the most effective methods of presenting usability evaluation results. This left a gap in finding the right defect report content for reporting usability defects that were found outside of formal usability evaluations, especially for users with limited usability knowledge.

While a variety of usability defect classification models exist in usability engineering that are widely accepted by industry and academia, these classification models were not appropriate for use in OSS development, in which the defect reports contained limited information and the development stage often ignored the usability aspects [134]. However, some features in these classification models could be extended to overcome these drawbacks. This was one of the focuses of this research: to revise the usability problem taxonomy for classifying OSS usability defects using a 'lightweight' classification mechanism. This research has not only filled the existing gap but has also contributed to an increased understanding of how open source defect classification models can be designed using software engineering and usability engineering literatures and concepts, respectively.

To this end, this thesis extends current open source defect report forms to capture information specific to usability defects. Our proposed usability defect report forms can be adopted by organizations with specific usability teams, in which the process of handing over findings from usability evaluations to software development teams could be simplified. Usability evaluation findings written in our proposed forms provide “just enough” documentation and do not require the summarization of comprehensive documents to be used directly by the software developers. This is an important issue in software development as fast feedback cycles are essential.

Abbreviations

ANOVA	Analysis of Variance
CUP	Classification of Usability Problem
DCART	Data Collection, Analysis, and Reporting Tool
GUI	Graphical User Interface
HCI	Human Computer Interaction
HP-DCS	Hewlett Packard Defect Classification Scheme
IRC	Internet Relay Chat
ODC	Orthogonal defect Classification
OSS	Open Source Software
OSUDC	Open Source Usability Defect Classification
RCA	Root Cause Analysis
SLR	Systematic Literature Review
UAF	Usability Action Framework
UPT	Usability Problem Taxonomy
URM	Usability Reporting Manager
UX	User Experience

Appendix A

List of Included Paper in the Systematic Literature Review

The references listed below correspond to these prefaced with the letter "P" throughout the Chapter 3:

[1] Bruun, A., & Stage, J. (2014). Barefoot usability evaluations. *Behaviour & Information Technology*, *33*(11), 1148–1167. Doi:10.1080/0144929X.2014.883552

[2] Dumas, B. J. S., Molich, B. R., & Jeffries, B. R. (2004). Describing usability problems: Are we sending the right message? *Interactions*, 0–4.

[3] Hornbaek, K., & Frokjaer, E. (2006). What Kinds of Usability-Problem Description are Useful to Developers? In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* (Vol. 50, pp. 2523–2527). Doi:10.1177/154193120605002402

[4] Howarth, J., & Hall, M. (2006). Identifying Immediate Intention during Usability Evaluation. In *Proceedings of the 44th Annual Southeast regional conference* (pp. 274–279).

[5] Capra, M. G. (2007). Comparing Usability Problem Identification and Description by Practitioners and Students. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* (pp. 474–477).

[6] Bruun, A., Gull, P., Hofmeister, L., & Stage, J. (2009). Let Your Users Do the Testing : A Comparison of Three Remote Asynchronous Usability Testing Methods. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1619–1628).

[7] Howarth, J., Smith-jackson, T., & Hartson, R. (2009). Supporting novice usability practitioners with usability engineering tools. *International Journal Human-Computer Studies*, *67*(6), 533–549. Doi:10.1016/j.ijhcs.2009.02.003

[8] Skov, M. B., & Stage, J. (2009). Training software developers and designers to conduct usability evaluations. *Behaviour & Information Technology*, *31*(4), 425–435. Doi:10.1080/01449290903398208

[9] Høegh, R. T., & Stage, J. (2006). The Impact of Usability Reports and User Test Observations on Developers ' Understanding of Usability Data : An Exploratory Study. *INTERNATIONAL JOURNAL OF HUMAN–COMPUTER INTERACTION*, *21*(2), 173–196.

[10] Hvannberg, E. T., Law, E. L.-C., & Lárusdóttir, M. K. (2007). Heuristic evaluation: Comparing ways of finding and reporting usability problems. *Interacting with Computers*, *19*(2), 225–240. Doi:10.1016/j.intcom.2006.10.001

[11] Morse, E. L. (2000). The IUSR project and the common industry reporting format. In *Proceedings on the 2000 conference on Universal Usability – CUU '00* (pp. 155–156). New York, New York, USA: ACM Press. Doi:10.1145/355460.355556

[12] Vermeerena, A. P. O. S., Attemaa, J., PhDa, E. A., Riddera, H. de, Doorna, A. J. von, Erbuğb, Ç., … Maguirec, M. C. (2008). Usability problem reports for comparative studies : consistency and inspectability. *Human–Computer Interaction*, *23*(4), 329–380.

[13] Hornbæk, K., & Frøkjær, E. (2008a). Comparison of techniques for matching of usability problem descriptions. *Interacting with Computers*, *20*(6), 505–514. Doi:10.1016/j.intcom.2008.08.005

[14] Skov, M. B., & Stage, J. (2005). Supporting problem identification in usability evaluations. In *Proceedings of OZCHI'05, the CHISIG Annual Conference on Human-Computer Interaction* (pp. 1–9). Retrieved from http://portal.acm.org/citation.cfm?id=1108368.1108410

[15] 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*(3), 455–482. Doi:10.1518/hfes.45.3.455.27255

[16] Cockton, G., Woolrych, A., Hall, L., & Hindmarch, M. (2003). Changing analysts' tunes: The surprising impact of a new instrument for usability inspection method assessment. *Proceedings of HCI 2003 on People and Computers XVII*, 145–162. Retrieved from http://www.cet.sunderland.ac.uk/~cs0awo/hci 2003 full.pdf

[17] Feiner, J., & Andrews, K. (2012). Usability Reporting with UsabML. *Proceedings of the 4th International Conference on Human-Centered Software Engineering*, *7623*, 342–351.

[18] Heller, F. (2011). Me Hates This : Exploring Different Levels of User Feedback for (Usability) Bug Reporting. In *Extended Abstracts on Human Factors in Computing Systems* (pp. 1357–1362).

[19] Lewis, J. R. (2006). Effect of Level of Problem Description on Problem Discovery Rates: Two Case Studies. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* (pp. 2567–2571)

[20] Zhao, L., & Deek, F. P. (2005). Improving Open Source Software Usability. In *Proceeedings of the Eleventh Americas Conference on Information Systems* (pp. 923–928).

[21] Vetro, A., Zazworka, N., Seaman, C., & Shull, F. (2012). Using the ISO/IEC 9126 product quality model to classify defects: a Controlled Experiment. In *International Conferenece on Evaluation and Assessment in Software Engineering* (pp. 87–96).

[22] Lal, S., & Sureka, A. (2012). Comparison of Seven Bug Report Types: A Case-Study of Google Chrome Browser Project. In *2012 19th Asia-Pacific Software Engineering Conference* (pp. 517–526). Ieee. Doi:10.1109/APSEC.2012.54

[23] Twidale, M. B., & Nichols, D. M. (2005). Exploring Usability Discussions in Open Source Development. In *Proceedings of the 38th Annual Hawaii Internatioal Conference on System Sciences* (pp. 1–10).

[24] Martin, S. (2007). Enhancing the Downstream Utility of Usability Evaluations with Pattern-based Recommendations. In *COST294-MAUSE Workshop on Downstream Utility* (pp. 1–4).

[25] Botella, F., & Peñalver, A. (2013). A new proposal for improving heuristic evaluation reports performed by novice evaluators. In *Proceedings of the 2013 Chilean Conference on Human – Computer Interaction* (pp. 72–75).

[26] Ko, A. J., & Chilana, P. K. (2011). Design, Discussion, and Dissent in Open Bug Reports. In *Proceedings of the 2011 iConference* (pp. 106–113). Doi:10.1145/1940761.1940776

[27] Faaborg, A., & Schwartz, D. (2010). Using a Distributed Heuristic Evaluation to Improve the Usability of Open Source Software. In *CHI Conference on Human Factors in Computing Systems* (pp. 4–5).

[28] Bernonville, S., Kolski, C., & Leroy, N. (2010). First Experimentation of the ErgoPNets Method Using Dynamic Modeling to Communicate Usability Evaluation Results. *Human Error, Safety and Systems Development*, *5962*, 81–95.

[29] Molich, R., Jeffries, R., & Dumas, J. S. (2007). Making Usability Recommendations Useful and Usable. *Journal of Usability Studies*, *2*(4), 162–179.

[30] Nichols, D. M., Mckay, D., & Twidale, M. B. (2003). Participatory Usability : supporting proactive users. In *Proceedings of the 4th Annual Conference of the ACM Special Interest Group on Computer-Human Interaction* (pp. 63–68).

[31] Raza, a., Capretz, L. F., & Ahmed, F. (2012). Usability bugs in open-source software and online forums. *IET Software*, *6*(November 2011), 226. Doi:10.1049/iet-sen.2011.0105

[32] Douglas, I. (2006). Collaborative International Usability Testing : Moving from Document-based Reporting to Information Object Sharing. In *IEEE International Conference on Global Software Engineering* (pp. 0–4).

[33] 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:10.1016/j.jss.2013.08.014

[34] Beirekdar, A., Keita, M., Noirhomme, M., & Randolet, F. (2005). Flexible Reporting for Automated Usability and Accessibility Evaluation of Web Sites. *Human-Computer Interaction*, *3585*, 281–294.

[35] Hornbæk, K., & Frøkjær, E. (2008b). Making Use of Business Goals in Usability Evaluation : An Experiment with Novice Evaluators. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 903–912).

[36] Law, E. L., & Hvannberg, E. T. (2008). Consolidating Usability Problems with Novice Evaluators. In *Proceedings of the 5th Nordic conference on Human-computer interaction: building bridges* (pp. 495–498).

[37] Nørgaard, M., & Høegh, R. T. (2008). Evaluating Usability – Using Models of Argumentation to Improve Persuasiveness of Usability Feedback. In *Proceedings of the 7th ACM conference on Designing interactive systems* (pp. 212–221).

[38] Çetin, G., Verzulli, D., & Frings, S. (2007). An Analysis of Involvement of HCI Experts in Distributed Software Development: Practical Issues. *Online Communities and Social Computing*, *4564*, 32–40. Doi:10.1007/978-3-540-73257-0

[39] Äijö, R., & Mantere, J. (2001). Are non-expert usability evaluations valuable ? In *International Symposium on Human factors in Telecommunications* (pp. 1–5).

[40] Nichols, D. M., & Twidale, M. B. (2006). Usability processes in open source projects. *Software Process: Improvement and Practice*, *11*(2), 149–162. Doi:10.1002/spip.256

[41] Jonasson, G. F., & Hvannberg, E. T. (2009). Sharing Usability Problem Sets within and between Groups. *Human-Computer Interaction*, *5727*, 596–599.

[42] Li, Z., Tan, L., Wang, X., Lu, S., Zhou, Y., & Zhai, C. (2006). Have Things Changed Now ? – An Empirical Study of Bug Characteristics in Modern Open Source Software. In *Proceedings of the 1st workshop on Architectural and system*

support for improving software dependability (pp. 25–33).

[43] Zhao, L., Deek, F. P., & Mchugh, J. A. (2010). Exploratory inspection — a user-based learning method for improving open source software usability. *Journal of Software Maintenance and Evolution: Research and Practice*, *22*(8), 653–675. Doi:10.1002/smr

[44] N∅rgaard, M., & Hornbæk, K. (2009). Exploring the Value of Usability Feedback Formats. *International Journal of Human-Computer Interaction*. Doi:10.1080/10447310802546708

[45] S. Herbold, J. Grabowski, S. Waack, and U. Bünting, "Improved Bug Reporting and Reproduction through Non-intrusive GUI Usage Monitoring and Automated Replaying," in *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, 2011, pp. 232–241.

[46] Kreyss, J., White, M., Selvaggio, S., & Zakharian, Z. (2003). Text Mining for a Clear Picture of Defect Reports : A Praxis Report. In *Third IEEE International Conference on Data Mining* (pp. 3–6).

[47] F. P. Simões, "Supporting End User Reporting of HCI Issues in Open Source Software,". PhD Thesis. Pontificia Universidade Catolica, Do Rio De Janeiro, 2013.

[48] Ko, A. J., Myers, B. A., & Chau, D. H. (2006). A Linguistic Analysis of How People Describe Software Problems. In *Visual Languages and Human-Centric Computing (VL/HCC'06)* (pp. 127–134). Ieee. Doi:10.1109/VLHCC.2006.3

[49] Pichler, J., & Ramler, R. (2008). How to test the intangible properties of graphical user interfaces? In *Proceedings of the 1st International Conference on Software Testing, Verification and Validation, ICST 2008* (pp. 494–497). Doi:10.1109/ICST.2008.52

[50] Xia, X., Zhou, X., Lo, D., & Zhao, X. (2013). An Empirical Study of Bugs in Software Build Systems. In *13th International Conference on Quality Software* (pp. 200–203). Ieee. Doi:10.1109/QSIC.2013.60

[51] T. Roehm, N. Gurbanova, B. Bruegge, C. Joubert, and W. Maalej, "Monitoring user interactions for supporting failure reproduction," in *2013 21st International Conference on Program Comprehension (ICPC)*, 2013, pp. 73–82.

[52] Avnon, Y., & Boggan, S. L. (2010). Fit and Finish using a bug tracking system: challenges and recommendations. In *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems* (pp. 4717–4720). Doi:10.1145/1753846.1754219

[53] K. Hornbæk and E. Frokjær, "Comparing usability problems and redesign proposals as input to practical systems development," *CHI 2005 Technol. Safety, Community Conf. Proc. – Conf. Hum. Factors Comput. Syst.*, pp. 391–400, 2005.

[54] C. Wilson and K. P. Coyne, "The whiteboard: Tracking usability issues: to bug or not to bug?," *Interactions*, pp. 15–19, 2001.

[55] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, and D. Poshyvanyk, "Auto-completing Bug Reports for Android Applications," *Proc. 2015 10th Jt. Meet. Found. Softw. Eng.*, pp. 673–686, 2015.

[56] A. Hindle, A. Alipour, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection and ranking," *Empir. Softw. Eng.*, pp. 368–409, 2015.

[57] N. Shahida, M. Yusop, J. Grundy, and R. Vasa, "Reporting Usability Defects : Limitations of Open Source Defect Repositories and Suggestions for Improvement," in *Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference*, 2015, pp. 38–43.

Appendix B

Survey Questionnaire

Background Information

1. What is your gender?

□ Male □ Female

2. What is your age?

□ Less than 24 years old □ 25 - 34 years old □ 35 - 44 years old

□ 45 - 54 years old □ 55 years old and above

3. What is your professional position?

□ Software developer □ Software tester □ Quality assurance engineer

□ Customer consultant/ support □ System engineer □ Test manager

□ Project manager □ Usability engineer □ User interface designer

□ Other

If you have chosen "other", please specify:

4. How long have you been in your current position?

□ Less than 1 year □ Between 1 and 3 years

□ Between 3 and 5 years □ More than 5 years

5. Do you contribute to open source project such as Firefox, Google Chrome, GNOME?

□ Yes □ No

6. How do you describe your role in dealing with usability defects?

□ I am fixing usability defects □ I am reporting usability defects

Training or Certification on Human-Computer Interaction

7. Have you received any training/ certification related to usability evaluation/ Human Computer Interaction (HCI)

□ Yes □ No

Note: If you have answered/ chosen item [2] in question 7, skip the following question

8. How useful was the training/ certification for reporting and/ or understanding usability defects?

□ Very useful □ Somewhat useful □ Neither useful or not useful

□ Not very useful □ Complete waste of time

Discovering Usability Defects

Note: If you have answered/ chosen item [1] in question 6, skip the following question

9. Do you have experience in software testing (no matter what is your current position)?

□ No experience □ Less than 1 year □ Between 1 and 3 years

□ Between 3 and 5 years □ More than 5 years

10. How do you discover the usability defects?

□ Exploratory testing □ Functional testing □ Usability testing

□ Beta/ alpha testing □ Complaints/ reports from customers □ Using the product

□ Other

If you have chosen "other", please specify:

11. The amount of information available for reporting usability defects varies according to how the defects are discovered. For instance, usability defects found during usability testing have more information to report in comparison to functional testing. Would you agree or disagree on the following statement in relation to the availability of usability defect information for different defect discovery methods?

Statement	**Completely disagree**	**Somewhat disagree**	**Neither disagree or agree**	**Somewhat agree**	**Completely agree**
The difficulties the actual user will encounter as a consequence of the usability problems cannot be known during functional testing	□	□	□	□	□
In exploratory testing, a usability defect is typically grounded in general usability knowledge, rather than in speculation on users task performance and response	□	□	□	□	□

The presence of actual users during usability testing can reveal user's knowledge, likely difficulties, actual task scenario and realistic redesign solutions	□	□	□	□	□
In usability testing, more usability-related information can be reported, such as violated heuristics, design principles, users response and feelings as compared to system testing	□	□	□	□	□
Describing complaints from customers or end users may not correctly describe the actual difficulties faced by them	□	□	□	□	□

12. Do you think defect reporting tool should provide custom forms for reporting defects depending on how usability defects are discovered?

□ 1 (Strongly disagree) □ 2 □ 3 □ 3 □ 5 (Strongly agree)

Reporting Usability Defects

Note: If you have answered/ chosen item [1] in question 6, skip the following question

13. Do you use the following items when describing usability defects?

□ 1 (Strongly disagree) □ 2 □ 3 □ 3 □ 5 (Strongly agree)

Items	**Never**	**Rarely**	**Sometimes**	**Often**	**Always**
Title/ Summary	□	□	□	□	□
Cause of the problem	□	□	□	□	□
Software context (part of the user interface,	□	□	□	□	□

interface component)					
Proposed solution	□	□	□	□	□
Observed result	□	□	□	□	□
Expected result	□	□	□	□	□
Steps to reproduce	□	□	□	□	□
Severity	□	□	□	□	□
Software information (product, component, and/ or version)	□	□	□	□	□
Test environment (Operating system, hardware, browser details)	□	□	□	□	□

Reporting Usability Defects

Note: If you have answered/ chosen at least one of the following items: [(2,2)] in question 13, skip the following question

Note: If you have answered/ chosen item [1] in question 6, skip the following question

14. When specifying the defect report title/ summary, I do include

□ explanation on the situation that was happening at the time the problem occured
□ build or version of the software or OS on which the problem occured
□ an error message that come up by copying and pasting the whole thing in
□ quality attributes affected (i.e., usability, performance, reliability)
□ Other
□ If you have chosen "other", please specify:

Note: If you have answered/ chosen at least one of the following items: [(2,3)] in question 13, skip the following question

Note: If you have answered/ chosen item [1] in question 6, skip the following question

15. When describing causes of usability defects, I

□ include heuristics that are violated
□ describe a design fault, such as how the interaction architecture contributed to the problem
□ describe my knowledge of performing and understanding a task or object interface

□ Other

□ If you have chosen “other”, please specify:

16. I support the description of usability defect causes with

□ UI event trace □ Screenshot with annotations □ Screenshot and accompanying text

□ Video with captions □ Textual description □ Other

If you have chosen “other”, please specify:

Note: If you have answered/ chosen at least one of the following items: [(2,4)] in question 13, skip the following question

Note: If you have answered/ chosen item [1] in question 6, skip the following question

17. To indicate the context of the usability defects, I

□ describe the location of the problem in the interface, such as screen title

□ mention the problematic user interface object, such as button, menu and dialogue box

□ describe the user’s task

□ describe exactly which system components are affected

□ Other

□ If you have chosen “other”, please specify:

18. I show the location of usability defects in the interface using

□ screenshot with annotations □ video with captions □ textual description

□ Other

If you have chosen “other”, please specify:

Note: If you have answered/ chosen at least one of the following items: [(2,5)] in question 13, skip the following question

Note: If you have answered/ chosen item [1] in question 6, skip the following question

19. In describing a proposed solution to a usability defect, I

□ include several alternate solutions

□ describe advantages and disadvantages for alternative solutions

□ mention usability design principles and/ or previous research

□ use my knowledge to interpret how design is supposed to work

□ propose the expected behavior (defects)

□ propose the behavior desired (enhancements)

□ Other

□ If you have chosen “other”, please specify:

20. I support the description of the proposed solutions with

□ fix patch

□ simple sketches

□ oral presentation

□ digital mock-ups

□ ASCII art

□ Other

□ annotated screenshots

□ textual description only

If you have chosen “other”, please specify:

Note: If you have answered/ chosen at least one of the following items: [(2,7)] in question 13, skip the following question

Note: If you have answered/ chosen item [1] in question 6, skip the following question

21. To describe the expected result, I

□ refer to usability requirements

□ use my knowledge and experience to interpret the expected result

□ refer to usability design guideline

□ Other

□ If you have chosen “other”, please specify:

Note: If you have answered/ chosen at least one of the following items: [(2,6)] in question 13, skip the following question

Note: If you have answered/ chosen item [1] in question 6, skip the following question

22. To describe the observed usability defects, I

□ describe the effect of the user’s performance

□ justify what was wrong and why is it wrong

□ attach error message

□ describe the user’s behavior following the issues

□ attach screenshot with annotations to the issue

□ Other

If you have chosen “other”, please specify:

Note: If you have answered/ chosen at least one of the following items: [(2,8)] in question 13, skip the following question

Note: If you have answered/ chosen item [1] in question 6, skip the following question

23. To describe the steps to reproduce usability defects, I

□ write a textual description on the user's navigation flow through the system
□ record the steps that I follow and attach or paste a link to the video
□ Other
□ If you have chosen "other", please specify:

24. To rate the severity of usability defects, I justify

□ impact of the issue □ business effects, such as costs and time loss

□ how often the issue will occur during usage □ Other

If you have chosen "other", please specify:

Reporting Usability Defects

Note: If you have answered/ chosen item [1] in question 6, skip the following question

25. How do you report information about usability defects?

Medium of reporting	**Never**	**Rarely**	**Sometimes**	**Often**	**Always**
Traditional written report	□	□	□	□	□
Verbally in meeting with designers/ developers	□	□	□	□	□
Edited videos	□	□	□	□	□
Entry in a defect reporting tool	□	□	□	□	□

26. Do you think that experience in usability testing will help you create a better usability defect report?

□ 1 (Strongly disagree) □ 2 □ 3 □ 3 □ 5 (Strongly agree)

27. Which five items were the most difficult to provide? Please rank the items in order of difficulty from 1 to 5 where 1 is most difficult to you and 5 is least difficult to you.

Items	**1**	**2**	**3**	**4**	**5**
Title/ Summary	□	□	□	□	□

Cause of the problem	□	□	□	□	□
Software context (part of the user interface, interface component)	□	□	□	□	□
Usability principle/ heuristic violated	□	□	□	□	□
Proposed solution	□	□	□	□	□
Video recording	□	□	□	□	□
Audio recording	□	□	□	□	□
Product	□	□	□	□	□
Component	□	□	□	□	□
Version	□	□	□	□	□
Severity	□	□	□	□	□
Hardware	□	□	□	□	□
Operating system	□	□	□	□	□
UI event trace	□	□	□	□	□
Observed result	□	□	□	□	□
Expected result	□	□	□	□	□
Steps to reproduce	□	□	□	□	□
Screenshots	□	□	□	□	□

Fixing usability Defects

Note: If you have answered/ chosen item [2] in question 6, skip the following question

28. Do you use the following items when fixing usability defects?

Items	**Never**	**Rarely**	**Sometimes**	**Often**	**Always**
Title/ Summary	□	□	□	□	□
Cause of the problem	□	□	□	□	□
Software context (part of the user interface, interface component)	□	□	□	□	□
Usability principle/ heuristic violated	□	□	□	□	□
Observed result	□	□	□	□	□
Expected result	□	□	□	□	□
Steps to reproduce	□	□	□	□	□

29. Does the following attachments helps you in fixing usability defects?

Items	Never	Rarely	Sometimes	Often	Always
Video recording	□	□	□	□	□
Audio recording	□	□	□	□	□
UI event trace	□	□	□	□	□
Proposed solution	□	□	□	□	□
Screenshots	□	□	□	□	□
Proposed fix patch	□	□	□	□	□
Digital mockups	□	□	□	□	□
ASCII art	□	□	□	□	□

30. Do you request the following general information to assist with defect management and reproduction?

Items	Never	Rarely	Sometimes	Often	Always
Product	□	□	□	□	□
Component	□	□	□	□	□
Version	□	□	□	□	□
Hardware	□	□	□	□	□
Operating system	□	□	□	□	□
Severity	□	□	□	□	□

31. Which five items help you the most? Please rank the items in order of importance from 1 to 5 where 1 is most important to you and 5 is least important to you.

Items	1	2	3	4	5
Title/ Summary	□	□	□	□	□
Cause of the problem	□	□	□	□	□
Software context (part of the user interface, interface component)	□	□	□	□	□
Usability principle/ heuristic violated	□	□	□	□	□
Proposed solution	□	□	□	□	□
Video recording	□	□	□	□	□

Audio recording	□	□	□	□	□
Product	□	□	□	□	□
Component	□	□	□	□	□
Version	□	□	□	□	□
Severity	□	□	□	□	□
Hardware	□	□	□	□	□
Operating system	□	□	□	□	□
UI event trace	□	□	□	□	□
Observed result	□	□	□	□	□
Expected result	□	□	□	□	□
Steps to reproduce	□	□	□	□	□
Screenshots	□	□	□	□	□

32. Which of the following problems have you encountered when fixing usability defects? (Please select ALL that apply)

You were given unclear:	
□	Title/ summary
□	Cause of the problem
□	Software context (the location of problem in the interface)
□	Usability principle/ heuristic violated
□	Proposed solution
□	Screenshots
□	Audio recording
□	Video recording
You were given incorrect:	
□	Component
□	Observe result
□	Expected result
□	Product
□	Version
□	Severity
□	Hardware
□	Operating system
There was insufficient information in:	
□	Steps to reproduce
□	UI event trace
The reported used:	
□	Bas grammar
□	Unstructured text/ format
□	Vague comment
□	Too long text
□	Non-technical language
□	Usability jargon/ term
Others:	
□	Duplicate
□	Spam
□	Viruses/ worms

33. When assessing usability defects, I found

Statement	**Never**	**Rarely**	**Sometimes**	**Often**	**Always**
The usability defects reported by potential end-users with direct experience from the work domain were given higher priority	□	□	□	□	□
Usability expert provides better usability defect report compared to general software tester	□	□	□	□	□
General software tester rarely supplies usability-related information, such as violated heuristics or design principles	□	□	□	□	□
Developer provides more meaningful usability defect description with proposed fix patch and logs	□	□	□	□	□
Non technical users rarely provide important information such as software context, product information and reproducibility	□	□	□	□	□

34. Do you think a defect reporter propose better solutions when they have usability experience (UX), rather than relying on individual speculations?

□ 1 (Strongly disagree) □ 2 □ 3 □ 3 □ 5 (Strongly agree)

Defect Reporting Tool

Note: If you have answered/ chosen at least one of the following items: [(2,5)] in question 25, skip the following question

Note: If you have answered/ chosen at least one of the following items: [(3,5), (4,5), (6,5)] in question 25, skip the following question

Note: If you have answered/ chosen item [1] in question 6, skip the following question

35. Does your organization use defect reporting tool to manage usability defects?

□ Yes □ No

Defect Reporting Tool

Note: If you have answered/ chosen at least one of the following items: [(2,5)] in question 25, skip the following question

Note: If you have answered/ chosen item [2] in question 35, skip the following question

36. Name your defect reporting tool:

37. Does your defect reporting tool offer sufficient flexibility to capture and manage usability defects?

□ 1 (Strongly disagree) □ 2 □ 3 □ 3 □ 5 (Strongly agree)

38. Do you have any positive/ negative feedback about your defect reporting tool for supporting usability issues? Please specify here:

39. Do you think the user experience (UX) of the defect reporting tool influences the quality of defect reports?

□ 1 (Strongly disagree) □ 2 □ 3 □ 3 □ 5 (Strongly agree)

40. Do you think your defect reporting tool should provide a custom forms for reporting defects depending on reporter's experience)new – intermediate – experienced reporter)?

□ 1 (Strongly disagree) □ 2 □ 3 □ 3 □ 5 (Strongly agree)

41. Do you think your defect reporting tool should provide custom forms for reporting defects depending on reporter's knowledge (non-technical, technical, Human-Computer Interaction expert)?

□ 1 (Strongly disagree) □ 2 □ 3 □ 3 □ 5 (Strongly agree)

Note: If you have answered/ chosen item [1] in question 6, skip the following question

42. Do you use an automated tool to capture information for reporting usability defects?

□ Yes □ No

Automation Tool

Note: If you have answered/ chosen item [2] in question 42, skip the following question

Note: If you have answered/ chosen item [1] in question 6, skip the following question

43. Name the automated tools that you have used:

44. Do you think the manual process used to capture usability defect information causes erroneous or incomplete reports?

□ 1 (Strongly disagree) □ 2 □ 3 □ 3 □ 5 (Strongly agree)

Knowledge and Experience in Usability Defect Reporting

45. " The level of detail of usability defect reports varies greatly from reporter to reporter" – Do you agree with this statement?

□ 1 (Strongly disagree) □ 2 □ 3 □ 3 □ 5 (Strongly agree)

46. Please indicate how much you agree with the following factors about their influence in reporting usability defects.

Factors	**Completely disagree**	**Somewhat disagree**	**Neither disagree or agree**	**Somewhat agree**	**Completely agree**
Practical knowledge of the usability of software systems	□	□	□	□	□
Ability of thinking from the user's perspective	□	□	□	□	□
Knowledge of users' mental model	□	□	□	□	□
Knowledge of usability principles	□	□	□	□	□
Domain expertise	□	□	□	□	□
Knowledge of technical aspects	□	□	□	□	□
Knowledge of defect reporting	□	□	□	□	□

47. "Level of experience of software tester cannot guarantee the completeness of usability defect reports" – Do you agree with this statement?

□ 1 (Strongly disagree) □ 2 □ 3 □ 3 □ 5 (Strongly agree)

48. "Long experience in software testing helps reporters to build their knowledge, particularly in describing usability defects" – Do you agree with this statement?

□ 1 (Strongly disagree) □ 2 □ 3 □ 3 □ 5 (Strongly agree)

49. Do you think previous experience in reporting usability defects helps in increasing the completeness of usability defect information reported?

□ 1 (Strongly disagree) □ 2 □ 3 □ 3 □ 5 (Strongly agree)

Comments

50. Please use the space provided below to share any ideas or experiences you have that might improve the quality of usability defect reporting:

Appendix C

The Revised Usability Defect Classification Model – Categories, Subcategories, and Example Defects

Categories	**Sub-categories**	**Defects**	**Example defects**
Interface	GUI structure and aesthetics	Object appearance (icon, menu item, scroll bar, button, favicon etc)	Object (screen) keeps blinking/ flashing/ flickering Unnecessary animation on object (screen) Incorrect/ unnecessary use of object icon Incorrect/ inconsistent object design Inappropriate use of object size Missing object Unwanted object Incorrect/ inconsistent use of object color Inappropriate use of color contrast between object and background Inappropriate object (screen) resolution Object (screen) is truncated/ chopped/ overlapped/ blocked/ cropped Object (screen) not persistently display Object (screen) looks very bare Words/ name on object are not completely visible
		Object (screen) layout	Improper use of screen width and length
		Object (screen) state	Incorrect object (screen) activation and deactivation behavior Indistinguishable between active and inactive object (screen) Incorrect object (screen) select / deselect behavior Incorrect object (screen) visibility behavior Incorrect/ unclear current object (screen) state Slow object state change Slow screen refresh Screen does not automatically load content/ refresh Cursor loss focus Incorrect object (screen) focus No visual cue to screen (object) existence Object action is not properly set Inappropriate/ incorrect screen navigational
	Information presentation	Data presentation	Missing default data Data not selected Object does not remember last selection Data is not populated in the object (screen) Incorrect data type and format
		Object (screen) naming and labeling	Misspelled/ mislabeled label Label name is not intuitive Incorrect use of labeling standard Inconsistent menu labeling format
		Non-message feedback	Missing visual cues Misunderstanding of visual cues Irritating/ unnecessary animation on the progress bar Incorrect/ missing/ non-informative indicator No warning indicator in the screen Inappropriate use of dialog feedback
		Error/ notification and feedback message	Inconsistent message Inappropriate/ irrelevant message Incorrect/ misleading/ insufficient message Message is too verbose/ non -informative
		On screen text and results	Missing/ incomplete/ misleading information Content is not updated Unsupported format to display content
		Menu structure	Incorrect connection of main options to sub-options Inappropriate menu option Unorganized menu organization/ structure Obsolete menu option Missing relevant menu option Misplaced menu option

	Audibleness	Voice and sounds	Inappropriate video/ audio behavior
		Audio cue	NS
		Text/ feedback in speech	NS
Interaction	Manipulation	Keyboard press	Cursor losses focus when typing
			Duplicate access keys
			Inappropriate use of access keys
			Inconsistent access keys
			Missing/ not working access keys
			Unable to navigate from one field to another using keyboard
			Unable to type
			Unsupported keyboard function
		Mouse click	Mouse click not working
			Mouse hover does not give focus
			Inconsistent mouse click behavior
			Too many clicks to perform task
		Finger touch	Inappropriate vibrating behavior
			Incorrect click event action
			Tap did not invoke action bar
		Voice control	NS
		Scrolling mechanism	Scrolling not working
			Inappropriate scrolling behavior
			Unable to scroll
			No purpose vertical/ horizontal scroll
			Unbreakable scrolling axis lock
		Drag and drop	Unable to drag screen (object)
		Zooming	Incorrect/ inappropriate zooming behavior
			Unclear visual cues that an editor is zoomed
	Functionality		Missing
			Inadequate
			Preference
			Irrelevant
			Misaligned
			Problematic
			Technical deficiency
	Task execution	Action	Incorrect action results
			No action executed
			Incorrect action executed
			Too many steps to task completion
			Unnecessary action to execute task
			Limited accessibility to execute action
		Reversibility	Unable to undo action
			Unclear to perform action reversal
			Reverting the current action works incorrectly
		Feedback	Missing notification/ warning/ feedback message
			No summary of task in progress/ completed
			Unnecessary confirmation dialogue box/ indicator/ feedback
			Missing permission request to access other device
			No indication about the current state of an action in progress
			No indication about what will happen when clicking on certain button/ menu/option
			No feedback when performing critical action

***** *NS – No sample defects from the 377 open source usability defects we studied*

Appendix D

Open Source Usability Defect Classification Taxonomy Evaluation

Background Information

1. What is your age?

   □ Less than 24 years old □ 25 - 34 years old □ 35 - 44 years old

   □ 45 - 54 years old □ 55 years old and above

2. What is your gender?

   □ Male □ Female

3. What best describes you?

   □ HCI/ UX/ usability expert
   □ Industry researcher
   □ Academic researcher
   □ Software developer with experience in both user interface and software development
   □ Software developer with experience in software development only
   □ Software tester with HCI knowledge
   □ Software tester without HCI knowledge
   □ End user with HCI/ UX/ usability knowledge
   □ End user without HCI/ UX/ usability knowledge

4. Have you received any training/ certification related to usability evaluation/ Human Computer Interaction (HCI)

   □ Yes □ No

5. How familiar are you with usability defects?

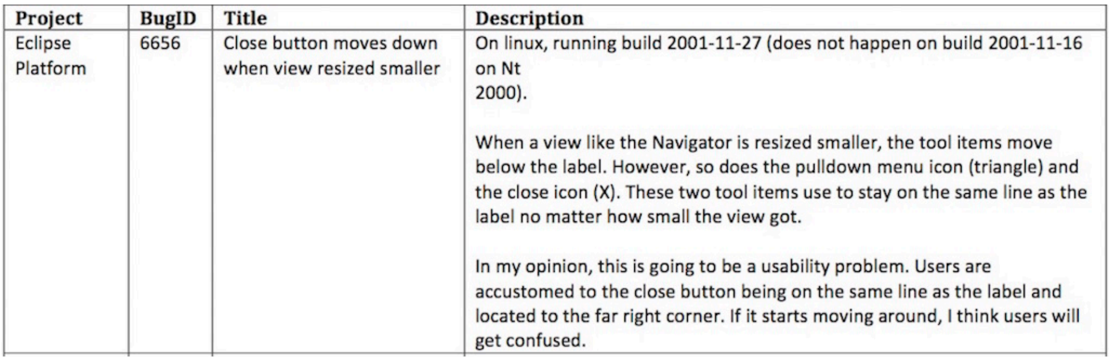
   □ Not at all familiar □ Slightly familiar □ Somewhat familiar

   □ Moderately familiar □ Extremely familiar

6. Have you used any of the following defect classification scheme?

   □ Orthogonal Defect Classification (ODC)
   □ Root Cause Analysis (RCA)
   □ Hewlett Packard Defect Classification Scheme (HP-DCS)
   □ Classification of Usability Problems (CUP)
   □ Usability Problem Taxonomy (UPT)
   □ Usability Action Framework (UAF)
   □ Other, please specify:

Defect 1: Eclipse Platform defect #6656



Project	BugID	Title	Description
Eclipse Platform	6656	Close button moves down when view resized smaller	On linux, running build 2001-11-27 (does not happen on build 2001-11-16 on Nt 2000). When a view like the Navigator is resized smaller, the tool items move below the label. However, so does the pulldown menu icon (triangle) and the close icon (X). These two tool items use to stay on the same line as the label no matter how small the view got. In my opinion, this is going to be a usability problem. Users are accustomed to the close button being on the same line as the label and located to the far right corner. If it starts moving around, I think users will get confused.

7. Study Eclipse Platform Defect #6656 above. What kind of usability defect is reported? View OSUDC taxonomy here.

□ Interface □ Interaction □ Both

Note: if you have answered/ chosen item [3] in question 7, skip the following question

Note: if you have answered/ chosen item [2] in question 7, skip the following question

8. What kind of interface defect is reported? View OSUDC taxonomy here.

□ GUI structure and aesthetics – Object (screen) appearance
□ GUI structure and aesthetics – Object (screen) layout
□ GUI structure and aesthetics – Object (screen) state
□ Information presentation – Data presentation
□ Information presentation – Object (screen) naming & labeling
□ Information presentation –Non message feedback
□ Information presentation – Error, notification and feedback message
□ Information presentation – On screen text and results
□ Information presentation – Menu structure
□ Audibleness – Voice and sound
□ Audibleness – Text and feedback in speech
□ Other interface problems not listed above, please specify:

Note: if you have answered/ chosen item [3] in question 7, skip the following question

Note: if you have answered/ chosen item [2] in question 7, skip the following question

9. Does the user seem to express any of the following emotions in the defect descriptions? View OSUDC taxonomy here.

Emotion	**Yes**	**No**
Annoyance	□	□
Distraction	□	□
Frustration	□	□

Note: if you have answered/ chosen item [3] in question 7, skip the following question

Note: if you have answered/ chosen item [2] in question 7, skip the following question

10. Does the user mention anything about how the issue affects the task execution? View OSUDC taxonomy here.

□ Complexity
□ Loss of data
□ Visibility
□ Understandability
□ Performance
□ Does not mention
□ Accessibility
□ Other, please specify:

Note: if you have answered/ chosen item [3] in question 7, skip the following question

Note: if you have answered/ chosen item [2] in question 7, skip the following question

11. Based on the above defect descriptions, on what basis has the user reported this issue? View OSUDC taxonomy here.

□ Missing
□ Irrelevant
□ I can't tell
□ Wrong
□ Overlooked
□ Incongruent mental model
□ Better way

Note: if you have answered/ chosen item [3] in question 7, skip the following question

Note: if you have answered/ chosen item [1] in question 7, skip the following question

12. What kind of interaction defect is reported? View OSUDC taxonomy here.

□ Manipulation- Keyboard press
□ Manipulation- Mouse click
□ Manipulation- Finger touch
□ Manipulation-Voice control
□ Manipulation-Scrolling mechanisms
□ Manipulation- Drag and drop
□ Manipulation-Zooming

□ Task execution - Action

□ Task execution - Reversibility

□ Task execution - Feedback

□ Functionality

□ Other interface problems not listed above, please specify:

Note: if you have answered/ chosen item [3] in question 7, skip the following question

Note: if you have answered/ chosen item [1] in question 7, skip the following question

13. Does the user seem to express any of the following emotions in the defect descriptions? View OSUDC taxonomy here.

Emotion	**Yes**	**No**
Annoyance	□	□
Distraction	□	□
Frustration	□	□

Note: if you have answered/ chosen item [3] in question 7, skip the following question

Note: if you have answered/ chosen item [1] in question 7, skip the following question

14. Does the user mention anything about how the issue affects the task execution? View OSUDC taxonomy here.

□ Complexity

□ Loss of data

□ Visibility

□ Understandability

□ Performance

□ Does not mention

□ Accessibility

□ Other, please specify:

Note: if you have answered/ chosen item [3] in question 7, skip the following question

Note: if you have answered/ chosen item [1] in question 7, skip the following question

15. Based on the above defect descriptions, on what basis has the user reported this issue? View OSUDC taxonomy here.

□ Missing

□ Irrelevant

□ I can't tell

□ Wrong

□ Overlooked

□ Incongruent mental model

□ Better way

Note: if you have answered/ chosen item [1] in question 7, skip the following question

Note: if you have answered/ chosen item [2] in question 7, skip the following question

16. What kind of interface defect is reported? View OSUDC taxonomy here.

□ GUI structure and aesthetics – Object (screen) appearance
□ GUI structure and aesthetics – Object (screen) layout
□ GUI structure and aesthetics – Object (screen) state
□ Information presentation – Data presentation
□ Information presentation – Object (screen) naming & labeling
□ Information presentation –Non message feedback
□ Information presentation – Error, notification and feedback message
□ Information presentation – On screen text and results
□ Information presentation – Menu structure
□ Audibleness – Voice and sound
□ Audibleness – Text and feedback in speech
□ Other interface problems not listed above, please specify:

Note: if you have answered/ chosen item [1] in question 7, skip the following question

Note: if you have answered/ chosen item [2] in question 7, skip the following question

17. What kind of interaction defect is reported? View OSUDC taxonomy here.

□ Manipulation- Keyboard press
□ Manipulation- Mouse click
□ Manipulation- Finger touch
□ Manipulation-Voice control
□ Manipulation-Scrolling mechanisms
□ Manipulation- Drag and drop
□ Manipulation-Zooming
□ Task execution - Action
□ Task execution - Reversibility
□ Task execution - Feedback
□ Functionality
□ Other interface problems not listed above, please specify:

Note: if you have answered/ chosen item [1] in question 7, skip the following question

Note: if you have answered/ chosen item [2] in question 7, skip the following question

18. Does the user seem to express any of the following emotions in the defect descriptions? View OSUDC taxonomy here.

Emotion	**Yes**	**No**
Annoyance	□	□

Distraction	□	□
Frustration	□	□

Note: if you have answered/ chosen item [1] in question 7, skip the following question

Note: if you have answered/ chosen item [2] in question 7, skip the following question

19. Does the user mention anything about how the issue affects the task execution? View OSUDC taxonomy here.

□ Complexity

□ Loss of data

□ Visibility

□ Understandability

□ Performance

□ Does not mention

□ Accessibility

□ Other, please specify:

Note: if you have answered/ chosen item [1] in question 7, skip the following question

Note: if you have answered/ chosen item [2] in question 7, skip the following question

20. Based on the above defect descriptions, on what basis has the user reported this issue? View OSUDC taxonomy here.

□ Missing

□ Irrelevant

□ I can't tell

□ Wrong

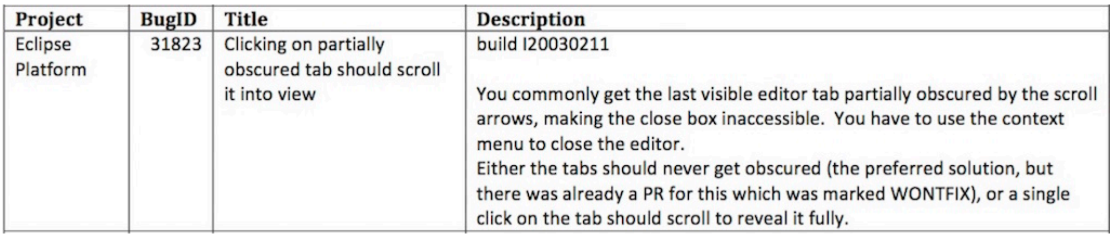
□ Overlooked

□ Incongruent mental model

□ Better way

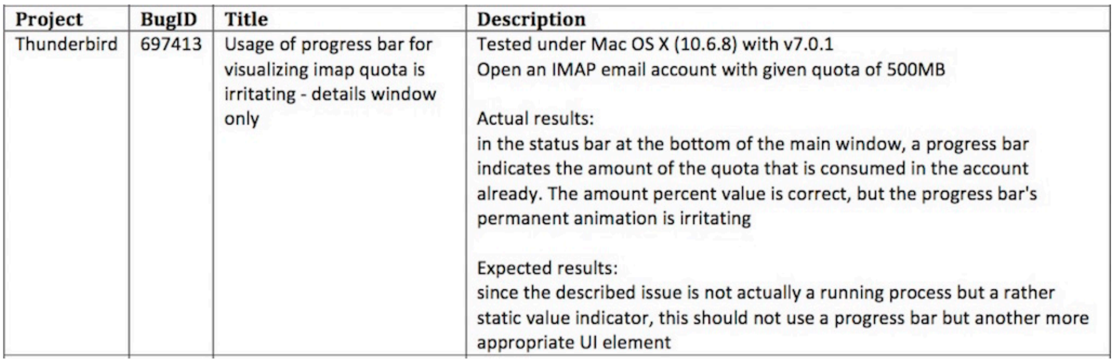
** The same questions were used for classifying the other 9 usability defect reports below:

Defect 2: Eclipse Platform Defect#31823



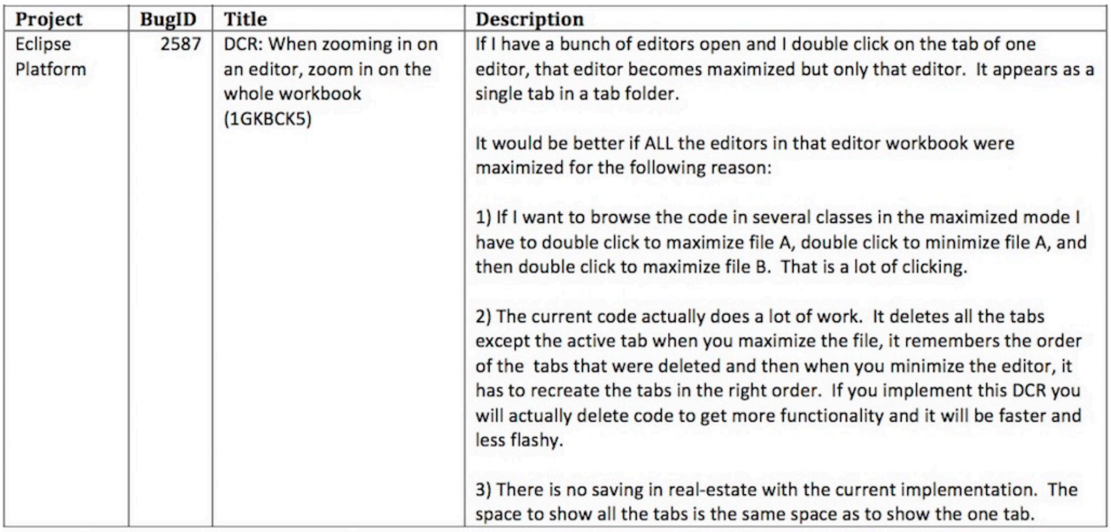
Project	BugID	Title	Description
Eclipse Platform	31823	Clicking on partially obscured tab should scroll it into view	build I20030211 You commonly get the last visible editor tab partially obscured by the scroll arrows, making the close box inaccessible. You have to use the context menu to close the editor. Either the tabs should never get obscured (the preferred solution, but there was already a PR for this which was marked WONTFIX), or a single click on the tab should scroll to reveal it fully.

Defect 3: Thunderbird Defect #697413



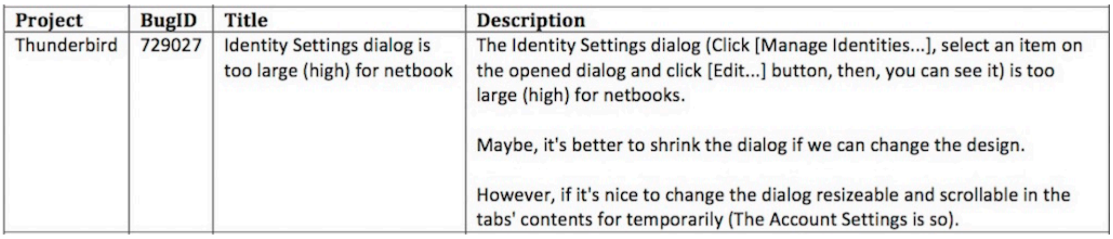
Project	BugID	Title	Description
Thunderbird	697413	Usage of progress bar for visualizing imap quota is irritating - details window only	Tested under Mac OS X (10.6.8) with v7.0.1 Open an IMAP email account with given quota of 500MB Actual results: in the status bar at the bottom of the main window, a progress bar indicates the amount of the quota that is consumed in the account already. The amount percent value is correct, but the progress bar's permanent animation is irritating Expected results: since the described issue is not actually a running process but a rather static value indicator, this should not use a progress bar but another more appropriate UI element

Defect 4: Eclipse Platform Defect #2587



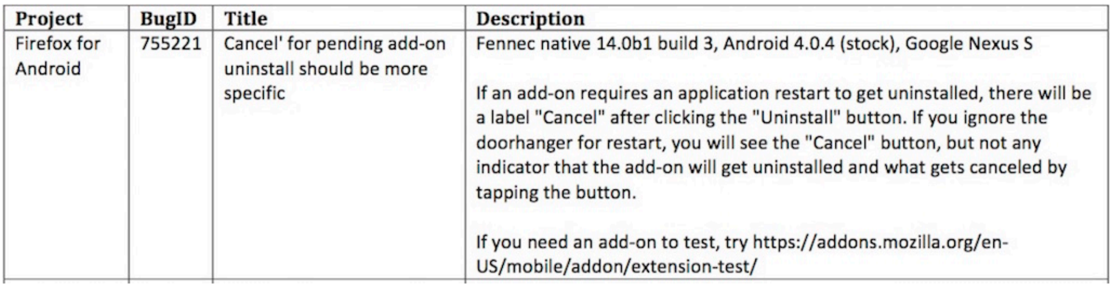
Project	BugID	Title	Description
Eclipse Platform	2587	DCR: When zooming in on an editor, zoom in on the whole workbook (1GKBCK5)	If I have a bunch of editors open and I double click on the tab of one editor, that editor becomes maximized but only that editor. It appears as a single tab in a tab folder. It would be better if ALL the editors in that editor workbook were maximized for the following reason: 1) If I want to browse the code in several classes in the maximized mode I have to double click to maximize file A, double click to minimize file A, and then double click to maximize file B. That is a lot of clicking. 2) The current code actually does a lot of work. It deletes all the tabs except the active tab when you maximize the file, it remembers the order of the tabs that were deleted and then when you minimize the editor, it has to recreate the tabs in the right order. If you implement this DCR you will actually delete code to get more functionality and it will be faster and less flashy. 3) There is no saving in real-estate with the current implementation. The space to show all the tabs is the same space as to show the one tab.

Defect 5: Thunderbird Defect #729027



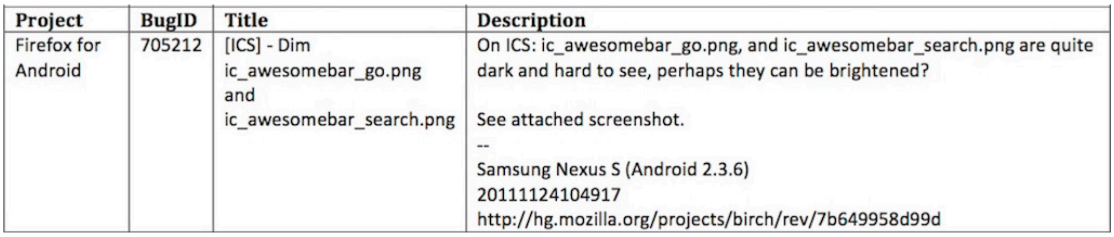
Project	BugID	Title	Description
Thunderbird	729027	Identity Settings dialog is too large (high) for netbook	The Identity Settings dialog (Click [Manage Identities...], select an item on the opened dialog and click [Edit...] button, then, you can see it) is too large (high) for netbooks. Maybe, it's better to shrink the dialog if we can change the design. However, if it's nice to change the dialog resizeable and scrollable in the tabs' contents for temporarily (The Account Settings is so).

Defect 6: Firefox for Android Defect #755221



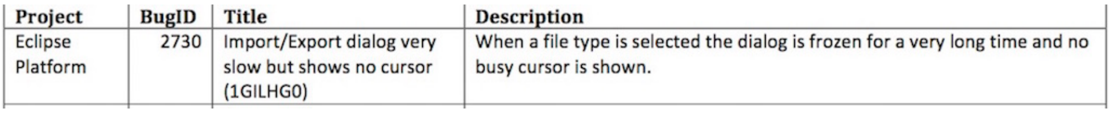
Project	BugID	Title	Description
Firefox for Android	755221	Cancel' for pending add-on uninstall should be more specific	Fennec native 14.0b1 build 3, Android 4.0.4 (stock), Google Nexus S If an add-on requires an application restart to get uninstalled, there will be a label "Cancel" after clicking the "Uninstall" button. If you ignore the doorhanger for restart, you will see the "Cancel" button, but not any indicator that the add-on will get uninstalled and what gets canceled by tapping the button. If you need an add-on to test, try https://addons.mozilla.org/en-US/mobile/addon/extension-test/

Defect 7: Firefox for Android Defect#705212



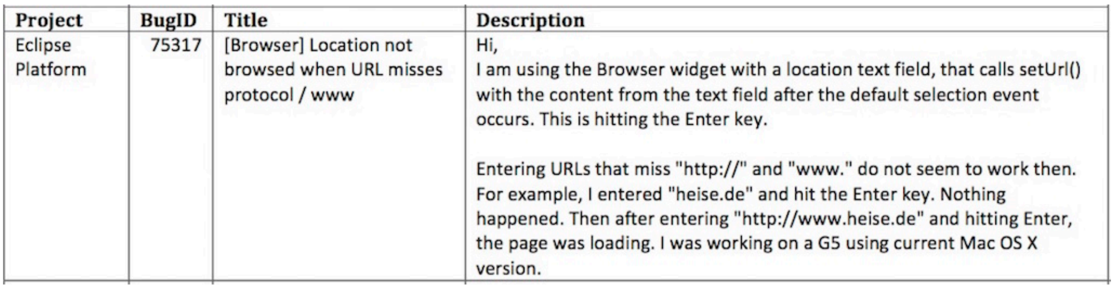
Project	BugID	Title	Description
Firefox for Android	705212	[ICS] - Dim ic_awesomebar_go.png and ic_awesomebar_search.png	On ICS: ic_awesomebar_go.png, and ic_awesomebar_search.png are quite dark and hard to see, perhaps they can be brightened? See attached screenshot. -- Samsung Nexus S (Android 2.3.6) 20111124104917 http://hg.mozilla.org/projects/birch/rev/7b649958d99d

Defect 8: Eclipse Platform Defect #2730



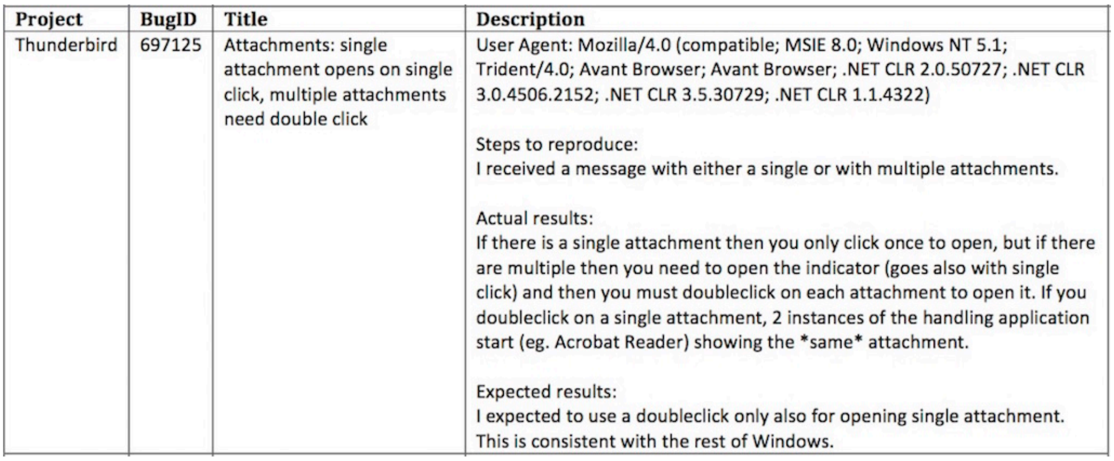
Project	BugID	Title	Description
Eclipse Platform	2730	Import/Export dialog very slow but shows no cursor (1GILHG0)	When a file type is selected the dialog is frozen for a very long time and no busy cursor is shown.

Defect 9: Eclipse Platform Defect #75317



Project	BugID	Title	Description
Eclipse Platform	75317	[Browser] Location not browsed when URL misses protocol / www	Hi, I am using the Browser widget with a location text field, that calls setUrl() with the content from the text field after the default selection event occurs. This is hitting the Enter key. Entering URLs that miss "http://" and "www." do not seem to work then. For example, I entered "heise.de" and hit the Enter key. Nothing happened. Then after entering "http://www.heise.de" and hitting Enter, the page was loading. I was working on a G5 using current Mac OS X version.

Defect 10: Thunderbird Defect #697125



Project	BugID	Title	Description
Thunderbird	697125	Attachments: single attachment opens on single click, multiple attachments need double click	User Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; Avant Browser; Avant Browser; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; .NET CLR 1.1.4322) Steps to reproduce: I received a message with either a single or with multiple attachments. Actual results: If there is a single attachment then you only click once to open, but if there are multiple then you need to open the indicator (goes also with single click) and then you must doubleclick on each attachment to open it. If you doubleclick on a single attachment, 2 instances of the handling application start (eg. Acrobat Reader) showing the *same* attachment. Expected results: I expected to use a doubleclick only also for opening single attachment. This is consistent with the rest of Windows.

Appendix E

Prototype Evaluation Form

Usability Defect Description Qualities

Instructions:

- Read each usability defect description
- Rate your impression of the quality for each of the information written in the usability defect description by answering the given questions
- For some questions, you need to reproduce the problem in your own mobile device

Defect Report	Assessment Questions	Answer
Bugzilla defect#1145590 **Report Title:** Meaning of icon is not intuitive **Description:** Meaning of icons is not intuitive. Users unaware the ‘book’ is for Reader View. Users confused what the ‘Hat’ is for. **Steps to Reproduce:** **1.** Launch Firefox browser. **2.** Look for Reader View icon. **Actual Results:** User did not notice that the “book” icon is for Reader View. **Expected Results:** Make the Reader View icon is more intuitive.	1. Was the defect report explained about proposals for solutions? For example, the descriptions provide alternatives and tradeoffs, and supplied rationale for the recommendations [9].	□ Yes □ No □ Partially
	2. Was the defect report described cause of the problem, including the justification what posed a problem, including system components that are affected or involved?	□ Yes □ No □ Partially
	3. Was the defect report described the main usability issue involved in the problem? For example, the description about what is wrong with the interaction architecture, interface and user task design.	□ Yes □ No □ Partially
	4. Was the defect report explained in detail steps by steps how to reproduce the problem, including user’s navigation flow through the system?	□ Yes □ No □ Partially
	5. Are you able to reproduce the problem on your own device and environment?	□ Yes □ No □ Partially
	6. Were the actual results you observed similar with the one in the defect descriptions?	□ Yes □ No □ Partially
	7. Was the defect report justified why the reporter thinks that it was a problem?	□ Yes □ No □ Partially
	8. Was the defect report explicitly mentioned about what poses a problem to the user?	□ Yes □ No □ Partially
	9. Was the defect report described the impact of the problem on business effect, impact on the user’s task and importance of the task?	□ Yes □ No □ Partially
	10. Was the defect report described reporters emotion, feeling or reactions upon the issues?	□ Yes □ No □ Partially
	11. Was the defect report mentioned about how often the problem occurred?	□ Yes □ No □ Partially

Defect Report	Assessment Questions	Answer
Bugzilla defect#1146389 **Report Title:** Unclear how to access Reader View **Description:** I have difficulty to find out Reader view icon. The reader View icon is not persistently displayed when user scroll down the page. In addition, the very small Reader view icon and its color are not intuitive for user to discover. In addition, several users also addressed their negative feedback on build #3 and #4 about the unintuitive meaning of "book" icon for Reader View icon. **Steps to reproduce:** 6. Click on Firefox browser icon. 7. Go to google search and find some articles – for example search on "software engineering". 8. Select on one of the search results, and wait for the information page to be loaded. 9. While the information is loading, scroll down the page. 10. Observe the "Reader View" icon and try to access Reader View. **Actual results:** When the information is loading and I scroll down the page, I couldn't find any sign to access Reader View. The "Reader View" icon is not immediately shown at the end of the URL bar and the icon is not discoverable when you scroll down the page. It is very frustrating when you have to scroll up to the very top of the page to find Reader View icon. This can be time consuming if you had a very long page. Furthermore, the "Reader View" icon is very small and the grey icon on the black background (in private browsing) is not discoverable enough. **Expected results:** I would expect that the "Reader View" icon be triggered while you are in any part of the page. So that user can access Reader View icon instantly without scrolling up to the top of the page.	1. 1.Was the defect report explained about proposals for solutions? For example, the descriptions provide alternatives and tradeoffs, and supplied rationale for the recommendations [9].	□ Yes □ No □ Partially
	2. Was the defect report described cause of the problem, including the technical justification of what posed a problem (i.e., system components that are affected or involved) or violations of any usability design and principles?	□ Yes □ No □ Partially
	3. Was the defect report described the main usability issue involved in the problem? For example, the description about what is wrong with the interaction, interface and user task design.	□ Yes □ No □ Partially
	4. Was the defect report explained in detail steps by steps how to reproduce the problem, including user's navigation flow through the system?	□ Yes □ No □ Partially
	5. Are you able to reproduce the problem on your own device and environment?	□ Yes □ No □ Partially
	6. Were the actual results you observed similar with the one in the defect descriptions?	□ Yes □ No □ Partially
	7. Was the defect report justified why the reporter thinks that it was a problem?	□ Yes □ No □ Partially
	8. Was the defect report explicitly mentioned about what poses a problem to the user?	□ Yes □ No □ Partially
	9. Was the defect report described the impact of the problem on business, user's task and importance of the task?	□ Yes □ No □ Partially
	10. Was the defect report described reporters emotion, feeling or reactions upon the issues (i.e, annoyance, confusion and distraction)?	□ Yes □ No □ Partially
	11. Was the defect report mentioned about how often the problem occurred or if other user had experiences the same problem?	□ Yes □ No □ Partially

Defect Report	Assessment Questions	Answer
Bugzilla defect#1145602 **Report Title:** New About:Home tabs experience is confusing **Description:** I can't directly find tabs that have been opened. The newly added tabs are represented as multiple blank/empty spots, which are hidden in multilayer page. User can create multiple new tabs without realizing they are doing so. Intention to create a new tab isn't clear. **Steps to reproduce:** 1. Click on Firefox browser icon. 2. Go to any webpage. For example open http://ebay.com.au 3. Press the middle menu at the bottom of the Tab manager page – then press on New Tab **Actual results:** There is no obvious indicator that shows new tab is created. If you are well noticed, only number on the tab icon (square box on the right side of URL bar) is updated/ increased whenever you add a new tab. However, if you did not aware the existing number of tabs open, you may not knew that a new tab has been added. It took me sometime to figure out if new tabs were successfully added or not, and I do not know where to find the existing open tabs. The only way you can know if the tab was created is by pressing tab icon on the right side of the URL bar. The Tab Manager experience is really confusing. **Expected results:** It would be good if there were an informative indicator other than the updated number of open tabs to inform user that new tab was added. For example, a toast message that indicate: "New tab is successfully added"	1. 1.Was the defect report explained about proposals for solutions? For example, the descriptions provide alternatives and tradeoffs, and supplied rationale for the recommendations [9].	□ Yes □ No □ Partially
	2. Was the defect report described cause of the problem, including the technical justification of what posed a problem (i.e., system components that are affected or involved) or violations of any usability design and principles?	□ Yes □ No □ Partially
	3. Was the defect report described the main usability issue involved in the problem? For example, the description about what is wrong with the interaction, interface and user task design.	□ Yes □ No □ Partially
	4. Was the defect report explained in detail steps by steps how to reproduce the problem, including user's navigation flow through the system?	□ Yes □ No □ Partially
	5. Are you able to reproduce the problem on your own device and environment?	□ Yes □ No □ Partially
	6. Were the actual results you observed similar with the one in the defect descriptions?	□ Yes □ No □ Partially
	7. Was the defect report justified why the reporter thinks that it was a problem?	□ Yes □ No □ Partially
	8. Was the defect report explicitly mentioned about what poses a problem to the user?	□ Yes □ No □ Partially
	9. Was the defect report described the impact of the problem on business, user's task and importance of the task?	□ Yes □ No □ Partially
	10. Was the defect report described reporters emotion, feeling or reactions upon the issues (i.e, annoyance, confusion and distraction)?	□ Yes □ No □ Partially
	11. Was the defect report mentioned about how often the problem occurred or if other user had experiences the same problem?	□ Yes □ No □ Partially

Defect Report	Assessment Questions	Answer
Bugzilla defect#1191376 **Report Title:** Always show toolbar when restoring app **Description:** My most frequent use case for Firefox is launching it quickly to search for something or load a specific page by URL (either in the clipboard, in my memory, or less frequently, from my history). I tend to sometimes keep tabs open to refer to later, but most of the times I launch the app I don't want to deal with them and just want a blank slate. **Steps to Reproduce:** **1.** Launch Firefox browser. **Actual Results:** When Firefox launches, depending on what I was doing last, sometimes the top toolbar is collapsed. So I need to do a bit of a "wiggle" (scroll the page slightly) to get it to appear, then tap the location bar, then start typing. **Expected Results:** Ideally, I would never have to do that wiggle. If the toolbar always appeared after a restore then I wouldn't have to.	1. Was the defect report explained about proposals for solutions? For example, the descriptions provide alternatives and tradeoffs, and supplied rationale for the recommendations [9].	□ Yes □ No □ Partially
	2. Was the defect report described cause of the problem, including the justification what posed a problem, including system components that are affected or involved?	□ Yes □ No □ Partially
	3. Was the defect report described the main usability issue involved in the problem? For example, the description about what is wrong with the interaction architecture, interface and user task design.	□ Yes □ No □ Partially
	4. Was the defect report explained in detail steps by steps how to reproduce the problem, including user's navigation flow through the system?	□ Yes □ No □ Partially
	5. Are you able to reproduce the problem on your own device and environment?	□ Yes □ No □ Partially
	6. Were the actual results you observed similar with the one in the defect descriptions?	□ Yes □ No □ Partially
	7. Was the defect report justified why the reporter thinks that it was a problem?	□ Yes □ No □ Partially
	8. Was the defect report explicitly mentioned about what poses a problem to the user?	□ Yes □ No □ Partially
	9. Was the defect report described the impact of the problem on business effect, impact on the user's task and importance of the task?	□ Yes □ No □ Partially
	10. Was the defect report described reporters emotion, feeling or reactions upon the issues?	□ Yes □ No □ Partially
	11. Was the defect report mentioned about how often the problem occurred?	□ Yes □ No □ Partially

Defect Report	Assessment Questions	Answer
Bugzilla defect#1231815 **Report Title:** Irrelevant steps to add new tab **Description:** I am not satisfied with the steps to open a new Tab. The current steps are irrelevant to speed up the process to add new Tab. **Steps to Reproduce:** 5. Click on Firefox browser icon. 6. Go to any webpage. For example open http://ebay.com.au 7. When a webpage is open, open a new tab. Press the Tab button on the right side of the URL bar. 8. Press the "+" button at the bottom of the Tab manager to add a new tab. On the newly open page, open a new page, for example – http://gumtree.com.au **Actual Results:** To open new tab, user requires two steps (step 3 and step 4 above). Even this issue is not affecting my task, but this is really annoying when I have to use only one hand to hold the phone and use several tabs to look for something. **Expected Results:** Long-press the tab button to add new Tab. Using long-press button could provide an easy way to interact with the web browser, especially when the user is moving.	1. Was the defect report explained about proposals for solutions? For example, the descriptions provide alternatives and tradeoffs, and supplied rationale for the recommendations [9].	□ Yes □ No □ Partially
	2. Was the defect report described cause of the problem, including the technical justification of what posed a problem (i.e., system components that are affected or involved) or violations of any usability design and principles?	□ Yes □ No □ Partially
	3. Was the defect report described the main usability issue involved in the problem? For example, the description about what is wrong with the interaction, interface and user task design.	□ Yes □ No □ Partially
	4. Was the defect report explained in detail steps by steps how to reproduce the problem, including user's navigation flow through the system?	□ Yes □ No □ Partially
	5. Are you able to reproduce the problem on your own device and environment?	□ Yes □ No □ Partially
	6. Were the actual results you observed similar with the one in the defect descriptions?	□ Yes □ No □ Partially
	7. Was the defect report justified why the reporter thinks that it was a problem?	□ Yes □ No □ Partially
	8. Was the defect report explicitly mentioned about what poses a problem to the user?	□ Yes □ No □ Partially
	9. Was the defect report described the impact of the problem on business, user's task and importance of the task?	□ Yes □ No □ Partially
	10. Was the defect report described reporters emotion, feeling or reactions upon the issues (i.e, annoyance, confusion and distraction)?	□ Yes □ No □ Partially
	11. Was the defect report mentioned about how often the problem occurred or if other user had experiences the same problem?	□ Yes □ No □ Partially

Defect Report	Assessment Questions	Answer
Bugzilla defect#1144758 **Report Title:** Setting icon not easily discoverable **Description:** Settings icon is not easily discoverable since it is ‘buried’ down a level in the Tabs screen. **Steps to Reproduce:** 7. Press on Firefox browser icon. **8.** Find Setting icon in the About: Home page. **9.** Press the middle menu at the bottom of the Tab manager page. **10.** Find Setting icon. **Actual Results:** The Setting icon cannot be found on the About: Home page. The Setting icon is exist in the Tabs screen and user have to swipe to the next screen. In this case, I have to perform unnecessary steps to find the icon, and it wasting my time. **Expected Results:** The Setting icon should be easily discoverable without too many steps to find the icon.	12. Was the defect report explained about proposals for solution? For example, the descriptions provide alternatives and tradeoffs, and supplied rationale for the recommendations [9].	□ Yes □ No □ Partially
	13. Was the defect report described cause of the problem, including the justification what posed a problem, including system components that are affected or involved?	□ Yes □ No □ Partially
	14. Was the defect report described the main usability issue involved in the problem? For example, the description about what is wrong with the interaction architecture, interface and user task design.	□ Yes □ No □ Partially
	15. Was the defect report explained in detail steps by steps how to reproduce the problem, including user’s navigation flow through the system?	□ Yes □ No □ Partially
	16. Are you able to reproduce the problem on your own device and environment?	□ Yes □ No □ Partially
	17. Were the actual results you observed similar with the one in the defect descriptions?	□ Yes □ No □ Partially
	18. Was the defect report justified why the reporter thinks that it was a problem?	□ Yes □ No □ Partially
	19. Was the defect report explicitly mentioned about what poses a problem to the user?	□ Yes □ No □ Partially
	20. Was the defect report described the impact of the problem on business effect, impact on the user’s task and importance of the task?	□ Yes □ No □ Partially
	21. Was the defect report described reporters emotion, feeling or reactions upon the issues?	□ Yes □ No □ Partially
	22. Was the defect report mentioned about how often the problem occurred?	□ Yes □ No □ Partially

Defect Report	Assessment Questions	Answer
Bugzilla defect#1237631 **Report Title:** Do not exit private mode on browser re-launch even if there are no private tabs to restore **Description:** I am not satisfied with the existing Private Tab browsing feature. Currently, Firefox browser only open browser in normal mode, even the last opened browser is in Private mode. It is contradict with my previous experience with Safari browser, in which browser will be re-launched based on the previously open mode. **Steps to Reproduce:** 1. Press on Firefox browser icon. 2. Press the middle menu at the bottom of the Tab manager page – then press on New Private Tab. 3. Repeat step 2 to open several private tabs. 4. Press on the Tab button (tiny purple box) on the right side of the URL bar. 5. Press the middle menu at the bottom of the Tab manager page – then press on Close All Tabs. 6. Re-launch Firefox **Actual Results:** Firefox is opened in normal browsing. I am confused because normally I browse using Private mode. Even this feature design is not affecting my task, but it actually affecting my browsing privacy. **Expected Results:** Firefox will open in private mode, similar to Safari browser.	1. 1.Was the defect report explained about proposals for solutions? For example, the descriptions provide alternatives and tradeoffs, and supplied rationale for the recommendations [9].	□ Yes □ No □ Partially
	2. Was the defect report described cause of the problem, including the technical justification of what posed a problem (i.e., system components that are affected or involved) or violations of any usability design and principles?	□ Yes □ No □ Partially
	3. Was the defect report described the main usability issue involved in the problem? For example, the description about what is wrong with the interaction, interface and user task design.	□ Yes □ No □ Partially
	4. Was the defect report explained in detail steps by steps how to reproduce the problem, including user's navigation flow through the system?	□ Yes □ No □ Partially
	5. Are you able to reproduce the problem on your own device and environment?	□ Yes □ No □ Partially
	6. Were the actual results you observed similar with the one in the defect descriptions?	□ Yes □ No □ Partially
	7. Was the defect report justified why the reporter thinks that it was a problem?	□ Yes □ No □ Partially
	8. Was the defect report explicitly mentioned about what poses a problem to the user?	□ Yes □ No □ Partially
	9. Was the defect report described the impact of the problem on business, user's task and importance of the task?	□ Yes □ No □ Partially
	10. Was the defect report described reporters emotion, feeling or reactions upon the issues (i.e, annoyance, confusion and distraction)?	□ Yes □ No □ Partially
	11. Was the defect report mentioned about how often the problem occurred or if other user had experiences the same problem?	□ Yes □ No □ Partially

Defect Report	Assessment Questions	Answer
Bugzilla defect#1145597 **Report Title:** Navigating between tabs is confusing **Description:** Navigating between tabs is confusing. Tabs are not well defined visually (blend together) and look is not polished/iOS feeling. **Steps to Reproduce:** 4. Click on Firefox browser icon. 5. Go to any webpage. For example open http://ebay.com.au 6. Press the middle menu at the bottom of the Tab manager page – then press on New Tab 7. Go to the previous open tabs. **Actual Results:** No indicator to navigate between tabs. **Expected Results:** There should be an indicator to navigate between tabs as similar to desktop version.	1. Was the defect report explained about proposals for solutions? For example, the descriptions provide alternatives and tradeoffs, and supplied rationale for the recommendations [9].	□ Yes □ No □ Partially
	2. Was the defect report described cause of the problem, including the justification what posed a problem, including system components that are affected or involved?	□ Yes □ No □ Partially
	3. Was the defect report described the main usability issue involved in the problem? For example, the description about what is wrong with the interaction architecture, interface and user task design.	□ Yes □ No □ Partially
	4. Was the defect report explained in detail steps by steps how to reproduce the problem, including user's navigation flow through the system?	□ Yes □ No □ Partially
	5. Are you able to reproduce the problem on your own device and environment?	□ Yes □ No □ Partially
	6. Were the actual results you observed similar with the one in the defect descriptions?	□ Yes □ No □ Partially
	7. Was the defect report justified why the reporter thinks that it was a problem?	□ Yes □ No □ Partially
	8. Was the defect report explicitly mentioned about what poses a problem to the user?	□ Yes □ No □ Partially
	9. Was the defect report described the impact of the problem on business effect, impact on the user's task and importance of the task?	□ Yes □ No □ Partially
	10. Was the defect report described reporters emotion, feeling or reactions upon the issues?	□ Yes □ No □ Partially
	11. Was the defect report mentioned about how often the problem occurred?	□ Yes □ No □ Partially

Defect Report	Assessment Questions	Answer
Bugzilla defect #1199983 **Report Title:** Unable to scroll long page using scrollbar **Description:** I have difficulty to manipulate object (scrollbar) in the user interface. When I open a really large page, like a Wikipedia page, it appears that I can only scroll further down with the swipe gesture. The scrollbar on the right page is not able to grab. The current scrolling mechanism of Firefox mobile browser violating the flexibility and ease of use usability principles to move within the screen. This is very annoying experience when you know the scrollbar is supposed to be used to move the screen viewing area up and down, but it does not work. **Steps to reproduce:** 4. Go to Wikipedia page and search for information, which may have long information. 5. Touch the screen. You will see the scroll indicator on the right side. 6. Try to grab the scrollbar and scroll down. **Actual results:** You were unable to scroll using the scroll indicator. Instead, you are scrolling using swipe gesture. It took so much time for me to scroll to the middle of the page. I spent literally 5 mins just with the swiping restore to get to the middle of the page, because it seems that I can't grab the scrollbar on the right and move it down with my finger to scroll faster like in other iOS apps. **Expected results:** I would expect the scrollbar function to work similar to any other scrollbar in desktop webpage and in other iOS app. User can grab the scrollbar and fast scrolling to the up/ bottom of the page.	1. Was the defect report explained about proposals for solutions? For example, the descriptions provide alternatives and tradeoffs, and supplied rationale for the recommendations [9].	□ Yes □ No □ Partially
	2. Was the defect report described cause of the problem, including the technical justification of what posed a problem (i.e., system components that are affected or involved) or violations of any usability design and principles?	□ Yes □ No □ Partially
	3. Was the defect report described the main usability issue involved in the problem? For example, the description about what is wrong with the interaction, interface and user task design.	□ Yes □ No □ Partially
	4. Was the defect report explained in detail steps by steps how to reproduce the problem, including user's navigation flow through the system?	□ Yes □ No □ Partially
	5. Are you able to reproduce the problem on your own device and environment?	□ Yes □ No □ Partially
	6. Were the actual results you observed similar with the one in the defect descriptions?	□ Yes □ No □ Partially
	7. Was the defect report justified why the reporter thinks that it was a problem?	□ Yes □ No □ Partially
	8. Was the defect report explicitly mentioned about what poses a problem to the user?	□ Yes □ No □ Partially
	9. Was the defect report described the impact of the problem on business, user's task and importance of the task?	□ Yes □ No □ Partially
	10. Was the defect report described reporters emotion, feeling or reactions upon the issues (i.e, annoyance, confusion and distraction)?	□ Yes □ No □ Partially
	11. Was the defect report mentioned about how often the problem occurred or if other user had experiences the same problem?	□ Yes □ No □ Partially

Defect Report	Assessment Questions	Answer
Bugzilla defect#1250238 **Report Title:** Open In New Private Tab doesn't offer an affordance to find the newly opened private tab **Description:** **Steps to Reproduce:** 1. Click on Firefox browser icon. 2. Go to google.com.au and search for something 3. Select one search result and long press the link **Actual Results:** Open In New Private Tab doesn't work. **Expected Results:** The Open In New Private Tab should be populated and user could be easily discovered the newly opened private tab.	1. Was the defect report explained about proposals for solutions? For example, the descriptions provide alternatives and tradeoffs, and supplied rationale for the recommendations [9].	□ Yes □ No □ Partially
	2. Was the defect report described cause of the problem, including the justification what posed a problem, including system components that are affected or involved?	□ Yes □ No □ Partially
	3. Was the defect report described the main usability issue involved in the problem? For example, the description about what is wrong with the interaction architecture, interface and user task design.	□ Yes □ No □ Partially
	4. Was the defect report explained in detail steps by steps how to reproduce the problem, including user's navigation flow through the system?	□ Yes □ No □ Partially
	5. Are you able to reproduce the problem on your own device and environment?	□ Yes □ No □ Partially
	6. Were the actual results you observed similar with the one in the defect descriptions?	□ Yes □ No □ Partially
	7. Was the defect report justified why the reporter thinks that it was a problem?	□ Yes □ No □ Partially
	8. Was the defect report explicitly mentioned about what poses a problem to the user?	□ Yes □ No □ Partially
	9. Was the defect report described the impact of the problem on business effect, impact on the user's task and importance of the task?	□ Yes □ No □ Partially
	10. Was the defect report described reporters emotion, feeling or reactions upon the issues?	□ Yes □ No □ Partially
	11. Was the defect report mentioned about how often the problem occurred?	□ Yes □ No □ Partially

Appendix F

Ethics Clearance

E.1 **SUHREC Project 2014/231 Ethics Clearance**

To: Prof John Grundy, FSET/Mrs Nor Shahida Mohamad Yusop

Dear Prof Grundy,

SHR Project 2014/231 What makes a good usability defect report?
Prof John Grundy, FSET/Mrs Nor Shahida Mohamad Yusop
Approved Duration: 21/10/2014 to 31/07/2017 [Adjusted]

I refer to the ethical review of the above project protocol by a Subcommittee (SHESC1) of Swinburne's Human Research Ethics Committee (SUHREC) at a meeting held 19 September 2014. Your responses to the review, as emailed on 1 and 20 October were reviewed by a SHESC1 delegate.

I am pleased to advise that, as submitted to date, the project may proceed in line with standard on-going ethics clearance conditions here outlined.

- All human research activity undertaken under Swinburne auspices must conform to Swinburne and external regulatory standards, including the current *National Statement on Ethical Conduct in Human Research* and with respect to secure data use, retention and disposal.

- The named Swinburne Chief Investigator/Supervisor remains responsible for any personnel appointed to or associated with the project being made aware of ethics clearance conditions, including research and consent procedures or instruments approved. Any change in chief investigator/supervisor requires timely notification and SUHREC endorsement.

- The above project has been approved as submitted for ethical review by or on behalf of SUHREC. Amendments to approved procedures or instruments ordinarily require prior ethical appraisal/clearance. SUHREC must be notified immediately or as soon as possible thereafter of (a) any serious or unexpected adverse effects on participants any redress measures; (b) proposed changes in protocols; and (c) unforeseen events which might affect continued ethical acceptability of the project.

- At a minimum, an annual report on the progress of the project is required as well as at the conclusion (or abandonment) of the project. Information on project monitoring, self-audits and progress reports can be found at: http://www.research.swinburne.edu.au/ethics/human/monitoringReportingChanges/

- A duly authorized external or internal audit of the project may be undertaken at any time.

Please contact the Research Ethics Office if you have any queries about on-going ethics clearance. The SHR project number should be quoted in communication. Researchers should retain a copy of this email as part of project recordkeeping.

Best wishes for the project.

Yours sincerely,

Kaye Goldenberg
Acting Secretary, SHESC1
Research Ethics Executive Officer (Acting)
Swinburne Research (H68)
Swinburne University of Technology
Level 1, SPS, 24 Wakefield Street
Hawthorn, VIC 3122
Tel: +61 3 9214 5218
Fax: +61 3 9214 5267
Email: kgoldenberg@swin.edu.au

E.2 SUHREC Project 2016/325 Ethics Clearance

To: A/Prof Jean-Guy Schneider, FSET

SHR Project 2016/325 - Evaluation of a New Usability Defect Classification
A/Prof Jean-Guy Schneider, Nor Shahida Mohamad Yusop (Student) – FSET/Prof John Grundy, Prof Rajesh Vasa – Deakin University
Approved duration: 20-01-2017 to 20-06-2017 [Adjusted]

I refer to the ethical review of the above project by a Subcommittee (SHESC3) of Swinburne's Human Research Ethics Committee (SUHREC). Your response to the review as e-mailed on 20 January 2017 was put to the Subcommittee delegate for consideration.

I am pleased to advise that, as submitted to date, ethics clearance has been given for the above project to proceed in line with standard on-going ethics clearance conditions outlined below.

- The approved duration is 20-01-2017 to 20-06-2017 unless an extension request is subsequently approved.

- All human research activity undertaken under Swinburne auspices must conform to Swinburne and external regulatory standards, including the *National Statement on Ethical Conduct in Human Research* and with respect to secure data use, retention and disposal.

- The named Swinburne Chief Investigator/Supervisor remains responsible for any personnel appointed to or associated with the project being made aware of ethics clearance conditions, including research and consent procedures or instruments approved. Any change in chief investigator/supervisor, and addition or removal of other personnel/students from the project, requires timely notification and SUHREC endorsement.

- The above project has been approved as submitted for ethical review by or on behalf of SUHREC. Amendments to approved procedures or instruments ordinarily require prior ethical appraisal/clearance. SUHREC must be notified immediately or as soon as possible thereafter of (a) any serious or unexpected adverse effects on participants and any redress measures; (b) proposed changes in protocols; and (c) unforeseen events which might affect continued ethical acceptability of the project.

- At a minimum, an annual report on the progress of the project is required as well as at the conclusion (or abandonment) of the project. Information on project monitoring and variations/additions, self-audits and progress reports can be found on the Research Internet pages.

- A duly authorised external or internal audit of the project may be undertaken at any time.

Please contact the Research Ethics Office if you have any queries about on-going ethics clearance, citing the Swinburne project number. A copy of this e-mail should be retained as part of project record keeping.

Best wishes for the project.

Yours sincerely,

Sally Fried

Secretary, SHESC3

References

[1] A. Fernandez, E. Insfran, and S. Abrahão, “Usability evaluation methods for the web: A systematic mapping study,” *Information and Software Technology*, vol. 53, no. 8, pp. 789–817, 2011.

[2] M. Theofanos and W. Quesenbery, “Towards the Design of Effective Formative Test Reports,” *Journal of Usability Studies*, vol. 1, no. 1, pp. 27–45, 2005.

[3] T. S. Andre, H. Rex Hartson, S. M. Belz, and F. a. Mccreary, “The user action framework: a reliable foundation for usability engineering support tools,” *International Journal on Human-Computer Studies*, vol. 54, pp. 107–136, 2001.

[4] A. J. Ko and P. K. Chilana, “How power users help and hinder open bug reporting,” in *Proceedings of the 28th international conference on Human factors in computing systems*, 2010, p. 1665.

[5] A. Raza, L. F. Capretz, and F. Ahmed, “Improvement of Open Source Software Usability: An Empirical Evaluation from Developers’ Perspective,” *Advances in Software Enginering*, vol. 2010, pp. 1–12, 2010.

[6] L. Zhao and F. P. Deek, “Improving Open Source Software Usability,” in *Proceeedings of the Eleventh Americas Conference on Information Systems*, 2005.

[7] G. Çetin, D. Verzulli, and S. Frings, “An Analysis of Involvement of HCI Experts in Distributed Software Development: Practical Issues,” *Online Communities and Social Computing*, vol. 4564, pp. 32–40, 2007.

[8] R. T. Høegh and J. Stage, “The Impact of Usability Reports and User Test Observations on Developers ’ Understanding of Usability Data : An Exploratory Study,” *International Journal of Human-Computer Interaction*, vol. 21, no. 2, pp. 173–196, 2006.

[9] K. Hornbaek and E. Frokjaer, “What Kinds of Usability-Problem Description are Useful to Developers?,” in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 2006, vol. 50, no. 24, pp. 2523–2527.

[10] M. B. Twidale and D. M. Nichols, “Exploring Usability Discussions in Open Source

Development," in *Proceedings of the 38th Annual Hawaii Internatioal Conference on System Sciences*, 2005, pp. 1–10.

[11] M. G. Capra, "Usability Problem Description and the Evaluator Effect in Usability Testing," . PhD Thesis. Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 2006.

[12] S. L. Keenan, H. R. Hartson, D. G. Kafura, and R. S. Schulman, "The Usability Problem Taxonomy : A Framework for Classification and Analysis," *Empirical Software Engineering*, vol. 4, pp. 71–104, 1999.

[13] T. S. Andre, S. M. Belz, F. A. McCrearys, and H. R. Hartson, "Testing a Framework for Reliable Classification of Usability Problems," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 2000, vol. 44, no. 37, pp. 573–576.

[14] M. Hertzum and N. E. Jacobsen, "The Evaluator Effect: A Chilling Fact About Usability Evaluation Methods," *International Journal of Human-Computer Interaction*, vol. 15, pp. 183–204, 2003.

[15] R. Molich, R. Jeffries, and J. S. Dumas, "Making Usability Recommendations Useful and Usable," *Journal of Usability Studies*, vol. 2, no. 4, pp. 162–179, 2007.

[16] B. J. S. Dumas, B. R. Molich, and B. R. Jeffries, "Describing usability problems: Are we sending the right message?," *Interactions*, pp. 0–4, 2004.

[17] N. S. M. Yusop, J. Grundy, and R. Vasa, "Reporting Usability Defects : Limitations of Open Source Defect Repositories and Suggestions for Improvement," in *Proceedings of the 24th Australasian Software Engineering Conference*, 2015, pp. 38–43.

[18] T. Zimmermann and S. Breu, "Improving Bug Tracking Systems," in *31st International Conference on Software Engineering - Companion Volume*, 2009, pp. 247–250.

[19] A. Bruun, P. Gull, L. Hofmeister, and J. Stage, "Let Your Users Do the Testing : A Comparison of Three Remote Asynchronous Usability Testing Methods," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2009, pp. 1619–1628.

[20] T. Zimmermann, R. Premraj, N. Bettenburg, C. Weiss, S. Just, and A. Schro, "What Makes a Good Bug Report ?," *IEEE Transactions on Software Engineering*, vol. 36, no. 5, pp. 618–643, 2010.

[21] E. I. Laukkanen and M. V. Mantyla, “Survey Reproduction of Defect Reporting in Industrial Software Development,” in *International Symposium on Empirical Software Engineering and Measurement*, 2011, pp. 197–206.

[22] D. M. Nichols and M. B. Twidale, “Usability processes in open source projects,” *Software Process Improvement and Practice*, vol. 11, no. 2, pp. 149–162, 2006.

[23] C. Wilson and K. P. Coyne, “The whiteboard: Tracking usability issues: to bug or not to bug?,” *Interactions*, pp. 15–19, 2001.

[24] F. P. Simões, “Supporting End User Reporting of HCI Issues in Open Source Software,” PhD Thesis, Pontificia Universidade Catolica, Do Rio De Janeiro, 2013.

[25] C. Sun, D. Lo, X. Wang, J. Jiang, and S. Khoo, “A Discriminative Model Approach for Accurate Duplicate Bug Report Retrieval,” in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, 2010, pp. 45–54.

[26] S. Banerjee, Z. Syed, J. Helmick, and B. Cukic, “A Fusion Approach for Classifying Duplicate Problem Reports,” in *IEEE 24th International Symposium on Software Reliability Engineering*, 2013, pp. 208–217.

[27] Y. Tian, D. Lo, and C. Sun, “Information Retrieval Based Nearest Neighbor Classification for Fine-Grained Bug Severity Prediction,” in *19th Working Conference on Reverse Engineering*, 2012, pp. 215–224.

[28] J. Xuan, “Developer Prioritization in Bug Repositories,” in *34th International Conference on Software Engineering*, 2012, pp. 25–35.

[29] P. Bhattacharya, L. Ulanova, I. Neamtiu, and S. C. Koduru, “An empirical analysis of bug reports and bug fixing in open source Android apps,” in *Proceedings of the European Conference on Software Maintenance and Reengineering*, 2013, pp. 133–143.

[30] B. a. Kitchenham, P. Brereton, M. Turner, M. K. Niazi, S. Linkman, R. Pretorius, and D. Budgen, “Refining the systematic literature review process—two participant-observer case studies,” *Empirical Software Engineering*, vol. 15, no. 6, pp. 618–653, 2010.

[31] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, “Systematic Mapping Studies in Software Engineering,” in *Proceedings of the 12th international conference on Evaluation and*

Assessment in Software Engineering, 2008, pp. 68–77.

[32] J. D. Strate and P. a. Laplante, “A Literature Review of Research in Software Defect Reporting,” *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 444–454, 2013.

[33] S. Davies and M. Roper, “What’s in a bug report?,” in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2014, pp. 1–10.

[34] S. Zaman, B. Adams, and A. E. Hassan, “Security Versus Performance Bugs : A Case Study on Firefox,” in *Proceedings of the 8th Working Conference on Mining Software Repositories*, 2011.

[35] A. Nistor, T. Jiang, and L. Tan, “Discovering, reporting, and fixing performance bugs,” in *10th Working Conference on Mining Software Repositories*, 2013, pp. 237–246.

[36] S. Zaman, B. Adams, and a. E. Hassan, “A qualitative study on performance bugs,” in *9th IEEE Working Conference on Mining Software Repositories*, 2012, pp. 199–208.

[37] L. Tan, C. Liu, Z. Li, X. Wang, Y. Zhou, and C. Zhai, “Bug characteristics in open source software,” *Empirical Software Engineering*, pp. 1–41, 2013.

[38] S. Lal and A. Sureka, “Comparison of Seven Bug Report Types: A Case-Study of Google Chrome Browser Project,” in *19th Asia-Pacific Software Engineering Conference*, 2012, pp. 517–526.

[39] A. J. Ko, B. A. Myers, and D. H. Chau, “A Linguistic Analysis of How People Describe Software Problems,” in *Visual Languages and Human-Centric Computing*, 2006, pp. 127–134.

[40] P. K. Chilana, A. J. Ko, and J. O. Wobbrock, “Understanding expressions of unwanted behaviors in open bug reporting,” in *IEEE Symposium on Visual Languages and Human-Centric Computing*, 2010, pp. 203–206.

[41] B. Dit and A. Marcus, “Improving the Readability of Defect Reports,” in *Proceedings of the International Workshop on Recommendation System for Software Engineering*, 2008, pp. 47–49.

[42] R. V. Lotufo, “Towards Next Generation Bug Tracking Systems,” PhD Thesis. University of

Waterloo, Canada, 2013.

[43] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, and D. Poshyvanyk, “Auto-completing Bug Reports for Android Applications,” in *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 673–686.

[44] M. Terry, M. Kay, and B. Lafreniere, “Perceptions and Practices of Usability in the Free/Open Source Software (FOSS) Community,” in *Proceedings of the 28th international conference on Human factors in computing systems*, 2010, pp. 1–10.

[45] A. Raza, L. F. Capretz, and F. Ahmed, “Usability bugs in open-source software and online forums,” *IET Software*, vol. 6, p. 226, 2012.

[46] R. Lotufo and K. Czarnecki, “Improving Bug Report Comprehension,”. Technical Report. Generative Software Development Laboratory, University of Waterloo, Canada, 2012.

[47] A. Raza, L. F. Capretz, and F. Ahmed, “Improvement of Open Source Software Usability: An Empirical Evaluation from Developers’ Perspective,” *Advances in Software Engineering*, vol. 2010, pp. 1–12, 2010.

[48] J. Howarth, T. Smith-jackson, and R. Hartson, “Supporting novice usability practitioners with usability engineering tools,” *International Journal Human-Computer Studies*, vol. 67, no. 6, pp. 533–549, 2009.

[49] J. Feiner and K. Andrews, “Usability Reporting with UsabML,” in *Proceedings of the 4th international conference on Human-Centered Software Engineering*, 2012, vol. 7623, pp. 342–351.

[50] A. Faaborg and D. Schwartz, “Using a Distributed Heuristic Evaluation to Improve the Usability of Open Source Software,” in *28th ACM Conference on Human Factors in Computing Systems*, 2010.

[51] B. Kitchenham and S. Charters, “Guidelines for performing Systematic Literature Reviews in Software Engineering,”. Technical Report. Keele University, United Kingdom, 2007.

[52] M. Petticrew and H. Roberts, “Systematic Reviews in the Social Sciences: A Practical Guide,” *Blackwell Publishing*, pp. 164–214, 2006.

[53] P. Mclnerney, C. Pantel, and K. Melder, “Managing Usability Defects from Identification to Closure,” in *Extended Abstracts on Human Factors in Computing Systems*, 2001, pp. 497–498.

[54] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, “Systematic literature reviews in software engineering – A systematic literature review,” *Information and Software Technology*, vol. 51, no. 1, pp. 7–15, 2009.

[55] D. Maplesden, E. Tempero, J. Hosking, and J. C. Grundy, “Performance Analysis for Object-Oriented Software: A Systematic Mapping,” *IEEE Transactions on Software Engineering*, vol. 41, pp. 691–710, 2015.

[56] N. Salleh, E. Mendes, and J. Grundy, “Empirical Studies of Pair Programming for CS/SE Teaching in Higher Education: A Systematic Literature Review,” *IEEE Trans. Softw. Eng.*, vol. 37, no. 4, pp. 509–525, Jul. 2011.

[57] P. Achimugu, A. Selamat, R. Ibrahim, and M. Naz, “A systematic literature review of software requirements prioritization research,” *Information and Software Technology*, vol. 56, no. 6, pp. 568–585, 2014.

[58] G. S. Walia and J. C. Carver, “A systematic literature review to identify and classify software requirement errors,” *Information and Software Technology*, vol. 51, no. 7, pp. 1087–1109, Jul. 2009.

[59] B. A. Kitchenham, E. Mendes, and G. H. Travassos, “Cross versus within-company cost estimation studies: A systematic review,” *IEEE Transactions on Software Engineering*, vol. 33, pp. 316–329, 2007.

[60] M. Nørgaard and R. T. Høegh, “Evaluating Usability – Using Models of Argumentation to Improve Persuasiveness of Usability Feedback,” in *Proceedings of the 7th ACM conference on Designing interactive systems*, 2008, pp. 212–221.

[61] E. T. Hvannberg, E. L.-C. Law, and M. K. Lárusdóttir, “Heuristic evaluation: Comparing ways of finding and reporting usability problems,” *Interacting with Computers*, vol. 19, no. 2, pp. 225–240, 2007.

[62] A. Bruun and J. Stage, “Barefoot usability evaluations,” *Behaviour & Information Technology,* vol. 33, no. 11, pp. 1148–1167, Feb. 2014.

[63] M. G. Capra, “Comparing Usability Problem Identification and Description by Practitioners and Students,” in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 2007.

[64] Y. C. Cavalcanti, P. A. da Mota Silveira Neto, D. Lucrédio, T. Vale, E. S. de Almeida, and S. R. de Lemos Meira, “The bug report duplication problem: An exploratory study,” *Software Quality Journal*, vol. 21, pp. 39–66, 2013.

[65] F. Botella and A. Peñalver, “A new proposal for improving heuristic evaluation reports performed by novice evaluators,” in *Proceedings of the Chilean Conference on Human - Computer Interaction*, 2013, pp. 72–75.

[66] J. Matejka, T. Grossman, and G. Fitzmaurice, “IP-QAT: In-Product Questions, Answers, & Tips,” in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, 2011, pp. 175–184.

[67] P. K. Chilana, A. J. Ko, and J. O. Wobbrock, “LemonAid : Selection-Based Crowdsourced Contextual Help for Web Applications,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2012, pp. 1549–1558.

[68] I. Herraiz, D. M. German, M. Jesus, U. Rey, and J. Carlos, “Towards a Simplification of the Bug Report form in Eclipse,” in *Mining Software Repositories*, 2008, pp. 145–148.

[69] D. Lavery, G. Cockton, and M. P. Atkinson, “Comparison of evaluation methods using structured usability problem reports,” *Behaviour & Information Technology*, vol. 16. pp. 246–266, 1997.

[70] T. S. Andre, H. R. Hartson, and R. C. Williges, “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 Ergonomics Society*, 2003.

[71] G. Cockton, A. Woolrych, and M. Hindmarch, “Reconditioned Merchandise : Extended Structured Report Formats in Usability Inspection,” in *Extended Abstracts on Human Factors in Computing Systems*, 2004, pp. 1433–1436.

[72] G. A. Bowen, “Document Analysis as a Qualitative Research Method,” *Qualitative Research Journal,* vol. 9, no. 2, pp. 27–40, 2009.

[73] A. Følstad, P. O. Box, E. L. Law, K. Hornbæk, and S. Copenhagen, “Analysis in Practical Usability Evaluation : A Survey Study,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2012, pp. 2127–2136.

[74] J. Itkonen and C. Lassenius, “The Role of the Tester ’ s Knowledge in Exploratory Software Testing,” *IEEE Transactions on Software Engineering*, vol. 39, no. 5, pp. 707–724, 2013.

[75] G. Guest, K. MacQueen, and E. Namey, *Introduction to applied thematic analysis*. London, UK: Sage, 2012.

[76] V. Lelli, A. Blouin, and B. Baudry, “Classifying and qualifying GUI defects,” in *IEEE 8th International Conference on Software Testing, Verification and Validation*, 2015.

[77] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, B. K. Ray, and D. S. Moebus, “Orthogonal Defect Classification - A Concept for In-Process Measurements,” *IEEE Transactions on Software Engineering*, vol. 18, no. 11, pp. 943–956, 1992.

[78] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, “An Approach to Detecting Duplicate Bug Reports using Natural Language and Execution Information,” in *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 461–470.

[79] Y. Tao, “Grammatical analysis of user interface events for task identification,” *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8517 LNCS, pp. 197–205, 2014.

[80] J. Itkonen and K. Rautiainen, “Exploratory testing: a multiple case study,” in *IEEE/ACM International Symposium on Empirical Software Engineering and Measurement*, vol. 0, pp. 82–91.

[81] C. R. L. Neto and E. S. de Almeida, “Five years of lessons learned from the Software Engineering course: Adapting best practices for Distributed Software Development,” in *Second International Workshop on Collaborative Teaching of Globally Distributed Software Development*, 2012, pp. 6–10.

[82] V. Kettunen, J. Kasurinen, O. Taipale, and K. Smolander, “A study on agility and testing processes in software organizations,” *Human Factors*, pp. 231–240, 2010.

[83] T. Kanij, R. Merkel, and J. Grundy, “A preliminary survey of factors affecting software

testers," in *Proceedings of the Australian Software Engineering Conference*, 2014, pp. 180–189.

[84] A. Beer, A.- Vienna, and R. Ramler, "The Role of Experience in Software Testing Practice," in *34th Euromicro Conference Software Engineering and Advanced Applications*, 2008, pp. 258–265.

[85] P. L. Poon, T. H. Tse, S. F. Tang, and F. C. Kuo, "Contributions of tester experience and a checklist guideline to the identification of categories and choices for software testing," *Software Quality Journal*, vol. 19, pp. 141–163, 2011.

[86] A. F∅lstad, "Work-Domain Experts as Evaluators: Usability Inspection of Domain-Specific Work-Support Systems," *International Journal of Human Computer Interaction*, vol. 22, pp. 217–245, 2007.

[87] T. Kanij, R. Merkel, and J. Grundy, "A Preliminary Study on Factors Affecting Software Testing Team Performance," in *International Symposium on Empirical Software Engineering and Measurement*, 2011, pp. 359–362.

[88] M. V. Mäntylä, J. Itkonen, and J. Iivonen, "Who tested my software? Testing as an organizationally cross-cutting activity," *Software Quality Journal*, vol. 20, no. 1, pp. 145–172, 2012.

[89] V. Kettunen, J. Kasurinen, O. Taipale, and K. Smolander, "A study on agility and testing processes in software organizations," *Human Factors*, pp. 231–240, 2010.

[90] S. Herbold, J. Grabowski, S. Waack, and U. Bünting, "Improved Bug Reporting and Reproduction through Non-intrusive GUI Usage Monitoring and Automated Replaying," in *IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, 2011, pp. 232–241.

[91] T. Roehm, N. Gurbanova, B. Bruegge, C. Joubert, and W. Maalej, "Monitoring user interactions for supporting failure reproduction," in *21st International Conference on Program Comprehension*, 2013, pp. 73–82.

[92] J. Pichler and R. Ramler, "How to test the intangible properties of graphical user interfaces?," in *Proceedings of the 1st International Conference on Software Testing, Verification and*

Validation, 2008, pp. 494–497.

[93] V. Arnicane, *Use of Non-IT Testers in Software Development*, vol. 4589. 2007.

[94] D.-H. Ham, “A model-based framework for classifying and diagnosing usability problems,” *Cognition, Technology & Work*, vol. 16, pp. 373–388, 2014.

[95] R. Geng, M. Chen, and J. Tian, “In-process Usability Problem Classification, Analysis and Improvement,” in *the 14th International Conference on Quality Software*, 2014, pp. 240–245.

[96] J. Nielsen and R. Molich, “Heuristic evaluation of user interfaces,” in *Proceedings of the SIGCHI conference on Human factors in computing systems Empowering people*, 1990, pp. 249–256.

[97] D. A. Norman, “Cognitive engineering,” *User centered System Design*, pp. 31–61, 1986.

[98] R. Khajouei, L. W. P. Peute, a. Hasman, and M. W. M. Jaspers, “Classification and prioritization of usability problems using an augmented classification scheme,” *Journal Biomedical Informatics*, vol. 44, no. 6, pp. 948–957, 2011.

[99] S. G. Vilbergsdóttir and E. L. Law, “Classification of Usability Problems (CUP) Scheme : Augmentation and Exploitation,” in *Proceedings of the 4th Nordic conference on Human-computer interaction: changing roles*, 2006, pp. 14–18.

[100] A. Faaborg and D. Schwartz, “Using a Distributed Heuristic Evaluation to Improve the Usability of Open Source Software,” in *CHI Conference on Human Factors in Computing Systems*, 2010, pp. 4–5.

[101] J. R. Wood and L. E. Wood, “Card Sorting: Current practices and beyond,” *Journal of Usability Studies*, vol. 4, no. 1, pp. 1–6, 2008.

[102] S. G. Vilbergsdottir, E. T. Hvannberg, and E. L. C. Law, “Assessing the reliability, validity and acceptance of a classification scheme of usability problems (CUP),” *Journal of System and Software*, vol. 87, pp. 18–37, 2014.

[103] V. Harkke and P. Reijonen, “Are We Testing Utility ? Analysis of Usability Problem Types,” in *Design, User Experience, and Usability: Design Discourse*, 2015.

[104] R. W. Reeder and R. A. Maxion, “User interface defect detection by hesitation analysis,” in

Proceedings of the International Conference on Dependable Systems and Networks, 2006, pp. 61–70.

[105] J. L. Fleiss, “Measuring nominal scale agreement among many raters,” *Psychological Bulletin*, vol. 76, no. 5, pp. 378–382, 1971.

[106] J. Cohen, “A coefficient of agreement for nominal scales,” *Educational and Psychological Measurement*, vol. 20, pp. 37–46, 1960.

[107] M. L. McHugh, “Interrater reliability : the kappa statistic,” *Biochemica Medica*, vol. 22, no. 3, pp. 276–282, 2012.

[108] K. Hornbæk and E. Frokjær, “Comparing usability problems and redesign proposals as input to practical systems development,” in *CHI 2005: Technology, Safety, Community: Conference Proceedings - Conference on Human Factors in Computing Systems*, 2005, pp. 391–400.

[109] D. M. Hilbert and D. F. Redmiles, “Extracting usability information from user interface events,” *ACM Computing Surveys*, vol. 32, pp. 384–421, 2000.

[110] S. Hedegaard and J. G. Simonsen, “Extracting usability and user experience information from online user reviews,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2013, pp. 2089–2098.

[111] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, “Extracting structural information from bug reports,” in *International workshop on Mining Software Repositories*, 2008, pp. 27–30.

[112] R. Souza, C. Chavez, and R. Bittencourt, “Patterns for extracting high level information from bug reports,” in *1st International Workshop on Data Analysis Patterns in Software Engineering*, 2013, pp. 29–31.

[113] E. L. Law and E. T. Hvannberg, “Consolidating Usability Problems with Novice Evaluators,” in *Proceedings of the 5th Nordic conference on Human-computer interaction: building bridges*, 2008, pp. 495–498.

[114] N. S. M. Yusop, J. Grundy, and R. Vasa, “Reporting Usability Defects: A Systematic Literature Review,” *IEEE Transactions on Software Engineering*, vol. 43, no. 9, pp. 848–867, 2017.

[115] N. S. M. Yusop, J. Grundy, and R. Vasa, “Reporting Usability Defects – Do Reporters Report What Software Developers Need ?,” in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, 2016.

[116] N. S. M. Yusop, J.-G. Schneider, J. Grundy, and R. Vasa, “What Influences Usability Defect Reporting ? – A Survey of Software Development Practitioners,” in *23rd Asia-Pasific Software Engineering Conference*, 2016.

[117] N. S. M. Yusop, J.-G. Schneider, J. Grundy, and R. Vasa, “Analysis of the Textual Content of Mined Open Source Usability Defect Reports,” in *24th Asia-Pasific Software Engineering Conference*, 2017.

[118] M. Torchiano, D. M. Fernández, G. H. Travassos, and R. M. de Mello, “Lessons Learnt in Conducting Survey Research,”. in *Proceedings of the 5th International Workshop on Conducting Empirical Studies in Industry*, 2017.

[119] J. Su, P. Shao, and J. Fang, “Effect of Incentives on Web-Based Surveys,” *Tsinghua Science and Technology*, vol. 13, no. 3, pp. 344–347, 2008.

[120] E. Singer and C. Ye, “The Use and Effects of Incentives in Surveys,” The *Annals of the American Academy of Political and Social Science*, vol. 645, no. 1, pp. 112–141, 2013.

[121] J. M. Fang and P. J. Shao, “The effect of material incentives on web survey completion: Evidence from three meta-analyses,” in *2010 International Conference on Management Science and Engineering*, 2010, pp. 75–81.

[122] H. Hedberg, N. Iivari, M. Rajanen, and L. Harjumaa, “Assuring Quality and Usability in Open Soruce Software Development,” in *First International Workshop on Emerging Trends in FLOSS Research and Development*, 2007, pp. 1–5.

[123] D. I. K. Sjoberg, T. Dyba, M. Jorgensen, D. I. K. Sjøberg, T. Dybå, and M. Jørgensen, “The Future of Empirical Methods in Software Engineering Research,” *Future of Software Engineering*, vol. SE-13, no. 1325, pp. 358–378, 2007.

[124] B. Kitchenham and S. L. Pfleeger, “Principles of survey research part 4: questionnaire evaluation,” in *ACM SIGSOFT Software Engineering Notes*, 2002, vol. 27, no. 3, p. 20.

[125] K. Petersen and C. Wohlin, “Context in industrial software engineering research,” in *3rd*

International Symposium on Empirical Software Engineering and Measurement, 2009, pp. 401–404.

[126] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2008.

[127] T. Kanij, R. Merkel, and J. Grundy, “An empirical study of the effects of personality on software testing,” in *26th International Conference on Software Engineering Education and Training*, 2013, pp. 239–248.

[128] L. Marks, Y. Zou, and A. E. Hassan, “Studying the Fix-Time for Bugs in Large Open Source Projects Categories and Subject Descriptors,” in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, 2011.

[129] E. Giger, M. Pinzger, and H. C. Gall, “Predicting the fix time of bugs,” in *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*, 2010, pp. 52–56.

[130] L. D. Panjer, “Predicting Eclipse Bug Lifetimes,” in *Fourth International Workshop on Mining Software Repositories*, 2007.

[131] P. Bhattacharya and I. Neamtiu, “Bug-fix Time Prediction Models : Can We Do Better ?,” in *Proceedings of the 8th Working Conference on Mining Software Repositories*, 2011, pp. 207–210.

[132] P. Hooimeijer and W. Weimer, “Modeling Bug Report Quality,” in *Proceedings of the 22nd IEEE/ACM international conference on Automated software engineering*, 2007, pp. 34–43.

[133] M. Kamalrudin, J. Grundy, and J. Hosking, “Supporting requirements modelling in the Malay language using essential use cases,” in *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing*, 2012, pp. 153–156.

[134] J. Hall, “Usability Themes in Open Source Software,”. PhD Thesis. University of Minnesota, 2014.