to appear in Information and Software Technology (c) Elsevier 2020

A Revised Open Source Usability Defect Classification Taxonomy

Nor Shahida Mohamad Yusop[1], John Grundy[2], Jean-Guy Schneider[3] and Rajesh Vasa[3]

[1]Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA Selangor, Malaysia
nor_shahida@fskm.uitm.edu.my

[2]Faculty of Information Technology Monash University Melbourne, Australia
{john.grundy}@monash.edu

[3]Faculty of Science, Engineering and Built Environment Deakin University Geelong, Australia
{jeanguy.schneider, rajesh.vasa}@deakin.edu.au

Abstract

Context: Reporting usability defects is a critical part of improving software. Accurately classifying these reported usability defects is critical for reporting, understanding, triaging, prioritizing and ultimately fixing such defects. However, existing usability defect classification taxonomies have several limitations when used for open source software (OSS) development. This includes incomplete coverage of usability defect problems, unclear criticality of defects, lack of formal usability training of most OSS defect reporters and developers, and inconsistent terminology and descriptions.

Objective: To address this gap, as part of our wider usability defect reporting research, we have developed a new usability defect taxonomy specifically designed for use on OSS projects.

Method: We used Usability Problem Taxonomy (UPT) to classify 377 usability defect reports from Mozilla Thunderbird, Firefox for Android, and the Eclipse Platform. At the same time, we also used the card-sorting technique to group defects that could not be classified using UPT. We looked for commonalities and similarities to further group the defects within each category as well as across categories.

Results: We constructed a new taxonomy for classifying OSS usability defects, called Open Source Usability Defect Classification (OSUDC). OSUDC was developed by incorporating software engineering and usability engineering needs to make it feasible to be used in open source software development. The use of the taxonomy has been validated on five real cases of usability defects. However, evaluation results using the OSUDC were only moderately successful.

Conclusion: The OSUDC serves as a common vocabulary to describe and classify usability defects with respect to graphical user interface issues. It may help software developers to better understand usability defects and prioritize them accordingly. For researchers, the OSUDC will be helpful when investigating both trends of usability defect types and understanding the root cause of usability defect problems.

Keywords: usability defect reporting, usability defect taxonomy, empirical evaluation, open source software development

1 Introduction

Usability is one of the prominent software quality characteristics that measures the understandability, learnability, operability and attractiveness of the software products [1]. In the context of community open source software in which no specific software development processes were carried out, usability activities are often ignored. Volunteers are more focused on functionality and features rather than appearance, design aesthetic, and how people will use the products [2]. As a result, open source projects often have poor interfaces and complex user interaction [2], [3].

Since usability is a key acceptance criterion of a software product, usability-related issues need to be reported. In this work, a usability issue is defined as any *unintended behaviour* by the product that is *noticed by the user* and has an *effect on user experience*. For example, consider a search job that uses a lot of computer resources. If the effect of high memory usage is only noticeable by software developers, then we consider this problem to be a performance defect. But, if a user experiences the slowness of retrieving the search results and is frustrated by a delay, in addition to performance it also affects usability.

However, reporting usability defects can be a challenging task, especially in convincing developers that the usability issue is indeed a real defect. The subjective nature of a usability issue in addressing a user's feelings, emotions, and struggling requires a mutual definition so that developers do not misinterpret the key information. In open source project development where most volunteers are "non usability – savvy" and work for limited time, a list of usability keywords and options to classify usability defects is a helpful solution to directly point to the causes and solutions [4]. Currently, existing defect repositories, such as Bugzilla, have used keyword functionality to label usability-related defects. For instance, a defect can be labelled 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, the lack of descriptions, examples and limited usability terms make it difficult for non-expert Human Computer Interaction evaluators to assign such labels for certain

usability defects [5]. Moreover, a recent literature review exploring usability defect reporting practices has discovered that usability defect reporting processes suffer from a number of limitations, including mixed data, inconsistency of terms and values of usability defect data, and insufficient attributes to classify usability defects [6]. These limitations encouraged us to revise existing usability defect classification models to produce a model suitable for the OSS domain. 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.

Based on our analysis of open source usability defect reports, we integrated and revised some existing usability defect classification models [8]–[10] to better incorporate Software Engineering and usability engineering needs. We also obtained and evaluated feedback on our new proposed open source usability defect classification model by requesting software development practitioners and novice users to classify a sample of usability defects. From an analysis of these classifications, we identified several strengths and weaknesses in our approach. In this paper, we report on the design and empirical evaluation of our new OSS usability defect taxonomy. Key contributions of this work include:

- a revised OSS usability defect taxonomy to classify usability defects in OSS environment that have limited usability engineering training and practices; and
- an evaluation of the taxonomy by practitioners to understand its strengths and weaknesses.

The rest of this paper is organized as follows. In Section 2, we describe an overview of usability defect classification schemes from the usability and software engineering disciplines. Section 3 follows with the rationale for revising existing usability defect classification schemes in open source software development. Section 4 explains the research process and methodology to construct the taxonomy. In Section 5, we elaborate our new usability defect classification model. We present our approach to evaluate the model in Section 6, and the evaluation results are presented in Section 7 and Section 8, respectively. We outline threats to validity in Sections 9 and we discuss some important issues in Section 10. The paper concludes with a summary, implications, and future work in Section 11.

2 Existing Usability Defect Classification Schemes

Research on usability defect classification is often studied in the field of human computer interaction (HCI). Earlier efforts to classify usability defects were done by Nielsen [7]. Nielsen refined the nine heuristics he identified earlier using factor analysis of 249 usability problems to derive a set of heuristics with maximum explanatory power, resulting in a revised set of 10 heuristics. Since Nielsen's heuristics only offer a high-level classification that only considers high level views of difficulties users encountered with the user interface, there are several limitations with using it to classify usability problems, as reported in [8]: 1) insufficient distinguishability, 2) lack of mutual exclusiveness, 3) incompleteness, and 4) lack of specificity.

To overcome these limitations, Keenan et al. [8] developed the Usability Problem Taxonomy (UPT) that classifies usability defects into artefact and task components. The artefact 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/labelling, and other wording. In the second level, other wording is further categorized into feedback messages, error messages, other system messages, 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.

However, Keenan's approach to classification relies on a high quality defect description which, as our earlier work demonstrates [9], 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. [10] have expanded the UPT to include other usability engineering support methods and tools. By adapting and extending Norman's [11] theory of action model, they developed Usability Action Framework (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.

Meanwhile, in ISO/IEC 25010 standard the quality of software product can be measured using eight characteristics (further divided into sub characteristics) – functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability. In the context of product usability, ISO/IEC 25010 defines usability as appropriateness recognisability, learnability, operability, user error protection, user interface aesthetics, and accessibility. However, the results from [12] indicated that the classification of defects using main characteristics

and sub characteristics were not reliable due to the limited information present in the defect reports. With little information, the functionality and usability issues were difficult to distinguish.

However, since 2000, many researchers have started to actively use software engineering classification models to classify usability defects. One of the most prominent approaches is the adoption of a cause-effect model. For example, Vilbergsdottir et al. [13] 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 detail. 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. Although the Post-CUP is useful for defects triaging, in which similar issues can be mapped into specific fixes, we postulate that some of the attributes in Pre-CUP are not relevant for novice OSS reporters to report informative usability defect descriptions. For example, technical information about defect removal activity, failure qualifier, expected phase, and frequency are difficult to obtain, especially for those who have limited usability-technical knowledge.

Khajouei et al. [14] argued that the lack of information on the effects of usability defects in UAF will cause a long discussion to convince developers of the validity of the usability defects. 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.

Although Geng et al. [15] agreed that CUP can capture important usability defect information and provide feedback for usability software, CUP could not be used to analyse the effect on users and task performance. Considering the importance of the cause – effect relationship, they have customized the ODC and UPT, as shown in Figure 1. They developed cause-effect usability problem classification model that consists of three causal attributes (artefact, task, and trigger) and four effects attributes (learning, performing, perception, and severity). However, in the absence of formal usability evaluation in OSS projects, the trigger attribute as suggested in the model cannot be sufficiently justified. 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 [16]. This model-based framework consists of three perspectives, in which each perspective is facilitated by the use of models: artefact-users-tasks-organization-situation model for Context of Use, abstraction hierarchy model for Design Knowledge, and function-behaviour-structure model for Design Activities - in which the usability problem needs to be analysed 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 analysed from the Design Knowledge and Design Activities perspectives. Such a reference framework allows usability evaluators to develop a specific classification scheme for a context. However, poor involvement of usability evaluators in OSS projects makes it rather 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 [17].

Several other related work support usability-related issues by focusing on GUI defects and functionality. Examples include the GUI fault model [18], which categorize GUI defects into interface and interaction defects, and Harkke et al. [19] classified usability defects into 11 categories – missing, misinterpreted, positive, inadequate, unexplored, misplaced, unnecessary, technical deficiencies, problematic change of work practice, preferenced, and misaligned.

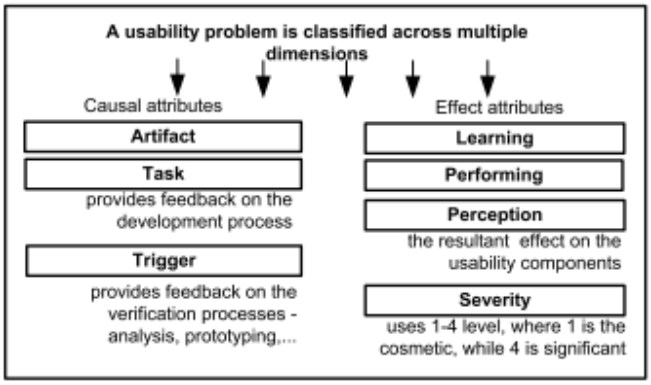

Figure 1. Geng's cause-effect usability problem classification model [15].

3 Rationale for Revising Existing Usability Defect Classification Schemes

From a software engineering perspective, cause-effect classification models 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 [15], in our view, is not appropriate to classify open source usability defects that often contain limited information [20]–[22]. The trigger component in the causal attributes can be limited. This is because in the absence of formal usability evaluations in OSS development, it is impossible to identify the usability evaluation methods that trigger usability defects.

Even if formal usability evaluations were to be conducted, OSS projects would still lack an effective mechanism to conduct the evaluations, mainly for two reasons. First, many of the volunteers who contribute to OSS development are developers, who generally 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 the time on this.

Considering all of these limitations, we revised Geng's classification [15] to better suit an OSS environment and adapted some elements of the ODC framework to address cause and effect attributes. In the following paragraphs we summarize the rationale for our revisions.

Defect category - In software development, quantitative measurements such as the amount of memory used, the time to load an 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 can 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 can learn about the heuristics and understand how the same types of defects could be resolved. However, current usability principles being used by Bugzilla's keyword functionality are too broad [23]. Some keywords are hard to distinguish and may lead to incorrect interpretation. Consider the following Bugzilla keywords and definitions:[1]

Ux-affordance – controls should visually express how the user should interact with them.

Ux-discovery – users should be able to discover functionality and information visually exploring the interface, they should not be forced to recall information from memory. (This is often the nemesis of ux-minimalism since additional visible items diminish the relative visibility of other items being displayed)

Based on these definitions, the two keywords refer to the ability of users to recognize and understand possible actions based on visual cues of user interface. The unclear separation between the keywords can lead to misclassification of defects that will eventually affect the identification of root cause and similar resolution strategies. In fact, the single perspective classification as used by Bugzilla is not relevant for classifying usability issues that often consist of graphical user interface and action issues. In this regard, a taxonomic classification such as UPT is a recommended approach to classify usability defects from an artefact and task aspect, respectively.

Effect – Previous studies have reported that usability defects are treated at a lower priority compared to functional defects [24]. 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 [24]. From our analysis of open source defect reports [9], 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.

4 Research Process and Methodology

The OSUDC taxonomy was created following a three-phase process and influenced by the design science methodology [32]: 1) problem identification, 2) artefact design, and 3) validation.

In the problem identification phase, we reviewed several usability defect classification models in the literature, in particular UPT [8], ODC [25], GUI fault model [18] and usability-ODC framework [15]. 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 with limited usability knowledge.

In contrast to the previous classifications [8], [15], 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 defect reports. In fact, in open source usability defect reports, we could not identify testing techniques used by the users that triggered the usability defect. For these reasons, we revised the previous classification models [8], [15], [18] to only consider information normally available in the open source defect reports.

In the artefact defect phase, we adapted the original ODC framework to better understand usability defect causes and effects and integrated it with usability practices. Figure 2 illustrates our high-level cause-effect usability defect classification model. We collected 377 usability defects from Mozilla Thunderbird, Firefox for Android, and Eclipse Platform projects. In constructing a list of potential metrics to be used for ODC attributes (cause and effect attributes), the first author began by randomly reading a few samples of defect reports in detail, trying to understand the defect content structure. Most of the defect descriptions that were read contained significant information about defect reproduction, actual results (i.e. what is wrong with the

[1] https://bugzilla.mozilla.org/describekeywords.cgi#ux-affordance

usability defects), expected results (i.e. what should be fixed) and user difficulty expressions.

Based on this common information, *usability defect categories* and *failure qualifier* were included as ODC-cause attributes and *user difficulty* as ODC-effect attribute in our first draft of a 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 resolution 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 analyse 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 [26]. 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 [8], [15], and failure qualifier attributes [27]. While in an open card sort no predefined groups were used, instead the groups 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 authors read half of the sample independently, applying the final classification model, and consulted the categories definitions and terminology until a consensus was reached. We reached agreement collaboratively as disagreements arose.

In the validation phase, we conducted a case survey methodology [33] and we summarize our findings in Section 7. We focused on investigating the ease of use and understandability of the OSUDC taxonomy rather than the effectiveness and accuracy of the classification.

5 The OSUDC Taxonomy version 1.0

The OSUDC taxonomy provides a structured way of characterizing usability defects for open source software through three top-level elements, namely *defect categories*, *failure qualifier* and *user difficulty*. These elements are further refined to provide a common terminology that can be used by researchers and practitioners to report and classify usability defects. The remainder of this section provides an overview of the three elements, including the changes and additions to the original UPT.

5.1 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.

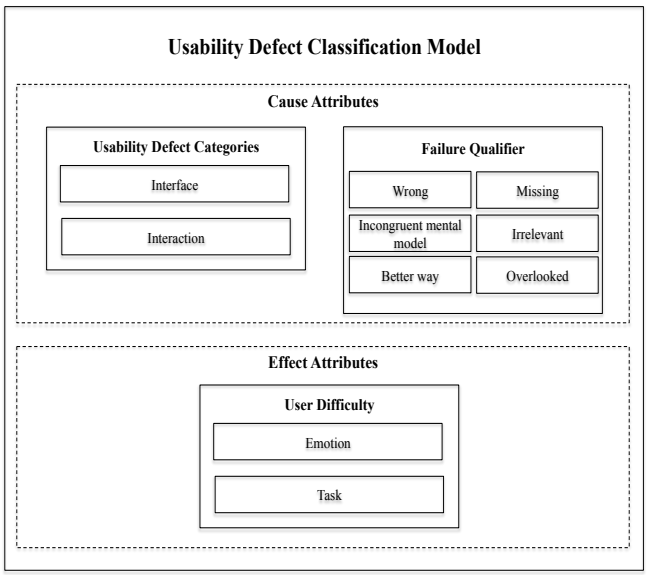

Figure 2. OSUDC Taxonomy based on [15]

5.2 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.

5.2.1 Usability Defect Categories

To classify usability defect categories, we used a closed card sort. We began by classifying the written usability defect description to a set of predefined usability defect categories as in [8], [15]. Since we experienced some difficulties – lack of specificity and insufficient definition when using [8], [15], 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 [18]. Interface defects refer to defects affecting the structure and behaviour 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 "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 - Visualness, information presentation, and audibleness. Visualness refers to GUI presentation, such as harmonious colours, object affordance and layout coherence, respectively. 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 the "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 [18], 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 [15] 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 [8]. We adapted four subcategories as in the ODC-usability framework – keyboard press, mouse click, finger touch, and voice control. We added three subcategories; scrolling mechanism, drag and drop, and zooming to support touch-based interaction for touch screen-based devices.
- Task execution focuses on the outcome of certain tasks. We adapted three subcategories as in the fault model [18] – 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 "feedback" subcategory.
- Functionality refers to any problem due to the capabilities provided by the product. We reused functionality definitions from [19] – missing functionality, misinterpreted functionality, inadequate functionality, misplaced functionality, unnecessary functionality, technical deficiencies, preference functionality, and misaligned functionality.

The resulting model is illustrated in detail in Figure 3, and a definition of each category used in the model is summarized in Table 1. The detailed categories, subcategories, and examples of such usability defects can be found in our online supplementary materials: http://bit.ly/Material_for_OSUDC.

5.2.2 Trigger

Another aspect of ODC 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 [25], *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 analysing trigger attributes from defect description we used *failure qualifier* from ODC as a set of predefined metrics.

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 through the observation from the recorded user test session [13]. For instance, the usability evaluator can ask the test participants why they did not notice the presence of a menu or if they think some elements on the user interface are missing. On the contrary, our approach determines the failure qualifier based on a statement written in the defect report. To reflect this, we have refined the definitions of the failure qualifier attributes to suit our usage, as shown in Table 2. We assume that this ODC failure qualifier would be a good ground on which to base the users' justification on how they discovered the usability defects.

5.3 Effect Attribute

In many defect classification models [15], [25], [27], 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 [24], we argue that defect severity is not a reliable metric for analysing usability defects for software quality improvement.

In Geng's classification [15], the effect attribute is studied from the perspective of three components – problem in learning, problem in performing given tasks, and user perception. Each of these components has multiple sub-components. For example, the learning component contains three sub-components - learnability, memorability, and retention over time. The performance component has two sub-components - 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 software development we recognize that the impact the defect has on the user and likelihood of occurrence is important to prioritize defect fixes. However, previous research has found that such information is rarely present in defect reports, or if it is reported, the information is not clear [9], [20], [21]. For this reason, we examined the effect of usability defects as the user difficulty, in terms of *human emotion* and *task performance* only.

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 first author reviewed statements and

generated labels of emotion and task difficulty, then merged and sorted the lists into meaningful descriptive labels, and created a set of codes. In 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 present in defect reports, we analysed the phrases such as [28]:

- "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 …"

Table 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), symbol, 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 colours, 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 behaviour 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 labelling	Any difficulty in language such as words/ terminology used as names on objects (such as buttons, title bars, field labels) and screens [8]. These problems also include inconsistencies of naming and labelling standard.
Non-message feedback	Any difficulty that is due to distracting, annoying, and confusing feedback [8], 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 menus 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 encountered when the user has difficulty to use mouse to click, including left/ right mouse clicking, double clicking
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.
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.
Missing	An element of the system that is necessary for the users' work is not available at all. The task/ work cannot be performed with the presented system. The functionality has not been implemented.
Misinterpreted	Refer to the terminology or symbols/ functions are not correctly understood by the user. One may think about a different meaning of the symbol, feature, function or information from the designed purpose.
Inadequate	A required element of the system is present, but the implementation is not sufficient for the task at hand. Functionality has been designed and implemented into the system, but it lacks a proper fit with the work and practices of users preventing or significantly hindering the performance.
Misplaced	The needed element is available and adequate but in a cumbersome format or requires unnecessary effort to find and use. The feature is implemented somewhere or somehow, but not available at a required place and point in time.
Unnecessary	Users do not use, notice, behaviourally or verbally ignore a function or a piece of information that has been implemented.
Technical deficiencies	The system is implemented with error/ bugs.
Preferenced	A way of doing something in the system is preferred to an alternative way.
Misaligned	The feature would require a change in the work practice to be useful. The way in which functionality is meant to be used differs from the existing or traditional use.

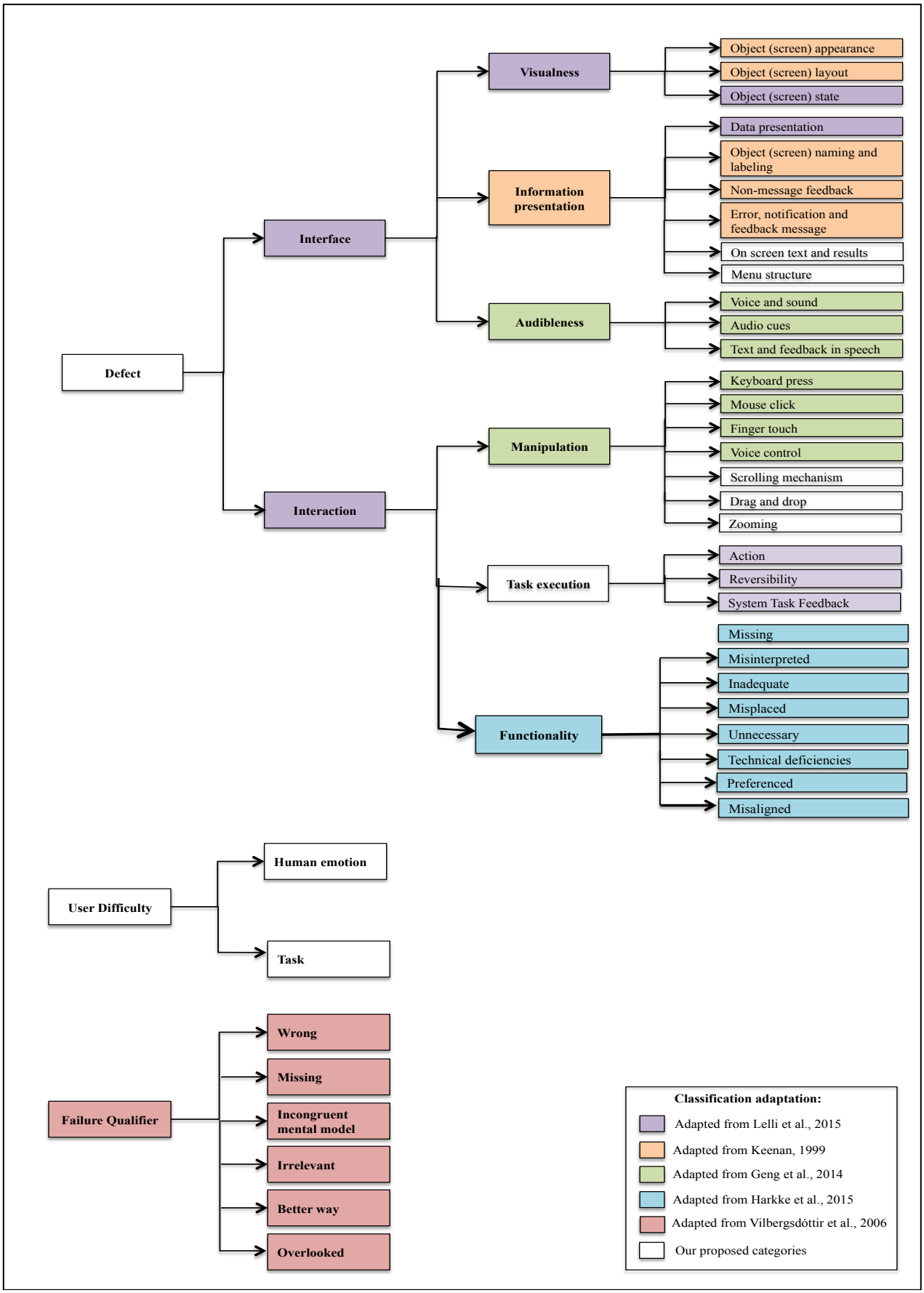

Figure 3. Hierarchical structure of defect types, effect and failure qualifier. The colours indicate the different source we adapted in our classification model.

We started the examination with the Mozilla Thunderbird dataset. Once the Mozilla Thunderbird dataset was classified, the process was repeated with the Firefox for Android and Eclipse Platform datasets. If there were inconsistencies, the draft codes were modified and refined again. Finally, the appropriateness of the resulting codes and definitions were thoroughly discussed. 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 3 and Table 4 define each of these user difficulty attributes and list some example phrases from open source defect reports.

Table 2. Failure Qualifier – Sample phrases from usability defect reports [13]

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 **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.
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.
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.

Table 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 cool items 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 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 OSUDC Evaluation

The goal of this evaluation is to verify the readability and understandability of our proposed usability defect reporting categories. In particular, the outcome of the evaluation focuses on the key aspects of ease of learning, ease of use, completeness, and clarity of the proposed terms and definitions, rather than just the effectiveness of the taxonomy to accurately classify usability defects. We evaluated the OSUDC taxonomy with respondents from various backgrounds. In order to do this, we used a web-based survey as a tool of evaluation. The following subsections describe evaluator selection, problem selection, and the protocol used in conducting the OSUDC evaluation.

6.1 Evaluator Selection

Our evaluators were recruited from the researchers' industrial contacts and students. The evaluators had varying levels of experience in industry and academic software development environments. Participation was voluntary and evaluators could 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).

6.2 Problem Selection

We randomly selected five usability defects (five reports from Eclipse Platform, three reports from Mozilla Thunderbird, and two reports from Firefox for Android) from the 377 usability defects that had been examined during the analysis phase prior to building the revised open source usability defect classification.

6.3 Protocol

We evaluated the classification model using self-administered evaluation survey designed in Google form survey (https://forms.gle/mCP7YbgcysT88uEu6). The survey was first conducted in May 2017. However, due to a low number of responses (12 evaluators), we reopened the survey in January 2020. In the survey, 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 had three sections. In the first section, the evaluators were required to fill out a small questionnaire about personal background. This pre-questionnaire contains a total of six questions. In the second section, evaluators were given 5 usability defects to be analysed according to the OSUDC taxonomy. Each usability defect contains a total of four to six questions depending on an 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 required to read five defect reports and assess the types of defects, as well as identify the presence of information about emotion, task difficulty, and failure qualifier. The categories of the defect type, emotion, task difficulty, and failure qualifier components of our OSUDC taxonomy are shown in Figure 3. When assessing the defect reports, the evaluator can choose more than one category for emotion and task difficulty component. This is because some usability issues, such as getting a feedback pop-up window appear in the middle of a task, may cause distraction and increase annoyance. The evaluators were not monitored and could classify the problems in any order, revisiting any problems they wished. There was no time limit imposed on the evaluators.

7 Results

7.1 Evaluator Demographic Information

A total of 41 evaluators from 26 to 55 years of age participated in the evaluation of the OSUDC taxonomy. As shown in Table 5, most of the evaluators are computing students and academic researchers, accounting for 48.8% and 29.3%, respectively. Almost 80% of the evaluators had received training or certification related to usability evaluation/ HCI/ UX. However, as indicated in Table 6, the majority of evaluators had limited familiarity in handling usability defects.

Table 5. Demographic information of the evaluators

Job responsibility	Evaluators
Academic researcher	29.3%
Computing students (undergraduate/ postgraduate)	48.8%
Software developer with experience in both user interface and software development	12.2%
HCI/ UX/ usability expert	7.3%
End user with HCI/ UX/ usability knowledge	2.4%

Table 6. Evaluators' familiarity with usability defects

Extremely familiar	2.4%
Moderately familiar	26.8%
Somewhat familiar	29.3%
Slightly familiar	41.5%
Not at all familiar	0.0%

7.2 Fleiss' Kappa Analysis

To measure the reliability of evaluators' agreement to classify usability defect reports using the OSUDC taxonomy, we used Fleiss' kappa [29]. 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[2] installed in an Excel spreadsheet to calculate the Fleiss' kappa values.

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

Table 7. Overall kappa for defect category, emotion, task difficulty, and failure qualifier

	Overall Kappa (K)	Asymptotic Standard Error	Z	P Value	Lower 95% Asymptotic CI Bound	Upper 95% Asymptotic CI Bound
Defect Category	**0.304**	**0.011**	**27.066**	**0.000**	**0.282**	**0.326**
Emotion	0.008	0.008	0.938	0.348	-0.009	0.024
Task Difficulty	0.021	0.007	2.882	0.004	0.007	0.035
Failure Qualifier	0.042	0.008	5.536	0.000	0.027	0.057

Table 8: Agreement in interface classification

Report	Total agreement in subcategories									Total Number of Evaluators
	Visualness			*Information Presentation*						
	Appearance	*Layout*	*State*	*Data presentation*	*Error, notification and message feedback*	*Non message feedback*	*Object naming and labelling*	*Menu structure*	*On screen text results*	
1	**6**	**22**	**5**	**0**	**0**	**0**	**2**	**1**	**0**	**36**
2	6	7	2	0	0	0	0	4	1	20
3	**25**	**4**	**0**	**3**	**3**	**1**	**1**	**1**	**2**	**40**
4	1	1	1	3	0	0	1	1	0	8
5	2	3	3	4	1	0	0	3	2	18

Table 9: Agreement in interaction classification

Report	Total agreement in subcategories											Total Number of Evaluators
	Manipulation							*Task execution*			*Functionality*	
	Keyboard Press	*Mouse click*	*Finger touch*	*Scrolling mechanism*	*Drag and drop*	*Zooming*	*Voice control*	*Action*	*Reversibility*	*Feedback*		
1	0	5	1	3	1	6	0	0	0	0	2	18
2	**0**	**15**	**0**	**12**	**3**	**0**	**1**	**2**	**1**	**1**	**1**	**36**
3	0	0	1	1	0	0	0	1	0	0	0	3
4	**15**	**1**	**4**	**1**	**0**	**0**	**0**	**11**	**1**	**0**	**4**	**37**
5	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**21**	**2**	**1**	**7**	**33**

For the classification of the defect category component, kappa was computed at the primary category level only (interface or interaction). Since the number of observations within each primary category varied, analysis at the subcategory level would have invalidated the kappa values. Landis and Koch [30] suggested that the Kappa result be interpreted as follows: values < 0 indicate no agreement, 0.00 - 0.20 poor agreement, 0.21-0.40 fair agreement, 0.41 – 0.60 moderate agreement, 0.61-0.80 good agreement, and 0.81 – 1.00 strong agreement.

The Fleiss' kappa results for each OSUDC component are reported in Table 7. As can be seen, **there was a fair agreement** between the evaluators' assessment on the *defect category* component, with a kappa value of 0.304 and a 95% confidence interval (CI) between 0.282 and 0.326. As for agreement on the *emotion, task difficulty*, and *failure qualifier* components, the Fleiss' kappa (K) < 0.200 represents **poor strength of agreement**. Reasons for this may be our choice of too many values defined for *emotion, task difficulty*, and *failure qualifier* component that adversely influenced the results, as reported in [27]. 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.

In addition to Fleiss' kappa analysis for the overall levels of agreement, we also inspected the classification data from a non-statistical perspective. We discovered that more than 30 evaluators agreed on report#1 and report#2 as interface problem and report#2, report#4, and report#5 as interaction problem. Table 8 shows the number of evaluators who classified the five reports in the same subcategories in the interface component. As can be seen, among the 40 evaluators who agreed that report#3 as interface problem, 25 of them agreed that issues described in defect report #3 are related to visual appearance aspect. For defect report #1, from the 36 evaluators, 22 of them agreed it as a layout problem.

Similarly, the entries in Table 9 are the number of evaluators who classified the five reports in the same subcategories in the interaction component. Although more than 35 evaluators agreed to classify report#2 and report#4 as interaction problems, only 15 of the evaluators agreeing that the problem was related to mouse click and keyboard press issue, respectively. For defect report #5, 21 out of 33 evaluators agreed that the issue is due to defective action.

8 Experiences and Feedback

This section presents and discusses the results from the post-evaluation questionnaire filled out by the evaluators at the end of our survey. The post-evaluation questionnaire had one closed question and one open-ended question. The closed question was measured on a 5-point Likert scale using the satisfaction-based statements as follows:

- Easy to learn – the degree to which an evaluator is satisfied that the OSUDC is easy to be learned with no training or demonstration

- Easy to use – the degree to which an evaluator is satisfied that the OSUDC is simple, user friendly, and flexible to be used
- Completeness – the degree to which an evaluator is satisfied that the 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 OSUDC are clearly written so that it is easily understandable

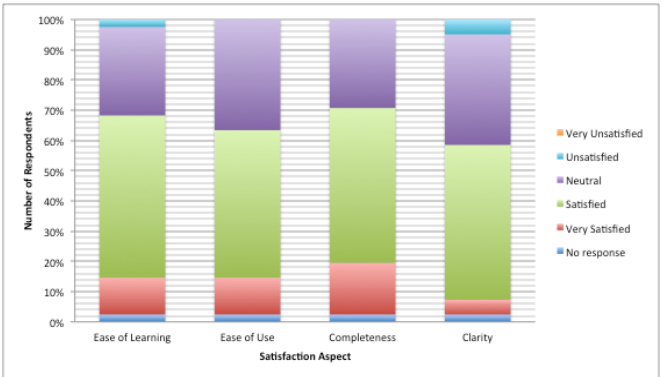

Figure 4. Responses on the four satisfaction aspects

The responses are depicted in Figure 4. Based on the "Very satisfied" and "satisfied" rating, we see that about 60% of the evaluators were satisfied with the OSUDC. However, some evaluators' comments indicate difficulty to understand the meaning of the terms and differentiate some options. This can be seen from the following comments:

"A bit confusing. Maybe should have some visualization examples for sample defects"

"Seems great. I'm not familiar with other approaches so it might be miles better than others or just a small step forward. It was difficult to decide on some options"

"Some examples to explain the taxonomy would be helpful"

"It would be helpful if the explanation in the classification scheme were explained in a layman term"

"Quite complex for non-technical users"

"The learning information should be in more detail"

Among the four satisfaction aspects, only one evaluator was dissatisfied with the ease of learning and clarity of OSUDC. These evaluators expressed their dissatisfaction comments in the accompanying open-ended question, as below:

"The model took a long time to get used to – due to uncertainty about the best category to select, it took me approximately 2 hours to classify the 5 projects – but I got quicker towards the end, so it will probably be okay for someone using it frequently"

Concerning the ease-of-use aspect, fifteen evaluators rated neutral in satisfaction. Possibly, the use of the Google survey as a medium of evaluation had a negative effect on the evaluator's experience. Switching back and forth between the Google form and the OSUDC reference document can increase the sense of annoyance because the numerous defect categories are unintuitive, making the understanding and selection of an appropriate category more difficult. If using a self-developed classification 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 expressed 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."

9 Threats to Validity

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

9.1 External Validity

One possible threat to the external validity of this validation survey is generalization of the findings. There are two factors to consider. First, the background of the participants in this survey that have limited knowledge and experience in usability defects. As shown in Table 5, only one respondent claimed to be very familiar with usability defects, while 23 respondents were moderately familiar (based on their self-assessed moderate and somewhat familiar ratings). Therefore, the understandability level about usability defects is unclear overall. 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. 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.

9.2 Internal Validity

The selection of usability defect reports is a key threat to internal validity. In our study we analysed 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 if this limitation may not be fully representative of other OSS projects. However, the elements identified in OSUDC cover a vast spectrum of elements from both usability (adaptation of UPT) and software (adaptation of ODC) engineering domains. Hence, we can stipulate that our OSUDC elements are generalizable and reflective of OSS projects in general.

9.3 Construct Validity

Construct validity concerns the relation between the studied concept and theory behind it. To mitigate this threat, we have used multiple sources of information in constructing the OSUDC, such as systematic literature review [6], international standards, and industrial practitioner opinions.

Thus, the solution is formulated from a wide spectrum of sources.

9.4 Conclusion Validity

One concern is regarding the misinterpretation of terms and cases when the researchers analysed the 377 usability defect reports. This creates a risk of bias for the categories and elements proposed in the OSUDC that could be mitigated by involving practitioners and usability experts instead of solely academic researchers. Although some evaluators involved in the OSUDC validation have industrial experiences, their understanding of the 5 cases is limited by simply reading the usability defect descriptions than direct involvement in finding the defects. It can be difficult for someone to select a category and give reason for selecting it if he/she was not directly involved in the discovery of the defects. Therefore, it remains future work to have open source communities and practitioners who are directly involved in the defect reporting process apply the OSUDC.

10 Discussion

For the purpose of practical usability defect reporting in conjunction with our proposed OSUDC, we recommend four characteristics for capturing usability defects:

1. State the types 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.

The feedback we obtained from the post-questionnaires provides a good insight into the needs of non-technical users when analysing and understanding usability defects. Especially in OSS project 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.

Although we received positive feedback from the majority of evaluators (based on the Satisfied and Very Satisfied rating in Figure 4), the classification result is disappointing. The level of agreement between our 41 evaluators was only fair for the *defect types* component, while agreement on *emotion, task difficulty* and *failure qualifier* component was poor. While fair agreement of overall defect type classification is certainly helpful, we expected there to be considerably more consistency between evaluators for most subtypes. This is because in defining our subtype categories and providing descriptions to users, we tried to be very clear about differences between each. In addition, for the evaluation we chose representative 5 example usability defect reports that were varied in type/subtype, with text descriptions being not too detailed but not very terse either.

We considered three possible key factors that affected the lack of significant results that we obtained for this evaluation. First, as pointed out by Keenan [8], poor quality of defect descriptions can 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], [32] 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 regards to *task difficulty*, results from our previous study, [9] indicate 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 that we used in the study, representative of the whole set of 377, possibly makes it still difficult for evaluators to interpret and classify the *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. We provided evaluators with our classification taxonomy and examples, but no specific training and demo of using it. Previous studies have demonstrated the necessity of initial training to increase users' familiarity and understanding of certain tools, aspects, and concepts [8], [27], [33]. We also acknowledge the need for more evaluator training, especially for novice usability evaluators, to help them better identify and rate usability defects. In future work, 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, respectively.

The third factor may be caused by the effect of having many relatively inexperienced or novice usability evaluators. As summarized in Table 5, nearly half of them were computing students with limited HCI knowledge. Closer analysis of these student respondents found that more than 90% of them never received any usability-related training, and less than 25% of them have used ODC, RCA or 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 fourth factor may be due to the use of insufficient examples and incomplete definitions of emotion, task difficulty, and failure qualifier components. Using manual classification during the development of OSUDC taxonomy, our analysis and interpretation might be inaccurate. Although the definitions have been reviewed and agreed by all the authors, the different interpretations from evaluators are inevitable, especially for those who first used this taxonomy. More research is needed to strengthen the model. For example, by using machine-learning tools such as Weka and RapidMiner, we hope that the glossary of terms for each category can be expanded and made more accurate.

11 Summary

This study presented the OSUDC taxonomy to classify and analyse usability defects. In the absence of formal usability evaluation in most OSS projects and limited information available in most usability defect descriptions, we revised the existing defect classification schemes to accommodate these limitations. We integrated the Geng's classification model and ODC framework to reflect the important element of classifying usability defects from the perspective of usability and software engineering. In our OSUDC, we introduced a cause-effect classification model that contains three main classification attributes, namely (1) usability defect categories, (2) failure qualifier, and (3) user difficulty.

The OSUDC was validated through an online survey. Overall, we obtained useful feedback and we refined the OSUDC based on the feedback received. Although the majority of evaluators have limited usability knowledge (refer Table 6), their feedback was important for us to understand the needs of such people who are not "usability experts" to be able to accurately classify usability defects. However, the feedback from the evaluators needs to be further explored in a further validation through interview sessions to get more details about their opinions, challenges, and difficulties when using the OSUDC taxonomy. We found some categories and values of the OSUDC to overlap and were found to be unclear to some of our evaluators. We thus refined the definitions of those categories to make them more understandable.

Despite our overall disappointing evaluation outcome around levels of agreement in classification of defect types and subtypes, we believe that our OSUDC can help software developers to better understand usability defects and prioritize them accordingly. For researchers, the 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.

Key future work is to further modify the taxonomy to clarify when to use each defect type and subtype, and provide users with more detailed examples of its application. In addition, we want to re-run the evaluations with more upfront training in using the taxonomy, and recruit more users with usability experience. We also want to try out our taxonomy with more open source usability defect developers and reporters, to provide further feedback on its refinement and explanation. Finally, we are using the OSUDC in a new OSS usability reporting tool under development [34]. We will also trial this with OSS developers and reporters to determine how effective the OSUDC is when used in this way.

Acknowledgements

Support for the first author from the Fundamental Research Grant Scheme (FRGS) under Contracts FRGS/1/2018/ICT 01/UITM/02/1, Universiti Teknologi MARA (UiTM), ARC Discovery Projects scheme project DP140102185 and Laureate Fellowship FL190100035, and from the Deakin Software and Technology Innovation Lab and Data61 for all authors, is gratefully acknowledged.

References

[1] I. Padayachee, "ISO 9126 external systems quality characteristics , sub- characteristics and domain specific criteria for evaluating e-Learning systems," in *The Southern African Computer Lecturers' Association, University of Pretoria, South Africa.*, 2010.

[2] A. Raza, L. F. Capretz, and F. Ahmed, "Improvement of Open Source Software Usability: An Empirical Evaluation from Developers' Perspective," *Adv. Softw. Eng.*, vol. 2010, pp. 1–12, 2010.

[3] L. Despalatović, "The Usability of Free / Libre / Open Source Projects," *Int. J. Comput. Inf. Technol.*, vol. 02, no. 05, pp. 958–963, 2013.

[4] 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.

[5] 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.

[6] N. S. M. Yusop, J. Grundy, and R. Vasa, "Reporting Usability Defects: A Systematic Literature Review," *IEEE Trans. Softw. Eng.*, vol. 43, no. 9, pp. 848–867, 2017.

[7] 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.

[8] S. L. Keenan, H. R. Hartson, D. G. Kafura, and R. S. Schulman, "The Usability Problem Taxonomy : A Framework for Classification and Analysis," *Empir. Softw. Eng.*, vol. 4, pp. 71–104, 1999.

[9] 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 (APSEC)*, 2017.

[10] 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," *Int. J. Hum. Comput. Stud.*, vol. 54, pp. 107–136, 2001.

[11] D. A. Norman, "Cognitive engineering," *User centered Syst. Des.*, pp. 31–61, 1986.

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

[13] 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.

[14] R. Khajouei, L. W. P. Peute, a. Hasman, and M. W. M. Jaspers, "Classification and prioritization of usability problems using an augmented classification scheme," *J. Biomed. Inform.*, vol. 44, no. 6, pp. 948–957, 2011.

[15] 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.

[16] D.-H. Ham, “A model-based framework for classifying and diagnosing usability problems,” *Cogn. Technol. Work*, vol. 16, pp. 373–388, 2014.

[17] G. Çetin, D. Verzulli, and S. Frings, “An Analysis of Involvement of HCI Experts in Distributed Software Development: Practical Issues,” *Online Communities Soc. Comput.*, vol. 4564, pp. 32–40, 2007.

[18] V. Lelli, A. Blouin, and B. Baudry, “Classifying and qualifying GUI defects,” in *IEEE 8th International Conference on Software Testing, Verification and Validation*, 2015.

[19] V. Harkke and P. Reijonen, “Are We Testing Utility ? Analysis of Usability Problem Types,” in *Design, User Experience, and Usability: Design Discourse*, 2015.

[20] T. Zimmermann, R. Premraj, N. Bettenburg, C. Weiss, S. Just, and A. Schro, “What Makes a Good Bug Report ?,” *IEEE Trans. Softw. Eng.*, vol. 36, no. 5, pp. 618–643, 2010.

[21] 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 - ESEM ’14*, 2014, pp. 1–10.

[22] D. M. Nichols and M. B. Twidale, “Usability processes in open source projects,” *Softw. Process Improv. Pract.*, vol. 11, no. 2, pp. 149–162, Mar. 2006.

[23] 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.

[24] C. Wilson and K. P. Coyne, “The whiteboard: Tracking usability issues: to bug or not to bug?,” *Interactions*, pp. 15–19, 2001.

[25] 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 Trans. Softw. Eng.*, vol. 18, no. 11, pp. 943–956, 1992.

[26] J. R. Wood and L. E. Wood, “Card Sorting: Current practices and beyond,” *J. Usability Stud.*, vol. 4, no. 1, pp. 1–6, 2008.

[27] 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),” *J. Syst. Softw.*, vol. 87, pp. 18–37, 2014.

[28] 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.

[29] J. L. Fleiss, “Measuring nominal scale agreement among many raters,” *Psychol. Bull.*, vol. 76, no. 5, pp. 378–382, 1971.

[30] J. R. Landis and G. G. Koch, “The Measurement of Observer Agreement for Categorical Data,” *Biometrics*, 1977.

[31] T. R. Nichols, P. M. Wisner, G. Cripe, and L. Gulabchand, “Putting the Kappa Statistic to Use,” *Qual. Assur. J.*, no. January 2010, pp. 64–65, 2010.

[32] 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, CSMR*, 2013, pp. 133–143.

[33] A. Bruun and J. Stage, “Barefoot usability evaluations,” *Behav. Inf. Technol.*, vol. 33, no. 11, pp. 1148–1167, Feb. 2014.

[34] N. S. M. Yusop, J. Grundy, J. G. Schneider, and R. Vasa, “Preliminary evaluation of a guided usability defect report form,” in *Proceedings - 25th Australasian Software Engineering Conference, ASWEC 2018*, 2018.