UNDERSTANDING AND IMPROVING REQUIREMENTS DISCOVERY IN OPEN SOURCE SOFTWARE DEVELOPMENT: AN INITIAL EXPLORATION

By

Jaison Kuriakose

A Dissertation submitted to the

School of Graduate Studies

In partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Faculty of Business Administration

Memorial University of Newfoundland

January, 2017

St. John's, Newfoundland and Labrador

ABSTRACT

In proprietary or closed source software (CSS) development, there is a formal requirements engineering (RE) phase for discovering the requirements for an application. The requirements engineering process in CSS development is comprised of many formal practices (e.g., elicitation/generation). With the advent of the Internet and web-based tools and technologies, a new and different form of software development has emerged – globally distributed, typically volunteer driven, open source software (OSS) development. OSS development largely occurs in an informal, ad hoc manner and often lacks the formal developmental practices and processes of CSS development. The goal of this research is to gain a better understanding of the current state of RE in OSS, to identify potential directions for improving RE in OSS, and to empirically investigate the potential of some specific RE practices to improve OSS development. In pursuit of the research goal, in the initial phase of this research a web-based survey of practicing OSS developers was conducted to explore the current state of RE in OSS. Results supported the claims about informality of RE in OSS. as well as pointed towards potential directions for improvement. In the second phase of the research, a web-based experiment was conducted to investigate the actual benefits from a particular CSS development requirements generation practice – requirements reuse (operationalized as the availability of a library of reusable requirements within OSS development environment) – for OSS development. Analysis of the experimental data revealed that that the experimental treatment (availability of a library of reusable requirements) had a *significant effect on the size of requirements message, requirements quantity and requirements completeness after controlling for covariates, indicating usefulness of the reusable library. The final phase of the research focused on OSS issue gathering approaches, a source of requirements for OSS. In this phase, a qualitative study of OSS developers explored how an OSS issue gathering approach, enforcing classification (versus free-form OSS issue gathering), may contribute to the misclassification problem (erroneous classification of OSS issues), and what can be done at the issue gathering interface level to mitigate the misclassification problem. Insights from the analysis of data from the final phase of the research shed light on the desirable characteristics that OSS issue gathering interfaces should possess for mitigating misclassification.*

ACKNOWLEDGEMENTS

I want to express my gratitude to Dr. Jeffrey Parsons for his valuable guidance and support during the entire course of my PhD program. I also want to thank Dr. Sherrie Komiak and Dr. Jennifer Jewer for their valuable feedback and suggestions on my dissertation research. I also want to thank Dr. Dennis Peters, Dr. Dale Foster and Dr. Joseph Feller for their valuable feedback and suggestions on my thesis. I also want to thank Faculty of Business Administration and Memorial University for the support provided for my dissertation research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVATIONS

OSS	**Open source software**
RE	**Requirements engineering**
CSS	**Closed source software**

CHAPTER ONE. INTRODUCTION

1.1 Background and Motivation

1.1.1 Open Source Software Development

The term *Open Source Software* (OSS) refers to software whose source code can be freely used, modified or redistributed (e.g., Linux, Mozilla Firefox). *OSS development* is an umbrella term used for denoting the activities carried out to develop such software (e.g., Crowston et al., 2012). OSS development is a specific case of what Prikladniciki et al. (2003) call multi-site, multi-cultural and globally distributed software development but which has the particular goal of delivering freely usable software with no or minimal license restrictions. OSS development can be characterized by several common features (Dietze 2005):

- Collaborative development
- Geographically distributed actors with diverse capabilities and qualifications
- Voluntary participation
- Interaction among actors entirely through web-based technologies
- Developmental activities carried out in parallel
- Dynamic releases
- Lack of centralized management
- Independent, crowd sourced peer-review.

There are several clearly observable differences between OSS development and proprietary/closed source software development (CSS development) (Mockus et al., 2002). While CSS development largely occurs in a centralized setting, OSS development largely occurs in a decentralized setting (e.g., Feller and Fitzgerald, 2000; Scacchi, 2006; Prikladniciki et al., 2003). CSS development involves salaried developers and project members; sharing a common organizational culture; and having rich formal and informal interactions, often facilitated by colocation working environments (e.g., Agerfalk et al., 2005). On the other hand, OSS development is largely carried out by volunteer developers and non-developer contributors who are often globally distributed and come from different organizational and cultural backgrounds (Crowston et al., 2012). These volunteering contributors, often working from arbitrary locations, collaborate almost entirely over the Internet using different tools and technologies, such as communication artifacts (e.g., discussion forums), source code management tools (e.g., Git, SVN), and issue reporting artifacts (e.g., BugZilla) (Detienne et al., 2006, Crowston et al., 2012). OSS developers are driven by different types of motivations, including the need for functionality, learning opportunities, career development, fun and altruistic intentions (Crowston et al., 2012; Feller and Fitzgerald, 2000; Shah, 2006; Hars and Ou, 2002). Many developers leave after contributing for a certain period, and a small group of developers remains to oversee the further evolution of the project (Shah, 2006). OSS projects are largely dependent on volunteering developers and being able to continuously attract and retain developers is important for their success (e.g., Crowston et al., 2003). In CSS development settings, the developers and project members are paid for their contribution to the project which can be expected to be a major motivation and the leaving-joining process can be expected to happen at a much lower rate in comparison to OSS projects. Also, CSS development projects are usually much more resource rich than OSS projects, and the exiting of team members can be expected to have a lower impact in comparison to OSS projects.

Distance between developers themselves and between developers and end-users is a major differentiating factor between CSS development and OSS development (c.f. Prikladniciki et al., 2003; Agerfalk et al., 2005). Distance involves not just the geographical distance but also the socio-cultural distance which emerges because of different individuals assigning different meanings to situations based on their socio-cultural backgrounds (c.f. Agerfalk et al., 2005). Individuals from different cultures may interpret and react to situations differently, and geographical distance contributes to the increase in social or cultural distance (Agerfalk et al., 2005). For example, an email from a sender belonging to a culture where communication is usually direct may be misunderstood as abrupt or rude by a recipient belonging to a different culture (Prikladniciki et al., 2003). Distance could also be temporal such as time zone differences (Agerfalk et al., 2005; Prikladniciki et al., 2003). For example, time zone differences between EU and India could mean very few overlapping hours between workers in EU and their counterparts in India and subsequently, high temporal distance (Agerfalk et al., 2005).

Geographical, temporal and socio-cultural distances give rise to many types of problems and challenges that OSS development may find more difficult to tackle (e.g., synchronization). The geographical and temporal distance could make it difficult to manage project artifacts (Agerfalk et al., 2005). Collocated environments (which are common in CSS development) facilitate rich informal face-to-face communication (e.g., coffee talk; coordinating by peeking around the cubicle wall) between team members that help build better-working relationships and allow better flow of project related information (Agerfalk et al., 2005; Prikladniciki et al., 2003). The geographical and temporal distance reduces chances of such communication; for example, time zone differences can be a barrier to a quick phone call to clarify issues on the fly (Agerfalk et al., 2005). Geographical distance can increase the effort needed to initiate contact which can result in scenarios such as developers proceeding with implementation without initiating contact with stakeholders, thus leading to errors (Agerfalk et al., 2005). The need to depend on technology for communication because of geographical and temporal distance can introduce its own problems; for example, different versions of tools posing a challenge in collaborating; issues with networks connecting geographically distributed locations (Agerfalk et al., 2005).

Temporal distance can lead to coordination problems (that OSS development may find more difficult to manage) such as reduced hours of collaboration. For example, individuals in US and Ireland can hope for at most three overlapping hours in a workday (Agerfalk et al., 2005). When efficient information sharing mechanisms are not in place (e.g., poor documentation) which is the case with a large number of OSS projects (Singh et al., 2009), geographical, temporal and socio-cultural distance can be major barriers in forming a shared understanding of situations at hand. For example, these distances could lead to inadequate information dispersion about the overall architecture vision of a project, which, in turn, could result in skewed perceptions about tasks to be done and ineffective collaboration (Agerfalk et al., 2005; Prikladniciki et al., 2003). The distances can also result in a lack of team spirit and awareness; for example, physical separation/lack of face to face contact resulting in a lack of awareness about the work activities of remote team members (Agerfalk et al., 2005). Also, it is more difficult to build and maintain trust among globally distributed project members in comparison to collocated project members (Agerfalk et al., 2005). These problems can be potentially challenging to manage in the OSS domain, given its constraints (e.g., lack of resources in comparison to CSS development).

OSS development largely occurs in an ad hoc, informal manner (Zhao and Elbaum, 2003) and OSS development communities typically do not have any formal organizational structure (Crowston et al., 2012). In contrast to CSS development, OSS development is mainly driven by volunteers (Mockus et al., 2002), OSS contributors often choose to do work that is of interest to them instead of work being assigned (Mockus et al., 2002; Detienne et al., 2006,) and ***there is often lack of system-level design, project plans, schedules and list of deliverables*** (Mockus et al., 2002; Detienne et al., 2006). OSS development often lacks the processes followed in CSS development (Mockus et al., 2002), and is marked instead by rapid development, frequent incremental releases, and parallel development and debugging (Feller and Fitzgerald, 2000). There are many web-based OSS development environments such as Sourceforge, Google Code, CodePlex and GitHub within which the OSS developmental activities are usually carried out. These development environments provide many types of tools (e.g., source code management tools, issue reporting artifacts, discussion forums) to support OSS development activities.

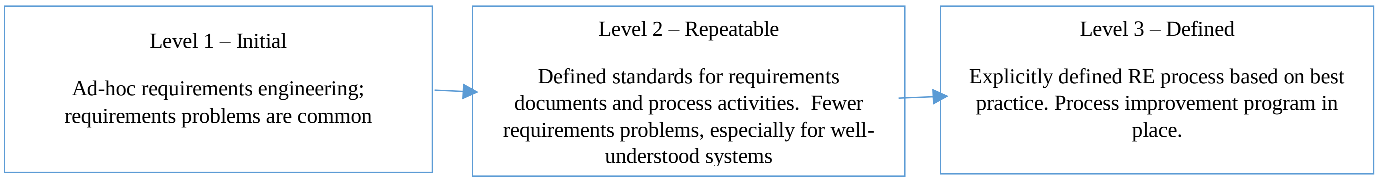
The often-observed lack of formal developmental practices in OSS domain is characteristic of its general informality/ lack of a project structure. Such a lack of project structure can potentially make many OSS projects less reliable and useful in comparison to CSS projects (Aksulu and Wade, 2010). This research aims to investigate the current state of and potential directions of improvement for requirements engineering in OSS development.

1.1.2 Requirements Discovery in Open Source Software Development

OSS development largely occurs in an ad hoc, informal manner (Zhao and Elbaum, 2003). In particular, requirements discovery in OSS development appears to be informal and ad hoc (e.g., Scacchi, 2002). Requirements describe the features and characteristics that software should possess and can be of type functional as well as non-functional (Dennis et al., 2012). In CSS development, there is a formal requirements engineering (RE) phase for discovering requirements. The RE process is comprised of many formal practices, including: requirements generation/elicitation; requirements analysis and negotiation; requirements validation; and requirements management (Sommerville and Swayer, 1997; Browne and Ramesh, 2002). On the other hand, OSS requirements discovery appear to be largely informal and devoid of any formal structure (Scacchi, 2002; Crowston et al., 2012; Vlas and Vlas, 2011). Existing OSS development environments such as Sourceforge, Google Code, CodePlex and GitHub also appear to be largely code centric and lacking support for carrying out many of the RE practices used in developing proprietary/closed source software.

Requirements in OSS projects often emerge informally as part of messages posted on communication artifacts, such as discussion forums and issue data submitted through issue reporting artifacts (Scacchi, 2002; Noll, 2008). Based on analysis of web-based artifacts (e.g., posted discussions) of a few OSS projects, the OSS literature reports a handful of informal requirements generation activities in OSS development, such as: assertion by developers; requirements data (e.g., feature requests) submitted though issue reporting artifacts by users; and deriving requirements from features appearing in other software (Noll, 2008; Noll and Liu, 2010). Dietze (2005) has called for improvements in requirements discovery practices in OSS development. There is an inherent tendency of OSS developers to focus on implementation activities, often leading to insufficient accomplishment of other developmental activities such as RE practices, an area needing improvement (e.g., in the form of defining additional and supportive processes) (Dietze, 2005). The need for improvement also becomes evident when the requirements process maturity model (Sommerville and Swayer, 1997) is used as a lens to analyze RE in OSS. Recent research (e.g., Cox et al., 2009) has empirically validated this model in industry. The model is shown below.

Figure 1. Requirements engineering process maturity levels (Sommerville and Swayer, 1997, p.22)


Level 1 – Initial
Ad-hoc requirements engineering; requirements problems are common
Level 2 – Repeatable
Defined standards for requirements documents and process activities. Fewer requirements problems, especially for well-understood systems
Level 3 – Defined
Explicitly defined RE process based on best practice. Process improvement program in place.

The model has three levels. In level one, there is no defined requirements engineering process, and there are many requirements related problems. There is no good quality requirements document produced and requirements generation is dependent on individual skills and experience. In level two, there are defined standards for requirements document and description, policies, and procedures for requirements management and use of some tools and techniques to support requirements engineering activities. In level three, there is a well-defined requirements engineering process in place (Sommerville and Swayer, 1997).

Requirements engineering in OSS development appears to be in level one, i.e., largely ad hoc. For example, as previously mentioned, requirements may originate and exist informally as part of some message posted on a discussion forum, but there is often no well specified formal requirements document (e.g., Scacchi, 2002). This informality goes against the founding principles of the formal requirements engineering (RE) process as stated by Bell and Thayer in 1976: "*The requirements for a system do not arise naturally; instead they need to be engineered and have continuing review and revision*" (p.5, Lamsweerde, 2000). Empirically, Hofmann and Lehner (2001) found that successful software development teams had to perform on average, three iterations of requirements engineering practices (e.g., identifying stakeholders, using modeling methods). As requirements for OSS projects are often simply asserted by developers (e.g., Noll, 2008) and there is a lack of formal elicitation of requirements information by analysts (Laurent and Cleland Huang, 2009), the quality and usefulness of requirements information generated for OSS projects may depend to a great extent on individual skills and abilities of OSS developers (e.g., their domain knowledge). This dependence is characteristic of level one in the RE maturity model. In fact, Rantalainen et al. (2011) report that, when OSS developers develop OSS software for which they (the developers) are potential users, they can be expected to have good domain knowledge but when the potential user base expands to include users other than the developers, the OSS developers may be lacking relevant domain knowledge (Rantalainen et al., 2011).

The RE process maturity model suggests that process improvement efforts should be undertaken when requirements engineering is at level one. OSS researchers in the past have mentioned the need for improvement in requirements discovery in OSS development (e.g., Dietze, 2005). Any process improvement initiative should involve efforts to understand existing processes and problems, identifying improvement goals and investigating ways and means of incorporating the improvements in existing workflows (c.f. Sommerville and Swayer, 1997). This research begins by investigating the current state of requirements discovery in OSS development to identify problems and challenges that exist in the context of requirements discovery in OSS. The RE process maturity model suggests that a direction for RE process improvement is the incorporation of more formal practices (e.g., defining a standard requirements document structure (Sommerville and Swayer, 1997)). Based on an analysis of forty successful and forty unsuccessful OSS projects, (Michlmayr, 2005) noted that OSS projects could benefit from the incorporation of mature processes. In fact, Ciolkowski and Soto writes:

> *"It is a widespread belief that OSS communities operate in an essentially chaotic way, and that, for this reason, no systematic development processes can be taking place during OSS development. Consequently, most casual observers would regard traditional maturity models as completely inappropriate for OSS software. We disagree with the previous idea. The main assumption underlying process assessment approaches is that more mature processes consistently lead to higher quality products, whereas for an organization with immature processes, the capacity to deliver high quality products is unreliable and cannot be predicted. There is no reason to believe this assumption is not valid for OSS"* (p.318, Ciolkowski and Soto, 2008)

This research follows the above line of thought and first carries out an exploratory investigation of whether the formal RE practices are perceived by OSS developers to be beneficial for OSS development. In doing so, this research attempt to uncover potential directions of improvement for requirements discovery in OSS development. This is important since recent research (Vlas, 2012) has found a positive association between quality of requirements data generated in artifacts such as discussion forums and OSS project success, indicating that good quality requirements can contribute to the success of OSS projects. Most OSS projects have been developed by developers for their own use (i.e., for the technical developet community (e.g., Foushee, 2013)). However, more recently the user base of OSS projects has expanded beyond the technical community, and taking the needs of non-developer users into account while developing OSS projects is critical for OSS project success (Choi and Chengalur-Smith, 2009). OSS projects are fast becoming components within IT infrastructure in various domains such as business, education and government (Terry et al., 2010). Hauge et al. (2008), in their survey of software companies, found that close to 50% of the companies surveyed integrated OSS components into their products. Nikula and Jantunen (2005), in their survey of business organizations, found that close to 50% of the surveyed organizations used OSS internally. For such adoption to happen efficiently, it is important that the OSS cover a major part of the requirements of the potential adopters (Ayala et al., 2013). Interestingly, such potential adopters often have to perform risk reduction activities (e.g., hiring an expert) before incorporating OSS (Ayala et al., 2013), and a large percentage of these potential adopters involve non-critical applications (Nikula and Jantunen, 2005). Sohn and Mok (2008) found that the capability of OSS software to provide functionality that met the stated and implied needs of users positively influenced OSS utilization in firms. They also found that OSS possessing non-functional requirements/characteristics (e.g., reliability, usability, efficiency) had positive impacts on OSS utilization in firms. Many OSS projects lack non-functional requirements, such as usability, with their developers not being aware of the importance of usability and requirements of users (Raza et al., 2012). Fitzgerald notes that, to support the needs of commercial organizations intending to adopt OSS, more rigorous project management is needed to deliver a more professional OSS product (Fitzgerald, 2006, p.591). Ad hoc approaches, such as OSS developers simply asserting requirements, may not be able to capture the requirements of an ever-increasing user base of OSS projects (e.g., Rantalainen et al., 2011). There is evidence for a large number of OSS projects ending up as failures (Khondu et al., 2013; Kalliamvakou et al., 2014). A potential contributor to such failures could be a lack of user interest (e.g., Stewart et al., 2006). If OSS projects are not able to visualize their potential user base and, subsequently, are not efficiently capturing the requirements of such a user base, then this might contribute to user disinterest. For example, often OSS developers ignore feature requests from users, causing user dissatisfaction (Laurent and Cleland-Huang, 2009).

With its focus on investigating OSS requirements discovery practices, this research will also help address research gaps highlighted by other OSS researchers. Crowston et al. (2012), in their review of the empirical work done on OSS development, highlight that much research is needed on the use of CSS development practices (e.g., requirements engineering practices) in OSS development. Alspaugh and Scacchi (2013) note that the answer to the question of whether OSS domain would benefit from requirements engineering practices (from CSS development) is not clear and requires further research.

The next section describes in detail, the objectives of this research.

1.2 Research Objectives and Design

Research is an activity that attempts to contribute to the understanding of some phenomenon (Vaishnavi and Kuechler, 2015). The phenomenon of interest in the context of this research is requirements discovery in OSS development. Requirements discovery in OSS development occurs in an ad hoc manner (e.g., Scacchi, 2002; Noll, 2008), is problematic and challenging (e.g., Kuriakose and Parsons, 2015), and needs improvement (e.g., Dietze, 2005). Little research has investigated requirements related activities in OSS development. As a result, we have a limited understanding of requirements engineering in

OSS and there is need for much research for furthering our understanding of it (e.g., Vlas and Vlas, 2011; Alspaugh and Scacchi, 2013; Crowston et al., 2012). Thus, one objective of this research is to gain an understanding of the current state of requirements engineering in OSS development including areas of improvement. Well-established theories such as the *software development life cycle (SDLC)* theory and the *RE process maturity model* theory can shed some light in this direction. For example, SDLC theory argues that requirements discovery, the initial phase of software development life cycle should be disciplined and comprised of formal requirements engineering practices (e.g., Sommerville and Swayer, 1997; Walls et al., 1992). When observed from the lens of SDLC, it becomes evident that OSS requirements discovery is largely ad hoc and informal since the requirements engineering phase in SDLC is comprised of several formal practices that appear to be largely missing during OSS development (e.g., Noll and Liu, 2010). The RE process maturity model also suggests that OSS requirements discovery is in an ad hoc and problematic state, as discussed in the preceding section. It also suggests gradual incorporation of formal RE practices as a potential direction of improvement. Requirements engineering is a complex process with a broad scope that needs to take into account, not only the target software but also the environment (often comprising of humans, devices, other software, etc.) surrounding it (Lamsweerde, 2000). Contributing to the complexity is the fact that the RE process is comprised of several intertwined activities, such as elicitation/gathering, negotiation, and documentation (Lamsweerde, 2000). When analyzing a complex problem, an efficient approach to follow is to decompose it into components (Simon, 1996; Burton-Jones and Meso, 2008). Given the complexity of requirements engineering process, any attempts to improve RE in OSS should ideally begin by focusing on one or more of the sub-processes within RE. This research focuses on the requirements generation activity, which is responsible for capturing and making available the requirements from potential sources (e.g., Arthur and Groner, 2005). Thus, after reviewing the current state of RE in OSS and identifying areas of improvement, this research addresses the specific objective of *whether and how requirements generation in OSS development could be improved*.

The first phase of this research starts with an exploration of the current state of the requirements engineering in OSS development. It is important to investigate, as part of any research initiative on improvement opportunities for OSS requirements discovery, the current state (e.g., what RE practices, formal and informal are used, if any) and the different problems and challenges that may exist during requirements discovery in OSS development. As suggested by the RE process maturity model, for the OSS development domain, the set of formal RE practices is a potential solution space for tackling the problems that arise from the informality of/ad hoc requirements discovery in OSS projects. Hence, the initial phase of this research sets out to investigate the current state of requirements engineering in OSS development, the problems, and challenges that may exist during requirements discovery in OSS development, and the extent to which incorporation of formal RE practices (from CSS development) may be beneficial for requirements discovery in OSS domain. In OSS development, it is mainly developers who are responsible for overseeing and managing the developmental activities (Crowston and Howison, 2005). Hence, they are a valuable source for gathering information about the current state of requirements discovery in OSS development and are used as the source of data collection in the first phase of the research.

Analysis of data obtained from the survey provides evidence that requirements discovery in OSS development is indeed ad hoc. While most of the formal RE practices (from CSS development) have significantly low reported usage in OSS development, they are largely perceived as beneficial for OSS development, thus highlighting many potential directions of improvement for requirements discovery in OSS development. The next step then becomes to empirically investigate the actual benefits from some of the formal RE practices perceived as beneficial. Responding OSS developers also indicate several problems and challenges with requirements discovery in OSS development, further establishing the need for improvement initiatives. Given the complexity of RE process, empirical investigation of all formal RE practices is not feasible within a scope of a single research project. In this research, I focus on requirements generation activities in RE. Requirements generation activities are used for discovering requirements about the software from various sources (e.g., users), and usually form the first stage of the requirements engineering phase in CSS development (Sommerville and Swayer, 1997; Browne and Ramesh, 2002). Results from the initial research phase indicate that most of the formal requirements generation practices (the practices under requirements elicitation category) are perceived as beneficial for OSS development by OSS developers, in spite of low usage.

In the second phase of this research, an empirical investigation of the benefits from a specific requirements generation practice, requirements reuse for OSS requirements discovery, is carried out. This particular practice is selected for further investigation because it can be easily incorporated into the web-based OSS development environments, for example, as a library of reusable requirements in the form of wiki pages or a centralized repository within OSS development environments (e.g., a GitHub repository). Moreover, the literature has reported OSS community reusing other artifacts such as code (e.g., Von Krogh et al., 2005), a potential indicator of their general willingness to reuse software development artifacts. It is potentially easier to incorporate requirements reuse into existing OSS workflows compared to some other formal requirements generation practices, such as analysts eliciting requirements from users using interviews or scenarios, because of the distributed (geographical, temporal, and socio-cultural distances) and ad hoc nature of OSS development and evolution, lack of financial resources and time constraints of contributors. For example, in the OSS domain, it is difficult for analysts to identify and bring together the right stakeholders for requirements elicitation (Laurent and Cleland-Huang, 2009).

For the empirical investigation, a web-based experiment is designed with two conditions, one in which a library of reusable requirements (specifically requirements patterns) is made available within the web-based OSS development environment that could be reused during requirements generation (experimental task) and a control condition in which no such library is available. Requirements patterns are artifacts containing reusable requirements knowledge, often as natural language descriptions, making them easy to understand and use (e.g., Breaux et al., 2012). Analysis of the experimental data reveals significant main effects of the availability of a library of reusable requirements on requirements size, quantity, and completeness (potential benefits) after controlling for covariates, and significant interactions between the availability of a library of reusable requirements and covariates (specifically, their technical and crowdsourcing experiences).

Different stakeholders of a software system have different perspectives about the software due to their different skills, knowledge, and experience (Finkelstein et al., 1992; Darke and Shanks, 1996). It is necessary to accommodate these various viewpoints during requirements generation (Darke and Shanks, 1996; Pacheco and Garcia, 2012). This also emerges from the initial survey where the requirements generation practice "*collect requirements from multiple viewpoints*" is perceived as beneficial for OSS development by OSS developers, in spite of low usage. To accommodate diverse views, it is important that information is gathered from individuals independent of any form of classification (Parsons and Wand, 2014). Imposing classification while gathering information reflects a fixed, limited view that can be detrimental to accommodating diverse views (Parsons and Wand, 2014). Interestingly in OSS issue repositories, which are a major source of requirements for OSS projects (e.g., Alspaugh and Scacchi, 2013), both types of issue gathering practices are popular, one that imposes classification and one that does not. Recent research (e.g., Herzig et al., 2013) has reported misclassification (erroneous classification of issue data) as an information quality problem with OSS issue data. For example, Herzig et al. found that about 39% of issues that were classified as bugs were not really bugs but rather enhancement requests, feature requests, and documentation issues. Similarly, a large percentage of issues that were classified as feature requests and enhancement requests were found to be actually of some other type (Herzig et al., 2013). Such misclassification can be a major challenge to OSS requirements discovery.

The third and final phase of this research explores how the two different OSS issue gathering approaches may differ in contributing to the misclassification problem and how the OSS issue gathering interface should be, especially for mitigating the misclassification problem. This final phase of the research uses a qualitative methodology involving OSS developers and data is collected using a web-based survey. OSS developers are often involved in both issue classification as well as issue reporting and with their knowledge and experience, are a good source of information for the final research phase. Many useful insights emerge from the analysis of data, for example, a simple issue gathering interface that does not require issue reporters to perform any classification/labeling at the time of submission of issues may be more effective in tackling the misclassification problem in comparison to a complex interface that requires issue reporters to perform classification/labeling at the time of submission. Based on the obtained insights, recommendations are proposed for OSS issue gathering interfaces.

The research questions for the three phases of this research are summarized below:

Main research objective: *Understanding the current state of requirements engineering in OSS development and whether and how requirements generation in OSS development could be improved?*

Research questions for phase one:

R1. *What are OSS developers' perceptions of the extent to which the formal requirements engineering practices (from CSS development) and the informal requirements generation practices reported in OSS literature are present in their developmental activities*?

R2. *What are OSS developers' perceptions of the extent to which requirements engineering activities (from CSS development) are beneficial for OSS development*?

R3. *What are OSS developers' perceptions about some of the problems and challenges that exist/could occur during requirements discovery in OSS development*?

Research question for phase two:

R4. *Does having access to a library of reusable requirements within OSS development environment have any benefits for requirements generation in OSS projects*?

Research question for phase three:

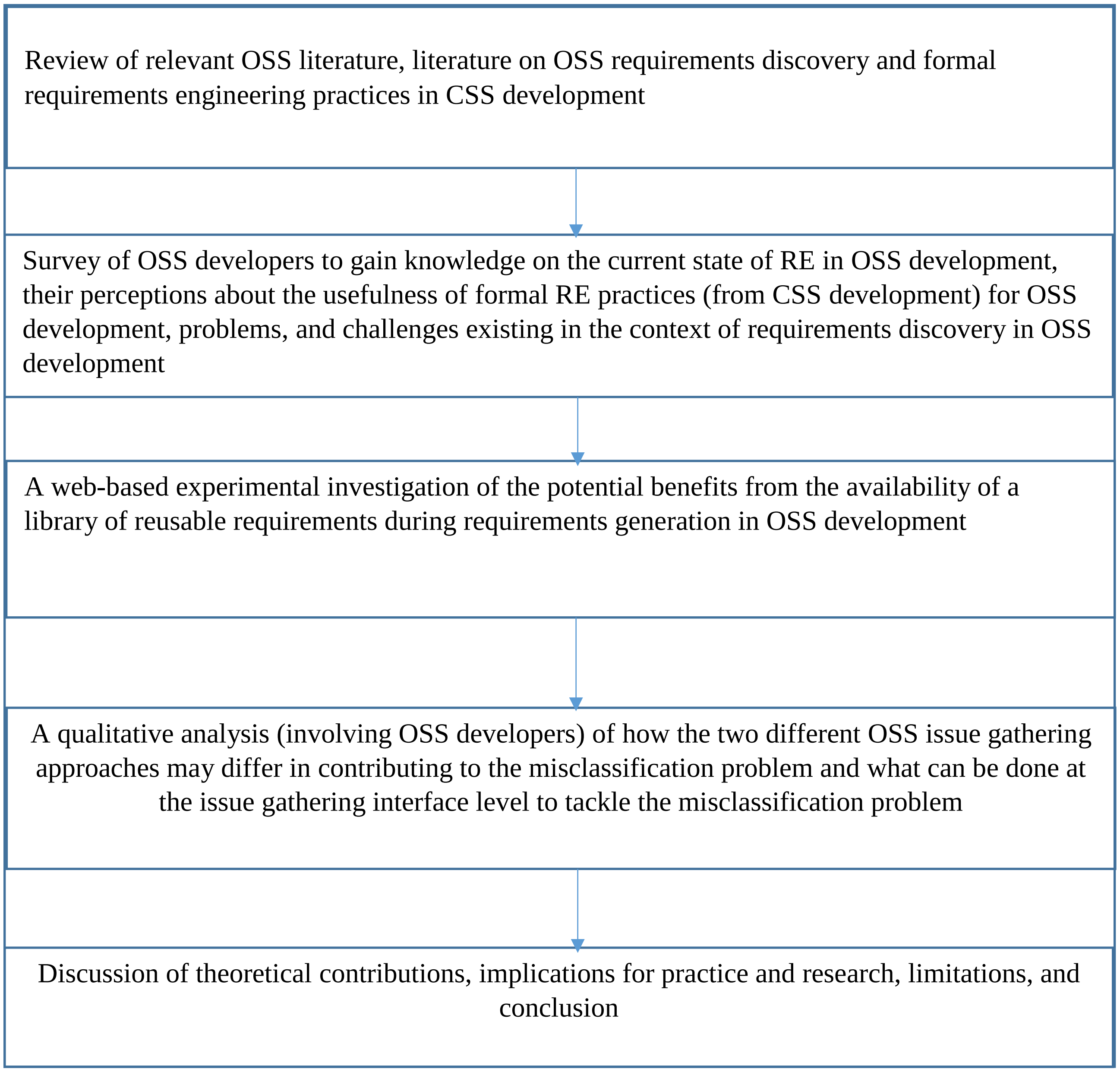
R5. *How do different OSS issue gathering approaches may differ in contributing to the misclassification problem?*

R6. *What could be done at the issue gathering interface level to tackle the misclassification problem*?

This research makes theoretical contribution in the area of the design of web-based open source software development environments by focusing on OSS requirements discovery, an under-researched and problematic aspect of OSS development. Requirements engineering process improvement is a design science contribution (e.g., Kabaale and Nabukenya, 2011; Fernandez and Penzenstadler, 2015, Fernandez and Wieringa, 2013). For example, Fernandez and colleagues (e.g., Fernandez and Penzenstadler, 2015; Fernandez and Wieringa, 2013) analyzed existing RE processes in traditional organizations such as Siemens and BMW, identified a need for improvement and subsequently, and empirically evaluated an alternative RE approach in those organizational contexts. They found that the alternative RE approach resulted in improvements such as completeness and ease of use. This study follows a similar approach for OSS domain by investigating the current state of requirements engineering in OSS development, identifying problems and challenges, and identifying and empirically investigating some potential directions for improvement for OSS requirements discovery. New knowledge about potential improvements to the design of IS artifacts is a design science contribution (Vaishnavi and Kuechler, 2015). Web-based OSS development environments are largely code centric and lack support for many formal requirements engineering practices (potential design deficiencies). Several potential directions for improvement of OSS requirements discovery emerged from the initial phase of this research many of which can also be potential design improvements if adapted efficiently within existing web-based OSS development environments. In phase two and three of the research, specific design improvements (availability of a library of reusable requirements as a feature in web-based OSS development environments, desirable characteristics for OSS issue gathering interfaces) are further explored. Results from the second phase indicate that a design enhancement in the form of a library of well-structured reusable requirements can bring in improved outcomes for web-based OSS development (e.g., greater quantity of requirements). The final phase of the research reveals new knowledge in the form of desirable characteristics in OSS issue gathering interfaces (design suggestions) that can help reduce the misclassification of OSS requirements artifacts. One way in which design knowledge can be output is in the form of models, which are descriptions/representations about how things are/should be, often representing the connection between problem and solution components which could then enable further exploration of the potential effects of design decisions; a key focus of models is often utility rather than truth (Vaishnavi and Kuechler, 2015; Hevner et al., 2004; March and Smith, 1995). The knowledge obtained from this research is largely in form of models, describing connections between problem and solution components in the domain of OSS requirements discovery; for example, a design enhancement in form of availability of a library of reusable requirements as a potential solution to incompleteness of OSS requirements and design suggestions for OSS issue gathering interfaces that can potentially help mitigate the misclassification problem. The theoretical contribution also emerges from the second phase of the research in the form of knowledge about the antecedents of requirements size, quantity, and completeness all of which are under-researched variables in IS literature.

1.3 Structure of the Research

The research proceeds as shown in the flowchart below:


Review of relevant OSS literature, literature on OSS requirements discovery and formal requirements engineering practices in CSS development
Survey of OSS developers to gain knowledge on the current state of RE in OSS development, their perceptions about the usefulness of formal RE practices (from CSS development) for OSS development, problems, and challenges existing in the context of requirements discovery in OSS development
A web-based experimental investigation of the potential benefits from the availability of a library of reusable requirements during requirements generation in OSS development
A qualitative analysis (involving OSS developers) of how the two different OSS issue gathering approaches may differ in contributing to the misclassification problem and what can be done at the issue gathering interface level to tackle the misclassification problem
Discussion of theoretical contributions, implications for practice and research, limitations, and conclusion

Chapter two provides the background of the research and review of relevant literature. Chapter three describes the findings from the analysis of survey data (e.g., the findings about the current state of requirements discovery in OSS development, potential directions of improvement). Chapter four describes the findings from the experimental investigation of the potential benefits from the availability of a library of reusable requirements during requirements generation in OSS development. Chapter five describes the insights from the analysis of qualitative data obtained from OSS developers on how the two different OSS issue gathering approaches may differ in contributing to the misclassification problem and suggestions about the desirable characteristics of issue gathering interface for tackling the misclassification problem. Chapter six describes theoretical contributions, implications for practice and research and limitations.

CHAPTER TWO. BACKGROUND AND REVIEW OF RELEVANT LITERATURE

2.1 Open Source Software (OSS) Development

Perens (1999) describes the definition for *open source* to include the following:

- Free redistribution: can make any number of copies of software, give them to anyone or even sell but no need to pay anyone for this privilege.
- Source code can be freely distributed.
- Modification of any sort is allowed and all modifications/derived works can also be freely distributed.
- No restrictions against any person/group of persons; no restrictions against fields of endeavour (anyone can use the software in any field of endeavour, for example business or research).

In addition to the code, an OSS project usually provides open access to all its development artifacts (e.g., issue data in issue repositories, data in mailing lists and discussion forums) (Robbins, 2003). The domain of this research is OSS development, an umbrella term that denotes the software development activities carried out to produce such software.

OSS development is often carried out by organizationally and geographically distributed developers and other contributors who voluntarily participate in the development (Crowston et al., 2012). The volunteering contributors collaborate and carry out developmental activities almost entirely over the Internet using tools such as mailing lists, web forums, version control systems, issue reporting artifacts, real-time chats and wikis (Robbins, 2003; Detienne et al., 2006; Rantalainen et al., 2011). OSS developmental activities are often carried out within web-based collaborative development environments (e.g., Source Forge, GitHub, Google Code and CodePlex) that provide many tools to support the developmental activities (e.g., Robbins, 2003).

Crowston and Howison (2005) describe the social structure of OSS development as largely comprised of core developers, co-developers, active users and passive users. Usually, it is a small group of core developers who do most of the code contributions and are responsible for overseeing the OSS project evolution. The co-developers contribute by submitting code for review (which may or may not be accepted) by core developers. Active users contribute by providing feature requests and bug reports and helping in testing new releases whereas passive users are mainly observers who do not contribute (Crowston and Howison, 2005). The core developers in OSS projects control code base and create most of the functionality (Mockus et al., 2002).

The participation of volunteer developers and other contributors in OSS development is often driven by different kinds of motivation, including: the need for certain functionality; learning opportunities; career development; and fun (Shah, 2006; Hars and Ou, 2002; Robbins, 2003). For example, many OSS developers participate in development because of their need for some software functionality (Nichols and Twidale, 2003). Because of this, many OSS projects are self-driven, where the developers write the code to meet their own needs (Rantalainen et al., 2011).

While the majority of contributing OSS developers leave after their needs are met, a small group (the core developers) remains; for these developers, participation becomes a hobby, and they oversee the further evolution of the project (Shah, 2006). A large number of OSS projects have to rely on part-time efforts of these small number of core developers for their continuous evolution (Robbins, 2003).

OSS development has evolved into a domain with characteristics that make it distinct from the CSS development. This is elaborated in the next section.

2.2 Open Source Software Development versus Closed Source Software Development

There are some clearly observable differences between the CSS development and OSS development domains (Mockus et al., 2002; Godfrey and Tu, 2000; Aksulu and Wade, 2010). As Scacchi puts it: "This is a world that differs in many ways from traditional software engineering, where it is common to assume centralized software development locales, development work, and administrative authority that controls and manages the resources and schedules for software development and maintenance" (p.5, Scacchi, 2006).

One major difference is in terms of the goals of the paradigms. CSS development aims at filling some commercial void, while OSS development aims to create something useful or interesting (Godfrey and Tu, 2000). In fact, often OSS projects have only loosely defined objectives (Aksulu and Wade, 2010). CSS projects have an externally defined lifespan whereas OSS projects do not have such defined life spans; they may survive as long as at least one contributor is willing to maintain them (Aksulu and Wade, 2010). While CSS development occurs in a relatively uniform and stable organizational environment, OSS development occurs in an open unstable environment that does not have hierarchical processes or enforcement of technology policies and lacks standards and regulations (Aksulu and Wade, 2010). Based on their review of OSS empirical work, Aksulu and Wade (2010) conclude that ***the lack of project structure in OSS development could make it less efficient than CSS development and result in non-reliable and non-useful outputs***

In contrast to CSS development, which is staffed and funded, OSS development is mainly driven by unpaid volunteer developers and non-developer contributors (Mockus et al., 2002; Godfrey and Tu, 2000). Baytiyeh and Pfaffman (2010) report that, while there is a small percentage of OSS developers who are paid by organizations for their contribution, payment does not have a significant impact on motivation to contribute; developers are motivated primarily by altruistic intentions and desire to create and learn. OSS contributors often choose to do work that is of interest to them (self-selection), instead of work assigned to them (Mockus et al., 2002; Detienne et al., 2006; Robbins, 2003; Godfrey and Tu, 2000; Aksulu and Wade, 2010). This highlights the freedom that OSS development provides, with no one having power over the contributors (Godfrey and Tu, 2000; Aksulu and Wade, 2010). OSS developers' self-selection of work can be influenced to a great extent by their familiarity with the application domain and developmental technologies (Robbins, 2003). The feedback from users and developers (often termed peer review) is a critical resource for the evolution of OSS projects, while it is a non-critical resource for CSS projects (Robbins, 2003). For example, a piece of code submitted by some contributor may need to be reviewed and discussed by the community before it is incorporated into the source code of the OSS project (Robbins, 2003).

OSS development is often ad hoc in nature (Zhao and Elbaum, 2003). OSS projects often lack the formal processes followed in closed source software development (Mockus et al., 2002); they are instead characterized by rapid development, frequent incremental releases, and parallel development and debugging (Feller and Fitzgerald, 2000). CSS projects have economic concerns and contractual agreements, which is generally not the case with OSS projects, facilitating “release early, release often” in OSS development (Robbins, 2003). Frequent releases may be advantageous to OSS projects for attracting volunteers (Robbins, 2003). OSS projects often lack system-level design, project plans, schedules, and list of deliverables (Mockus et al., 2002; Detienne et al., 2006). In particular, *OSS development appears to lack formal requirements engineering practices* (Crowston et al., 2012; Robbins, 2003; Scacchi 2002).

The next section discusses some major research themes in OSS literature.

2.3 Major Research Themes in OSS Literature

Crowston et al. (2012) and Aksulu and Wade (2010) have reviewed a large number of empirical OSS studies and identified major research themes in OSS domain. These are listed in the Table 1. This research, centered on requirements discovery in OSS development, can be put into OSS software development practices category in the Crowston et al. taxonomy or the OSS production category in the Aksulu and Wade taxonomy. Aksulu and Wade put into OSS production category research on issues such as OSS methodology, design structures, architecture, work practices.

Table 1. OSS research themes

OSS research themes (Aksulu and Wade, 2010)	**Example categories (Aksulu and Wade, 2010)**
Conceptual	OSS benefits/drawbacks, OSS versus proprietary software, Business/economic models, and strategies/policies for OSS
Performance metrics	Software quality- testing and bug fixes, OSS security, Development team performance, OSS success
Legal and Regulatory	OSS licensing and legal issues, OSS standards, and regulation
OSS production	Process, team formation, governance, collaboration and knowledge sharing, users and developers' motivation, Role of commercial corporations
OSS applications	Telecommunications, education, Imaging, supply chain, gaming, academic, biomedical, natural sciences
OSS diffusion	OSS adoption, OSS implementation
Beyond software	Open paradigms, open innovation, open standards, open knowledge flows
OSS research themes (Crowston et al., 2012)	**Example categories (Crowston et al., 2012)**
Member characteristics	Geographic location, motivation for participation
Project characteristics	License types
Technology use	Types of technology used in development
Software development practices	Project management planning, requirements, coding, testing, maintenance
Social processes	Socialization, decision making, coordination and collaboration, knowledge management
Firm involvement practices	Adoption by firms, OSS commercialization
Social states	Trust
Task-related states	Roles, commitment levels, shared mental models
Software implementation	OSS use in different contexts
Team performance	Success measures, relationship between success and other variables
Software evolution	Software evolution, OSS community evolution

An analysis of the abstracts of the studies in OSS production category by Aksulu and Wade revealed that only one study (Scacchi, 2004) had looked at OSS requirements discovery, indicating that OSS requirements discovery is an under-researched area in OSS domain. This is also the case with the Crowston et al. review, which found only two studies that looked at requirements discovery, leading to their main claim was that OSS domain lacked traditional requirements engineering practices and that OSS requirements discovery largely occurs in an ad hoc manner. Crowston et al. highlight the need for research on OSS development practices, in particular, on the use of closed source software development practices (e.g., requirements engineering practices) in OSS development. Alspaugh and Scacchi (2013) make a similar call, and mentions that the answer to the question of whether OSS domain would benefit from requirements engineering practices (from CSS development) is unclear and awaits further research. There has been a lack of work on requirements engineering in OSS since these reviews. For example, a search with the keyword “requirements” in the OSS literature specific database, flosshub (http://flosshub.org/biblio), indicated a lack of research on RE in OSS since 2011.

This research aims to fill the above-mentioned gap with its specific focus on requirements discovery in OSS development. This is elaborated in the next section.

2.4 Requirements Discovery in OSS Development

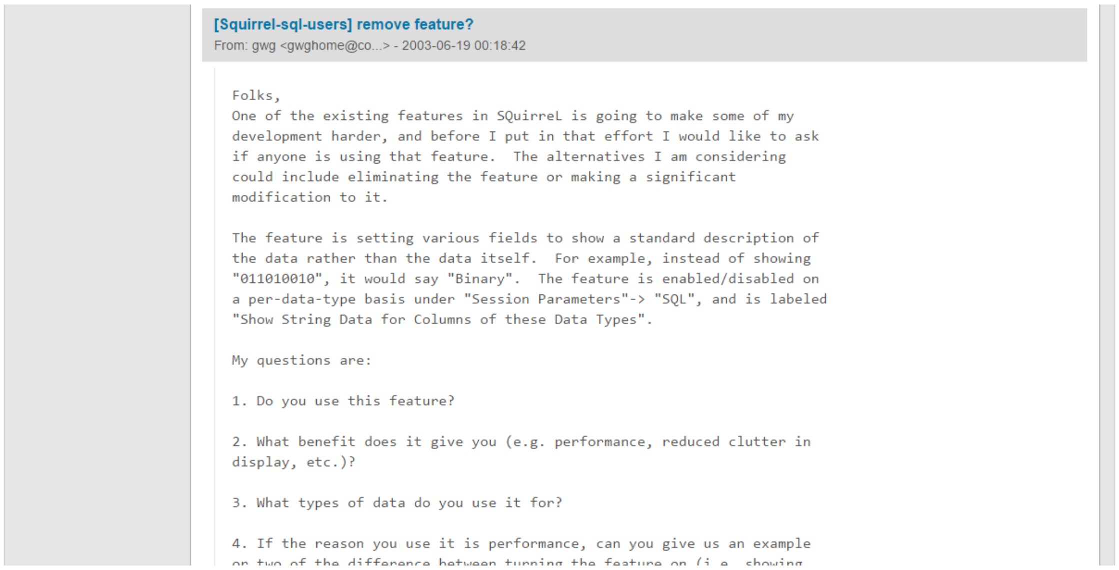
2.4.1 Requirements and Requirements Engineering

Requirements are statements about what features and characteristics software should have (Dennis et al., 2012). Requirements could be functional: they describe what features should be in software. An example of a functional requirement is the need for a software to allow registered users to review order history. Requirements could be of a non-functional type that describe what characteristics other than the functional features should the software have. For example, there could be performance requirements (e.g., capacity and reliability of the software) or cultural/political requirements (e.g., the need for software to be compliant with data protection laws) (Dennis et al., 2012). Requirements can also describe how the software should be built (system requirements, such as the hardware and software needed to support development), the business needs (business requirements; such as reducing order processing time) and what tasks users perform (user requirements; such as scheduling an appointment) (Dennis et al., 2012).

In CSS development, there is a formal requirements engineering phase comprising of several sub-phases such as requirements elicitation, requirements documentation, requirements validation, and requirements management (Sommerville and Swayer, 1997; Browne and Ramesh, 2002). Requirements elicitation is the process of discovering requirements for software from different potential sources, such as users and documents. During this stage, analysts use different techniques such as interviews, observation, and document analysis to elicit requirements from different sources (Sommerville and Swayer, 1997; Browne and Ramesh, 2002; Dennis et al., 2012). During requirements documentation, requirements are formally documented (e.g., software requirements specification document) for the purpose of communicating with stakeholders such as users and developers (Sommerville and Swayer, 1997). After the requirements have been documented, they are formally validated with stakeholders to check for omissions, conflicts and ambiguities (Sommerville and Swayer, 1997; Browne and Ramesh, 2002).

2.4.2 What We Know about OSS Requirements Discovery

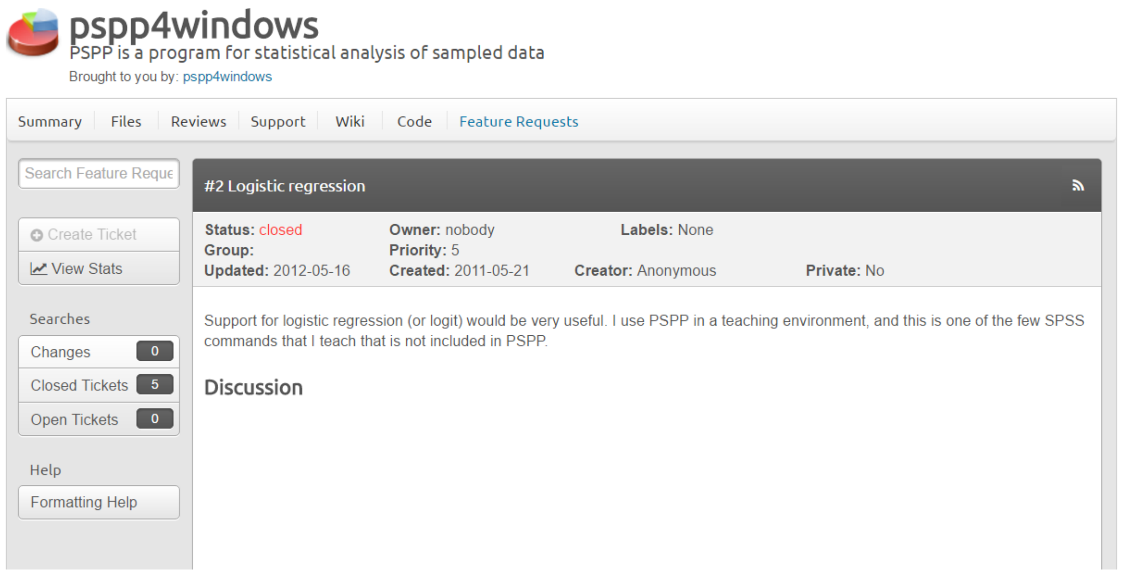
For the most part, OSS development lacks the formal requirements engineering practices from the CSS domain (e.g., Crowston et al., 2012; Scacchi, 2002). In fact, Crowston et al. (based on their systematic literature review of 184 empirical works in OSS domain shortlisted from 52 journals and 40 conferences) write: "*FLOSS projects are often said to not conduct formal requirements analyses*" (p.17, Crowston et al., 2012). As a specific example, there is often no formal requirements elicitation by analysts in OSS development development (Laurent and Cleland-Huang, 2009). In OSS development, requirements may exist informally as part of some message posted on communication artifacts such as discussion forums, issue reporting artifacts, bulletin boards, project wikis, and chat tools (Scacchi, 2009; Noll, 2008; Vlas and Vlas, 2011). Some examples are shown in Figures 2 and 3.



[Squirrel-sql-users] remove feature?
From: gwg <gwghome@co...> - 2003-06-19 00:18:42

```
Folks,
One of the existing features in SQuirreL is going to make some of my
development harder, and before I put in that effort I would like to ask
if anyone is using that feature.  The alternatives I am considering
could include eliminating the feature or making a significant
modification to it.

The feature is setting various fields to show a standard description of
the data rather than the data itself.  For example, instead of showing
"011010010", it would say "Binary".  The feature is enabled/disabled on
a per-data-type basis under "Session Parameters"-> "SQL", and is labeled
"Show String Data for Columns of these Data Types".

My questions are:

1. Do you use this feature?

2. What benefit does it give you (e.g. performance, reduced clutter in
display, etc.)?

3. What types of data do you use it for?

4. If the reason you use it is performance, can you give us an example
```

Figure 2. Example of a requirement existing as part of an informal message posted on discussion forum (https://sourceforge.net/p/squirrel-sql/mailman/squirrel-sql-users/?viewmonth=200306)



pspp4windows

PSPP is a program for statistical analysis of sampled data

Brought to you by: pspp4windows

#2 Logistic regression

Status: closed **Owner:** nobody **Labels:** None
Group: **Priority:** 5
Updated: 2012-05-16 **Created:** 2011-05-21 **Creator:** Anonymous **Private:** No

Support for logistic regression (or logit) would be very useful. I use PSPP in a teaching environment, and this is one of the few SPSS commands that I teach that is not included in PSPP.

Discussion

Figure 3. Example of a requirement submitted as a feature request through an issue reporting artifact (https://sourceforge.net/p/pspp4windows/feature-requests/2/)

Requirements may be simply asserted by developers based on personal experience and knowledge, emerge as part of the bug reports and feature requests submitted through issue reporting artifacts, or be ideas obtained from features in other commercial products (Noll, 2008). Table 2 summarizes major findings reported in the existing literature on requirements discovery in OSS development.

It appears from the findings reported in Table 2 that requirements engineering in OSS development is largely informal and ad hoc. All the studies reviewed in Table 2 have reported their findings based on qualitative analysis of project artifacts (e.g., data in discussion forums, mailing lists and issue repositories) of a few OSS projects. To the best of my knowledge, there have not been other types of studies (e.g. quantitative, experimental). A search for other type of studies, including a search with the keyword *requirements* in the OSS literature specific database flosshub (http://flosshub.org/biblio) was unsuccessful (i.e., no results). For example, Noll (2008) analyzed archival data of Firefox, an OSS project. whereas Noll and Liu (2010) analyzed archival data of OpenEMR, an open source electronic medical record software.

Table 2. Major findings reported in existing literature on requirements discovery in OSS development

Findings on requirements engineering practices reported in OSS literature	**Scacchi (2002)**	**Noll (2008).**	**Ernst and Murphy (2012)**	**Llanos and Castillo (2012)**	**Massey (2002)**	**Noll and Liu (2010)**
Requirements are asserted by developers	√	√	√	√	√	
Requirements specification mainly exists informally as part of communication messages in emails, discussion forums, and such artifacts.	√			√		√
Lack of formal requirements elicitation			√	√		
Lack of formal requirements validation	√			√		
Lack of formal requirements prioritization			√			
Requirements are contributed by users through bug reports and feature requests		√		√	√	
A source of requirements is features appearing in commercial products		√			√	

In OSS development, it is mainly the OSS developers who are responsible for overseeing and managing the developmental activities (Crowston and Howison, 2005). Hence, they would be a valuable source of information about the current state of requirements engineering in OSS development, as well as about potential directions for improvement. Surprisingly, there is a lack of such an empirical study. To address this omission, the first phase of this study (chapter three) reports a survey that was carried out among OSS developers that sought to gather information from them about their usage of requirements engineering practices during OSS development.

2.5 Importance of Requirements Discovery for OSS Projects

2.5.1 Ever Expanding OSS User Base

The user base of OSS projects is increasing significantly (Scacchi, 2007; Choi and Chengalur-Smith, 2009), and OSS projects are fast becoming components within IT infrastructure in various domains such as business, education, and government (Terry et al., 2010). For example, companies such as Red Hat and Novell make use of OSS (Terry et al., 2010). Foushee (2013) also found that the number of OSS projects written by developers for themselves has decreased over the years from a high in 2001 to a low in 2011, whereas OSS projects written for non-developer users have increased over the years. Choi and Chengalur-Smith (2009) also made similar observations. OSS developers often carry out the development activities in an informal ad hoc manner (Mockus et al., 2002). The informal ad hoc approach works fine for self-driven OSS projects as OSS developers know the needs and can just implement those needs (Rantalainen et al., 2011). The informal ad hoc approach becomes problematic when the user base of OSS projects extends beyond the developers themselves, in which case the developers are no longer domain experts (Rantalainen et al., 2011). For example, such an approach fails to take into account end-user usability (Nichols and Twidale, 2003). OSS developers' lack of domain knowledge could contribute to observations such as that of Foushee (2013), who found that OSS projects that were written by developers for themselves were much more likely to be successful in comparison to OSS projects that were written for general users (i.e., not for the developers themselves). Choi and Chengalur-Smith write that developing OSS projects for general end-users is "*critical factor for OSS success, yet critical issue holding back the OSS movement*" (p.1, Choi and Chengalur-Smith, 2009). In fact, Heppler et al. (2016) found statistical empirical evidence that OSS developers tend to ignore feature requests from non-developer users.

User interest in OSS projects is an indicator of OSS project success (Stewart et al., 2006; Subramaniam et al., 2009). Stewart et al. found that for OSS projects to be successful, they must signal to the users that they will provide a high level of utility. They also found that user interest in OSS projects has a positive effect on OSS development activity, which indicates that having an interested user base for OSS projects motivates OSS developers to continuously carry out developmental activities (Stewart et al., 2006). Chengalur-Smith et al. (2010) also mention that OSS developers may not be motivated to work on OSS projects that only a few users are interested in. As noted by one OSS developer in the study by Shah (2006, p. 1008): "why work on something that no one will use? There is no satisfaction there" (p.1008) and by another, "it is rewarding when you see that what you helped create is used by many people. I want to let many people know about this software, and I want them to use it" (p.1008) (Shah, 2006). These findings from the OSS development literature are in line with the literature on CSS development, which has consistently found that user involvement is one of the most important factors for project success (e.g., Verner et al., 2005). For example, a series of surveys by Standish group consistently found that user involvement is the most important factor for software project success (Standish group, 1995; Standish group, 1999). More recent research (e.g., Bano and Zowghi, 2013) has made similar observations. If good requirements are generated for OSS projects (for example, from users), they could address some of the above-mentioned issues. For example, developers could be more confident that they are working on things that are of interest to the user community and, when users find that the developers are working on things that they desire, it could be a signal to them about the utility of the OSS project.

Wixom and Todd (2005) empirically demonstrated that user satisfaction with an information system positively influences user's intention to use an information system. A major antecedent of user satisfaction with an information system is information system quality (e.g., information system having the functionalities needed by the user) (Wixom and Todd, 2005). Lee et al. (2009) empirically demonstrated that OSS quality positively influences OSS user satisfaction which in turn positively influence OSS use. This implies that, if OSS software is capable of meeting the expectations, needs and wants of potential users, then it is more likely to capture the interest of users and get them involved, which should then have favorable outcomes for the OSS project. Sohn and Mok (2008) found that the capability of OSS software to provide functionality that met the stated and implied needs of users positively influenced OSS utilization in firms. They also found that OSS possessing non-functional requirements/characteristics (e.g., reliability, usability, efficiency) also had positive impacts on OSS utilization in firms. Thus, generating good requirements can help OSS projects have functionality meeting user needs and expectations, as well as have good non-functional characteristics.

2.5.2 Good Requirements can Drive OSS Project Success

A major reason for software not being able to do what the users want or expect is problems with the software requirements gathered (Chakraborty et al., 2010). For example, incomplete requirements and changing requirements are major factors contributing to software project failures (Standish group, 1995; Kulk et al., 2008). An example is that of Qantas, Australia's national airline, which had to cancel a $40 million software project after potential users (specifically union of aircraft mechanics) refused to use it. A major reason for this was that management did not care about user perspectives on the new software, but, rather, focused on implementing what the management thought was appropriate (Dennis et al., 2012). In the OSS domain, there is evidence for a large number of projects being abandoned soon after they start (e.g., Chengalur-Smith et al., 2010; Khondu et al., 2013). Sourceforge (a web-based OSS development environment) has over 150000 OSS projects but most of them become inactive soon after registration or within the first year (Chengalur-Smith et al., 2010). Khondu et al. (2013) found that over 86% of projects in their analysis were inactive with no recorded activity during the last year, and over 65% of projects had no activity within the last two years. In Sourceforge, abandoned projects can be tagged as stale by developers to indicate that they have been properly abandoned by original developers and need new volunteers and support to be maintained. Khondu et al. (2013) found that over 60% of the inactive projects were tagged as stale. Kalliamvakou et al. (2014) reported that the majority of GitHub projects in their analysis were inactive. Clearly, having such a large number of unsuccessful OSS projects is undesirable since it represents wasted time, effort and other resources that went into development.

Vlas (2012) empirically demonstrated that there is an association between quality of requirements data posted on OSS discussion forums and OSS project success. For example, PHPMyAdmin is a successful OSS project analyzed in the Vlas study. It was found that PHPMyAdmin had accumulated a rich set of meaningful requirements in its discussion forums (requirements that were mainly about the application domain of PHPMyAdmin software) (Vlas, 2012). The findings of Vlas in the OSS domain are similar to findings reported in the literature on CSS development about the importance of good quality requirements for software projects. For example, Kamata and Tamai (2007) found evidence for good quality software requirements specification in successful CSS development projects. Thus, the poor quality of requirements generated during the evolution of OSS projects could potentially contribute to their failures.

The findings of Vlas indicate that, if OSS projects have access to better quality requirements data, then they are in a better position to be successful over the long run. This is in line with Chengalur-Smith et al. (2010), who found that OSS projects can remain active and have continuous developmental activities if they can get good ideas from users and developers. Such ideas can broaden the scope, purpose, and functionality of the OSS projects. Chengalur-Smith et al. further found that even successful projects (both OSS and industrial) require continuous developmental activities. They back up this claim with examples of continuous new releases by Apple and Microsoft, and thereby illustrate that good quality ideas, such as new features, are important even for successful software projects (Chengalur-Smith et al., 2010). Thus, the generation of good quality requirements is beneficial for OSS projects.

The first phase of this research focuses on exploring in depth, the current state of requirements discovery in OSS development and identify areas and directions for improvement. To begin with, an exploratory survey was carried out among OSS developers. The survey and findings from it are discussed in greater detail in the next chapter.

CHAPTER THREE. AN EXPLORATION OF THE CURRENT STATE OF REQUIREMENTS DISCOVERY IN OPEN SOURCE SOFTWARE DEVELOPMENT AND POTENTIAL DIRECTIONS OF IMPROVEMENT

3.1 Survey on the Current State of Requirements Discovery in OSS Development

As described in previous chapters, a handful of existing OSS literature (e.g., Scacchi, 2002; Noll 2008; Noll and Liu, 2010), based on qualitative analysis of web-based archival data (e.g., messages in OSS discussion forums), have claimed that requirements discovery in OSS development is ad hoc and informal. However, there is lack of direct data involving the views of practitioners themselves – OSS developers and other contributors – on requirements in OSS development. In fact, researchers (e.g., Alspaugh and Scacchi, 2013; Crowston et al., 2012) have mentioned that much research is needed for bettering our understanding of requirements discovery in OSS development. Also, Dietze (2005) has called for improvements in requirements discovery practices in OSS development, and any such effort would need a detailed analysis of the current state of the practice of requirements discovery in OSS domain.

In OSS development, it is mainly OSS developers who are responsible for overseeing and managing developmental activities (Crowston and Howison, 2005) and, hence, are a valuable source of information about different aspects of requirements discovery. A web-based survey was developed from the review of RE and OSS literature (Sommerville and Swayer, 1997; Noll, 2008; Noll and Liu, 2010; Damian and Zowghi, 2003; Schmid, 2014; Lintula et al., 2006) to gather information of interest from OSS developers. The survey was developed using surveygizmo, a web-based survey application. The survey software provided insights into the design of the survey (e.g., average time needed for completion). The survey had four sections. The first section consisted of questions about the extent to which formal requirements engineering practices (from CSS development) and informal practices (reported in OSS literature (see Noll (2008)) were used during OSS development. The list of formal requirements engineering practices was obtained from Sommerville and Swayer (1999). This list is fairly comprehensive. There is empirical evidence for the usage of these practices in industries (e.g., Cox et al., 2009). Each practice listed was accompanied by a detailed description that could be view by placing the cursor over an * adjacent to the practice. The second section asked about perceived usefulness of each formal requirements engineering practice (from CSS development) for OSS development. The third section asked about problems and challenges that exist /could occur during requirements discovery in OSS development, and the fourth section asked for demographic information. For each section and individual groups of questions of the survey, participants could freely provide comments and any additional information that they wished to provide. The survey questions and structure can be found in Appendix 1.

The university research ethics committee approved the survey questionnaire. Subsequently, email invitations containing the link to the survey were sent to OSS developers registered on the OSS development environments GitHub and Sourceforge. Also, a link to the survey was posted on the webpages of online OSS communities such as Mozilla and Apache. No incentives were provided for completion of the survey. It is difficult to determine the exact number of OSS developers who received the survey link. Eighty four usable (complete) responses were obtained.

3.1.1 Demographic Profile of Respondents

The demographic profile of survey respondents is shown in Table 3.

Table 3. Demographic Profile of survey respondents

AGE	**<20**	**20-29**	**30-39**	**40-49**	**50+**
	7.1%	22.4%	41.2%	17.6%	11.8%
RESIDENCE	**North America**	**South America**	**Europe**	**Asia and rest of the world**	
	65.9%	4.7%	12.9%	16.5%	
EDUCATION	**Non-University education**	**Undergraduate or equivalent**	**Graduate or equivalent**	**Ph.D. and higher**	
	24.4%	36%	29.1%	10.5%	
TASK PROFILE IN OSS DEVELOPMENT	**Includes writing code**	**Does not include writing code**			
	94.2%	5.8%			
NUMBER OF OSS PROJECTS WORKED ON	**1-4**	**5-9**	**10-14**	**15+**	
	25.6%	29.1%	10.5%	34.9%	
AVERAGE NUMBER OF HOURS SPENT PER WEEK ON OSS DEVELOPMENT	**1-4**	**5-9**	**10-19**	**20+**	
	20%	25.9%	20%	34.1%	

The demographic question items, along with response categories (e.g., place of residence and its four possible options), were obtained from Sojer and Henkel (2010). In general, the data suggest that respondents were qualified OSS developers. The majority of the participants were from North America, but there was also participation from other parts of the world. This is illustrative of the globally distributed nature of OSS development, with contributors coming from different locations across the globe. More than 90% of the respondents participated in the coding tasks of OSS development. The sample of respondents had a good mix of less experienced and highly experienced OSS developers. Slightly more than 45% of the respondents had worked on less than ten OSS projects while almost 55% of the respondents had worked on more than ten 0SS projects.

3.1.2 Descriptive Findings on Use of Formal Requirements Engineering Practices and Informal Practices in OSS Development

The survey covered seven major categories of requirements engineering practices, obtained from Sommerville and Swayer (1997): requirements *documentation* practices, requirements *elicitation* practices, requirements *analysis and negotiation* practices, requirements *describing* practices, requirements *modeling* practices, requirements *validation* practices and requirements *management* practices. Within each category, questions were asked about the use of several specific practices in OSS development. Survey respondents were asked to indicate the level of usage of each practice by selecting one of the following seven options: always used (coded as 5), mostly used, sometimes used, rarely used, never used (coded as 1), not applicable and I do not know. Not applicable and I do not know were coded as 0. There were very few responses in not applicable, and I do not know categories.

Table 4 shows the means and standard deviations for respondents' reported usage for requirements documentation practices.

Table 4. Descriptive statistics: Usage of requirements documentation practices in OSS Development

Requirements documentation practices (Sommerville et al, 1997)	Mean	Std. Dev.
Define a standard document structure	2.34	1.346
Explain how to use the document	2.17	1.404
Include a summary of the requirements	2.78	1.556
Make a business case for the software	1.99	1.329
Define specialized terms	2.63	1.299
Make document layout readable	3.09	1.565
Help readers find information	2.85	1.483
Make the document easy to change	3.27	1.707

Comments of survey respondents indicate low usage of formal requirements documentation practices, for example: "*I have never formally documented the requirements in any of my open source projects: all have started as small apps/libraries to scratch an itch and have grown from there more or less organically.*" In Table four, not surprisingly, the practice, "make a business case for the software" have the lowest mean, which can be expected given the in-general non-commercial nature of OSS development.

Many self-driven OSS projects (i.e., developers develop the projects for their own use) (Rantalainen et al., 2011) could be experiencing this informality, as suggested by the following comment of a respondent: "*why would we bother writing a never used* *document when we could just make what we need*." The narrow focus of developers (e.g., not visualizing a large user base, rather developing for themselves; OSS developers simply asserting requirements (Noll, 2008)) might contribute to the informality as evident from the following comment: "*A lot of this survey may not apply to me. The open source software I have released is technology made for companies I am involved with. I select code I have written to be a candidate for making open source, consult with my colleagues and then post it online*." Instead of a formal specification, developers may go with what exists as knowledge inside their minds, as this comment from a respondent suggests: "*Very rarely worked with requirements document as far as I know. They are more of a piece of knowledge living through maintainers, not explicit data*."

Large and successful OSS projects (e.g., Mozilla), could contribute to the observed usage means as suggested by the following comment of a respondent: "*Honestly, only the largest OSS projects will take on the overhead of specification/design documentation.*" Some practices, such as including a summary of requirements, may be used under certain conditions (e.g., project owners may want to give newcomers the ability to participate in coding and influence project vision (Dabbish et al., 2012)), as suggested by the following comment: "*a written version of summary requirements often appears in a project when lead/team developers feel comfortable enough with having people contributing to their code. It works as a way of reducing people management overhead. When a business case exists, it is often de-attached from the project in order to allow anyone to contribute. Terms tend to be defined when the domain does not explain them and as for technicality, almost nobody likes to write them*." Such occurrences could contribute to the observed usage of these practices. The average OSS projects that are often driven by hobbyist developers (e.g., Shah (2006)) may largely lack formal requirements documentation practices, as indicated by the following comment of a respondent: "*Open source projects that I worked on were mostly hobbyist small individual projects where I was the sole developer, or research grade software developed in an academic setting where engineering methodologies were not applied or small tools I developed during the course of other (commercial or academic work). Thus, there was no formal engineering process or planning of any kind.*"

There may be some ad hoc/ informal documentation, such as project read me files having some screenshots about the project and some instructions (Begel et al., 2013). Such ad hoc documentation would not constitute a formal requirements document as suggested by a responding OSS developer: "*I have never been involved in an open source project that had a formal requirements document. For these small-scale projects, requirements are intrinsically linked to purpose and features which are listed on the project website (if available), or more commonly, the README.*" Another respondent's comment makes a similar suggestion: "*many open source projects that I have worked on have a read me. Occasionally a wiki with ad hoc other documentation.*" The practices such as "make documents layout readable" and "make the document easy to change," for which usage means appear to be little higher compared to other practices, may indicate the perceptions of responding OSS developers about documents such as README (e.g., making a README file easy to change).

Table 5 shows the means and standard deviations for respondents' reported usage for requirements elicitation practices.

Table 5. Descriptive statistics: Usage of requirements elicitation practices in OSS development

Requirements elicitation practices (Sommerville et al, 1997)	**Mean**	**Std.**
Assess software feasibility	2.7	1.552
Be sensitive to political and organizational consideration	2.23	1.434
Identify and consult software users	2.84	1.338
Record requirement sources	2.48	1.468
Define the software's operating environment	3.62	1.411
Use business concerns to drive requirement elicitation	2.35	1.328
Look for domain constraints	2.7	1.504
Record requirements rationale	2.57	1.334
Collect requirements from multiple view points	2.61	1.245
Prototype poorly understood requirements	2.83	1.367
Use scenarios to elicit requirements	2.66	1.355
Define operational processes	2.35	1.501
Reuse requirements	2.6	1.411

The following comments of survey respondents indicate low usage of formal requirements elicitation practices in OSS development: "*These questions all imply a much higher degree of formality than in projects I have worked on*"; "*Again these questions assume a high degree of formality around requirements elicitation. This has not been my experience*" and "*I had to think hard about some of these. These practices we do, we certainly don't think of in the terms you used. Our development is mostly informal, but sometimes use some of those if they sound like a good idea, rather than as a conscious decision to do so.*"

Freedom of development and no one having any power over the contributing developers are important characteristics of OSS development and this freedom also means developers often ignoring formal developmental practices (from CSS development) that may not be as exciting as writing code (Godfrey and Tu, 2000). Thus, developers' individual perceptions could contribute to the informality as illustrated from the third comment above; if they don't feel like using a formal practice, then it may not be part of the OSS development.

Certain OSS project characteristics, such as being large and successful, could contribute to the observed usage means in Table 5. Other characteristics, such as being not-for-profit, could impede the usage of certain practices such as feasibility assessment as suggested by the following respondent's comment: "*feasibility assessment: this is rarely applicable as the basic premises assume ROI (return of investment) expectations which are mostly non-existent, domain constraints are in my experience always used since most of the times people move towards things that 1) they feel they have a constrain* [sic] *and want to learn and they understand and want to improve.*" It appears from the preceding comment that, during the course of development, developers may obtain some understanding of constraints in the application domain that may direct their subsequent programming activities. This could be a potential informal manifestation of the practice: looking for domain constraints that in turn could account for some of the reported usages.

The ad hoc nature of OSS evolution, where, for example, some developer(s) driven by some vision or need, make an initial piece of source code openly available and gradually many interested developers joining in (Nakakoji et al., 2002; Godfrey and Tu, 2000), might contribute to the informality, as is consistent with the following respondent's comment: "*I have yet to work on or be a part of an open source project that* *had a formal requirements gathering process. Open source projects seem to arise out of a specific need and grow somewhat organically rather than through formal solicitation.*"

In Table 5, reported usage of the practice *define the software's operating environment* was higher than for other practices. The operating environment of the software includes the host computer and hardware and software with which the proposed software would be interacting (Sommerville and Swayer, 1997). Developers are expected to have an understanding of such technical aspects which in turn could contribute to the observed usage. This is consistent with the following comment of a responding OSS developer: "*software environment is usually obvious, in my experience. Never built open source for business yet, it's mostly to get something done.*" Note that the preceding comment also points to low usage of the practice "use business concerns to drive requirements elicitation," potentially because of the general objective of OSS projects to produce something useful/interesting for the community, instead of commercial goals as in CSS development (e.g., Godfrey and Tu, 2000).

Table 6. Descriptive statistics: Usage of requirements analysis and negotiation practices in OSS development

Requirements analysis and negotiation practices (Sommerville et al, 1997)	**Mean**	**Std. Dev.**
Define software boundaries	3.11	1.362
Use checklists for requirements analysis	2.36	1.293
Provide software to support negotiations	2.47	1.580
Plan for conflict and conflict resolution	2.11	1.352
Prioritize requirements	3.57	1.286
Classify requirements using a multidimensional approach	2.11	1.440
Use interaction matrices to find conflicts and overlaps	1.8	1.368
Assess requirements risk	2.34	1.364

Table 6 shows the means, and standard deviations for respondents' reported usage for requirements analysis and negotiation practices.

The usage of formal requirements analysis and negotiation practices appears to be low during OSS development, as indicated by the following comments of responding OSS developers: "*Most of these actions are only required in a formal format, when taking place within a corporate or business setting, as so many technically incompetent people are generally involved in development*" and "*A lot of these are pretty heavyweight processes that I would associate more with large slow moving software companies rather than with free software projects.*" The first of the immediately preceding comments is indicative of the differences in the focus of OSS domain versus CSS domain. CSS development gives much greater attention to the needs of the user base ("technically incompetent people") in comparison to OSS development; a large number of OSS projects are developed primarily for developers (e.g., Foushee, 2013). This widespread self-centeredness of OSS developers could contribute to the informality, as illustrated by the following comments of responding OSS developers: "*As an open source developer, I rarely negotiate on my own projects with others as they are built by myself only. I do not need to negotiate or compromise most of the time*" and "*Plan for conflict and conflict resolution is used trivially; implementers make the final decision.*" As evident from the comments, OSS developers, potentially because of self-centredness, often do not feel the need to engage in negotiation or conflict resolution. Practices such as assessing requirements risk may be used only in certain situations (e.g., OSS projects having some legal implications, for example, health care projects) as suggested by the following respondent's comment: "*Assess requirements risk is mostly not applicable, however when used are mostly to prevent legal implications.*"

The average OSS project may be lacking these practices, while large and successful OSS projects may use them, a potential contributor to the observed usage. This is suggested by the following respondent's comment: "*It's a small project. Our process for conflict avoidance is 'ask that guy or that guy,' depending on domain or both. On IRC.*" This comment illustrates the simple, informal communication/discussions (on mediums such as internet relay chat (IRC)) that may occur between developers over some issue. This is in line with the observations of Noll and Liu who noted that brief discussions might occur between developers, for example, to discuss the merits of some feature; disagreements can occur during such discussions leading to additional discussions (Noll and Liu, 2010). Many OSS projects are small in size (e.g., single developer projects, projects having less than five developers) (e.g., Krishnamurthy, 2002) and such projects may lack formal requirements analysis and negotiation practices, as indicated by the following comment: "*All my open source projects thus far been primarily solo efforts (thus no conflicts); on rare occasions, users have requested new functionality, but that rarely constitutes negotiation.*"

The ad hoc nature of OSS evolution could make the usage of certain practices, such as conflict resolution, a gradual emergence process. "*Seriously? You are asking if people use e-mail to talk to other people about software. Also, it's not clear WHEN in the development you are talking about. Many projects start out completely ad hoc and develop conflict resolution plans if they become sufficiently large. Am I supposed to* *answer this question as of a new open source project one guy is writing, or for Linux?*" For a large number of OSS projects, some developer(s) driven by a vision or need might make an initial piece of source code openly available, and gradually many interested developers and contributors join in (Nakakoji et al., 2002; Godfrey and Tu, 2000). With the increase in the number of contributors, the likelihood of occurrence of conflicts and disagreements can go up which, in turn, could result in the occurrence of some informal or formal conflict resolution activity, as the preceding comment points out. The ad hoc nature of OSS evolution and ad hoc emergence of requirements information could impede usage of formal analysis and negotiation practices as the following comment stresses: "*Again important to note in our practice, requirements evolve alongside the software, rather than being agreed to in advance of implementation.*" Evolution of the OSS software could go in any direction, being highly active with continuous developmental activities or becoming dormant with much less activity (e.g., Khondu et al., 2013) which could, in turn, facilitate or deter evolution of formal developmental practices including RE practices. Linux is an example of an OSS project whose successful evolution led to the subsequent emergence of formal developmental practices; Linus Torvalds wrote and made available the initial version of source code of Linux and over time, it became a successful OSS project by attracting and retaining many contributors, which resulted in the emergence of new practices such as coordination practices (Iannacci, 2005).

From Table 6, it can be seen that the practice "prioritize requirements" has the highest mean. Laurent and Cleland-Huang (2009) in their analysis of the discussion forums of OSS projects, found that some prioritization techniques might be available, such as using a voting button through which users could demonstrate their support for some feature request, and in one project they found that users could specifically assign priority to their feature request information. The issue reporting artifacts in OSS development usually have a specific field labeled priority (see the screenshot in Appendix 4). Availability of such techniques could contribute to the observed usage of "prioritizing requirements". Laurent and Cleland-Huang also mention that these techniques have their own problems and hence there is a lack of sophisticated support for requirements prioritization.

In their analysis of OSS projects, Ernst and Murphy (2012), found that there was no separate prioritization phase, but rather interests of developers were the important deciding factor. The interests of developers being a critical factor in requirements prioritization is also illustrated by the following comment of a responding OSS developer: "*Prioritize requirements mostly happen in the core team while others engage with the project on their own rhythm.*" Laurent and Cleland-Huang also found that prioritization decisions are made by project administrators and they may use information provided by users while making these decisions, but there was also evidence that administrators often ignored user provided information. Thus, the self-prioritization of OSS developers about requirements to implement could also contribute to the observed usage of requirements prioritization practice.

Some informal discussions concerning requirements negotiation/agreement about requirements may happen between OSS developers, for example on discussion forums (Llanos and Castillo, 2012; Noll and Liu, 2010). For example, developers may discuss feature implementation in development mailing lists (Guzzi et al., 2013). Such discussions could contribute to the observed usage of the practice "define software boundaries," which is deciding what should be kept within the scope of software and what should be outside (Sommerville and Swayer, 1997).

Table 7 shows the means and standard deviations for respondents' reported usage for requirements describing practices.

Table 7. Descriptive statistics: Usage of requirements describing practices in OSS development

Requirements describing practices (Sommerville et al, 1997)	**Mean**	**Std.**
Define standard templates for defining requirements	2.2	1.401
Use languages simply and concisely	3.45	1.335
Supplement natural language with descriptions of requirement	3.05	1.264
Specify requirements quantitatively	2.49	1.308
Use diagrams appropriately	2.95	1.203

Responding developers' comments indicate that code fragments may actually serve as requirements description: "*I will say simply writing a unit-test fits the description of the first question, which is a very popular way of both defining and enforcing a requirement*" and "*concise language may be hard (esp. if one is attached to a project/ or one wants to let the code do the talking)*" and "*supplements exist more in the form of code then math*"

The practice "use languages simply and concisely," has the highest mean. Requirements may exist informally as part of natural language text descriptions found in communication artifacts such as discussion forums within OSS development environments (Vlas and Robinson, 2012; Scacchi, 2002). The emergence of natural language artifacts as part of the evolution of OSS projects, such as messages posted in discussion forums and issue data in issue repositories, could contribute to the observed usage of the practice "use languages simply and concisely."

Table 8 shows the means and standard deviations for respondents' reported usage for requirements modeling practices.

Table 8. Descriptive statistics: Usage of requirements modeling practices in OSS development

Requirements modeling practices (Sommerville et al, 1997)	**Mean**	**Std. Dev.**
Develop complementary models of the proposed software	2.13	1.303
Model the software's environment	2.57	1.324
Model the software's architecture	2.93	1.376
Use structured methods for software modeling	2.26	1.421
Use a data dictionary	2.10	1.311
Document the links between user requirements and models	1.87	1.312

Comments of responding OSS developers indicate a low usage of requirements modeling practices (which is in line with Badreddin et al. (2013) who, in their analysis of twenty OSS projects, did not find any evidence for usage of modeling practices): "*Rather than model a large program, we typically modularize our stuff*" and "*We have separate deploy scripts (called DevStack) for making a minimal working environment developers can work against. Smaller than a typical production environment*" (see additional comments below). In the first of the immediately preceding comments, modularization refers to a technique of programming that separates a large program into separate independent modules, each module focusing on some particular functionality or a part of it (e.g., https://en.wikipedia.org/wiki/Modular_programming). Both of the immediately preceding comments illustrate the code-centric approach that is characteristic of OSS development, an approach different from that in a CSS development environment and a potential contributor to the informality. In fact, OSS developers may use the code as a substitute artifact for some of the modeling artifacts as suggested by a responding OSS developer: "*Is the code a data dictionary? I think the code is a data dictionary. It shows model names inherently and does not support casual variations*." Such usage occurrences could contribute to the observed usage.

The self-centeredness of OSS developers could contribute to the informality as suggested by the following respondents' comments: "*If the project is in the head of one developer, it is a waste of time to model the project out using UML*" and "*The engine guy knew what he was doing and did it.*" Large and successful OSS projects could be potential contributors to the observed values in Table 8: "*None of the open source projects that I have worked on have been large enough or complex enough to demand requirements modeling.*" The ad hoc nature of OSS evolution could impede usage of certain modeling practices as indicated by the following comment: "*A data dictionary expects a large group which is not true for most of the cases. When the group grows, there is either enough base that it is no longer possible to break the convention or a list has been built via documentation; is also worth noting that generally speaking new contributors consult existing team for guidance on implementing their intended feature or fix*." As the preceding comment suggests, the small size of many OSS projects could impede practices such as a data dictionary that expect large team size but, interestingly, even if a small OSS project manages to grow large through successful evolution, the existing conventions and norms in the project may make it difficult to use practices such as a data dictionary.

Table 9 shows means and standard deviations for respondents' reported usage for requirements validation practices.

Table 9. Descriptive statistics: Usage of requirements validation practices in OSS development

Requirements validation practices (Sommerville et al, 1997)	**Mean**	**Std. Dev.**
Check that the requirements document meets your standards	2.23	1.551
Organize formal requirements inspection	1.96	1.427
Use multidisciplinary teams to review requirements	2.03	1.301
Define validation checklists	1.89	1.251
Use prototyping to animate requirements	2.51	1.373
Write a draft user manual	2.64	1.352
Propose requirements test cases	2.99	1.325
Paraphrase models	1.89	1.378

The usage of requirements validation practices appears to be low during OSS development, as indicated by the following comment of a responding OSS developer: "*The standards met are ad hoc and non-explicit in most cases. Most open source projects I worked on have been too small to bother or be able to afford domain experts and what not, or even have sizeable teams.*" This comment indicates that because of the ad hoc nature of OSS development, there are often no explicit standards, potentially impeding the usage of practice checking whether requirements have met standards or not. Also, the comment indicates that the lack of resources (e.g., money) for many OSS projects can mean a lack of domain experts, hindering the usage of practices such as formal requirements inspection; the small size (e.g., small number of contributors) for many OSS projects can hinder the usage of practices such as using multidisciplinary teams for reviewing requirements. Another developer comment indicates that many OSS projects are small in size, especially in the initial stages of evolution and even if they manage to gain successful evolution and become large in size (e.g., many contributors), there may still be a lack of organized development and developmental activities may be limited to issue generation and submission of code fragments by contributors: "*Again, most open source projects start with one developer and an idea. If the project gains public attention/motivation, public development is done in a less organized manner via issues and pull requests on GitHub.*" This again suggests low usage of formal requirements validation activities. This is also in line with what Noll and Liu (2010) report based on their analysis of OpenEMR, an OSS project. They found instances when no validation happened; a requirement message would be posted by someone and after some time, an implementation of it emerged. In other instances, some informal discussions about validating some feature (e.g., its merits) happened among developers (Noll and Liu, 2010), a potential contributor to the observed usage in Table 9. In Table 9, it can be seen that the practice, "propose requirements test cases", has the highest usage mean. An example of test cases associated with a requirement is: for the requirement *store last name,* possible test cases can be regular length, maximum length, longer than allowed, blank, etc.; for example, what would be the software behavior on entering last names of these types (Zielczynski, 2007). The direct association of testing with coding (e.g., testing code fragments to see if its behavior is as expected or not) could contribute to the observed usage of the practice, proposing requirements test cases. This is also indicated by the following developer comment: "*We unit-test the living beans out of our software if that counts.*"

Table 10 shows means and standard deviations for respondents' reported usage for requirements management practices.

Table 10. Descriptive statistics: usage of requirements management practices in OSS development

Requirements management practices (Sommerville et al, 1997)	**Mean**	**Std. Dev.**
Uniquely identify each requirement	2.6	1.431
Define policies for requirements management	2.05	1.342
Define traceability policies	1.83	1.350
Maintain a traceability manual	1.57	1.144
Use a database to manage requirements	2.28	1.509
Define change management policies	2.13	1.421
Identify global software requirements	2.52	1.460
Identify volatile requirements	2.27	1.415
Record rejected requirements	2.43	1.424

The usage of requirements management practices appears to be low during OSS development as indicated by the following comments of responding OSS developers: "*Much of what is done is done in a non-formal manner*" and "*No formal requirements management has been performed for this very small project.*" The second of the immediately preceding comments indicates the possibility that it may be the large OSS projects that may have some of these practices, a potential contributor to the observed usage in Table 10.

Issue tracker emerged as a potential substitute for a requirements database as indicated through the following comments: "*We are using the Google Code issue tracker to keep track of requirements*" and "*I use a database to track requirements in so far as I frequently use enhancement tickets in bug trackers to define functionality that needs implementing.*" Thus, issue tracker usage could be a potential contributor to the observed usage for the practice: using a database to manage requirements. Some of the issue trackers may have features for uniquely identifying each issue (e.g., issue id) which could potentially contribute to the observed usage of the practice: uniquely identify each requirement. This is indicated by the following comment: "*The database is typically an issue manager, e.g., GitHub issue tracker, Bugzilla or JIRA, which automatically includes an identifier. Depending on the project, these may be individual work items, larger goals or both (usually with links from the small items to the requirement item).*" Version control systems, such as Git, also emerged as a potential substitute for requirements database as evident from the following comment: "*Our blueprints (which generally outline implementation strategy but could include requirements/rationale) are all stored in a git-based repo. This would count as a DB and handle revision control/versioning.*" Git repositories allow storing and tracking revisions/changes to not only code but also any text files or manuscripts (Blischak et al., 2016), thus facilitating the management of requirements as well, as the preceding comment indicates.

Table 11. Descriptive statistics: usage of informal requirements generation activities in OSS development

Informal requirements generation activities reported in OSS literature [see Table2 in chapter two]	**Mean**	**Std. Dev.**
Requirements are asserted by an open source software developer based on his or her personal experience	4.09	1.031
Requirements are asserted by an open source software developer based on his or her personal knowledge of user needs	3.83	1.107
Requirements are contributed by users through bug reports	3.81	1.035
Requirements are contributed by users through feature requests	3.65	1.043
Requirements are derived from features found in some other software	3.36	1.143

Table 11 shows the means and standard deviations for respondents' reported usage for informal requirements generation activities.

It can be seen from Table 11 that the usage means are higher compared to the means for most of the practices in Tables 4-10. The usage of informal requirements generation activities in OSS development appears to be high as indicated by the following comments of responding OSS developers: "*sudden change in survey tone to include directly applicable questions!*"; "*not very formal here either*" and "*it's mostly what you (dev) wants to build that determines the product- but kind of has to make it useful to people, so that includes some reqs. That he/she may not have wanted to do.*" The third of the immediately preceding comments indicate how developers' perceptions and interests can be the dominant factor in OSS development, for example, developers' personal views about what is useful can determine what features get included in the software, as the comment suggests.

Issue data such as bug reports and feature requests that are usually generated in issue repositories of OSS projects are a major source of requirements, as results in Table 11 indicate and are consistent with existing literature (e.g., Crowston et al., 2012, Noll, 2008). The following comment of a responding OSS developer also suggests this: "*GitHub issues play a huge role.*" In addition to issue data, contributed code fragments may also serve as a source of requirements as the following comment indicates: "*Requirements are often contributed by users as fully formed patches (or GitHub pull requests which can be merged into an existing codebase.*" Here patches/pull requests refer to code fragments submitted by contributors.

3.1.3 Quantitative Evidence for Informality of Requirements Generation in OSS Development

Two informal practices associated with OSS developers are asserting requirements based on personal experience and asserting based on personal knowledge of user needs.

Table 12. Informal assertion versus formal elicitation of requirements

Use of informal requirements generation practices versus use of formal requirements elicitation practices	**Mean Std. Dev.**	**Paired samples t**	**Sig.**
Requirements are asserted by an open source software developer based on his or her personal experience (Informal)	4.09 1.031		
Identify and consult software users (formal)	2.85 1.314	8.609	.000
Collect requirements from multiple viewpoints (formal)	2.61 1.232	8.991	.000
Use business concerns to drive requirements elicitation (formal)	2.35 1.320	9.735	.000
Use scenarios to elicit requirements (formal)	2.62 1.371	7.749	.000
Requirements are asserted by an open source software developer based on his or her personal knowledge of user needs (informal)	3.83 1.107		
Identify and consult software users (formal)	2.85 1.314	6.998	.000
Collect requirements from multiple viewpoints (formal)	2.61 1.232	7.502	.000
Use business concerns to drive requirements elicitation (formal)	2.35 1.320	8.422	.000
Use scenarios to elicit requirements (formal)	2.62 1.371	6.270	.000

This assertive approach of requirements generation does not involve formal elicitation of requirements from potential users, which is different from CSS development, where analysts formally elicit requirements from users using different methods such as interviewing and scenario based elicitation (e.g., Laurent and Cleland

Huang, 2009; Scacchi, 2002).A paired samples t-test was run between usage scores of the two informal assertive practices and the usage scores of four formal requirements elicitation practices that involve direct elicitation of requirements from users in order to investigate if there were any statistically significant differences. Results are reported in Table 12. As Table 12 shows, informal assertive approaches have significantly higher reported usage in OSS development than the formal elicitive approaches of requirements generation. This provides quantitative evidence for the general informality of requirements generation in OSS development and also quantitative evidence for claims about the assertion of requirements made in qualitative OSS literature. For example, Scacchi writes: "*We also observe the assertion of requirements that simply appear to exist without question or without trace to a point of origination, rather than somehow being elicited from stakeholders, customers, or prospective end users of open software systems*" (p.10, Scacchi, 2002). His observation is from qualitative analysis of web-based artifacts of an OSS project.

The ad hoc generation/emergence of issue data (e.g., users may submit some bug report or feature request randomly) in OSS issue repositories is another informal requirements generation practice that can be observed in OSS development (e.g., Noll, 2008). This practice is non–elicitive; that is, there are no analysts formally eliciting the requirements data from users on the web-based issue generation forums (c.f. Laurent and Cleland Huang, 2009). A paired samples t-test was run between usage scores for the two types of issue data, namely bug reports and feature requests as sources of requirements and four formal elicitation approaches. Paired samples t test was used since the paired scores came from the same respondents. The results are shown in Table 13.

Table 13. Informal issue data generation versus formal elicitation of requirements

Use of informal requirements generation practices versus use of formal requirements elicitation practices	**Mean Std. Dev.**	**Paired samples t**	**Sig.**
Requirements are contributed by users through feature requests (Informal)	3.65 1.043		
Identify and consult software users (formal)	2.85 1.314	4.785	.000
Collect requirements from multiple viewpoints (formal)	2.61 1.232	6.694	.000
Use business concerns to drive requirements elicitation (formal)	2.35 1.320	7.776	.000
Use scenarios to elicit requirements (formal)	2.62 1.371	5.303	.000
Requirements are contributed by users through bug reports: (informal)	3.81 1.035		
Identify and consult software users (formal)	2.85 1.314	5.921	.000
Collect requirements from multiple viewpoints (formal)	2.61 1.232	8.355	.000
Use business concerns to drive requirements elicitation (formal)	2.35 1.320	8.900	.000
Use scenarios to elicit requirements (formal)	2.62 1.371	6.768	.000

Table 13 indicates that the usage of issue data generated (bug reports, feature requests) as sources of requirements in OSS development, is significantly higher than the usage of formal elicitive approaches of requirements generation. This provides further quantitative evidence for the general informality of requirements generation in OSS development and also provides quantitative evidence for claims about the usage of issue data (bug reports, feature requests) as sources of requirements in qualitative OSS literature. For example, Alspaugh and Scacchi write: "*Perhaps the most common requirement-like OSS artifacts are isolated feature requests or bug reports submitted to tracking systems like Bugzilla………., Each feature request or bug report can be taken to imply a requirement but in themselves they rarely constitute a Classical Requirements artifact*" (p.3-4, Alspaugh and Scacchi, 2013).

3.2 Quantitative Findings on Perceptions of OSS Developers about the Usefulness of Formal Requirements Engineering Practices from CSS Development for OSS Development

The second section of the survey asked respondents to indicate what they thought about the usefulness of adopting formal requirements engineering practices (from CSS development) in OSS development. They were asked to indicate the usefulness of each requirements engineering practice for OSS development by selecting one of the following seven options: extremely useful (coded as 5), very useful (coded as 4), useful (coded as 3), not useful (coded as 2), harmful (coded as 1), not applicable (coded as 0) and I do not know (coded as 0). To determine whether there was a statistically significant difference in the perceptions of OSS developers about the usefulness of RE practices in OSS development and their reported usage (measured as described at the beginning of section 3.1.2) of RE practices, a paired sample t-test was run between usage ratings and usefulness ratings for each RE practice. Past research from other domains such as healthcare has used a similar approach for analyzing rating data (e.g., Bruce and Ritchie, 1997; Chu and Choi, 2000).

One assumption of a paired sample t-test is that the differences of paired observations are normally distributed. The tests of normality are many times not reliable and an alternative approach for testing normality is the analysis of descriptive statistics (Siau and Long, 2009). One rule of thumb that can be used to investigate whether there is a serious violation of normality assumption or not is to see if absolute values of both skewness and kurtosis are less than 1 (Siau and Long, 2009). If so, there is acceptable normality (Siau and Long, 2009). For the data, most of the skewness scores (except for two) are less than 1. If skewness is greater than 1 but less than 2, then also the distribution is not highly skewed (Siau and Long, 2009). The two greater than one skewness scores are 1.151 and 1.217. Many of the kurtosis values also have absolute values less than 1 and many of the remaining ones having values greater than 1 but less than 2 (see Appendix 6). West et al. (1995) mention that substantial departure from normality happens when skewness > 2 and kurtosis > 7 whereas Kline mentions that skew with an absolute value greater than 3 and kurtosis with an absolute value greater than 10 are problematic (c.f. Stull, 2008). The skewness and kurtosis of the data are below all of these threshold values.

Sawilowsky and Blair (1992) investigated the robustness of t-test with eight real world non-normal distributions and different sample sizes. The t-test was found to produce robust results with the different non-normal real world distributions (Sawilowsky and Blair, 1992). Schmider et al. (2010) analyzed robustness of ANOVA (of which t-test is a special case) with non-normal rectangular distribution (skewness = 0; kurtosis = 1.8) and exponential distributions (skewness = 2; kurtosis = 9) and found ANOVA to produce robust results. The findings of Sawilowsky and Blair and Schmider et al. show that kurtosis values like 4 and 9 do not cause problems for parametric tests such as t-test and ANOVA. Seven kurtosis values were between 2 and 3 and three kurtosis values were greater than 3. The findings of Sawilowsky and Blair and Schmider et al. indicate that these values should not be a hindrance in t-test producing accurate results for the corresponding RE practices.

Bootstrapping is a method that can be used in the context of non-normal data (Lumley et al., 2002). Bootstrapping was found to provide similar p-values and confidence intervals as that of t-tests (Lumley et al., 2002). Kang and Harring (under review) report that a bootstrap t-test has power advantages over a normal t-test with non-normal data. SPSS provides the option to do bootstrapping in combination with parametric tests such as t-test. Another alternative for non-normal data is using non-parametric tests, which do not have normality assumptions (Nevo et al., 2012). One such test is sign test which can be used to test whether there is a statistically significant difference between scores in two conditions. To determine whether there were significant differences in perceptions of OSS developers about the use and perceived usefulness of RE practices, a paired samples t-test, bootstrap paired samples t-test and sign test were run for use and perceived usefulness scores of each RE practice. The results for paired samples t-test and sign test are presented in Tables 14 to 20. Running bootstrapped paired sample t-test produced similar significance values as shown in Tables 14 to 20 and confidence intervals that did not contain zero. This means that the true difference between means cannot be zero, thus supporting the findings of normal paired samples t-test and sign test. In addition, the Bonferroni correction method was also applied since this is a case of multiple comparisons (discussed in detail at the end of this section; also Appendix 2).

In general, perceptions of OSS developers about the extent to which the formal RE practices (from CSS development) could be advantageous for OSS development do not match their reported usage of these practices. For most of the RE practices, their perceptions about usefulness was significantly higher than their reported usage, indicating a gap in their perceptions and practice. Thus, it appears that when it comes to requirements related activities in OSS development, OSS developers may not be actually practicing what they believe could be beneficial. Many potential reasons for this observed gap emerged from responding developers' comments; for example, limited size and scope of OSS projects and limited project resources such as a small number of contributors available and time constraints of volunteering developers. It appears that manifestations of some of the formal RE practices within the existing OSS development workflows could help account for these constraining factors while allowing the domain to pursue and enjoy the benefits of these practices. A detailed discussion of the findings follows.

The results of the paired samples t-test and sign test for requirements documentation practices are shown in Table 14.

Table 14. Perceived usage versus usefulness: requirements documentation

Requirements documentation practices (Sommerville et al, 1997)	**Use**		**Usefulness**		**Paired samples t-test (t value: use - usefulness)**	**Sig. (2-tailed)**	**Sign test (use - usefulness) p-value**
	Mean	**SD**	**Mean**	**SD**			
Define a standard document structure	2.34	1.346	3.18	1.106	-6.924	.000	.000
Explain how to use the document	2.17	1.404	3.11	1.169	-6.154	.000	.000
Include a summary of the requirements	2.78	1.556	3.5	1.033	-4.183	.000	.008
Make a business case for the software	1.99	1.329	2.7	1.429	-4.925	.000	.000
Define specialized terms	2.63	1.299	3.33	1.245	-4.333	.000	.000
Make document layout readable	3.09	1.565	3.77	1.210	-4.128	.000	.002
Help readers find information	2.85	1.483	3.77	1.169	-5.795	.000	.000
Make the document easy to change	3.27	1.707	3.79	1.204	-3.130	.002	.003

As evident from the test results, for all the requirements documentation practices, the beliefs of OSS developers about the usefulness of these practices for OSS development was higher than their reported usage of these practices during OSS development, indicating a gap between their perceptions and practice.

The usefulness perceptions are also indicated by the following comments of responding OSS developers: "*I am in charge of keeping our software's docs clean, and a style template for it has been a huge help*" and "*Documentation whether of requirements or implementation becomes more crucial as a project grows. For small projects, it can be detrimental, but for larger ones, important.*" The second of the immediately preceding comments indicates that large OSS projects may find requirements documentation more useful, a potential indicator that the variable OSS project size can influence requirements documentation usefulness. OSS project documentation may help in the evolution of the project, for example, Aggarwal et al. (2014) found a positive association between OSS project documentation and OSS project popularity. Environmental cues can influence perceptions of individuals (Baker et al., 2002) and thus the lack of documentation could negatively influence perceptions of many potential users and contributors about the OSS projects. Thus, the availability of documentation (e.g., requirements documentation) could be potentially beneficial for the growth of small OSS projects.

Some potential reasons for the observed gap in perceptions and practice emerged from responding developers' comments. Having a small number of contributors and a small project scope could potentially contribute to the observed gap as suggested by the following respondent's comment: "*My answers assume some large, general use project with many developers (who I assume would be the primary users of requirements docs). Many projects are small in scope and targeted to a small subset of users and don't generally need requirements documentation.*" The limited amount of time that many volunteering OSS developers have available to contribute could be another potential contributor to the observed gap as suggested by the following respondent's comment: "*Useful or no, time constraints will likely prevent me from doing this. I am involved with very small development houses, we try to turn out clean code as quickly as possible by passing unnecessary steps whether they are useful or not.*" Thus, any improvement efforts involving requirements documentation in OSS development will have to take these constraints into account.

Future research could explore additional reasons for the observed gap in perceptions and practice. Efforts could be made to explore how some of the formal requirements documentation practices perceived as beneficial for OSS development could be incorporated into existing OSS development workflows. For example, requirements may be specified as postings on discussion forums and such artifacts (Noll and Liu, 2010; Scacchi, 2002). One possible research direction could be to evaluate whether some standard structure could be enforced on the posting of requirements related messages on such forums. Many OSS projects may have sort of informal documentation, such as project README files (Begel et al., 2013); future research could look at whether and how some standard structure could be brought into such informal documentation. The requirements related messages may contain technical terms that may be difficult to comprehend. For example, within posted messages, code fragments may be included (e.g., Scacchi, 2002; Vlas and Robinson, 2013). Descriptions and explanations of such technical information items could accompany a posted message, for example, as a pop-up window, a potential adaptation of the practice, defining specialized terms.

The results of the paired samples t-test and sign test for requirements elicitation practices is shown in Table 15. As can be seen from Table 15, for most of the requirements elicitation practices, the perceptions of OSS developers about the usefulness of these practices for OSS development are significantly higher than their reported usage of these practices during OSS development.

Table 15. Perceived usage versus usefulness: requirements elicitation

Requirements elicitation practices (Sommerville et al, 1997)	Use		Usefulness		Paired samples t-test (t value: use - usefulness)	Sig. (2-tailed)	Sign test: (use - usefulness) p-value
	Mean	SD	Mean	SD			
Assess software feasibility	2.7	1.552	3.19	1.106	-6.924	.005	.013
Be sensitive to political and organizational consideration	2.23	1.434	2.73	1.221	-3.400	.001	.009
Identify and consult software users	2.84	1.338	3.71	1.048	-5.932	.000	.000
Record requirement sources	2.48	1.468	3.10	1.206	-4.063	.000	.000
Define the software's operating environment	3.62	1.411	3.5	1.230	.691	.491	.203
Use business concerns to drive requirement elicitation	2.35	1.328	2.62	1.321	-1.888	.063	.298
Look for domain constraints	2.7	1.504	3.18	1.308	-3.025	.003	.005
Record requirements rationale	2.57	1.334	3.39	0.940	-5.397	.000	.000
Collect requirements from multiple view points	2.61	1.245	3.45	1.056	-6.185	.000	.000
Prototype poorly understood requirements	2.83	1.367	3.51	1.185	-4.498	.000	.001
Use scenarios to elicit requirements	2.66	1.355	3.13	1.350	-2.868	.005	.004
Define operational processes	2.35	1.501	3.04	1.259	-4.586	.000	.000
Reuse requirements	2.6	1.411	3.1	1.168	-3.140	.002	.003

This indicates a gap between their perceptions and practice of requirements elicitation. The perceptions about the usefulness of requirements elicitation practices for OSS development is supported by the following comment of a responding OSS developer: "*I am not going to go through all of these, but they generally seem useful*." There was no significant difference between perceptions of usefulness and use for the practice: "Define the software's operating environment". The operating environment of the software is often comprised of host computer, hardware, and software with which the proposed software has to interact. Developers may naturally think about and work on such technical issues as part of the implementation and thus the practice "Define the software's operating environment" may naturally become part of their developmental activities, a potential reason for the lack of a significant difference.

Some potential reasons for the observed gap between perceptions and practice also emerged from the comments of responding OSS developers. For example, small project size (e.g., small number of contributors) may contribute to the observed gap, as suggested by the following comment: "*Those marked useful above would be, but only in the context of a sufficiently large project. For small open source projects (i.e., the scale I generally work at), requirements gathering is a personal thing because the project starts to scratch the developer's itch – hence there is no need for formalizing it. That said, I rarely see a need to pander to political and organizational considerations*." The preceding comment also illustrates the case of many small OSS projects being self-driven with developers focused on their personal needs instead of the needs of the potential user base and may be asserting requirements. Laurent and Cleland Huang (2009) report that

OSS users often got annoyed or perplexed because their feature requests were getting ignored. Such experiences could lead to user disinterest in the project which, over the long term, can negatively influence interest of the OSS developers in the project and OSS project developmental activities (e.g., Subramaniam et al., 2009; Stewart et al., 2006). In fact, user interest and developer interest can positively influence each other (Subramaniam et al., 2009). There is evidence that a large number of OSS projects have failed to deliver operational software, failed to attract volunteers or have become inactive (Katsamakas and Georgantzas, 2007; Khondu et al., 2013). Incorporating some form of formal requirements elicitation could potentially help small OSS projects expand their scope and size, getting users interested when finding that their needs are getting attention and getting contributors interested by providing them with a wider range of useful things on which to work (e.g., user interest has a positive impact on OSS project activity levels (p.579, Subramaniam et al., 2009)). Expanded project size and scope (e.g., higher OSS project activity levels, number of users interested in OSS projects, number of developers interested in OSS projects) are measures of OSS success (Subramaniam et al., 2009).

Other potential reasons for the observed gap between perceptions and practice of requirements elicitation in OSS development could be limited time available for developers to contribute, small team size and the geographically distributed nature of OSS development as suggested by the following respondent's comment: "*Since, this is, basically, a hobby for us, time is a very expensive resource. Being a small team, without physical proximity to each other, some of these things would simply kill the project dead.*"

Thus, any improvement effort involving requirements generation in OSS development should take into account some of these potential constraining factors.

Future research could explore additional reasons for the observed gap in perceptions and practice and the potential ways in which some of the formal requirements elicitation practices could be incorporated into existing OSS development workflows. For example, requirements reuse could be facilitated within web-based OSS development environments by making available a library or repository of reusable requirements. Geographically distributed contributors could then work with this central library or repository in parallel, using and making modifications as needed. Moreover, requirements reuse avoids the need to start from scratch thus saving a lot of time and effort (Hoffmann et al., 2013) which fits well with the time constraints of OSS developers. The actual benefits from some of the formal requirements elicitation practices perceived as useful for OSS development could be investigated using methodologies such as a field experiment with control and treatment conditions.

The results of the paired samples t-test and sign test for requirements analysis and negotiation practices is shown in Table 16. As Table 16 indicates, for most of the requirements analysis and negotiation practices, the perceptions of OSS developers about the usefulness of these practices for OSS development is significantly higher than their reported usage of them. This indicates a gap between their perceptions and practice of requirements analysis and negotiation.

Table 16. Perceived usage versus usefulness: requirements analysis and negotiation

Requirements analysis and negotiation practices (Sommerville et al, 1997)	**Use**		**Usefulness**		**Paired samples t-test (t value: use - usefulness)**	**Sig. (2-tailed)**	**Sign test: (usefulness - use) p-value**
	Mean	**SD**	**Mean**	**SD**			
Define software boundaries	3.11	1.362	2.99	1.329	-2.061	.042	.289
Use checklists for requirements analysis	2.36	1.293	2.99	1.215	-4.770	.000	.000
Provide software to support negotiations	2.47	1.580	2.78	1.498	-1.767	.081	.131
Plan for conflict and conflict resolution	2.11	1.352	3.01	1.222	-6.279	.000	.000
Prioritize requirements	3.57	1.286	3.85	1.032	-1.863	.066	.077
Classify requirements using a multidimensional approach	2.11	1.440	2.67	1.449	-3.689	.000	.000
Use interaction matrices to find conflicts and overlaps	1.8	1.368	2.36	1.384	-4.487	.000	.000
Assess requirements risk	2.34	1.364	2.9	1.462	-4.223	.000	.000

The perceptions about the usefulness of requirements analysis and negotiation practices is also supported by the following comment of a responding OSS developer: "*It is extremely important to define software boundaries, as developers that are not held by deadline constraints (due to lack of management in open source projects) are prone to creating code bloat and useless features that are fun to develop.*"

Some potential reasons for the observed gap between perceptions and practice of requirements analysis and negotiation emerged from the comments of responding OSS developers. For example, small project size and scope could contribute to the observed gap, as suggested by the following respondent’s comment: “*Again much of this will vary with project size and scope. Here multidimensional classification may be extremely valuable for a large project, but trivial or needlessly complex for a smaller one.*” Lack of financial resources could be another contributor as suggested by the following respondent’s comment: “*Small team, small conflict. But money is a thing.*” The largely non-commercial nature of OSS development could also contribute to the observed gap for practices such as assessing risk, as indicated by the following comment: “*Risk to schedules are rare enough in the open source projects that I worked on.*” It would be useful for any improvement effort involving requirements analysis and negotiation in OSS development to consider these constraining factors as part of the analysis and planning.

Future research could investigate other plausible reasons for the observed gap in perceptions and practice and look at ways of efficiently incorporating some of the requirements analysis and negotiation practices into existing OSS development workflows. The responding OSS developers pointed out some informal ways of carrying out some of the analysis and negotiation practices, and research efforts could focus on how to improve them. For example, mailing lists and issue repositories could be potential substitute artifacts for software supporting negotiation as suggested by the following comment of a responding OSS developer: “*Provide software to support negotiations: mailing lists and bug tracking systems are often used for this as well via assignment and debate.*” Research effort could be directed at how to improve these artifacts to better support requirements negotiation. For example, Schoop et al. (2003) have proposed a categorization of messages sent between negotiators, based on speech act theory (request, offer, counter-offer, accept, reject, question and clarification). These message categories could be incorporated into mailing lists and issue repositories, for example as message or issue headers.

Forking (creating a duplicate project that is a copy of parent OSS project and which evolves in parallel and has a different development team (e.g., Robles and Gonzalez-Barahona, 2012)) appears to be an informal conflict resolution practice in OSS development. This is suggested by the following comment of a responding OSS developer: "*I would say fork is the natural conflict resolution method in open source: typically an open source project will have some central authority (either individual or group based) which will decide strategy for the project. If people disagree, they are free to fork the project (assuming they feel that strongly about the conflict).*" In fact, GitHub has a "fork" feature with each OSS project repository that allows anyone to create a separate copy of an existing OSS project along with its own infrastructures such as versioning systems and mailing lists. The central authority in the preceding comment is referring to core developers who may decide what to include or exclude. The diversity of interests among OSS developers can give rise to conflicts (e.g., Joode and Vendel, 2004) and subsequently to behaviors such as forking as the preceding comment points out. Research efforts could be directed at investigating whether the method of forking could be improved as a conflict resolution mechanism, for example, making features such as forking more restricted and allowing it to proceed only if a list of identified conflicts remains unresolved.

The results of the paired samples t-test and sign test for requirements description practices is shown in Table 17.

Table 17. Perceived usage versus usefulness: requirements description

Requirements description practices (Sommerville et al, 1997)	**Use**		**Usefulness**		**Paired samples t-test (t value: use - usefulness)**	**Sig. (2-tailed)**	**Sign test (use - usefulness) p-value**
	Mean	**SD**	**Mean**	**SD**			
Define standard templates for defining requirements	2.2	1.401	3.05	1.304	-5.154	.000	.000
Use languages simply and concisely	3.45	1.335	3.89	1.018	-2.863	.005	.041
Supplement natural language with descriptions of requirement	3.05	1.264	3.59	1.127	-3.990	.000	.000
Specify requirements quantitatively	2.49	1.308	3.12	1.391	-4.234	.000	.000
Use diagrams appropriately	2.95	1.203	3.79	0.984	-6.795	.000	.000

The results in Table 17 point towards a gap in the perceptions and practice of OSS developers about requirements description, since their perceptions about the usefulness of requirements description practices for OSS development do not match their reported usage of these practices. For all the requirements description practices, the extent to which OSS developers perceived them as beneficial for OSS development was significantly greater than their reported usage of these practices.

The perceptions about the usefulness of requirements description practices are also suggested by the following comment of a responding OSS developer: "*Much of this* *is already good practice, but it does not just live in a requirements document. It lives in the project readme, in the software tests, in the project page, and GitHub issues.*" The preceding comment, while indicating the usefulness of requirements describing practices, also points out some related informal OSS artifacts and practices. A comment from another responding developer indicates how the practice using language simply and concisely may be useful for code writing: "*It is vital to use as little/simple code as possible to accomplish goals/features in a software project.*" This practice could be potentially incorporated, for example as a documented guideline, especially when code fragments become manifestations of some requirement. The guideline could also apply to natural language messages (many of which may be informal requirements descriptions (e.g., Scacchi, 2002)) posted in discussion forums and issue repositories.

Future research could look at whether and how the existing OSS artifacts and practices could be improved to be adaptations of some of the formal requirements description practices. For example, requirements information may exist informally as part of the natural language text in the postings in discussion forums and issue repositories (e.g., Vlas and Robinson, 2011). The practice "define standard templates for defining requirements" could be potentially adapted by attempting to bring in some standards and guidelines for construction of requirements messages. This is suggested by the following comment of a responding OSS developer: "*Standard templates for requirements are useful, but assuming a bug tracker is used to track requirements the template can be provided implicitly by the fields of the database.*"

The results of the paired samples t-test and sign test for requirements modeling practices is shown in Table 18.

Table 18. Perceived usage versus usefulness: requirements modeling

Requirements modeling practices (Sommerville et al, 1997)	**Use**		**Usefulness**		**Paired samples t-test (t value: use - usefulness)**	**Sig. (2-tailed)**	**Sign test: (use - usefulness) p-value**
	Mean	**SD**	**Mean**	**SD**			
Develop complementary models of the proposed software	2.13	1.303	2.74	1.377	-4.436	.000	.000
Model the software's environment	2.57	1.324	3.10	1.292	-4.544	.000	.000
Model the software's architecture	2.93	1.376	3.32	1.312	-3.327	.001	.000
Use structured methods for software modeling	2.26	1.421	2.7	1.471	-2.881	.005	.001
Use a data dictionary	2.10	1.311	2.61	1.497	-3.358	.001	.000
Document the links between user requirements and models	1.87	1.312	2.76	1.393	-6.176	.000	.000

As indicated in Table 18, the extent to which OSS developers view requirements modeling practices as beneficial for OSS development was significantly greater than the extent to which they reported using these practices. This indicates a gap between their perceptions and practice of requirements modeling.

Usefulness perceptions, as well as potential reasons for the observed gap, can be gleaned from the comments of responding OSS developers. Small project size and scope and limited time available for the OSS developers to contribute can contribute to the observed gap as suggested by the following comments: "*All the above might be useful,* *but again only in large scale projects; for small projects, they are a waste of time*" and "*This would again, take time away from development. Might make a more successful product, though… But not at the moment, I don't think it's worth it for us.*" Characteristics of the software itself could be another potential constraining factor as suggested by the following comment of a responding OSS developer: "*More worthwhile for the case where you have software that is all bundled together. Modularity is the way to go though.*" The practice, *modeling the software architecture* which shows how a software system is decomposed into its sub-systems (Sommerville and Swayer, 1997) fits well with the idea of modularity which is dividing a large program into separate independent modules with each module providing a distinct functionality (e.g., https://en.wikipedia.org/wiki/Modular_programming). The different subsystems in the architectural model could correspond to the different modules in the program. It is interesting to note that the practice, *modeling the software architecture*, was rated highest on the usefulness scale.

Future research could explore additional reasons for the observed gap in perceptions and practice and also, potential ways in which some of the modeling practices perceived as beneficial could be incorporated into existing OSS development workflows. The source code is a naturally occurring model artifact in OSS domain. Source code models some real world domain and contain knowledge about domain concepts and their relationships (Ratiu, 2009; Offen, 2002). The following comment of a responding OSS developer hints at this: "*Code is the accurate model of itself. Maintaining or re-presenting code in UML like environments is either an extra chore, a distraction or a new* *source of errors. Languages that are more expressive and readable improve the readability of the code are good or a new source of errors, as are experimental environments like LightTable that allow for contextual understanding of code.*" Feature support in OSS development environments for practices such as *modeling software architecture* (that can be more directly associated with the source code) may be welcomed by the OSS community, since they can improve the readability of the code.

Table 19. Perceived usage versus usefulness: requirements validation

Requirements validation practices (Sommerville et al, 1997)	**Use**		**Usefulness**		**Paired samples t-test (t value: use - usefulness)**	**Sig. (2-tailed)**	**Sign test: (use - usefulness) p-value**
	Mean	**SD**	**Mean**	**SD**			
Check that the requirements document meets your standards	2.23	1.551	2.94	1.215	-5.206	.000	.000
Organize formal requirements inspection	1.96	1.427	2.54	1.302	-3.738	.000	.000
Use multidisciplinary teams to review requirements	2.03	1.301	2.97	1.396	-6.258	.000	.000
Define validation checklists	1.89	1.251	2.77	1.120	-6.633	.000	.000
Use prototyping to animate requirements	2.51	1.373	3.08	1.222	-4.488	.000	.000
Write a draft user manual	2.64	1.352	3.38	1.095	-4.504	.000	.000
Propose requirements test cases	2.99	1.325	3.73	0.983	-5.203	.000	.000
Paraphrase models	1.89	1.378	2.78	1.312	-6.095	.000	.000

New OSS specific modeling languages are emerging. For example, Badreddin et al. (2013) proposed a modeling method known as Umple for OSS projects. Tools that support conversion of models into the code and vice versa (e.g., StarUML) are also emerging. Future research could investigate the usefulness of such tools.

The results of the paired samples t-test and sign test for requirements validation practices is shown below in Table 19.

As evident from Table 19, there is a gap in the perceptions and practice of OSS developers about requirements validation. For all of the requirements validation practices, the extent to which OSS developers perceived these practices as beneficial for OSS development was significantly greater than the extent to which they reported using them.

Usefulness perceptions, as well as potential reasons for the observed gap, emerged from the comments of responding OSS developers. The practice *proposing requirements test cases* was rated highest on the usefulness scale. A supportive comment from a responding OSS developer was: "*Unit-testing is extremely important.*" Research can focus on how to link effectively, requirements existing within OSS artifacts such as mailing lists and issue repositories with the unit-testing that may be carried out by developers. Limited project resources, such as only a small number of contributors available, could contribute to the observed gap as suggested by the following comment of a responding OSS developer: "*We do not have enough people to do this teams' thing, although we do sometimes review our work with other individuals.*" Another responding developer called for the participation of users to support validation activities through the use of tools such as Wikipedia: "*Get people to participate by being open ala Wikipedia this should not be hard for a user to start helping in open source.*"

Because of the in general code-centric nature of OSS development, prototyping may occur naturally in OSS development as the following comment of a responding OSS developer indicates: "*In open source, prototypes frequently become the implementations.*" Research can investigate how such coding activities can be more efficiently directed towards requirements validation. A large percentage of coding activities in OSS projects end up as failures; for example, in Linux Kernel, only about 33% of patches (code fragments) submitted by volunteering developers get accepted to be part of the main source code; the rest get rejected. For Apache and many other projects, only 40% get accepted (Jiang et al., 2013). A major reason for these failures is that the submitted code pieces do not implement some relevant, working feature or bug fix (Jiang et al., 2013). These failures indicate a large percentage of wasted coding effort. Such effort wastage can be potentially reduced through research investigating how to better focus the OSS coding efforts on relevant requirements; for example, how to efficiently direct potential contributors to requirements existing in discussion forums, issue repositories, and project readme files.

The results of the paired samples t-test and sign test for requirements management practices is shown in Table 20. Table 20 shows a gap in the perceptions and practice of OSS developers about requirements management. For all of the requirements management practices, the extent to which OSS developers perceived these practices as beneficial for OSS development was significantly greater than the extent to which they reported using them. Small project size and scope again emerged as a potential contributor to the observed gap as suggested by the following comment: "*I am using the* *biggest projects to answer all these questions. Small FOSS projects I have been involved in have all been very sketchy about documentation and policies.*"

Table 20. Perceived usage versus usefulness: requirements management

Requirements management practices (Sommerville et al, 1997)	**Use**		**Usefulness**		**Paired samples t-test (t value: use - usefulness)**	**Sig. (2-tailed)**	**Sign test: (use - usefulness) p-value**
	Mean	**SD**	**Mean**	**SD**			
Uniquely identify each requirement	2.6	1.431	3.48	1.141	-5.989	.000	.000
Define policies for requirements management	2.05	1.342	2.85	1.278	-5.527	.000	.000
Define traceability policies	1.83	1.350	2.56	1.389	-5.487	.000	.000
Maintain a traceability manual	1.57	1.144	2.32	1.378	-6.477	.000	.000
Use a database to manage requirements	2.28	1.509	3.18	1.270	-7.337	.000	.000
Define change management policies	2.13	1.421	3.01	1.242	-5.789	.000	.000
Identify global software requirements	2.52	1.460	3.23	1.193	-5.834	.000	.000
Identify volatile requirements	2.27	1.415	3.06	1.243	-6.712	.000	.000
Record rejected requirements	2.43	1.424	3.3	1.079	-5.932	.000	.000

Future research could investigate other potential contributors to the observed gap in practice and perceptions.

Additionally, research can also focus on how to efficiently incorporate some of the requirements management practices into existing OSS development workflows. One developer pointed out that the practice traceability could be incorporated into artifacts such as OSS issue repositories through dependency links: "*Traceability is trivial assuming the database used to store the requirements permits dependency links (hence no need for a traceability manual).*" Dependency links between two artifacts (e.g., requirements) X and Y indicate that existence of Y is dependent on the existence of X and changes in X will cause changes in Y (Winkler and Pilgrim, 2010). Winkler and Pilgrim note that techniques such as traceability matrix and graphs can be used for representing traceability. Research can look on how to incorporate such traceability representation techniques into OSS artifacts such as issue repositories. The traceability knowledge itself, i.e., whether two requirements or such artifacts are linked, could be made a crowdsourcing task that interested contributors can contribute to, for example by having a feature that allows users and developers to specify whether two bug requests or feature requests are related or not. Change requests (e.g., request to fix a bug) frequently emerge as OSS projects evolve (Schackmann and Lichter, 2009). Policies that state how such change requests are to be handled (such as historical data on how some of the past change requests were handled) could be made available in a documented form within OSS development environments.

The tests in Tables 14 to 20 are a case of multiple comparisons, and hence the Bonferroni correction method was also applied. When using the original, most conservative Bonferroni correction, the new adjusted alpha is 0.05/57=0.0008772. By using this adjusted alpha, most of the t-test results in Tables 11 to 17 remain significant. A less strict but more powerful method is the Holm-Bonferroni stepwise method in which the p-values are arranged from smallest to largest, and the adjusted alpha for the smallest p-value is 0.05/57, the adjusted alpha for the second smallest p-value is 0.05/56 and so on. The detailed Holm-Bonferroni computation is shown in Appendix 2. When using Holm-Bonferroni correction, most of the results are found to be indeed significant, even with the adjusted alphas. For example, the adjusted alpha for p-value 0.002 (the last row) is 0.004. Since 0.002 is less than 0.004, the result is significant.

There could be many problems and challenges existing in the context of requirements discovery in OSS development that could potentially contribute to the observed gap in practice and perceptions of OSS developers in Tables 14 to 20. Some of these are described further in the next section.

3.3. Problems and Challenges in the Context of Requirements Discovery in OSS Development

The third section of the survey asked respondents to indicate problems and challenges that exist or can occur during requirements discovery in OSS development. The list of problems was obtained from Damian and Zowghi (2003), who identified them in the context of a multi-site geographically distributed CSS development organization and also Schmid (2014). Table 21 shows the percentage of respondents who reported each issue as a problem for requirements discovery in OSS development.

Language barriers appear to be a major challenge for requirements discovery in OSS development. This indicates that communication artifacts such as discussion forums within OSS development environments need to provide support for different types of languages for carrying out requirements discovery activities more effectively. OSS contributors are often organizationally and geographically distributed (Crowston et al., 2012). The geographically distributed nature of OSS development can result in many problems and challenges for requirements discovery, as evident from the results. About 48% indicated geographical distance and time difference between countries and delays caused from communication media such as email as a challenge.

Table 21. Problems and challenges with OSS requirements discovery

Potential problems and challenges during requirements discovery (Damian and Zowghi, 2003; Schmid, 2014)	**%**
Differences in corporate culture (members being from different corporate environments having different system usage characteristics)	63.9
Differences in educational background (e.g., high skilled and low skilled members may have different expectations about how a system works)	59
Language barriers	56.6
Different members may use different standards	49.4
Geographic distance and time difference between countries as barriers in interaction between members	48.2
Delay (e.g., Delayed email response)	48.2
Requirements being expressed using diverse terminologies and diverse level of details by members making it difficult to analyze the requirements for discovering conflicts and redundancies	39.8
Difficulty in achieving a common understanding of requirements	36.1
Difficulty in managing conflict and having open discussions of interest (difficulty in managing conflicting interests of members during development because geographical distance makes it difficult to openly discuss about interests of members)	34.9
Diminished understanding of the working context of other members (members do not have enough familiarity with/knowledge of activities of remote group members and other background information thus leading to diminished understanding of the work context of remote members; this, in turn, leads to unwanted outcomes such as members in one region not being able to have adequate understanding of the requirements of the members in other region)	34.9
Difficulty to achieve cohesion/form coalition with others	33.7
Difficulty in negotiating requirements and prioritizing requirements	32.5
Ineffective decision-making meetings (e.g., ineffective meetings because of use of poor communication technologies)	31.3

Members from different language and cultural backgrounds may demand different types of user interfaces	30.1
Difficulty in trusting others	28.9
Differences in national culture (requirements may be influenced by cultural beliefs; members from different cultural background may have different expectations about functionality, behaviour, and design)	25.3
It is difficult to identify and include relevant members in the communication process for gathering requirements	25.3
Different laws and regulations in different countries	22.9

Communication artifacts used in OSS development have many limitations and many proposed ideas of improvement have not been actually implemented (Rantalainen et al., 2011). Thus, future research can look at ways of improving existing communication artifacts such as discussion forums within OSS development environments in order to better handle challenges such as geographical distance and time difference.

Differences in organizational/corporate culture were indicated as a challenge for requirements discovery by more than 60% of the respondents. An example of two different organizational cultures is corporate cultures that are supportive and unsupportive of OSS development (Diamant and Daniel, 2010). An OSS supportive organization may provide benefits such as flexible time and pay to its employees for participation in OSS development (Diamant and Daniel, 2010), which could be beneficial for requirements discovery. A challenge that is related to organizational culture is diminished understanding of the working context of other members (indicated by 34.9%). It may be useful to gather information from members about their organizational/working context which may help in making better sense of the messages posted by them and discussions that involve them. A large number of respondents (59%) indicated differences in education background as a challenge for requirements discovery. This could mean situations where developers with technical education and non-developer users with no technical education may have difficulty interacting, which in turn could be a major problem for requirements discovery. Research efforts could be directed at investigating ways of facilitating efficient interaction between OSS developers and users. The design of OSS communication artifacts based on theories of communication (e.g., Teeni, 2001) could be a step in this direction.

A smaller percentage of respondents indicated differences in cultural backgrounds (e.g., national culture) as a problem for requirements discovery. Diamant and Daniel argue that cultural homogeneity involves being exposed to similar views and this facilitate reinforcing and confirming views that an individual has adopted and things that he/she know (Diamant and Daniel, 2010). Diamant and Daniel further argue that when an OSS developer gets to interact with a culturally similar co-developer, this could help reinforce the problem-solving approaches that he or she has been following. On similar lines, it can be expected that culturally similar OSS community members may help reinforce and may be better able to understand each other's' requirements. A related finding is that about 30% of the respondents indicated that members from different cultural backgrounds might demand different types of user interfaces which indicate that OSS members from different cultural backgrounds may have difficulty understanding each other's user interface requirements. On the other hand, McLeod and Lobel (1992) found that ethnically diverse groups generated higher quality ideas in comparison to the ethnically homogeneous group which implies that different cultural backgrounds may have some benefits for requirements discovery in OSS development. Future research can look at how to efficiently utilize diversity and similarity of OSS contributors for the benefit of requirements discovery in OSS development.

Network theories (e.g., Monge and Contractor, 2003) can inform the design of features to support networking based on cultural (corporate or national) or educational similarity or heterogeneity within OSS development environments. For example, the theory of homophily argues that individuals tend to form ties with other individuals who are similar to themselves (Contractor et al., 2006). This would imply that OSS community members may be more interested in interacting with other similar OSS community members. An example of a supportive feature could be displaying profile information of a limited number of other similar OSS project members to a new member who registers. Features based on network theory and knowledge may also be solutions for some other problems reported, such as difficulty in achieving cohesion/ forming coalition with other members (34%) and difficulty in trusting others (29%). For example, cohesion and network density are concepts in network analysis which capture the extent of connectedness in a network and can be computed using mathematical measures (Borgatti et al., 2009). It is possible to compute such network analysis measures for larger OSS projects. For example, one measure of cohesion is the number of OSS projects in which two developers have worked together or working together as this indicates repeat collaboration among the developers (Singh et al., 2011). A feature that displays cohesion score for each pair of developers could potentially assist them in deciding whether to trust each other or not.

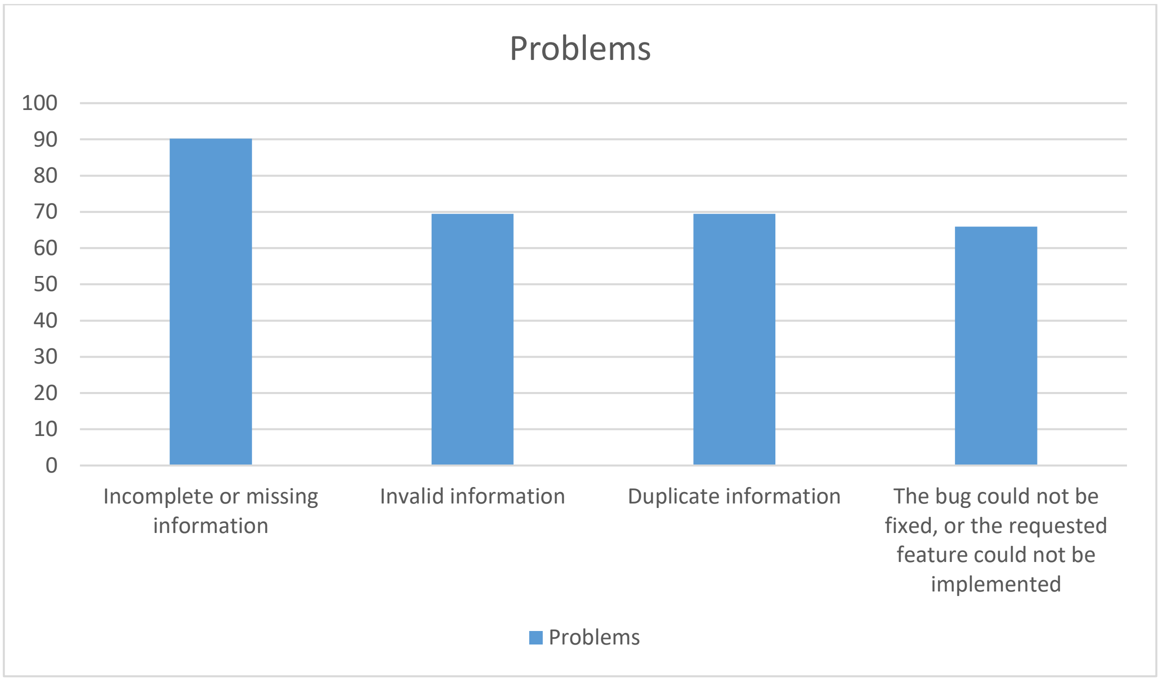
About 25% of the respondents indicated that it is difficult to identify and include relevant members in the communication process for requirements generation. This is similar to the finding of Laurent and Cleland-Huang (2009) who found that OSS development environments lacked support for bringing together, the right group of users to discuss related requirements. Laurent and Cleland-Huang mention that support should be provided within OSS development environments for placing users with similar interests in similar discussion groups, and they suggest techniques such as creating a predefined hierarchy of discussion topics for doing so.

About 23% of the respondents indicated that different laws and regulations in different countries might be a challenge in requirements generation. For example, there are many open source electronic medical record software (e.g., http://www.open-emr.org; http://openmrs.org/). Since these are OSS, contributors can be from all over the world. In the US, there is a strict health information law known as HIPAA that is applicable in the context of electronic medical records (Miller and Catherine, 2009). Since this is a US specific law, contributors from countries like India may be unaware of such laws. Thus, it may be useful to have a description of such laws (e.g., as a separate webpage within the OSS project website and links provided within communication artifacts such as discussion forums).

About 32% of respondents indicated that difficulty in negotiating and prioritizing requirements is a challenge for requirements generation in OSS development. Related findings are difficulty in managing conflict and having open discussions of interest (34.9%) and requirements being expressed using diverse terminologies and diverse level of details by members making it difficult to analyze the requirements for discovering conflicts and redundancies (39.8%). Conflicts can contribute to the difficulty in negotiating and prioritizing requirements. Scacchi (2002) has pointed out the lack of tool support within OSS development environments for RE activities such as analysis and negotiation. This limitation can be addressed by future research by investigating different possible feature supports within OSS development environments. The survey results on the usefulness of requirements analysis and negotiation practice (Table 16) provides some insights in this direction. For example, interaction matrix when implemented as a feature within OSS development environments, could potentially assist in conflict detection.

OSS literature (e.g., Bettenburg et al., 2008, Lintula et al., 2006) has reported some problems such as incompleteness with bug reports (a specific type of issue data generated in OSS issue repositories). Other types of issue data (e.g., feature requests) also gets generated in OSS issue repositories, and these issue data are a source of requirements for OSS projects (e.g., Noll, 2008). In order to investigate about problems that may exist in general with issue data irrespective of their specific type, respondents were asked to indicate the problems that may exist the issue data generated in OSS issue repositories. Figure 4 below provides results on some problems that can occur with issue data generated in issue repositories of OSS projects.

Figure 4. Problems with issue data generated in OSS issue repositories



About 90% of the respondents indicated incomplete information as a problem with bug reports and feature requests submitted by users and about 70% of the respondents indicated invalid information and duplicate information as a problem with bug reports and feature requests. These results are similar to the results by Bettenburg et al. (2008), who surveyed only in the context of bug reports. Bettenburg et al. found that incomplete information was the most commonly encountered problem, and invalid information was also a significant problem (Bettenburg et al., 2008). Future research can look at ways of mitigating these problems.

3.4 Conclusion

In general, the survey results provide evidence that OSS developers' reported usage of formal RE practices (from CSS development) do not match their perceptions about the extent to which these practices could be advantageous for OSS development. Thus, when it comes to requirements-related activities in OSS development, OSS developers do not appear to be actually practicing what they believe could be useful. Also, OSS developers indicated several problems and challenges that could exist in the context of requirements discovery in OSS development which could be potential barriers as well as motivations for usage of formal RE practices in OSS development. Most of the formal RE practices were perceived as beneficial for OSS development, in spite of significantly low reported usage. This indicates many potential directions for improving requirements discovery in OSS development, especially if some of these practices could be successfully manifested into existing OSS workflows. This would require additional research efforts focused on investigating whether and how some of the formal RE practices could be efficiently incorporated into existing OSS workflows and whether there would be any actual benefits from such incorporation. Given the complexity of the RE process, empirical investigation of all formal RE practices is not feasible within a scope of a single research project. Because of this, the subsequent phases of this research focus on requirements generation, one of the first stages of RE. Chapter 4 describes research investigating a specific requirements generation practice – requirements reuse. Requirements reuse can be an effective aid for facilitating more efficient and complete requirements generation while substantially reducing the needed effort (Hoffmann et al., 2013). This could be useful for OSS development considering problems such as incomplete requirements and limited resource availability (e.g., limited time available for volunteering developers to contribute). As the result in Table 15 indicate, requirements reuse was perceived as beneficial for OSS development but had significantly lower reported usage. I posit that this might be because of the lack of adequate support within OSS development environments. It is potentially easier to incorporate requirements reuse into existing OSS workflows compared to some other formal requirements generation practices, such as analysts eliciting requirements from users using interviews or scenarios, given the geographically distributed and ad hoc nature of OSS development and evolution and lack of resources such as money and limited time availability of contributors. The ad hoc and open nature of OSS development and evolution can make it difficult for analysts to identify right stakeholders and elicit requirements (Laurent and Cleland-Huang, 2009).

Requirements reuse can be easily incorporated into OSS development environments, in the form of the availability of a library of reusable requirements. For example, Git repositories allow storing and tracking revisions/changes to not only code but also any text files or manuscripts (Blischak et al., 2016). A library of reusable requirements could be incorporated as a centralized repository that the geographically distributed OSS community members can work with concurrently. The library of reusable requirements could even be incorporated as wiki pages. Researchers (e.g., Von Krogh et al., 2005) have reported the positive attitude of OSS contributors towards reusing development artifacts such as code, especially if they are easily available (reduced search efforts), for example, available within the development environments. A library of reusable requirements available within OSS development environments can thus appeal to

OSS developers and other contributors since there is no search overhead and facilitate reuse.

An experimental investigation is carried out to investigate the actual benefits from this practice for OSS development when incorporated into a web-based OSS development environment. This is the topic of the next chapter.

CHAPTER FOUR: AN INVESTIGATION OF THE USEFULNESS OF REQUIREMENTS REUSE FOR OSS REQUIREMENTS DISCOVERY

4.1 Introduction

A major obstacle to getting a correct and complete set of requirements during requirements gathering is the cognitive limitations of human participants; in other words, the "constraints in humans as information processors and problem solvers" (Davis, 1982, p.5). Davis describes three types of memories that humans can use for information processing: external memory, short-term memory, and long-term memory. Short-term memory has a limited capacity and can hold only a small number of items. This limitation of working memory can negatively affect the output of requirements generation, resulting in incomplete and incorrect requirements generation (Davis, 1982). Thus, the cognitive limitations of the participants in OSS development can negatively affect their requirements output. Researchers have reported incompleteness and invalidity as major problems with OSS requirements sources such as issue data (e.g., Bettenburg et al., 2008) – finding also evident from the first phase of this research (see Figure 4 in chapter three). In fact, project developers of many OSS projects on GitHub got together and submitted an open letter to the GitHub management listing the problems they faced. They requested many enhancements, and one of the top problems listed was the poor quality of issues submitted (a major source of requirements for OSS projects)[1]. The cognitive limitations of OSS contributors could contribute to such problems, especially given the distributed (geographical, temporal, and social-cultural distances between contributors) and ad hoc

[1] https://github.com/dear-github/dear-github

nature of OSS development. The effects of the limitations of short-term memory can be reduced by using external memory (e.g., something as simple as a pad of paper) (Davis, 1982). This phase of the research investigates whether the availability of an external memory – specifically, a library of reusable requirements in OSS development environments – is beneficial for requirements discovery in OSS development in the sense of improving requirements quantity and completeness.

A potential external memory is a library of reusable requirements. In a survey of requirements analysts in a traditional collocated software development setting, respondents indicated that, with requirements reuse, the need to start from scratch was eliminated, thus saving effort (Hoffmann et al., 2013). Respondents reported many potential benefits, such as more efficient and complete requirements generation, better quality requirements, better understandability of requirements and more complete requirements specification (Hoffmann et al., 2013). Chernak (2012) conducted a web-based survey of IT professionals to determine the state of practice of requirements reuse in industrial settings. He found that, while the majority of the survey respondents (more than 80%) believed that requirements reuse is beneficial, only about 59% reported actually using it. The results from the survey of OSS developers reported in Chapter 3 revealed that the extent to which OSS developers perceived requirements reuse practice as being useful for OSS development was significantly higher than the extent to which they reported using this practice (mean perceived usefulness = 3.1, mean reported usage = 2.6, t= -3.140; p=.002), thus indicating a significant gap between thought and action with respect to requirements reuse in OSS. The lack of adequate support within OSS development environments for requirements reuse is a potential contributor to this discrepancy. In OSS projects, requirements may exist informally as part of the textual data posted in communication artifacts such as discussion forums (e.g., Scacchi, 2002). Such poorly organized, ill-structured requirements information in OSS development environments can hinder reuse (c.f. Chernak, 2012).

Requirements reuse can be easily incorporated into OSS development environments in the form of a library of reusable requirements. For example, Git repositories allow storing and tracking revisions/changes to not only code but also any text files or manuscripts (Blischak et al., 2016). Hence, a library of reusable requirements could be incorporated as a repository into popular OSS development environments such as GitHub, which OSS contributors in different geographical locations can work with, making additions, deletions, or updating as needed. Providing a library of reusable requirements within OSS development environments could provide a well-organized, easily accessible collection of requirements to the participants in OSS development. Requirements can be efficiently organized within such a library through the use of meaningful headings and standard formats. For example, a security-related requirement can be stored under the heading Security, and the content can be organized according to some standard format. Researchers (e.g., Von Krogh et al., 2005) have reported the positive attitude of OSS developers towards reusing development artifacts such as, especially if they are easily available (e.g., available within the development environment) (see Table 22 in the next section). Sharif et al. (2015) report that while OSS developers are implementation-centric, they often seek external documentation to gain design knowledge that can inform implementation. Developers often face difficulty in locating such knowledge sources. The availability of a library of reusable requirements within OSS development environments, as a centralized repository or wiki pages, can facilitate reuse because of the easy availability (no search effort needed) and well-organized structure.

To the best of my knowledge, no prior research has experimentally investigated the benefits from a library of reusable requirements for requirements generation, neither in OSS development nor in CSS development (e.g., Hoffmann et al., 2013; also elaborated further in literature review section). Motivated by this and the potential benefits discussed above, the specific objective of this phase of the research is to investigate *whether access to a library of reusable requirements within OSS development environment provides any benefits for requirements generation in OSS projects*?

4.2 Literature Review: Requirements Reuse

Based on surveys and interviews with OSS developers, Von Krogh et al. (2005) reported that the types of knowledge generally reused during OSS development were algorithms, methods, and lines of code. Von Krogh et al. also note the lack of empirical work on knowledge reuse in the OSS development domain. They did not report any evidence of requirements reuse during OSS development. Also, a search using the keyword *requirements reuse* in a database specifically containing OSS literature (http://flosshub.org/biblio) returned zero results. This, combined with searches in other databases such as Google Scholar, provided no evidence of any prior empirical work on requirements reuse in OSS development domain.

The qualitative study by Von Krogh et al. provides many insights for research on requirements reuse in open source software development. These are discussed in Table 22.

Table 22. Findings of Von Krogh et al. and implications for reuse research in OSS development

Quotes from OSS developers (Von Krogh et al., 2005)	Findings by Von Krogh et al. based on analysis of quotes	Implication for requirements reuse research in OSS development
"Most of the time, I am looking for components that people have written specifically for reuse……..” (p.6).	actually practicing/willing to practice knowledge reuse (Von Krogh et al., 2005)	
"…..I downloaded the source code ……….you look at the code and use that to understand how something works" (p.5)	Often not reusing code itself but rather knowledge obtained (e.g., knowledge about algorithms used) through analysis of code (Von Krogh et al., 2005)	
"So, in this, we had a look at publications, papers and things like that. So we did not reuse a lot of code in this part but reused knowledge from academic publications" (p.4)	OSS developers were spending large amount of time analyzing scientific publications and documentation of other software projects with the goal of obtaining reusable knowledge (Von Krogh et al., 2005)	OSS contributors may be willing to spend time analyzing and attempting to reuse knowledge available in form of reusable requirements descriptions
I will post on the mailing list-the developer list. I will ask everybody: does anyone know about this ....." (p.4) "…….If you are looking for a library, it can take between 4-8 hours……this is not worth looking for one…..the overhead of searching……" (p.5)	The effort needed to search for reusable artifacts impacted knowledge reuse by OSS developers (Von Krogh et al., 2005). Discussion forums and OSS development environments such as source forge were highly preferred places to search for reusable knowledge (Von Krogh et al., 2005).	A library of reusable requirements made available within an OSS development environment such as Sourceforge and Google Code would reduce the search efforts needed to a great extent. This could potentially be a preferable way for OSS contributors to engage in requirements reuse (requirements knowledge reuse).

From the comments of the participants and interpretations of the authors as described in the table, it appears that participants in OSS development can be expected to have a favorable attitude towards requirements reuse (something that also emerged from the survey results described in chapter three). But when it comes to actual practice, they may prefer to expend minimal search effort to obtain reusable requirements artifacts, while expecting the maximum possible usability from those artifacts. This expectation is justified considering the voluntary nature of OSS development and the time constraints of OSS contributors.

Subsequent sections review work done on requirements reuse within CSS development and its potential implications for OSS development. Many of the works described in this section were obtained by making a search in the requirements bibliography provided by Alan M Davis (http://www.reqbib.com) with the keyword "reuse".

Lam et al. (1997) note that, although there was no documented empirical evidence on requirements reuse benefits at the time of writing, logically it can be argued that requirements reuse should lead to economic savings (Lam et al., 1997). Specifically, they mention that, within engineering domains such as aero-engine control systems, the possibility of even a small amount of reuse would mean large financial savings. Similar application domains exist in OSS development (e.g., http://jsbsim.sourceforge.net/), which could potentially experience benefits from reuse. Lam et al. also mention that reusing requirements may lead stakeholders to trust such requirements more in comparison to requirements generated from scratch. Requirements with proven reusability may be thus good candidates to be in a library of reusable requirements within OSS development environments as it may be easier for OSS contributors to trust their usability.

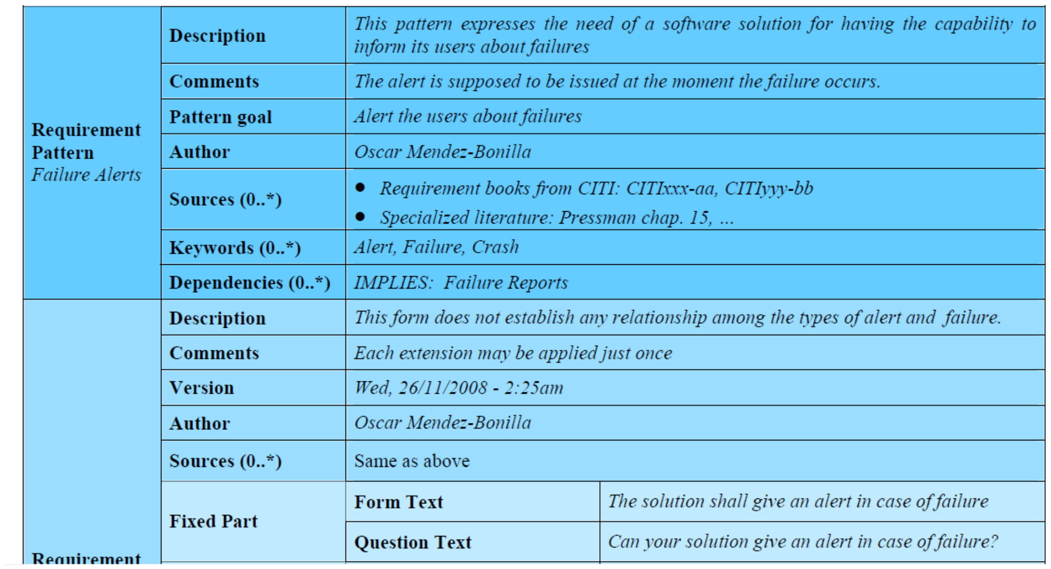
Lam et al. make several suggestions for systematic reuse of requirements. One suggestion is to identify a family of systems, which involves identifying similarities between systems and identifying similar working patterns among different systems (Lam et al., 1997). A related suggestion by Lam et al. is to identify requirements patterns through domain analysis. They stress that, while capturing requirements patterns, the focus should be given to pattern content, pattern representation, and dealing with exceptions, which may indicate the need to revise a particular pattern. Within the OSS development domain, there are several subdomains (e.g., games, internet/web infrastructure, astronomy, etc.) (Scacchi, 2002). Identifying similarities among projects within these domains may be useful for requirements reuse in OSS development. For example, similarities among different OSS game projects may provide insights about reusable requirements/requirements patterns for that domain, especially if similar characteristics are observed in a large number of projects.

Lam et al. (1997) warn against the narrow focus on specific reuse technologies only, and advise to broaden the focus to the potential impact that a reuse technology can have. This involves analyzing the current requirements engineering practices, how requirements reuse would be implemented within the current circumstances, and how the implementation of requirements reuse will change the current circumstances such as processes used, organizational structure and finances. For OSS development, this would mean analyzing the current requirements-related practices (which are largely ad hoc, as described in Chapters 2 and 3), whether and how specific requirements reuse technologies could be incorporated into existing OSS development workflows, and what impacts they could have if incorporated.

Different requirements reuse techniques have been proposed, but remain untested (Lam et al., 1997). Maiden et al. (1991) proposed the analogy technique for requirements reuse. An analogy captures similarities between the problem domain of a reusable requirements specification and the target problem domain. One example is an analogy that captures similarities between a video rental system and an industrial production planning system (for example, a video allocation list created by a video rental system is analogous to job allocation list created by a production planning system) (Maiden et al., 1991). Li et al. (2007) proposed a multiple ontology approach for requirements reuse. Li et al.'s framework consists of multiple ontologies: top-level ontology (describing basic concepts such as goals, tasks and events), domain ontology (describes concepts and relationships between concepts for a given domain), task ontology (describes concepts and relationships between concepts for a given task), and application ontology (describes concepts and relationships between concepts for a given application).

Requirements patterns are another type of requirements reuse technique (Palomares et al., 2014). Requirements patterns help avoid the need to build requirements from scratch (Franch et al., 2011). Patterns contain knowledge (gained from real world projects) about recurring situations (Lopez et al., 2013). An example of a requirement pattern is shown in Figure 5:

Figure 5. Example of a requirement pattern (Renault et al., 2009, p.10)



Requirement Pattern *Failure Alerts*	**Description**	*This pattern expresses the need of a software solution for having the capability to inform its users about failures*	
	Comments	*The alert is supposed to be issued at the moment the failure occurs.*	
	Pattern goal	*Alert the users about failures*	
	Author	*Oscar Mendez-Bonilla*	
	Sources (0..*)	● *Requirement books from CITI: CITIxxx-aa, CITIyyy-bb* ● *Specialized literature: Pressman chap. 15, …*	
	Keywords (0..*)	*Alert, Failure, Crash*	
	Dependencies (0..*)	*IMPLIES: Failure Reports*	
Requirement	**Description**	*This form does not establish any relationship among the types of alert and failure.*	
	Comments	*Each extension may be applied just once*	
	Version	*Wed, 26/11/2008 - 2:25am*	
	Author	*Oscar Mendez-Bonilla*	
	Sources (0..*)	Same as above	
	Fixed Part	**Form Text**	*The solution shall give an alert in case of failure*
		Question Text	*Can your solution give an alert in case of failure?*

A requirements pattern usually consists of many sections, such as shown in Figure 5. For example, one section could be metadata about the pattern such as goal, author name, and description; another could be the different forms or types a requirements pattern can have (Renault et al., 2009). After determining that a pattern is applicable to a particular project, it can be used for requirements extraction (Renault et al., 2009). For example, after it is determined that *failure alerts provided* form of *failure alerts* requirement pattern is applicable for a project, the requirement can be stated by using the form text section as "An alert in case of failure has to be provided, and the alert provided as solution shall be an email" (Renault et al., 2009).

A well-known work in the area of requirements patterns is by Withall (2007), who identified over thirty requirements patterns in real world organizational software projects. The structure of each requirement pattern proposed by Withall consists of basic details, applicability, different possible types, content, template for writing, possibly related extra requirements, and suggestions for implementation and testing. An example of a requirement pattern identified by Withall is the *data type* requirement pattern for which applicability is where there is a need to display some piece of information. The information piece can be in any form such as characters, numbers, and lists. The content of the data type requirement includes data type name, form, display format and constraints (Withall, 2007). Another example of a requirement pattern identified by Withall is inter-system interface requirement pattern, which can be used to specify requirements for an interface between proposed software and an external system or component with which it has to interact.

The requirements patterns described by Withall appear to incorporate some of the suggestions by Lam et al. (1997). For example, Lam et al. suggested that, while capturing requirements patterns, the focus should be given to pattern content, pattern representation and dealing with exceptions that may indicate the need to revise a particular pattern. Another suggestion is to provide contextual information along with reusable requirements (e.g., assumptions associated with a reusable requirements artifact) to prevent misuse of requirements reuse (e.g., reusing a requirement in a non-applicable context) (Lam et al., 1997). The requirements patterns described by Withall appear to follow these guidelines, as each of them has a content description, a well-defined structure and applicability description describing the context within which the pattern is applicable.

It appears that very little empirical work exists investigating the impacts of requirements reuse, even in traditional software development domain. Chernak (2012) conducted a web-based survey among IT professionals to determine the state of the practice of requirements reuse. In that survey, the majority of respondents (more than 80%) believed that requirements reuse is beneficial and important. Two of the top reported benefits were faster time-to-market and lower development cost (Chernak, 2012). Chernak reports that, even though a large majority of survey respondents believed requirements reuse to be beneficial, only about 59% actually practiced it. Some potential obstacles to requirements reuse were poor quality (e.g., incompleteness) of existing requirements, poor structure of existing requirements (making it difficult to identify reusable requirements), and existing requirements not being updated (Chernak, 2012). These problems appear to be similar to problems with the informal sources of requirements in OSS development. As described in Chapter 3, almost 90% of responding OSS developers indicated incomplete or missing information as a problem and almost 70% indicated invalid information as a problem with the issue data submitted in issue repositories. These obstacles could potentially pose challenges in reusing issue data such as bug reports or feature requests. Requirements may also emerge informally as part of some message posted on discussion forums (e.g., Scacchi, 2002), thus being poorly structured. Also, as evident from survey results in Tables 4 and 11 (Chapter 3), the usage of standard structure for requirements specification in OSS development was very low. The poorly structured requirements information that may exist in OSS development environments may not be easily reusable.

Hoffmann et al. (2013) conducted semi-structured interviews with five experienced requirements analysts. Based on the opinions of interviewed analysts, they report some potential benefits of using requirements patterns. Breaux et al. (2012) also hypothesize about some similar benefits from requirements patterns usage. One is the possibility of improvements in requirements elicitation. Hoffmann et al. mention that requirements patterns could potentially reduce the effort needed for elicitation since analysts don't need to start from scratch. Requirements patterns could potentially facilitate more complete elicitation and specification (e.g., facilitating thinking about non-functional requirements in participants who may be focused only on functional requirements, thus reducing the non-occurrence of non-functional requirements) (Hoffmann et al., 2013). Both Hoffman et al. and Breaux et al. mention that requirements patterns could potentially help improve requirements quality. For example, requirements patterns descriptions often include important attributes of associated requirements that can help structure requirements and mitigate the omission of important requirements related information (Hoffmann et al., 2013). Because of their well-defined content and structure, requirements patterns could potentially reduce ambiguities and inconsistencies in the requirements generated using them (Breaux et al., 2012). The structure and pre-defined templates in requirements patterns could potentially facilitate a better understanding of associated requirements (Hoffmann et al., 2013).

While researchers have hypothesized about potential benefits from the usage of requirements patterns, there is a lack of empirical evidence of the benefits from the usage of requirements patterns. Hoffmann et al. call for such research. This chapter addresses this gap by investigating the potential benefits from requirements reuse for requirements generation in OSS development. A library of reusable requirements (specifically requirements patterns) within OSS development environments could be an efficient way to facilitate requirements reuse practice in the OSS domain. Because of their applicability to a large number of application domains, requirements patterns could be ideal reusable requirements artifacts. The availability of a library of requirements patterns within OSS development environments fits well with the general desires of the OSS community concerning reuse (see Table 22 above) to have reusable artifacts with maximum usability while expending minimum search efforts. Note also that the availability of a library of requirements patterns could encompass some other proposed requirements reuse techniques such as the analogy technique. Analogy technique involves analyzing similarities between an available requirements specification and target problem domain (Maiden et al., 1991). This cognitive process occurs as OSS contributors go through the library of requirements patterns and attempt to reuse them in the context of some OSS project. Thus, the specific goal of this phase of the research is to experimentally investigate the actual benefits from the availability of a library of requirements patterns within OSS development environments for requirements generation in OSS projects.

The next section describes theoretical background and hypothesis development for the study.

4.3 Theoretical Background and Hypothesis Development

Individuals rely on information available in memory or from some external source while making judgments (Dixon, 1997). When an external database of reusable artifacts is made available to individuals, this can be seen as an attempt to expand the available knowledge to individuals without creating cognitive burden and, thus, will be an addition to the reusable knowledge existing in the memory of individuals (Dixon, 1997). Benbasat and Lim (2000) present a model of information availability that can provide insights on the theoretical view of the access of OSS contributors to reusable requirements library as an expansion of the *available* reusable knowledge they can utilize during OSS requirements discovery. Benbasat and Lim define two aspects of information availability, namely information search scope (ISS) and solution design scope (SDS).

Information search scope is the set of all relevant information available. For example, when an individual is asked to access the risk of death from heart attack among a population in comparison to other illnesses, the information search scope can be the set of all recalled instances about diseases that may lead to death (Benbasat and Lim, 2000). Solution design scope is the set of information items from the information search scope that can generate solutions to the problem at hand. Careful evaluation of the information items in the information search scope based on demands of the problem becomes the basis for generating solution design scope; for example, solution design scope in the example scenario could be generated by accessing the degree to which each identified disease may or may not cause death in comparison to others (Benbasat and Lim, 2000). Another example that illustrates ISS and SDS is that of a manager working on the problem of how to make his organization more successful. The decision maker (manager) first searches for a solution within a limited amount of information that becomes available in mind (such as ideas of new innovations, ideas for increasing customer demands) and this limited available information for decision-making is the ISS (Benbasat and Lim, 2000).

The size of ISS will significantly influence the size of the SDS. Smaller ISS leads to smaller SDS which in turn leads to biased/less satisfactory solutions. More specifically, smaller ISS implies that many potential solutions are not part of the initially generated ISS. In turn, this means they are not part of the subsequently generated SDS as well, since SDS is formed based on the initially generated ISS (Benbasat and Lim, 2000). Thus, to avoid negative effects/biases arising from limited available information for problem solving, the focus should be on the enlargement of ISS. Expanding the information search scope (ISS) in the example scenario would mean recalling more examples of diseases that may or may not be fatal (Benbasat and Lim, 2000). Providing external support tools can facilitate enlargement of ISS and SDS (Benbasat and Lim, 2000). Benbasat and Lim argued that the availability of support tools such as electronic brain storming (EBS) and electronic mail (EM) should facilitate enlargement of ISS and SDS, and found supporting empirical evidence in the form of generation of greater number of ideas and solutions in the availability condition compared to the non-availability condition.

The notion of expanding information search scope (ISS); that is, expanding the limited available information for decision making by providing support tools (Benbasat and Lim, 2000), is in line with Dixon's (1997) assertion that, in the context of reuse, "provision of databases of reusable design solutions can be viewed as an attempt to expand the knowledge that is available to designers without putting burden on cognitive abilities of the designers" (p. 23, Dixon, 1997). When participants in open source software development have access to a library of reusable requirements, this can be viewed as an expansion of the reusable knowledge that is *available* to them during the discovery of requirements for some proposed open source software. Having access to a library of reusable requirements should increase the information search scope (ISS) during requirements discovery for some proposed OSS development. The availability of a database of reusable content means that the designers can reuse content that is available both in the database as well as in their minds and also combine both (Dixon, 1997).

The cognitive phenomenon of anchoring and adjustment can also provide insights on usage of a library of reusable requirements that is available in an OSS development environment. In fact, Davis writes: "*Information requirements from users will tend to be a result of an adjustment from an anchor of the information currently available*" (p.10, Davis, 1982). When artifacts containing reusable knowledge become available, it facilitates two things: a) expanding the currently available information b) enabling individuals to anchor to this available information and make adjustments to it (e.g., modifying the available information) to make it usable in a new context (Dixon, 1997). Similarly, when a library of reusable requirements becomes available within OSS development environments, it expands the information available to OSS contributors to which they can anchor, make adjustments and use during requirements generation for OSS projects.

The theoretical arguments of Benbasat and Lim are in line with Newell and Simon's human information processing theory (See Benbasat and Lim, 2000; Nagasundaram and Dennis, 1993). This theory argues that human information processing system consists of short-term memory (STM) that has a limited capacity and contains a small amount of information, and it is this small amount of information *available* in the working memory that is available for immediate processing (Nagasundaram and Dennis, 1993). The limitations of short-term memory put constraints on human cognitive abilities. For example, it can affect individual's ability to define requirements; during a requirements elicitation interview, a user without the aid of any external storage can hold only a small amount of requirements related information in the short-term memory and consequently, provide very limited requirements information (Davis, 1982). Providing an external memory (e.g., electronic, or even a pad of paper) can increase the virtual capacity of the short-term memory and help reduce its limitations (Davis, 1982; Nagasundaram and Dennis, 1993). For example, providing external memory can increase productivity during idea generation (Nagasundaram and Dennis, 1993).

From the above discussion, it can be argued that when a library of reusable requirements becomes available to participants in OSS development, it can be thought of as an external memory which should increase their information search scope (in line with the discussion by Benbasat and Lim) and help overcome the limited capacity of human working memory. This should facilitate generation of a greater quantity of requirements. Researchers have found empirical evidence that individuals working alone without any external support can generate only a tiny proportion of information from the potential solution space (Connolly et al., 1993; Benbasat and Lim, 2000; Dennis et al., 1996), but they tend to hold the faulty belief that they have generated a highly complete set of solutions (Dennis et al., 1996). This implies that participants in OSS development who are working on their own without any form of external cognitive assistance within an online open source software development environment, such as Sourceforge, would be able to generate only a small amount of requirements that forms only a small proportion of the larger set of all possible requirements and they may even hold faulty beliefs about the completeness of generated requirements. But when a library of reusable requirements becomes available to them, it can be expected to increase their information search scope, enabling them to generate a greater number of meaningful requirements.

Based on the above discussion, I hypothesize:

H1: Participants in OSS development who have a reusable requirements library available within the OSS development environment in which they are working will generate a significantly greater quantity of requirements than participants who do not have a reusable requirements library.

Measuring the quality of requirements through accuracy is not feasible until the much later stages of software development (Pitts and Browne, 2007). Therefore, an alternative is to use a surrogate measure of quality – the completeness of gathered requirements – measured through depth and breadth of the gathered requirements (Pitts and Browne, 2007). Pitts and Browne define measures for breadth and depth of gathered requirements based on the coding of the gathered requirements according to a context-independent generic requirements categories' taxonomy developed by Browne and Rogich (2001). There are four main categories in the Browne and Rogich taxonomy: goal, process, task, and information; each category may, in turn, have many subcategories. Goal level requirements are organizational/strategic issues; sub-categories of requirements in this category are the actual goals, factors that may prevent goal achievement, causes of the current problem, and types of stakeholders. Process requirements are the processes required to achieve goals; subcategories of requirements in this category are the steps in some production process and factors that may impede the process. Tasks requirements are the tasks associated with a process; subcategories of requirements in this category are actions, assumptions, and rules of tasks, and roles and responsibilities. Information requirements are data-related requirements; subcategories are data to be displayed and data to be entered. Breadth is measured as the number of distinct requirements categories in the coded data, and depth is measured as the number of requirements within in each category in the coded data (Browne and Rogich, 2001; Pitts and Browne, 2007).

Research on idea generation holds the view that "quantity breeds quality." According to this view, when people express a large quantity of ideas, the chance of them being of good quality also increases (Rietzschel et al., 2007). Rietzschel et al. note that the underlying reasoning behind this is the view that each generated idea has an equal probability of having good quality and by the law of chance, as more number of ideas are generated, the number of good quality ideas generated should increase linearly. There is empirical evidence that the quantity of ideas generated is a reliable predictor of overall quality of the generated ideas (Dennis et al., 1996; Rietzschel et al., 2007). For example, Diehl and Stroebe (1987) found that increase in the quantity of generated ideas was associated with an increase in good quality ideas although it was also found to be associated with an increase in poor quality ideas as well. Similarly, Dennis et al. (1996) found that the experimental group that generated the greater quantity of ideas also had a higher total quality of ideas.

Hypothesis 1 argues that participants in OSS development who have a library of reusable requirements patterns available to them will generate a significantly greater quantity of requirements. As described previously, the quantity of ideas generated is a reliable predictor of total quality of ideas generated (Dennis et al., 1996; Rietzschel et al., 2007). Thus, it can be expected that if the availability of a library of reusable requirements can facilitate generation of a greater quantity of requirements, it can also facilitate generation of a greater quality of requirements (for example, greater quantity could be associated with greater number of requirements belonging to distinct categories, i.e., greater breadth). Moreover, Gettys et al. (1987) found that unaided decision makers generated incomplete solutions for ill-defined problems and suggested that aiding decision makers with some interventions should lead to the generation of more complete solutions (Gettys et al., 1987, p.43-44). Similarly, it can be argued that the completeness of requirements generated by unaided participants would be low, whereas when participants who are aided by external cognitive assistance such as availability of a library of reusable requirements, it should enable them to generate a more complete set of requirements. Hence, I hypothesize:

H2a: Participants in OSS development who have a reusable requirements library available within the OSS development environment in which they are working will generate a significantly greater breadth of requirements than participants who do not have any reusable requirements library.

H2b: Participants in OSS development who have a reusable requirements library available within the OSS development environment in which they are working will

generate a significantly greater depth of requirements than participants who do not have any reusable requirements library.

The size of a message can influence the understanding of the message (Teeni, 2001), thus making the size of the requirements description an important variable, especially in the context of web-based requirements generation. I expect that the more detailed a message in a web-based medium is, better the chances of its understandability by potential readers such as OSS developers and project members, which can lead to desirable outcomes such as being processed or implemented. Laurent and Cleland Huang report instances where OSS developers specifically requested well-described requirements information, as indicated from the following advice listed on an OSS project website: "*Features are more likely to get implemented if the description of the feature is clear. For a complicated feature, a link to a specification on the wiki is a great way to help flesh out the idea*" (p.250, Laurent and Cleland-Huang, 2009). As previously described, external memory should increase the information search scope and subsequently solution domain scope for each individual (Benbasat and Lim, 2000). Thus, the availability of a library of reusable requirements within OSS development environments should increase the information search scope for requirements discovery for OSS projects. In the web environment, a textual message is a normal way of constructing and conveying requirements information. The increased information search scope and solution domain scope should facilitate the construction of more detailed textual requirements messages since that is one practical way in which individuals could deliver the increased/enhanced requirements information generated in their minds.

Hence, I hypothesize:

H3: Participants in OSS development who have a reusable requirements library available within the OSS development environment in which they are working will construct requirements messages of significantly greater size than participants who do not have any reusable requirements library available.

The next section describes the research methodology for the hypotheses.

4.4 Research Methodology

4.4.1 Method

A web experiment methodology was used to investigate the above hypotheses. A web experiment is an extension of computer-based laboratory experiment (Reips, 2002). A Web experiment is a suitable methodology, considering the web-based, geographically distributed nature of OSS development. In the current study, participants were recruited from Crowdflower. They were paid a small incentive ($1 each) for their participation. Crowdflower provides a geographically distributed workforce, a characteristic of OSS development. Participants included individuals from both technical and non-technical background. Contributors to real world OSS projects are comprised of individuals from both technical and non-technical background (e.g., Crowston and Howison, 2005). The experimental group of participants thus closely match the mixture of technical and non-technical individuals that is characteristic of real world OSS development communities.

Both types of individuals are important for OSS projects. Individuals from technical background can contribute code whereas individuals from non-technical background can contribute issues and participate in discussions; in addition to such active contributors, even the passive users who do not contribute but have a general interest in the OSS projects are also important since their existence can attract potential active contributors for whom the interest of these users is a reward (Ye and Kishida, 2003).

The experimental webpages (treatment and control) were constructed in an OSS development environment - Google Code.

4.4.2 Experimental Design

4.4.2.1 Experimental Task

The experimental task was to generate requirements for a to-be-developed open source electronic medical record that is to be integrated with IBM Watson technology. A real world web-based open source software development environment, Google Code, was used to conduct the experiment. The detailed experimental task description is provided in Appendix 3.

One desirable characteristic of an experimental task used in an experimental design for testing effectiveness of a requirements discovery support tool is novelty. For a novel case, subjects will need to be imaginative while providing requirements, in addition to relying on prior experiences and beliefs (Browne and Rogich, 2001). The application of Watson to different domains such as health care and finance is new. Thus, it is expected that the experimental task has a high level of novelty, requiring subjects to use a high level of imagination. The demographics of participants are presented in Appendix 8. It can be seen from Appendix 8 that they were largely new to open source software development and hence a potential population of newcomers (Steinmacher et al., 2014). Newcomers are important for OSS development since they form the pool of potential future contributors that are vital to the long-term survival and growth of OSS projects (Jensen et al., 2011).

Implementation of electronic medical records in developing countries is a real research issue (Fraser et al., 2005). Fraser et al. mention that developing countries often face health crisis such as HIV/AIDS and tuberculosis, that threaten millions of lives, but for which lack of infrastructure often poses barriers to handling such crises. Efficient information management (e.g., by using information systems such as electronic medical record systems) can help eradicate many problems in health in developing countries (Fraser et al., 2005). Open source electronic medical record development is a feasible and efficient solution to managing many healthcare problems in developing countries since there is a large-scale need for clinical data management, need for rapid response, but with the challenge of adjusting with limited technical resources in developing countries (Mamlin et al., 2006). Mamlin et al. note that collaborative open source electronic medical record development can lay the foundation upon which other health information systems and specific applications can be later built. Hence, the context of the experimental task (development of an open source electronic medical record software to be integrated with Watson technology) of this phase of the current research is realistic and of practical significance. The requirements data that was generated as part of the experiment can be expected to provide useful knowledge about the design of such systems, and this should be an additional contribution of this study even though it is not the main focus.

4.4.2.2 Experimental Groups

There were two experimental groups. The treatment group had access to a library of reusable requirements (specifically the list of requirements patterns from Withall (2007). Withall describes over thirty requirement patterns. These patterns are generic and recur again and again in the context of systems development in a large number of domains (Withall, 2007). Table 23 below lists many of the requirements patterns described by Withall and connects them with examples of electronic medical record requirements listed by Manitoba-eHealth (retrieved from http://www.manitoba-ehealth.ca/commphysicians/files/emrreq.pdf). The control group did not have access to any reusable requirements library and were simply asked to provide requirements for the proposed open source electronic medical records software to be integrated with IBM Watson technology. All experimental materials available to both groups (the case description, Google Code project development environment) were the same except for the library of reusable requirements available to the treatment group.

Table 23. Requirements patterns of Withall and related examples from EMR domain

Generic Requirements patterns (Withall, 2007)	**Specific examples of requirements from electronic medical record (EMR) domain (http://www.manitoba-ehealth.ca/commphysicians/files/emrreq.pdf)**
Inter-system interface requirement pattern: can be used to describe some requirement about interface between proposed software and some external system or component with which the proposed software needs to interact	EMR system needs to provide access to external information resources such as health care literature.
Technology requirement pattern: can be used to describe some requirement about technology that should or should not be used in the development or running of proposed software or with which the proposed software needs to be compatible	EMR system does not need local client application functionality but needs web browser plugins.
Comply with standard requirement pattern: can be used to describe some requirement about standards that the proposed software needs to comply with	EMR system must comply with industry acceptable standards in areas such as clinical terminology and messaging (e.g., HL7 V3)
Documentation requirement pattern: can be used to describe some requirement about documentation that the proposed software needs to produce	EMR system needs to provide documentation of prescribed medications and immunizations given.
Living entity requirement pattern: can be used to describe some requirement about some entity for which the associated information has a lifespan, i.e., it is created, modified multiple times and after some time terminated.	A patient record needs to be deactivated/archived under conditions such as death.
Inquiry requirement pattern: can be used to describe some requirement about displaying specified information to the user.	EMR system needs to display context-specific help/instructions to users.
Extendability requirement pattern: can be used to describe some requirement about extending some aspect of the proposed software	EMR system needs to be extensible, i.e., it should be possible to modify and add features with minimal economic burden and minimal impact on existing functionalities.
Multiness requirement pattern: can be used to describe some requirement about accommodating multiple something simultaneously.	The same patient data should be accessible from multiple locations.
Authentication requirement pattern: can be used to describe some requirement about checking identities before allowing access	All communications with EMR system must be authenticated and secure.

4.4.2.3 Procedure

The treatment and control conditions of the web experiment were two separate Google Code web pages that were same in all aspects except for the availability of a library of reusable requirements. Both web pages contained the same experimental task description shown in Appendix 3. In the treatment condition, the library of reusable requirements was available on the Google Code web page as wiki page links (on the side of the task description) that participants could click to read the detailed description of individual requirement patterns. The treatment and control condition web-pages were made available as tasks on Crowdflower platform to which participants (individuals from the Crowdflower workforce) were randomly assigned by the Crowdflower application. Participants in each group were asked to read the description (on the Google Code web page) of the proposed open source electronic medical record software that was to be integrated with IBM Watson technology and to think about potential requirements for the proposed open source software. In addition, participants in the treatment group were also instructed that, if they wished, they could use the requirements from the reusable requirements library as a starting point for thinking about the requirements for the proposed open source electronic medical record software.

Participants were asked to submit the requirements they identified in textual format using the text box on the Crowdflower platform. The task was designed such that after providing the requirements information in the textbox, they were required to answer the questionnaire items on technical background, prior usage of electronic medical record systems/personal health record systems/other health information systems, and prior experience with OSS development.

4.4.2.4 Variables and Measures

In line with the hypotheses, the dependent variables of interest were requirements size, requirements quantity, and requirements completeness (depth and breadth). Requirements size was measured as the number of words used to construct the requirements message. Requirements quantity was the total number of requirements provided by a participant (e.g., Browne and Rogich, 2001). In line with Pitts and Browne (2007), requirements completeness was comprised of the measures of depth and breadth of requirements. Breadth was measured as the number of distinct requirements categories (from the Browne and Rogich requirements taxonomy shown below) provided by subjects. Depth was measured as the number of requirements within each category (from the taxonomy) provided by subjects. The taxonomy has been used in many studies to code requirements data (e.g., Browne and Rogich (2001), Pitts and Browne (2007)). For the purpose of analysis involving requirements quantity and completeness, a coding procedure (similar to Browne and Rogich (2001)) for the textual requirements data was used (described later).

The taxonomy (Pitts and Browne, 2007**)** is shown in Table 24.

Table 24. Requirements taxonomy (Pitts and Browne, 2007)

GENERIC REQUIREMENTS CATEGORIES (Pitts and Browne, 2007, p.101)	**DESCRIPTION (Pitts and Browne, 2007, p.101)**
GOAL	
Goal state specification	Identifying the particular goal state to be achieved
Goal specification	Comparing existing and desired states
Difficulties and constraints	Identifying factors inhibiting goal achievement
Ultimate values and preferences	Stating the final ends served by a solution
Difficulties and constraints	Identifying factors inhibiting goal achievement
Ultimate values and preferences	Stating the final ends served by a solution
Means and strategies	Specifying how a solution might be achieved
Causal diagnosis	Identifying the causes of the problematic state
Knowledge specification	Stating facts and beliefs pertinent to the problem
Perspective	Adopting appropriate point of view on the situation

Existing support environment	Existing technological environment to support new system
Stakeholders	Organization units, customers, suppliers, competitors
PROCESS	
Process description	Steps or tasks designed to produce a product
Process knowledge specification	Facts, rules, beliefs, decisions required to perform process
Difficulties and constraints	Facts that may prohibit process completion
Roles and responsibilities	Individuals/departments charged with performing process
TASK	
Task description	Sequence of actions required to complete a task
Task knowledge specification	Facts, rules, beliefs, decisions required to perform task
Performance criteria	Statement that link outcome with condition/constraints
Roles and responsibilities	Individuals/departments charged with performing task
Justification	Explanation of specific action to be/not to be taken
INFORMATION	
Displayed information	Data to be presented to users; paper/electronic format
Interface design	Language and format used for displayed information
Inputs	Data that must be entered in to system
Stored information	Data saved by the system
Objects and events	Physical entities and occurrences relevant to the system
Relationship among object/event	How one object/event is associated with another object/event
Data attributes	Characteristics of objects and events
Validation criteria	Rules that govern the validity of data
Computations	Information created by the system

Some control variables that were expected to be present in the context of requirements generation in OSS development were also measured. The experience and knowledge of software project team members and users can influence the outcomes of the requirements capture stage of a software project (Chatzoglou and Macaulay, 1997). Technical knowledge (e.g., knowledge about commonly occurring functional requirements in the context of information systems development, general knowledge about systems development such as commonly occurring implementation issues) can be important knowledge sources during requirements determination for systems development (Vitalari, 1985). Thus, one's technical background (technical career and technical education) can potentially influence requirements generation in OSS development and were included as covariates in the analysis. Questionnaire items for measuring for the extent to which the participants' career and education were technical were obtained from Enns et al. (2003).

Web-based requirements generation is different from the traditional face-to-face setting of requirements elicitation, as web-based requirements generation is mediated by the internet. Hence experience with the relevant web-based applications could be a potential confounding variable. One such variable is the crowdsourcing experience of participants, which is the number of crowdsourcing tasks (hosted on some crowdsourcing platform) that participants had completed in the past. For the crowdflower platform, this would be the total number of crowdflower tasks (tasks hosted on that platform) that a participant has completed. This measure was obtained for each participant from their Crowdflower profile. Crowdsourcing platforms, such as Crowdflower, are web-based platforms/applications that support crowdsourcing tasks in a variety of domains (Doan et al., 2011). Example categories of such crowdsourcing tasks are reviewing (e.g., books, movies), tagging (e.g., webpages) and constructing/sharing textual knowledge (e.g., answering questions) (Doan et al., 2011). Participants who have successfully completed many such crowdsourcing tasks could potentially gain knowledge of many different application domains. They may become adept at common web-based activities such as constructing online textual messages. Such knowledge could potentially play an important role during requirements generation in OSS development. Hence, crowdsourcing experience of participants was included as a covariate.

The prior experience of participants with medical software (e.g., personal health record software) could also be a potential source of knowledge for them during requirements generation; hence, it was also included as a covariate. Participants were asked to indicate their usage experience with electronic medical record software systems, personal health record software systems, and other health related software systems on a Likert scale and the scores were aggregated and included in the analysis as the control variable, *experience with medical software*.

4.5 Data Analysis and Results

4.5.1 Results: Requirements Size

Initial analysis focused on investigating the effect of experimental treatment (availability of a library of reusable requirements) on the requirements size. Requirements size was measured as the number of words used to construct the requirements message. This particular dependent variable is a continuous variable. General linear model (GLM) method in SPSS was used for the data analysis. GLM is suitable for situations where the dependent variable of interest is a continuous variable, the focal predictor is categorical, and confounding variables are expected to be present (e.g., Hawkins, 2014). The confounding variables are controlled for by including them as covariates in the GLM. The focal predictor for requirements size is the availability of reusable requirements library, a categorical variable that takes values 1 (reusable requirements library available) and 0 (reusable requirements library not available). The covariates are experience of contributing to crowdsourcing tasks, technical background – career, technical background - education and experience with medical software. The covariates and their measures were discussed in the preceding section.

The interaction between the predictor variable of interest, availability of a reusable requirements library, and each of the covariates was also included in the model in order to investigate whether the effect of the availability of reusable requirements library varied significantly with the variations in the covariates.

Two assumptions of the general linear model approach are that errors are normally distributed and have common variance across all values of predictor variables (homogeneity of variance) (Rutherford, 2012; Hawkins, 2014). The checks for these assumptions are presented in Appendix 5. The histogram of the standardized residuals with normal curve imposed on it show approximate normality. West et al. (1995) mention that substantial departure from normality happens when skewness > 2 and kurtosis > 7 whereas Kline mention that skew with an absolute value greater than 3 and kurtosis with an absolute value greater than 10 are problematic (c.f. Stull, 2008). The skewness and kurtosis, as shown in Appendix 5, were below all of these threshold values. For homogeneity of variance assumption, Levene's test should be non-significant. As can be seen in Appendix 5, Levene's test is indeed non-significant.

The equation for the general linear model is shown below:

Number of words used to construct the requirements = b1*X1 + b2*X2 + b3*X3 + b4*X4 + b5*X5 + b6* X1* X2+ b7*X1*X3+ b8*X1*X4+ b9*X1*X5 + c

where,

X1 = Availability of reusable requirements library

X2 = Experience of contributing to crowdsourcing tasks

X3 = Technical background - career

X4 = Technical background - education

X5 = Experience with medical software

The results are shown in Table 25 below.

Table 25. Model parameters (Dependent variable: requirements size)

Source	F	Significance
Corrected Model	2.572	.012
Intercept	11.878	.001
Availability of reusable requirements library	8.341	.005
Experience of contributing to crowdsourcing tasks	2.288	.135
Technical background - career	.465	.497
Technical background - education	.308	.581
Experience with medical software	.264	.609
Availability of reusable requirements library* Experience of contributing to crowdsourcing tasks	5.509	.022
Availability of reusable requirements library* Technical background - career	7.470	.008
Availability of reusable requirements library* Technical background - education	3.421	.068
Availability of reusable requirements library* Experience with medical software	.764	.385

R square = .238 indicating that the model accounted for about 24 % of the variance in the dependent variable, requirements size. According to Cohen’s criteria, accounting for 2% of the variance is considered small, accounting for 13% of the variance is considered medium and accounting for 26% of the variance is considered large (c.f.

Fritz and MacKinnon, 2007). By this criteria, the variance accounted for by the model here is close to being large.

The main effect for the availability of a reusable requirements library is significant, as well as the interactions between the availability of reusable requirements library with crowdsourcing experience and Technical background - career. This provides evidence for the significant effect of the availability of reusable requirements library on requirements size after controlling for covariates. The effect of the availability of a library of reusable requirements varies with varying levels of technical and crowdsourcing experience of participants. It appears that during the construction of textual requirements messages for web-based requirements generation, the availability of a library of reusable requirements may be more useful to individuals with lower levels of technical and crowdsourcing experience. The parameter estimates for the significant interactions are provided below:

Table 26. Interactions

Parameter	B	Std. err.	Sig.
[Availability of reusable requirements library=0]* Experience of contributing to crowdsourcing tasks	.044	.019	.022
[Availability of reusable requirements library=1]* Experience of contributing to crowdsourcing tasks	0		
[Availability of reusable requirements library=0]* Technical background - career	30.836	11.282	.008
[Availability of reusable requirements library=1]* Technical background - career	0		

Crowdsourcing experience has a stronger effect in the non-availability condition in comparison to the availability condition. Similarly, technical background – career has a stronger effect in the non-availability condition in comparison to the availability condition. Consider the situation where a library of reusable requirements is not available. Individuals who are new to crowdsourcing or individuals from a non-technical background would have nothing to rely on while participating in requirements generation, whereas individuals with higher levels of technical and crowdsourcing experience can rely on the knowledge gained from those experiences. A library of reusable requirements can be an aid to participants with low levels of technical and crowdsourcing experience while constructing/submitting requirements messages in OSS development.

Posting messages on OSS discussion forums and submitting issue information such as bug reports and feature requests in OSS issue repositories are also crowdsourcing tasks. Non-developer users often participate in such activities that happen as part of the evolution of OSS projects (e.g., Crowston and Howison, 2005; Crowston et al., 2012). Non-developer participants in OSS development who are just starting off (e.g., interested users who have never submitted an issue) could potentially benefit from reusable requirements artifacts, for example in constructing more detailed requirements messages. Steinmacher et al. (2014) report that newcomers to OSS development often face problems such as finding a way to start. Newcomers' lack of technical knowledge (e.g., lack of knowledge of tools and technologies used in a project) and the large amount of learning effort that may be needed in becoming familiar with OSS project aspects may pose a significant barrier to their participation (Steinmacher et al., 2014). Reusable requirements artifacts could potentially assist such newcomer users in their participation in OSS requirements generation, for example, in submitting a new feature request through issue repository, a potentially good way to start participating while making a useful contribution.

4.5.2 Coding of Textual Requirements Data

For the purpose of analysis involving requirements quantity and completeness, the requirements data collected from the experiment were coded independently by me and a second independent coder (a graduate student in MIS) who was not involved in the study. For the coding, the taxonomy proposed by Browne and Rogich (2001) was used. For the coding, the textual requirements data from participants were segmented into smallest substantive units, i.e., individual, distinguishable phrases and sentences (e.g., Curley et al., 1995). Initially, in terms of requirements quantity, there was 66% agreement between the initial coding between the two coders. Wherever disagreements occurred over the categories assigned to requirements pieces, I resolved it through a careful analysis of the definition of requirements categories and the requirements pieces to assign a final acceptable category. A sample of the coded data is shown below:

Table 27. Sample coding

Requirements data	Code
Must be usable by people with limited technical knowledge Must cater to the personal user's needs Must be safe in terms of data protection Must be affordable Must be widely supported	Goal level specification (5)

4.5.3 Results: Requirements Quantity

Similar to Browne and Rogich (2001), the quantity of requirements was measured as the total number of requirements generated by subjects. This is a continuous variable.

As before, general linear model (SPSS) was run with requirements quantity as the dependent variable, the availability of reusable requirements library as the categorical predictor variable and experience of contributing to crowdsourcing tasks, technical background – career, technical background - education and experience with medical software as covariates. The histogram of the standardized residuals with normal curve imposed on it shows approximate normality (see Appendix 5). As per previously mentioned rules of thumb, (West et al., 1995; Stull, 2008), the skewness and kurtosis shown in Appendix 5 were well below all of the threshold values. For homogeneity of variance assumption, Levene's test should be non-significant. As can be seen in Appendix 5, Levene's test is indeed non-significant.

The equation for the general linear model is shown below:

Requirements quantity = c1*X1 + c2*X2 + c3*X3 + c4*X4 + c5*X5 + c6* X1* X2+ c7*X1*X3+ c8*X1*X4+ c9*X1*X5 + d

where,

X1 = Availability of reusable requirements library

X2 = Experience of contributing to crowdsourcing tasks

X3 = Technical background - career

X4 = Technical background – education

X5 = Experience with medical software

The results from running the general linear model are shown in Table 28.

Table 28. Model parameters (Dependent variable: requirements quantity)

Source	F	Significance
Corrected Model	3.728	.001
Intercept	24.281	.000
Availability of reusable requirements library	9.591	.003
Experience of contributing to crowdsourcing tasks	3.912	.052
Technical background - career	.570	.453
Technical background - education	.720	.399
Experience with medical software	2.867	.095
Availability of reusable requirements library* Experience of contributing to crowdsourcing tasks	10.200	.002
Availability of reusable requirements library* Technical background - career	5.057	.027
Availability of reusable requirements library* Technical background - education	1.337	.251
Availability of reusable requirements library* Experience with medical software	.336	.564

R square = .312 indicating that the model accounted for about 31 % of the variance in the dependent variable, requirements quantity. By Cohen's criteria, the variance accounted for by the model here is large.

The main effects for the availability of reusable requirements library and crowdsourcing experience are significant, as well as the interactions of the availability of reusable requirements library with crowdsourcing experience and Technical background - career. This provides evidence for the significant effect of the availability of reusable requirements library on requirements quantity after controlling for covariates. The above results show that the effect of the availability of a library of reusable requirements on requirements quantity varies with varying levels of technical and crowdsourcing experience of participants. Specifically, during web-based requirements generation, the availability of a library of reusable requirements may be more useful to individuals with lower levels of technical and crowdsourcing experience. The parameter estimates for significant interactions are provided in Table 29. Based on these results, it appears that reliance on crowdsourcing experience and technical knowledge is significantly more when the reusable requirements library is not available in comparison to when a reusable requirements library is available.

Table 29. Interactions

Parameter	B	Std. err.	Sig.
[Availability of reusable requirements library=0]* Experience of contributing to crowdsourcing tasks	.003	.001	.002
[Availability of reusable requirements library=1]* Experience of contributing to crowdsourcing tasks	0		
[Availability of reusable requirements library=0]* Technical background - career	1.094	.486	.027
[Availability of reusable requirements library=1]* Technical background - career	0		

Participants in OSS development who are from a technical background or who have sufficient experience of contributing to OSS development (one type of crowdsourcing experience) have the advantage that they can potentially utilize such knowledge during requirements generation in OSS development. Individuals from a non-technical background or who are new to crowdsourcing would have nothing to rely on while participating in OSS requirements generation. For such participants, external aid such as a library of reusable requirements can be beneficial for discovering requirements that may otherwise be unavailable in their minds.

4.5.4 Results: Requirements Completeness

Similar to Browne and Rogich (2001), breadth was measured as the number of distinct requirements categories provided by subjects. As before, general linear model (SPSS) was run with breadth as the dependent variable, the availability of reusable requirements library as the categorical predictor variable and experience of contributing to crowdsourcing tasks, technical background – career, technical background - education and experience with medical software, as covariates.

Table 30. Model parameters (Dependent variable: requirements breadth)

Source	F	Significance
Corrected Model	2.346	.022
Intercept	83.371	.000
Availability of reusable requirements library	7.718	.007
Experience of contributing to crowdsourcing tasks	1.300	.258
Technical background - career	.052	.821
Technical background - education	.042	.839
Experience with medical software	1.438	.234
Availability of reusable requirements library* Experience of contributing to crowdsourcing tasks	.510	.478
Availability of reusable requirements library* Technical background - career	9.119	.003
Availability of reusable requirements library* Technical background - education	2.227	.140
Availability of reusable requirements library* Experience with medical software	.003	.955

The interaction between the predictor variable of interest, availability of reusable requirements library and each of the covariates was also included in the model. The results are shown in Table 30. Levene's test was again non-significant. R square = .222, indicating that the model accounted for about 22 % of the variance in the dependent variable, breadth of requirements. By Cohen's criteria, the variance accounted for by the model here is close to being large.

The main effect for the availability of reusable requirements library is significant as well as the interactions of the availability of reusable requirements library with technical background - career. This provides evidence for the significant effect of the availability of reusable requirements library on requirements breadth after controlling for covariates. The effect of the availability of a library of reusable requirements on the breadth of requirements appears to vary with varying levels of the technical background of participants. When it comes to generating more diverse requirements, it appears that the availability of a library of reusable requirements may be more useful to individuals with lower levels of technical experience. The parameter estimates for significant interaction is provided in Table 31.

Table 31. Interaction

Parameter	B	Std. err.	Sig.
[Availability of reusable requirements library=0]* Technical background - career	.401	.133	.003
[Availability of reusable requirements library=1]* Technical background - career	0		

Similar to Browne and Rogich (2001) and Pitts and Browne (2007), depth is measured as the number of requirements within each category (from the taxonomy) provided by subjects. For the goals category, the result is shown in Table 32:

Table 32. Model parameters (Dependent variable: requirements depth)

Source	F	Significance
Corrected Model	2.318	.023
Intercept	16.299	.000
Availability of reusable requirements library	4.306	.041
Experience of contributing to crowdsourcing tasks	1.133	.291
Technical background - career	.335	.565
Technical background - education	.299	.586
Experience with medical software	1.778	.186
Availability of reusable requirements library* Experience of contributing to crowdsourcing tasks	7.967	.006
Availability of reusable requirements library* Technical background - career	3.058	.084
Availability of reusable requirements library* Technical background - education	.828	.366
Availability of reusable requirements library* Experience with medical software	.029	.864

Levene's test was again non-significant. R square = .220, indicating that the model accounted for about 22 % of the variance in the dependent variable, depth of requirements. By Cohen's criteria, the variance accounted for by the model here is close to being large.

The main effect for the availability of reusable requirements library is significant, as well as the interactions of the availability of reusable requirements library with crowdsourcing experience. This provides evidence for the significant effect of the availability of reusable requirements library on requirements depth after controlling for covariates. The effect of the availability of a library of reusable requirements on the number of requirements generated in goals category appears to vary with varying levels of crowdsourcing experience of participants. When it comes to generating requirements of type goal, it appears that the availability of a library of reusable requirements may be more useful to individuals with lower levels of crowdsourcing experience. The parameter estimates for significant interactions are provided in Table 33.

Table 33. Interactions

Parameter	B	Std. err.	Sig.
[Availability of reusable requirements library=0]* Experience of contributing to crowdsourcing tasks	.002	.001	.006
[Availability of reusable requirements library=1]* Experience of contributing to crowdsourcing tasks	0		

For the other requirements categories, there were largely no significant effects. Thus, the effect of reusable requirements library appears to be largely on goals category which could be because of the generic and non-functional nature of the requirements patterns that were used in the experiment.

The results for requirements breadth and depth together provide evidence for the significant effect of the availability of a library of requirements patterns on requirements completeness. As with requirements quantity, the effect of the availability of a library of reusable requirements on requirements completeness appears to vary with varying levels of technical background and crowdsourcing experience of participants. During web-based requirements generation, the availability of a library of reusable requirements could be potentially beneficial to individuals with lower levels of technical and crowdsourcing experience, for example, in generating more diverse requirements.

4.6 Discussion and Conclusion

The quantitative results show that there is a significant effect of the availability of a reusable requirements library (after controlling for covariates) on the quantity, breadth, depth, and size of requirements, thus providing support for hypotheses one, two and three. The results provide evidence for the usefulness of the availability of a library of reusable requirements in web-based OSS development environments. Such a library can potentially help improve the size of requirements messages generated. Joyce and Kraut (2006) found that there was a significant positive association between length (word count) of the initial message posted by newcomers in the discussion forums of OSS projects and getting a reply to it, indicating that there was greater likelihood of project maintainers responding when newcomers wrote a longer initial message (Joyce and Kraut, 2006). Receiving a reply, in turn, had significant positive effect on the subsequent participation of newcomers (Joyce and Kraut, 2006). As the quantitative results indicate, a library of reusable requirements can potentially help newcomers construct requirements messages of greater size that in turn could facilitate rich communication between newcomers and OSS project maintainers. Arguello et al. (2006) found that topic coherence (message topic being related to the goal and purpose of an online community) significantly increases the likelihood of getting a reply to a posting (Arguello et al., 2006). Steinmacher et al. report that there is a high likelihood of OSS project administrators not responding to messages that do not have meaningful content since they may not understand it (Steinmacher et al., 2015). Such non-responsiveness may hinder subsequent participation by newcomers (Steinmacher et al., 2015). A library of reusable requirements could be useful in improving the quantity and completeness of requirements-related postings, especially by novice contributors and contributors from a non-technical background. For example, contributors from non-technical background may be unaware of non-functional requirements; a good quality requirements library can be a useful external aid in broadening their requirements related thinking and subsequently facilitating generation of information rich and meaningful requirements messages. In a free and open access medium such as web-based OSS development environments, a library of reusable requirements can be easily incorporated, for example, as wiki pages that are not enforced upon the potential contributors, but rather something that they can access by choice and for convenience. Inexperienced newcomers who are interested in becoming potential contributors to open source software projects could potentially go through such artifacts if they wish before submitting any requirements information that they may have. As previously described, both researchers and practitioners have noted incompleteness (e.g., missing of crucial information) and invalidity as major problems with OSS requirements sources such as issue data. A library of reusable requirements could help broaden the requirements-related thinking, especially for newcomers, and help reduce instances of incompleteness and invalidity.

Just as incompleteness and invalidity are major information quality problems with requirements information that gets generated in OSS development environments, so is the erroneous classification of requirements information (misclassification) which, like incompleteness and invalidity, can be a major hindrance to OSS requirements discovery. The next chapter describes research investigating how two popular, but different, types of issue gathering approaches may contribute to the misclassification problem and what can be done at the issue gathering interface level to tackle misclassification.

CHAPTER FIVE: RECOMMENDATIONS FOR TACKLING MISCLASSIFICATION DURING OSS REQUIREMENTS GENERATION

5.1 Introduction

Issue data generated in the issue repositories of OSS projects are a major source of requirements for OSS projects (e.g., Crowston et al., 2012; Noll, 2008). Indeed, Alspaugh and Scacchi (2013) write: "*Perhaps the most common requirements like OSS artifacts are isolated feature requests or bug reports submitted to tracking systems like Bugzill*a" (p.167). In OSS development environments such as Sourceforge, GitHub, and CodePlex, users and developers can submit requirements information by using issue reporting artifacts provided within the issue repositories of these development environment (e.g., Vlas and Robinson (2012)). Examples of such issue data are shown in chapter two (see Figures 2 and 3). Screenshots of many OSS issue reporting artifacts are shown in Appendix 4. There are many information quality issues with the issue data generated in OSS issue repositories, such as incompleteness and invalidity (e.g., Bettenburg et al., 2008). Cavalcanti et al. (2014), based on their review of the empirical work done on OSS issue repositories, note the quality of issue data descriptions and issue data management processes as major research issues (e.g., causes and impacts) requiring more attention. Recently the developers and maintainers of many popular OSS projects on GitHub got together and wrote an open letter to the GitHub management about the most frequent problems they faced. Among the top problems was the absence of crucial information in issues, leading developers to request the incorporation of a mandatory issue template with custom fields into the GitHub issue reporting interface as a potential solution[2]. GitHub currently uses a simple issue reporting interface with just a text box as shown in Figure 6. Only project maintainers can add labels to a submitted issue. The requested enhancement by developers of OSS projects on GitHub, a mandatory issue template with custom fields, is similar to those provided by some other OSS issue repository platforms such as Bugzilla and Jira. An example (Bugzilla issue reporting interface for open office) is shown in Figure 7. The GitHub and Bugzilla issue gathering interfaces represent two widespread, but different, approaches to gathering issues in OSS development. The first approach does not impose any classification/labeling at the time of issue creation and submission, examples of which are GitHub, GitLab and CodePlex. Issue classification is the process of classifying issues according to some taxonomy or schema (e.g., Falessi et al., 2014). The second approach imposes classification/labeling at the time of issue creation and submission, examples of which are Bugzilla, Jira and Trac. The goal of this chapter is to investigate whether these two different approaches of OSS issue gathering could contribute to the misclassification problem and what can be done at the interface level for tackling the misclassification problem.

Different stakeholders involved in software development have different perspectives/perceptions/views about the software because of their different skills, knowledge, and experience (Finkelstein et al., 1992; Darke and Shanks, 1996). It is necessary to accommodate these various viewpoints during requirements gathering (Darke

[2] Source: https://github.com/dear-github/dear-github ; http://www.infoq.com/news/2016/01/dear-github-letter

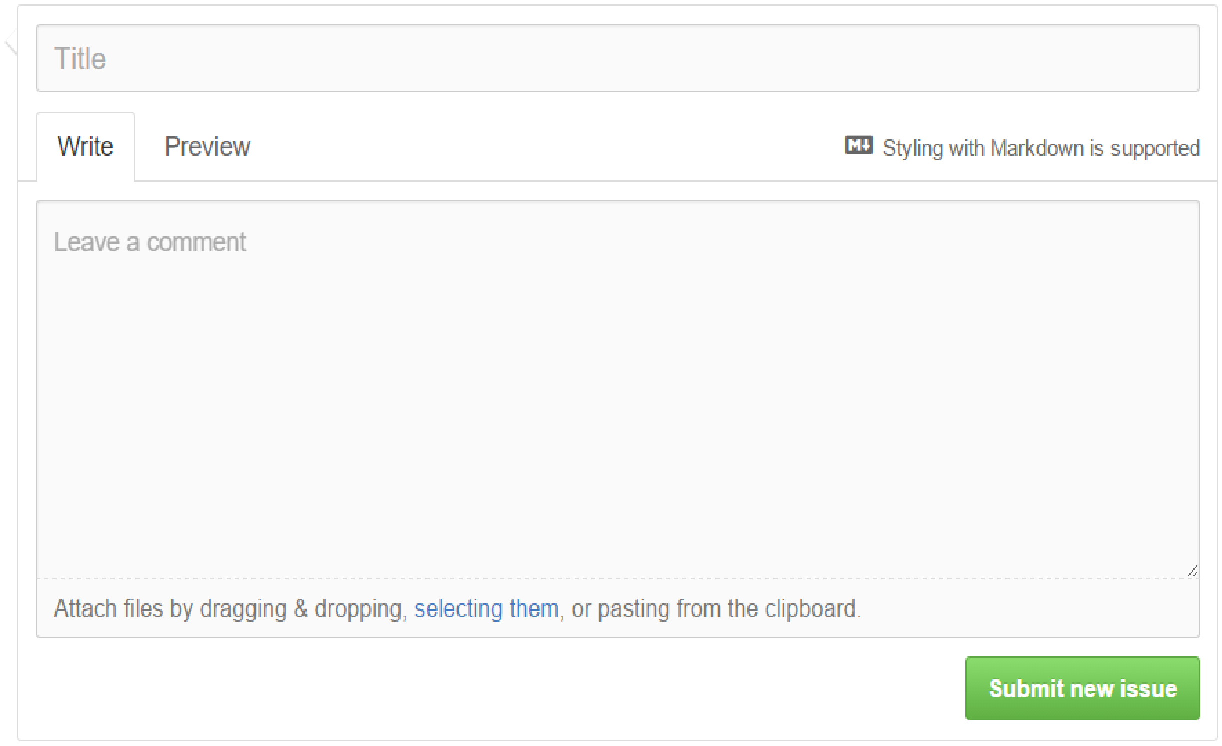
and Shanks, 1996). Software usage is heterogeneous and it is important to gather requirements information from all types of potential end users and stakeholders. In addition, different types of requirements information need to be gathered, including requirements about application domain, software environment, and the engineering – the development itself (e.g., technology to be used in the development) (Sommerville and Swayer, 1997b). Considering a single perspective about the software will not allow this, instead requirements need to be collected from diverse viewpoints (Sommerville and Swayer, 1997b). Results from the first research phase (chapter three) indicate that the requirements elicitation practice "*collect requirements from multiple viewpoints*" is perceived as beneficial for OSS development by OSS developers, in spite of low usage. When requirements are collected from a restricted viewpoint, needs of many stakeholders are not efficiently captured (Sommerville and Swayer, 1997). When requirements are collected from multiple viewpoints, this can have many benefits (e.g., inferring priority). For example, requirements suggested by many viewpoints could be potentially important and, subsequently, of high priority (Sommerville and Swayer, 1997). Choi and Chengalur-Smith (2009) note that it is critical to develop OSS projects for general end-users. OSS developers can often lack domain knowledge in many areas (Rantalainen et al., 2011) and often OSS projects written for end-users can end up as failures (Foushee, 2013). This makes it important in OSS domain to gather requirements from multiple viewpoints, especially from non-technical users.

To accommodate diverse views, it is important that information is gathered from individuals independent of any form of classification (Parsons and Wand, 2014).

Classification of information is usually carried out according to some accepted taxonomy or schema (e.g., Falessi et al., 2014). Imposing classification while gathering information (a priori classification) reflects a fixed, limited view (e.g., of the decision makers in a domain), which can be detrimental to accommodating diverse views (Parsons and Wand, 2014). Fixed views imposed by a priori classification may match with the views of only a limited number of information contributors and prevent inclusion of the views of others (Lukyanenko and Parsons, 2015; Lukyanenko et al., 2014[b]). The classification imposed during OSS issue gathering (e.g., Bugzilla) reflects a fixed, limited view, usually of a few project developers (this can vary from project to project), which can be a potential obstruction to gathering OSS issues and requirements from diverse viewpoints (e.g., non-technical users new to the specific OSS project).

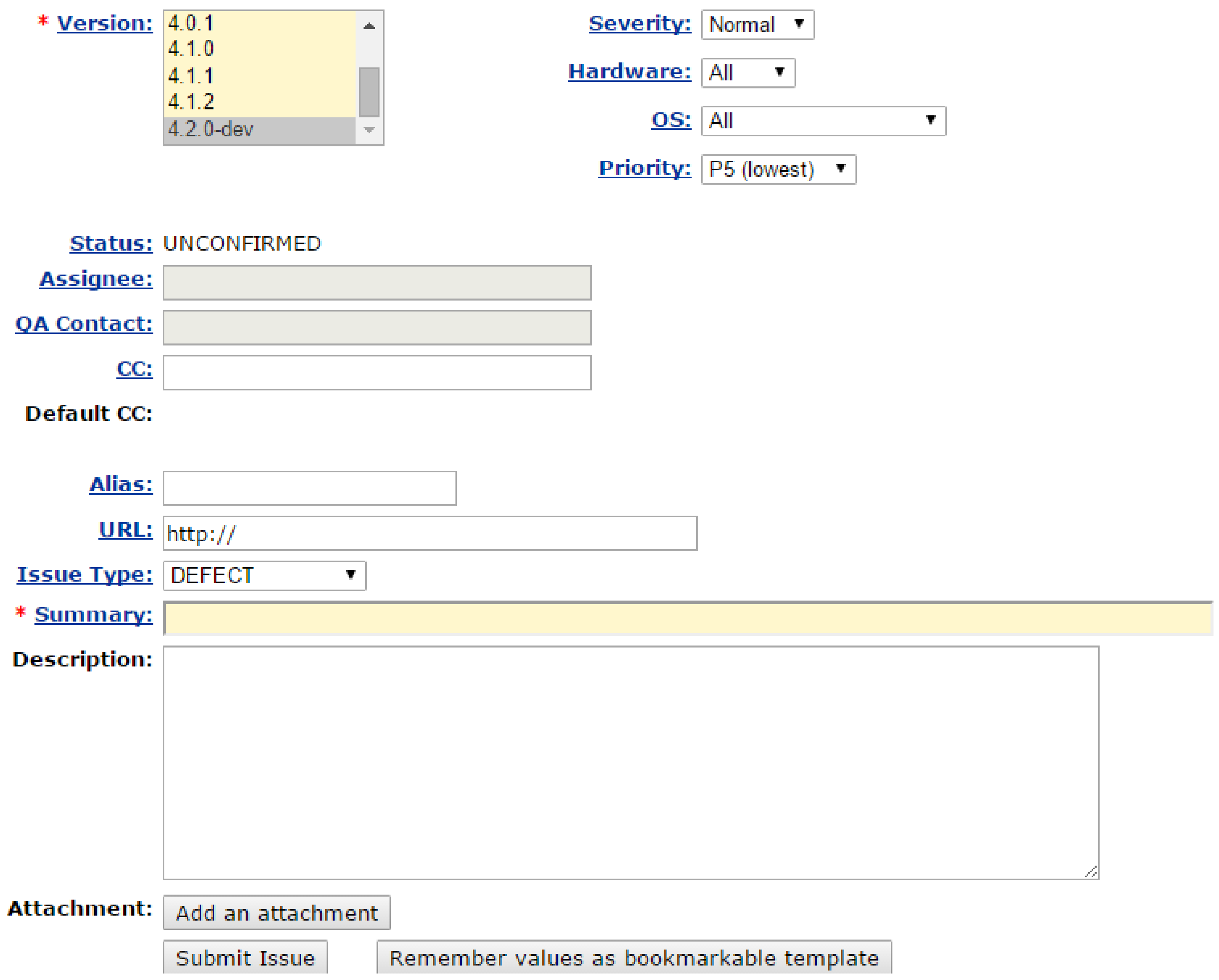
Enforcing a priori classification during information gathering can have negative impacts such as lower information quality and information loss (Lukyanenko et al., 2014[b]). Recent research has reported a major information quality problem with the OSS issue data, namely the misclassification (erroneous classification of issues) problem (e.g., Herzig et al., 2013). Herzig et al. found that about 39% of the issues that were classified as bugs in the issue data set that they analyzed were not really bugs but, rather, enhancement requests, feature requests and documentation issues. Similarly, a large percentage of issues that were classified as feature requests and enhancement requests were found to be actually of some other type (Herzig et al., 2013). It is important to investigate whether and how the two different OSS issue gathering approaches contribute to the misclassification problem.

Figure 6. GitHub issue reporting interface


Title
Write
Preview
Styling with Markdown is supported
Leave a comment
Attach files by dragging & dropping, selecting them, or pasting from the clipboard.
Submit new issue

Falessi et al. (2014) found empirical evidence that errors in issue classification can decrease verification and validation effectiveness; for example, making it hard to find issues of interest. Falessi et al. mention that such errors in issue data can decrease the usefulness of developing and deploying data mining techniques, as well as technologies (e.g., defect prediction models) that utilize issue data. Kochhar et al. (2014) provide an example of this by empirically demonstrating that misclassification has significant negative impact on bug localization (techniques that take as input an issue report and process them to locate source code files that are likely to be related to the issue). Falessi et al. call for research aimed at investigating support for decreasing human errors in issue classification.

Figure 7. Bugzilla issue reporting interface



* Version:	4.0.1 4.1.0 4.1.1 4.1.2 4.2.0-dev	Severity:	Normal
		Hardware:	All
		OS:	All
		Priority:	P5 (lowest)
Status:	UNCONFIRMED		
Assignee:			
QA Contact:			
CC:			
Default CC:			
Alias:			
URL:	http://		
Issue Type:	DEFECT		
* Summary:			
Description:			
Attachment:	Add an attachment		

From the preceding discussion, it can be seen that the misclassification of issues could be a major challenge for requirements discovery during OSS development. For example, a feature or improvement request that has been misclassified as a bug may be concluded as invalid by developers since they would not be able to reproduce it. It is important to investigate potential sources and solutions of the misclassification problem. In this chapter, I explore through a qualitative methodology involving OSS developers, whether and how the two popular but different approaches of OSS issue gathering contribute to the misclassification problem and what can be done at the issue gathering interface level for tackling misclassification. The next section describes a theoretical foundation for exploring the misclassification problem.

5.2 Issue Classification and Misclassification: Theoretical Underpinnings

Open information environments (OIE) are environments in which new sources and uses of information emerge and where the users of information, while having access to different sources of information, often have no control over them (Parsons and Wand, 2014). The stakeholders in OIEs are usually information contributors (sources), information consumers (users) and OIE sponsors (Parsons and Wand, 2014). OSS issue repositories are OIEs in which the sources of information (issue data) are the developers and non-developer users who report issues, and the users are the specific project developers/maintainers who use the issue data for implementation purposes. The sponsors could be the management team of web-based OSS environments (e.g., GitHub) and organizations interested in OSS development (e.g., Redhat), who may pay their employees to participate in OSS development.

Parsons and Wand identify the need to accommodate semantic diversity and ensuring information quality as key requirements for OIEs to be successful. Different users and sources are likely to have different views/interpretations of the information generated in OIEs; these different views can lead to different meanings being assigned to the data generated or data available, and views can change over time (Parsons and Wand, 2014). Accommodating semantic diversity implies the need for OIE applications to have mechanisms in place for accommodating the different and evolving views of sources and users. The different contributors of OSS issue data (non-developer users and developers) and users of these information (project developers, maintainers, and the interested readers) come from diverse backgrounds (e.g., different organizations and nationalities), and are geographically distributed (e.g., Crowston et al., 2012) and, therefore, can be expected to possess diverse views about the issues concerning OSS projects. For example, individuals from different countries could have very different views about user interface (Schmid, 2014).

Individuals can observe some characteristics of objects in a domain and form their individual perceptions about what they have observed; they could form different conceptualizations about the same characteristics of an object (diverse views), and these conceptualizations could even change over time for an individual (evolving views) (Lukyanenko and Parsons, 2015). Issue data submitted by an individual issue reporter can be viewed as information about some phenomena in the application domain of the particular OSS project. Multiple issue reporters could observe the same issue, form diverse perceptions about it, and report their individual descriptions about the observed issue. This could result in issue information from different reporters being perceived as duplicate by the project developers/maintainers. An example of duplicate issue information is provided below:

Three different submitted bug reports perceived as duplicates of each other by project developers (source: open office Bugzilla https://bz.apache.org/ooo)

[1] "*The rows are way too small (in fact I can't see a thing). I had to upsize the fonts to 22 to get a decent view of the sheet.*"
[2] "*The default row height is set to 0.0 for all cells when first starting. I have been unable to find a place to override this setting.*"

[3] "*Just installed 1.0 on redhat 7.2 with KDE 2.2. Open up a new spreadsheet. The rows are invisibly tiny. I select all rows with ctrl-a, then go to menu format/row/height and the height is showing as 0.03 cm with the default checkbox checked on. I enter 0.5 cm in the edit box and the default checkbox turns itself of. I close the dialog and the rows are now large enough to type in. Bug: The rows should not open so small. Where did the default of 0.03 come from?*"

In the above example, the three bug reports were submitted by different reporters who observed the same issue, formed different individual conceptualizations of it (diverse views) and reported their individual descriptions about the issue. Bettenburg et al. report that OSS developers may not perceive duplicate issue information as a serious problem; instead, they may add useful information about an issue (Bettenburg et al., 2008). Hence, diverse individual descriptions about the same issue could potentially help enrich the issue information content. Since different issue reporters may make observations about some phenomenon related to an OSS project in different ways, some may make rich observations/mental visualizations and subsequently provide rich, detailed issue information, while others may end up providing incomplete or incorrect issue information as perceived by the project developers/maintainers. Incompleteness and incorrectness are commonly occurring problems with the issue data in OSS issue repositories (Bettenburg et al., 2008). In the above example on duplicate bug reports, it can be seen that the third bug report has detailed information content, whereas the first bug report has limited information content, potentially illustrating the differences in the mental conceptualizations of their reporters at the time of reporting.

To support diverse and evolving views, that is, facilitate semantic diversity, a desired property from OIE applications is that they should allow capturing and storing information from contributors independent of any form of classification (Parsons and

Wand, 2014; Lukyanenko and Parsons, 2015). Classification is a human dependent activity and can be greatly influenced by human characteristics such as experience and knowledge (Lukyanenko et al., 2014[b]). The same thing may be classified differently by different individuals or the same thing may be classified differently by an individual at different times (Lukyanenko et al., 2014[b]). For example, one individual may classify a passport as an identity document while another individual may classify it as a travel document (Lukyanenko et al., 2014[b]). A priori classification presented in any IS artifact reflects fixed views that cannot easily accommodate the multiple and rapidly evolving views that are commonplace in OIEs (Parsons and Wand, 2014). Fixed views imposed by a priori classification can bias user-generated content to the views of a limited set of contributors and prevent the inclusion of views of others (Lukyanenko and Parsons, 2015). This is because individuals can widely differ in their conceptualizations of objects in some domain and individual conceptualizations can vary over time as well. As a result, the views of many potential contributors may not match with the limited view that an a priori classification imposes (Lukyanenko and Parsons, 2015). When information contributors are unfamiliar with the classes presented by an information system artifact to them, the result is a forced choice which does not match with the perceptions of the information contributors (Lukyanenko et al., 2014[b]). This can have negative impacts such as lower quality of contributed information and information loss (Lukyanenko et al., 2014[b]). As an example in the context of OSS issue repositories, comment 27 (Table 38) points out issues that are edge cases; for example, issues that are both a bug and an enhancement. Other combinations are possible like bug-documentation or bug-enhancement-user interface (source: GitHub). A restrictive a priori classification provided by issue gathering interface such as Bugzilla cannot accommodate such diverse cases and may result in loss of such information.

In OSS issue repositories, this would mean that the issue reporters should be able to specify their issue information as it is in their minds without having to worry about assigning them to some a priori classes/labels that an issue gathering interface provides . In other words, OSS issue gathering interfaces should capture issue information from reporters without imposing the need to assign specific class labels to them while creating and submitting issues. This can clearly support the diverse views of many different issue reporters distributed across the globe. For example, consider the label *issue type* in the Bugzilla issue reporting interface. If a reporter chooses enhancement as the type of his/her issue, in order to be certain it is indeed an enhancement, he or she needs to be certain that the requested characteristic is not already in the software which would mean having a good knowledge of the current functionalities and characteristics of the software. It is highly likely that often this is not the case. Consider the other label *priority* (*severity* is similar to this). Prioritization of requirements often involves groups of requirements and stakeholders, for example, high priority mould mean a requirement is likely to be implemented much before several other requirements or that it is more important in comparison to several other requirements to a group of stakeholders (Firesmith, 2004). A lone issue reporter submitting a single issue at some time point may not have a very good idea of priority and severity of the issue he or she is submitting. This is also indicated by the following comment of a responding OSS developer in the second survey: #1: "*Reporters are very rarely able to accurately decide priority, severity or any of the other fields presented in interface two. Only version is really valuable and that is easily* *established with a quick back and forth with the reporter*." Hence, by asking the issue reporter to assign such labels to their issue description, issue reporting interfaces such as that of Bugzilla appear not to be accommodating diversity well in the views of issue reporters (e.g., those issue reporters who do not have enough knowledge to provide all labels). As a result, many issue reporters may provide incorrect labels (e.g., see developer comment 18, Table 37), and the issue description gets stored along with those incorrect labels. Thus enforcing a priori classification at the time of creation of issues is a potential contributor to misclassification.

On the other hand, many OSS issue gathering interfaces (e.g., GitHub) provide a simple interface that seeks to capture just the issue description from the issue reporter. The issue reporters do not need to add any labels or classes to their issue information and the issue information gets stored independent of any classes/labels. In GitHub, only project developers/maintainers can assign labels to the submitted issues (https://help.github.com/articles/creating-an-issue/). Thus, the decision makers (project developers/maintainers) can infer the labels/classes (e.g., whether an enhancement, feature request or a bug) for a particular issue from the issue description itself, provided sufficient information has been provided in the description (c.f., Lukyanenko et al., 2014) and the issue reporters are not forced to classify/label their issues at the time of creation of their issues.

GitHub and Bugzilla issue gathering interfaces represent two popular, but different, ways of capturing and storing issue information from reporters in OSS domain. This is also indicated by the following comment of a survey respondent: #2: "*IMHO both* *these interfaces are widespread and need improvement. Specifically, the true goal is getting the problem fixed. Therefore both interfaces would benefit by having a prominent area to accelerate any fix….*" Google Code, Gitlab and Codeplex are examples of OSS development environments that use an issue reporting interface (shown in Appendix 4) similar to that of GitHub whereas Jira is an example that is similar to Bugzilla.

Differences in how an information system captures information from contributors can influence the quality of that information; for example, putting restrictions on contributors can result in information loss (Lukyanenko et al., 2014). Therefore, it is important to investigate how the two different approaches to issue data gathering in OSS issue repositories may affect the information quality of issue data. This becomes even more important considering the recent demands to GitHub management from some GitHub developers for a complex issue reporting interface similar to that of Bugzilla.[3] .In this third phase of the research, I take a qualitative approach to explore how the two different issue gathering approaches in OSS development may contribute to the misclassification problem (an information quality problem with OSS issue data) and what can be done at the interface level for mitigating the misclassification problem.

The next section describes in greater detail the research methodology.

[3] See, for example: https://github.com/dear-github/dear-github/issues/59 ; https://github.com/dear-github/dear-github/issues/72

5.3 Research methodology

A web-based survey was used to gather OSS developers' perceptions about how the two different issue gathering approaches in OSS development could contribute to the misclassification problem and what can be done at the interface level to tackle misclassification. Email invitations containing the survey link were sent out to OSS developers. OSS developers often participate in both issue reporting as well as issue classification and, with their knowledge and experience, are a good source of information for potential misclassification sources and solutions. Screenshots of GitHub and Bugzilla issue reporting interfaces were shown to the participants, and they were asked to indicate, based on their knowledge and experience, which interface was more capable of reducing misclassification. The questionnaire is shown in Appendix 7. Eighty-six OSS developers responded. In addition, participants were asked to provide their comments in text form in order to capture their thoughts and perceptions about the two types of issue reporting interfaces, especially in the context of misclassification problem. Fifty-seven comments were obtained. The coding process for comments was carried out by me (researcher) which is "sufficient and preferred" as suggested by experts (Bradley et al., 2007, (p.1761); Morse, 1994; Morse and Richards, 2002; Janesick, 2000). In fact, Morse writes: "The quantitative model of ensuring reliability and validity by using external raters is not recommended for qualitative research" (p.231, Morse, 1994). A reason for this is the violation of the process of induction that is characteristic of qualitative research (Morse, 1994). Card sorting technique [in which the comments are split into atomic parts (i.e., statements) and then put together and analyzed to identify common themes] (Zimmermann et al., 2010) and NVivo software were used to analyze the comments.

Examples are provided in Appendix 9. Open coding (researcher identifying themes emerging from the data (e.g., Hoepfl, 1997)) was matched with automatic coding by NVivo (e.g., Auld et al., 2007) to identify commonalities. For example, automatic coding indicated *user* as a theme and *user misclassification, average user* and *allowing user* as some of the subthemes. This had commonalities with the coding *incorrect classification when users themselves classify issues* by the researcher. The obtained insights are described in detail in the findings section (next). In addition, sentiment analysis (identifying positive and negative opinions in a text (e.g., Liu, 2012)) was also carried out using NVivo (discussed in section 5.5). Examples are provided in Appendix 9.

5.4 Findings

Eighty-six OSS developers participated in the survey. The majority of the respondents (61.6 %) indicated that the simple issue reporting interface of GitHub was more capable of reducing misclassification/incorrect labeling. The results are shown in Table 34:

Table 34. Respondent perceptions on which interface best reduces issue misclassification

GitHub interface	**Bugzilla interface that requires issue submitters to provide many label values at the time of submission of issues**	**Both interfaces**	**Neither interface**
61.6%	24.4%	4.7%	9.3%

In general, OSS developers appear to perceive the simple interface as more capable of reducing misclassification as it does not require issue submitters to classify/label their issues in contrast to the complex interface. This is indicated by the following comment: #3: "*I think interface one is better to avoid misclassification because it does not allow you to choose and the second one almost encourages choosing wrong* *because it requires so many choices to be made that some will be mistaken.*" (Others examples: see comment 6 (Table 35), comment 19 (Table 37), comment 22 (Table 37))

The developers' comments contained their opinions and perceptions about the pros and cons of the two types of interfaces in the context of misclassification and their views about how issue reporting interfaces should be for tackling misclassification. The insights that were obtained from the analysis of developers' comments are described.

Insight one: OSS developers appear to perceive themselves as more capable of performing accurate classification of issues in comparison to users (supportive comments in Table 35 below**)**

Table 35. Supportive comments for insight one

C.N.	OSS developers' comments	My description of comments
4	"*Usually the developers can choose labels that are more accurate and useful*"	Expressing confidence in OSS developers' ability to assign labels with relatively high accuracy to issue data
5	"*In my experience, it is better to let developers/admins triage the issues. Users can be unreliable when it comes to this*"	Expressing confidence in OSS developers' and project administrators' ability to classify issues and being skeptical of the users' ability to classify issues; appears to rely on past experience
6	"*An issue interface should not allow the reporter to indicate the type of issue, as they are more likely to give an incorrect classification than a developer who reads the issue*"	Expressing lack of confidence in issue reporters' ability to correctly classify issues and expressing confidence in developers' ability to classify issues with relatively higher accuracy
7	"*Leaving the classification, triage and de-duping to the development team that knows the backlog, the decisions made (and why) and reduces data issues*"	Expressing confidence in the OSS project development team's ability to achieve better classification of issues; reasons that the project development team has better knowledge of what has already been implemented and why and what remains to be implemented; this knowledge can help in issue classification, for example, for assigning labels such as duplicate

8	"*In my experience, the project maintainers have a much higher accuracy rate*"	Expressing confidence in project maintainers' ability to classify issues with relatively higher accuracy (in comparison to users); appears to rely on experience
9	"*Leave the analysis to the developers who are familiar with the project*"	Expressing confidence in OSS developers' ability to classify issues; reasons that developers' familiarity with the project could help in issue classification
10	"*Classification probably should not be done by users but rather by those trained in classifying the issue*"	Being skeptical of the users' ability to classify issues; having the opinion that individuals with some training in issue classification should classify issues; this could be possibly developers who would have higher likelihood of having such training while pursuing their education and while working on technical tasks
11	"*Assuming that project owners are more effective at classifying issues, it would appear self-evident that an interface that requires all classification be completed by project owners should deliver more accurate classification overall. However, another way to look at it is that interface two probably results in much higher classification coverage, though at the expense of some accuracy*"	Being positive about project team's ability to achieve better issue classification; appears to like the many label categories that are part of the second interface
12	"*Assuming that the developers are better assigning labels and that the submitters make mis-classification, not letting submitters label at all solves that, I guess*"	Being positive about OSS developers' ability to achieve better issue classification; being skeptical of the issue submitters' ability to classify issues

OSS project developers/maintainers' ability to better classify issues than issue submitters (who are not themselves OSS developers/project maintainers) can be expected because of reasons such as their familiarity with the application domain of the OSS project (comment 9, Table 35), their knowledge of what has been implemented and what remains to be done (comment 7, Table 35), and their overall technical knowledge and experience (comment 10, Table 35). For an issue reporting interface, this could potentially mean requiring the non-developer users to perform as little classification/labeling as possible. The simple GitHub issue interface appears to achieve this by not requiring any labeling from the issue submitters. After the issue has been submitted, the project developers/maintainers can infer classes/labels as needed by reading the issue description (e.g., comment 6, Table 35; comment 16, Table 36).

Insight two: OSS developers appear to perceive issue classification as a part of the routine project management activities that they need to do (supportive comments in Table 36 below).

Table 36. Supportive comments for insight two

C.N.	OSS developers' comments	My description of comments
13	"*Classification is a project management issue. Users should not be required to perform that function to get their issues addressed*"	Expresses the view that issue classification is just like any other project management task that the project development team needs to do and that users should be kept free of the burden of classification during issue submission.
14	"*Bug versus feature is really a function of the engineering team. Therefore, (interface one) from GitHub is preferably simple because it allows the engineers to manage assignment*"	Expresses the view that issue classification is a task that project development team needs to do; positive opinion about interface one as it is based on this philosophy
15	"*It is not up to the user how to classify an issue. The user should only be given a basic form to submit an issue*"	Expresses strong opinion that issue classification is not a task that users should do; positive opinion about interface one as it is based on this philosophy
16	"*Labeling is not relevant in issue interface one, as labeling is handled post submission by project maintainers, and usually not by the issue submitter*"	Considers issue classification by project development team as the norm; expresses belief that interface one is based on this philosophy
17	"*It seems far better to me to push issue tagging, organization and pretty much anything and everything that isn't directly related to a user explaining their issue in conventional language off onto the project maintainers than to let the issue creator (provided they aren't a project maintainer) do all of that*"	Expresses strong opinion that issue classification should be handled by project developers; expresses the view that only responsibility on users should be to report their issues in natural language

Classification of issues would have an important role to play in the implementation activities of OSS projects especially since the implementation activities that need to be done can differ greatly for different types of issues. For example, a bug may need semantic changes to be made in the code to support corrective maintenance, whereas a feature request may need implementing new methods and functionalities (Herzig et al., 2013). Hence, an OSS development team may view issue classification as a natural part of their project maintenance activities, (comment 13, Table 36), instead of viewing it as a task that users should perform (comment 15, Table 36). For an issue reporting interface, this could mean focusing on asking users only to provide the issue description and not any classification information (e.g., comment 15, Table 36; comment 17, Table 36).

Insight three: Not requiring issue submitters to perform classification/labeling of their issues can increase the workload of OSS project developers/maintainers but so can misclassification (supportive comments in Table 37 below)

A simple issue reporting interface like that of GitHub defers classification/labeling to the project developers/maintainers, meaning extra work for them (e.g., comment 18, Table 37; comment 21, Table 37), but misclassification by issue submitters can increase the workload of developers as well, since they would need to relabel the misclassified issues correctly (e.g., comment 18, Table 37; comment 19, Table 37). In both cases, developers would have to read through the issue description to understand it. However, the first scenario (leaving issue classification to project developers) may be more desirable since, if developers choose not to classify, then issues remain unclassified but there is no error.

Table 37. Supportive comments for insight three

C.N.	OSS developers' comments	My description of comments
18	"*Maintainers of a project are in a better position to triage issues. This requires a small amount of work upfront on the part of the maintainer, but it's probably a wash given that users will end up guessing at the complexity of choices in interface two, and the maintainer will still need to clean things up. Interface one is also more likely to be used*"	Expresses the view that when project developers do issue classification, it will be some extra work for them but much less when compared to the work of cleaning up the misclassified data generated by users. Expresses the view that project developers would be able to achieve better issue classification in comparison to users who may simply end up guessing.
19	"*In issue interface one, the issue remains unclassified until a contributor assigns a label. This reduces user misclassification to zero but depending on the number of developers, there may be disagreements over whether it is an enhancement or a bug. With issue interface two, it immediately asks a user to classify their issue, but this is definitely open to users mischaracterizing their issues, resulting in more developer time, ensuring it goes to the right place*"	Expresses the view that users classifying issues (which is the case with interface two) can actually increase developers' workload by requiring them to clean up the misclassification by users. Expresses the view that issue interface one drastically reduces chances of occurrence of misclassification; the only thing that may happen is some disagreements over issue classification in case there are many developers in the development team.
20	"*I suppose interface one reduces misclassification but does so by not classifying at all, it defers the work of classifying to someone else. Presumably, some who is better able to make classifications? Of course, the question is, is there any one with the time and temperament available to classify all the tickets being filed*"	Points out that interface one defers the issue classification task to project team members who can be expected to be better at issue classification, thus reducing misclassification. The additional workload should not be an issue if the project team members have the time and motivation needed.
21	"*Leaving the project maintainers or issue triagers in charge of labeling reduces mislabeling but increases their workload*"	Expresses the view that classification of issues by project team members would mean additional workload for them but would reduce issue misclassification.
22	"*Interface two offers a lot more field to screw up. The simpler interface one does not offer any way to classify an issue, so the burden is on the project maintainer to do so, but that will inherently reduce mis-categorization*"	Expresses the view that in the case of interface one, project developers would have the additional workload of issue classification but there would be the advantage of a reduction in misclassification.
23	*"One makes the developer or some other agent of the developer responsible for triage. Two enables the reporter to attempt to triage themselves, saving developer time, but also more strongly enabling mislabeling. Neither handles this correctly"*	Expresses the view that with complex interface two, issue reporters themselves classify issues which mean there is a high likelihood of misclassification which could offset developers' time savings

On the other hand, in the second scenario (issue reporters themselves classifying the issues), if the developer does not reclassify an incorrectly labeled issue, the misclassification stays, resulting in significant information quality issues. This can happen in the second scenario, especially if the developers do not want to put in much effort, as indicated by the following comment: #24: "*This is not a strong bias since both really come down to how much effort the development team puts into consistency. I have seen GitHub projects which are well organized and Bugzilla projects where nobody looks at the tags because it is seen as too much effort to clean them up*" Thus, it appears that many OSS developers may not be willing to take the effort to correct the misclassified issues submitted by issue reporters, which means the misclassification stays. This may explain the large percentage of misclassification in Bugzilla and Jira issue data reported by Herzig et al.

Insight four: A simple issue reporting interface may better facilitate participation of non-developer users in OSS issue reporting in comparison to a complex interface (supportive comments in Table 38 below)

It appears that a simple issue reporting interface, such as that of GitHub, may better facilitate participation of non-developer users who are often unfamiliar with OSS development (e.g., comment 25; Table 38; comment 26, Table 38; comment 36, Table 40), and who may not have sufficient information about their issues in mind at the time the issues are submitted (e.g., comment 34, Table 39). In spite of not having all needed information, in a complex issue reporting interface users may be psychologically pressured to provide incorrect label information just because of the presence of several label fields in the issue reporting interface (e.g., comment 26, Table 38; comment 29, Table38; comment 33, Table 39; comment 35, Table 39).

Table 38. Supportive comments for insight four

C.N.	OSS developers' comments	My description of comments
25	"*Interface one leaves classification to project maintainers and has a lower barrier to entry for reporting issues*"	Expresses the view that it would be easier for users to participate in issue reporting with the simple interface in comparison to the complex interface, one of the reasons being that they don't have to do any classification when using interface one as it is left out to the project development team
26	"*Both have issues. I prefer first (looks like GitHub) because it is so much simpler, which makes it more likely an issue will actually be submitted. Two which looks like Jira etc. is far, far too complex and is likely to get people to mis-categorize or just give up. First would work well if a simple categorization UI component were added*"	Expresses the view that issues are more likely to get submitted when the simple interface one is used. Points out that when the complex interface two is used, there is a high likelihood for both misclassification and as well as non-participation.
27	"*In addition to improper labeling, a complex UI can be confusing to users and inflexible to edge cases (e.g., what if the issue is both a bug and a feature). Keeping it simple with a message thread allows for flexibility through the conversation itself*"	Points out that a complex interface can lead to misclassified issues, confuse users and not flexible enough to allow reporting of special case issues, for example, issues that can be categorized as both bug and enhancement. Suggests that a simple interface would allow for more flexible issue reporting.
28	"*I think that, if interface two is provided, then perhaps issues would be better classified, but a huge percentage of issues would not be filed at all. Faced with that huge form, I would not open half the issues I do at current. Perhaps I am answering this question poorly as your question asked about classification, but I think when the choice is between an interface with marginally more poor classification is put against one so complicated that the amount of issues is drastically reduced I think the open source software development process will suffer greatly*"	Expresses the view that a complex interface would drastically reduce issue submission.
29	"*Tough decision, interface one is definitely more attractive and approachable, but it does not capture the complexity of the problem enough. The second interface captures much more complexity, but it can possibly intimidate bug-reporters with its many options*"	Points out that for many potential issue submitters, the complex interface two can be intimidating whereas the simple interface one can be attractive and approachable

A complex interface such as that of Bugzilla may dissuade many potential issue submitters from submitting their issues as they may feel that they cannot provide what all information is being asked for (e.g., comment 28, Table 38).

Comment 27 (Table 38) brings up an interesting scenario – that of edge cases; for example, issues that are both a bug and a feature or issues that are user interface issue and a feature request. Clearly, the predefined label values in a complex interface such as Bugzilla do not provide issue submitters with the flexibility to assign multiple categories when the need arise, as comment 27 points out. This is another potential example of non-accommodation of diverse views when reporters are forced to select from a predefined label value for an issue that does not fit in, resulting in misclassification of their issues.

Insight five: A simple issue reporting interface would provide ease of use for potential issue reporters (supportive comments in Table 39 below)

Ease of use is an important factor in the use of technology artifacts (e.g., Davis, 1989). Ease of use appears to be an important factor in the context of issue reporting interfaces as well (e.g., comment 34, Table 39). The simple interface encourages users to focus mainly on the content of the issue instead of the accompanying metadata (comment 32, Table 39; comment 35, Table 39), which can result in potential time savings for issue reporters (comment 30, Table 39). Such factors can potentially make a simple issue reporting interface easy to use and facilitate greater actual use. The complexity of the second interface can frustrate/overwhelm the user (comment 33, Table 39; comment 35, Table 39) and make it difficult to use (comment 31, Table 39). Thus, the complexity of the second interface can be a potential barrier to its actual use.

Table 39. Supportive comments for insight five

C.N.	OSS developers' comments	My description of comments
30	*"Interface one, however, takes less time and is often easier to move labels around afterward"*	Expresses the view that issue submission process using interface one would save time and that the project development team can easily handle the labeling process post submission
31	*"Second form needs major streamlining to be truly usable"*	Points out that the complex interface two need to undergo major changes
32	*"One wins on human factors consideration. By encouraging the user to describe the problem rather than forcing them to tender non-applicable details, the developer who receives the issue is much more likely to get information that is more useful to addressing the issue at hand"*	Expresses the view that when considering human factors, especially for users, the simple interface one is better since it does not demand anything from the user other than the issue itself.
33	*"The first (GitHub) is bit too simple. There is no way to require any classification at all. The second (Jira?) is a total mess and asks for too much information. In the first case there is no opportunity to gather any information and in the second it is likely to be wrong because it asks for too much and frustrates the user into picking false options"*	Expresses the view that the complex interface two demands too much meta data from users which can frustrate them and result in submission of incorrect meta data along with the issue. Points out that issue interface one is simple, probably very simple, for users to use.
34	"*Allows me to submit easier. Many times when I am submitting issues, I do not know that much information about what is happening- better to be simpler and let the people debugging it worry about specifics*"	Points out that issue submitters may not have the necessary information/knowledge for issue classification at the time of submitting of issues; suggests that it would be better if project development team handles issues classification in such circumstances
35	"*I think that the average user submitting a bug report will feel a bit overwhelmed by all the choices in interface two and feel like they have to provide input for every UI element shown even if they are not sure what they should select or input. In interface one, the focus seems more like getting the raw bug information from the user and letting the developer/project admin determine the appropriate metadata that gets attached to the bug submission*"	Expresses the view that a complex interface such as interface two that asks issue submitters to perform several types of labeling for the issues being submitted can leave the users feeling overwhelmed, especially if they do not have the necessary information/knowledge for the labeling. Appears to suggest that simple interface one is not that way since with interface one, the overhead of classifying issues is on the project development team and the users have to focus only on submitting the issue data.

Insight six: Only OSS developers/contributors with technical skills and experienced users may be capable of doing good quality classification at the time of issue creation (supportive comments in Table 40 below)

Table 40. Supportive comments for insight six

C.N.	OSS developers' comments	My description of comments
36	"*It depends on who is reporting. If it is an end user, interface one is far preferable, as it encourages reports even when they are not familiar with open source development. If it is a contributor reporting, then interface two alleviates a lot of the responsibility for categorization on the part of the repository owners*"	Expresses the view that interface one would encourage end users who are often not familiar with OSS development to participate in issue reporting. Also, suggests that a complex interface such as interface two may be suitable only for skilled reporters such as OSS developers who are likely to be capable of achieving good quality issue classification at the time of submission and thus could help reduce the workload of the project development team.
37	"*It really depends on who is submitting the bug. In either case, for open source projects that have open issue tracking systems, issues submitted by people that are unfamiliar with the project and people that are less technical; for maximum accuracy, I think a combination of either interface and communication between the reporter, issue tracker moderator (if they are different from the project developers) and project contributors will suffice. For experienced and highly technical people, interface two combined with communication between the reporter and project contributor would probably produce the most accurate results*"	Expresses the view that the complex interface two would be suitable only for experience individuals and technically skilled individuals. Also points out that communication between issue reporters and project development team is important for the non-technical reporters, reporters unfamiliar with the project as well as technically skilled and experienced reporters.

When project developers/maintainers (or other developers/individuals with good technical knowledge or users experienced in issue reporting) report issues, they may have the necessary knowledge and skill for providing issue classification information at the time of issue creation/submission (e.g., comment 36, comment 37, in Table 40). Such individuals could potentially achieve better quality issue classification at the time of issue creation that, in turn, could help reduce some of the workload of the project developers/work maintainers (comment 36, Table 40).

Factors such as prior experience and domain expertise can result in different views of the same domain among different contributors and different views for the same contributor at different time points (Lukyanenko and Parsons, 2015). Comments in Table

40 indicate that domain experts (project developers/maintainers) and experienced individuals (developers/experienced users) can have views different from those of novice users about the same OSS project for which they are submitting their issues. To accommodate such diverse views, applications should allow for capturing information from contributors mainly in a free form manner without imposing mandatory classification (Lukyanenko and Parsons, 2015). Contributors can be encouraged (not required) to provide classification information as attached to the focal information if they wish to do so (Lukyanenko and Parsons, 2015). The mandatory classification imposed in a complex issue reporting interface such as Bugzilla may be convenient for individuals, such as domain experts, who have a higher likelihood of having that information in their mind, but inconvenient for the average users who may not have that information (e.g., comment 34, Table 39). Hence the complex Bugzilla interface may not work for such users. On the other hand, a simple interface such as that of GitHub, while convenient for average users (e.g., comment 36, Table 40), can also accommodate the views of project developers/maintainers, developers, individuals with technical knowledge, and experienced users, since they can simply provide the classification information as additional text data to the main issue content if they wish to do so.

Insight seven: OSS developers may prefer some meta-data from issue submitters that would be useful in issue data classification.

(Supportive comments in Table 41 below)

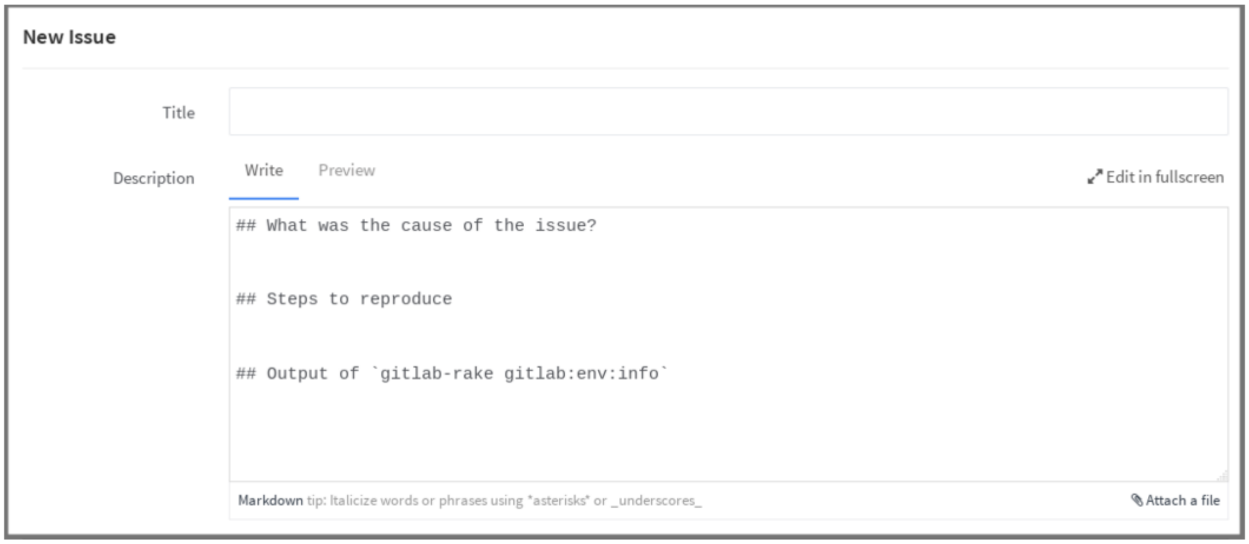
Table 41. Supportive comments for insight seven

C.N.	OSS developers' comments	My description of comments
38	"*I think maintainers/developers of a project still need to triage all issues that come into the system, but allowing users to provide a starting point of where they feel the issues belong, or their perceived severity allows maintainers/developers to prioritize the triaging*"	Expresses the view that while project developers should be largely responsible for issue classification, it may be helpful for the developers if they can get a sense of what the issue reporters think about where the issue being submitted fit in or how important it is.
39	"*The more people who can contribute to the classification, the better the classification would be (generally speaking)*"	Suggests that inputs from multiple individuals to the classification of an issue can improve the classification quality
40	"*Option one is much cleaner, but if it had optional things to add like assignee or category, it would be nice*"	Appears to like the simplicity of interface one; suggests that some additional information accompanying the issue can be useful; also suggests that requests for additional information should be optional, not mandatory
41	"*Simple interface, some labeling, not all fields needed from second screen though*"	Appears to like the simplicity of interface one; suggests that some additional information accompanying the issue can be useful
42	"*I wish GitHub had more fields. It is often useful to know particular version of application associated with a bug report and various other details*"	Suggests that additional information accompanying the issue can be useful
43	"*Interface two has some nice features, such as hardware and priority. Reminding the person that is submitting the issue that things such as OS and version are important for the bug. Those fields do not make sense in a feature request. Generally, I feel one works better, because if information is left out, it can just be requested so that the users gets the correct data. You would want android OS version, not just android, which a novice user might submit, but a more experienced user would most likely have "android 3.4.4, HTC 1, gen.2" listed anyways*"	Expresses the view that in general, the simple interface one would work better; An example scenario is when a feature request is being submitted, many of the label fields in the complex interface two will not make sense. Also, suggests that some additional information accompanying the issue can be useful; Reminders could be helpful for getting such additional information, especially in the case of novice users.
44	"*I like allowing people to apply an initial label, but that second form (is it from trac?)is too much detail IMO*"	Expresses the view that some additional information accompanying an issue can be useful but also points out that asking for as much detail as interface two does is undesirable

It appears that developers may prefer some classification information from issue submitters if they can provide it, such as version number and hardware (e.g., comment 42, comment 43, in Table 41). This could be a potential indicator of the desire of project developers/maintainers to get some meta-data on a submitted issue. While the complexity of an interface such as Bugzilla is acknowledged (e.g., Comment 33, Table 39), it is also expected that more brains on a cognitive activity such as classification could potentially improve it (e.g., comment 39, Table 41). Knowing issue submitters perceptions may be useful for the project developers (e.g., comment 38, Table 41).

A simple interface such as that of GitHub could support OSS developers' desire for some issue meta-data as comment 43 points out. Issue reporters with rich conceptualizations of the OSS projects could provide additional classification information as text data accompanying the main issue content through the simple GitHub interface. This can help avoid the dangers of misclassification through a complex interface, such as that of Bugzilla, while allowing issue reporters to provide supportive information to the project developers/maintainers. Some sort of reminder to the issue submitters for additional information may be useful to remind, but not require, them to provide useful classification information. In fact, Gitlab and Google Code provides a simple issue reporting interface like that of GitHub but with simple textual prompts to the submitters as shown in the Gitlab interface in Figure 8; this can be deleted if the submitter wants. Such prompts have potential benefits such as improving the completeness of requirements information elicited (e.g., Pitts and Browne, 2007).

Figure 8. Textual prompts in Gitlab issue reporting interface


New Issue
Title
Description
Write
Preview
Edit in fullscreen
Markdown tip: Italicize words or phrases using *asterisks* or _underscores_
Attach a file

```
## What was the cause of the issue?


## Steps to reproduce


## Output of `gitlab-rake gitlab:env:info`
```

Source: https://gitlab.com/gitlab-org/gitlab-ee/blob/master/doc/customization/issue_and_merge_request_template.md)

Different types of textual prompts may be needed for different types of issues. In fact, this is an active issue being considered by Gitlab (seeking different textual prompts for different types of issues).[4]

Insight eight: In a complex interface that requires labeling issues and which provides default label values, issue reporters could accidentally or because of lack of knowledge stick with default values which could contribute to misclassification.

(Supportive comments in Table 42 below)

[4] See https://gitlab.com/gitlab-org/gitlab-ce/issues/9088

Table 42. Supportive comments for insight eight

C.N.	OSS developers' comments	My description of comments
45	"*Interface one does not actually have any visible way to tag an issue with anything at all. Interface two by default has "DEFECT" selected as an issue type, which I can imagine would cause confusion or cause the user to skip it by accident*"	Points out that in the case of the complex interface two, because of confusion or by accident, issue reporters may just stick with default label values which can facilitate misclassification
46	"*Both have problems. Interface one gives no indication that there is the ability to classify. Interface two gives a default issue classification, which can be easily overlooked, and the default can just be accepted. There are also too many elements in interface two. Interface two is the better of the two, but it should not give a default, it should have a drop down say, pick classification, and have an indicator that it is a required field*"	Point out that in the case of interface two, issue reporters, by oversight, may just stick with default label values, which can facilitate misclassification.

It appears that, in a complex issue reporting interface such as Bugzilla, there is often chances of issue submitters sticking with default label values either by accident/confusion (comment 45, comment 46, Table 42), or because they may not have the knowledge/information to provide accurate values and, instead, decide to go with what is available, thus potentially contributing to the misclassification.

Insights one, two, four and five point out that enforcing classification at the time of creation of issues can facilitate misclassification, decrease participation, especially of non-technical users and decrease ease of use for issue reporters. Based on these insights, it is recommended that:

Recommendation one: Issue gathering interface should not enforce classification/labeling at the time of creation/submission of issues.

Insights three, six and seven point out that developers may prefer some meta-data accompanying the issue description that would be useful in issue classification. In fact, there is a good chance that issue descriptions from experienced reporters may contain such meta-data, as comment 43 points out. Hence it is recommended that:

Recommendation two: Qualified issue reporters (developers, experienced users) can provide useful meta-data that can assist project developers in issue classification as text data (e.g., phrases, sentences) accompanying the main issue description, if they wish to. The issue gathering interface should inform the issue reporter that they can provide such meta-data if they wish to but that it is not mandatory.

The next section discusses sentiment analysis of developers' comments.

5.5 Sentiment Analysis of Developers' Comments

Sentiment analysis at the sentence level involves determining whether the sentence expresses a positive or negative opinion (e.g., Liu, 2012). NVivo software was used to carry out sentiment analysis at the sentence level for OSS developers' comments. Table 43 lists sentences from developers' comments that were classified as containing negative opinions and inferences from them. In the context of the first approach to issue gathering in OSS issue repositories (which does not enforce classification at the time of issue gathering), the negative opinions expressed by OSS developers include disagreements that can arise among project developers over issue classification, limited time availability of OSS developers, and increased workload.

Table 43. Negative opinions of OSS developers

Classified as negative	Inference
"depending on the number of developers, there may be disagreements over whether it is an enhancement or a bug"	When developers themselves have to classify issues, a challenge can be the disagreements that can arise over an issue type, especially if there are many developers in the project development team.
"Of course, the question is, is there any one with the time and temperament available to classify all the tickets being filed"	The limited time availability of OSS developers can be a challenge when it comes to issue classification.
"Leaving the project maintainers or issue triagers in charge of labeling reduces mislabeling but increases their workload"	Issue classification by project developers would mean reduced misclassification but increased workload.
"Interface two offers a lot more field to screw up"	The many label fields in interface two mean a higher likelihood of errors in the submitted issue data
"The second interface captures much more complexity, but it can possibly intimidate bug-reporters with its many options"	The many label fields in interface two can be overwhelming to issue reporters
"Users can be unreliable when it comes to this"	Non-technical users can be unreliable when it comes to issue classification
"An issue interface should not allow the reporter to indicate the type of issue, as they are more likely to give an incorrect classification than a developer who reads the issue"	There is a higher likelihood of incorrect classification when issue reporters submit classification information along with issues than when project developers read issues and then assign classification information to it.
"Two which looks like Jira etc. is far, far too complex and is likely to get people to mis-categorize or just give up"	The complexity of the second interface can facilitate misclassification and deter participation
"In addition to improper labeling, a complex UI can be confusing to users and inflexible to edge cases (e.g., what if the issue is both a bug and a feature)"	The complexity of the second interface can cause confusion to users. It lacks flexibility to accommodate issues that can be classified in multiple ways, for example, issues that are both bug and feature.
"The second (Jira?) is a total mess and asks for too much information."	The second interface demands too much information from issue reporters which can be undesirable.
"In the first case there is no opportunity to gather any information and in the second it is likely to be wrong because it asks for too much and frustrates the user into picking false options"	The second interface demands too much information from issue reporters which can cause frustration and facilitate misclassification.
"Many times when I am submitting issues, I do not know that much information about what is happening"	Issue reporters may not have enough knowledge or information in mind to provide all the information that the complex second interface is asking for.

In the context of second approach to issue gathering in OSS issue repositories (which enforces classification at the time of issue gathering), the negative opinions expressed by OSS developers include the lower information quality of issue data (many errors in submitted data largely due to incorrect classification/labeling), cognitive burden in issue reporters, reduced participation especially of non-technical users, the lack of flexibility in accommodating many types of issue reporters (e.g., those who do not have sufficient information in mind to provide all the classification information) and the lack of flexibility in accommodating many types of issues (e.g., issues that are both a bug and a feature). (e.g., those who do not have sufficient information in mind to provide all the classification information) and the lack of flexibility in accommodating many types of issues (e.g., issues that are both a bug and a feature).

Table 44 lists sentences from developers' comments that were classified as containing positive opinions and inferences from them. In the context of first approach to issue gathering in OSS issue repositories (which does not enforce classification at the time of issue gathering), the positive opinions expressed by OSS developers include higher information quality (e.g., fewer errors due to project developers classifying issues instead of issue reporters), higher usability and greater control for project developers. There was also positive opinion about multiple individuals contributing to issue classification (e.g., getting useful metadata from experienced reporters), a potential goal of the second approach to issue gathering.

Table 44. Positive opinions of OSS developers

Classified as positive	**Inference**
"Leave the analysis to the developers who are familiar with the project"	OSS developers' familiarity with OSS projects can be beneficial for issue classification.
"Assuming that project owners are more effective at classifying issues, it would appear self-evident that an interface that requires all classification be completed by project owners should deliver more accurate classification overall"	The first interface, by not asking any classification information from issue reporters and deferring issue classification to project developers, can improve the accuracy of issue classification.
"Therefore, (interface one) from GitHub is preferably simple because it allows the engineers to manage assignment"	Not asking for classification information makes issue interface one simple and (desirably) allows project developers to control issue classification.
"By encouraging the user to describe the problem rather than forcing them to tender non-applicable details, the developer who receives the issue is much more likely to get information that is more useful to addressing the issue at hand"	The simple interface that does not ask for any classification information can potentially encourage issue reporters to focus entirely on the issue at hand and subsequently, project developers getting more useful information.
"It is often useful to know particular version of application associated with a bug report and various other details"	Meta data associated with issues can be useful.
"One wins on human factors consideration"	When it comes to human use, the simple interface is better.
"The more people who can contribute to the classification, the better the classification would be (generally speaking)"	More individuals contributing to issue classification can improve the quality of issue classification.
"Option one is much cleaner, but if it had optional things to add like assignee or category, it would be nice"	The simplicity of first interface is desirable. Some meta data accompanying issue description is desirable as well.
"Simple interface, some labeling, not all fields needed from second screen though"	The simplicity of first interface is desirable. Some meta data accompanying issue description is desirable as well.
"You would want android OS version, not just android, which a novice user might submit, but a more experienced user would most likely have "android 3.4.4, HTC 1, gen.2" listed anyways"	There is a high likelihood of issue data from experienced users containing useful metadata such as version related information.

From recommendation two, it can be seen that when using a simple interface (that does not require classification), experienced users and OSS developers could include metadata (that can be useful for project developers during issue classification) as text data accompanying the main issue description. There could be textual prompts or similar reminders that notify the issue reporter that they can provide such information if they wish to, but that it is not compulsory. In this way, an issue gathering approach (that does not enforce classification at the time of issue gathering) can accommodate reporters who do not have sufficient knowledge/information to provide useful meta-data as well as reporters who are capable and willing to provide such information.

5.6 Discussion and Conclusion

As previously described, in open information environments different contributors are likely to have different views/interpretations about some phenomena and, in order to accommodate the diverse and evolving views of information contributors, a desired characteristic in OIEs is that they should allow the capturing of information from contributors independent of any form of classification (Parsons and Wand, 2014). When an information system enforces a priori classification on contributors of user-generated content, it can have negative impacts such as information loss and lower information quality (Lukyanenko et al., 2014). A similar picture emerged from the research discussed in this chapter about OSS issue gathering approaches that enforce classification at the time of issue creation. An OSS issue gathering approach that forces issue reporters to classify their issues at the creation time can potentially lead to misclassification. It can also lead to the loss of issue information, especially when contributors are novice non-developer users. An issue gathering approach that does not force issue reporters to classify their issues at the time of creation of issues could potentially reduce misclassification and the potential loss of participation from such enforcement. An issue gathering interface based on such an approach could still accommodate metadata useful for classification from capable and willing contributors (e.g., OSS developers, experienced users) who can provide such information as additional text data accompanying the main issue description. Simple textual prompts (that can be deleted if a contributor wants to) could potentially serve as efficient reminders for such contributors.

CHAPTER SIX: THEORETICAL CONTRIBUTIONS, IMPLICATIONS FOR PRACTICE AND RESEARCH AND LIMITATIONS

6.1 Theoretical Contributions

This research makes theoretical contribution in the area of the design of web-based OSS development environments by focusing on OSS requirements discovery, an under-researched aspect of OSS development. Findings from the first and second phase of the research provide potential design improvement knowledge for making web-based OSS development environments friendlier towards requirements engineering. Findings from the third phase of the research provide preliminary insights into the potential association between OSS issue gathering approaches and the information quality of OSS issues (a major source of requirements for OSS projects). Phase three also provides insights into the desirable characteristics in OSS issue gathering interfaces for reducing misclassification (potential design knowledge about OSS issue gathering interfaces). The research also makes theoretical contribution by developing a model of the antecedents of requirements size, quantity and completeness and empirically assessing the antecedents. This research started with an exploration of the current state of the requirements discovery practices in OSS development. Relevant OSS literature was reviewed. Few works exist on OSS requirements discovery. These works have carried out qualitative analysis of a few web-based OSS projects (e.g., analysis of their discussion forums) and have claimed that requirements discovery in OSS development is informal and ad hoc. The first phase of this research extended the existing work by conducting a survey of OSS developers (described in chapter three) who are largely responsible for overseeing the developmental and maintenance activities (e.g., Crowston and Howison, 2005). The survey inquired about the usage of formal requirements engineering practices (from CSS development) and informal practices during OSS development, the perceived usefulness of formal requirements engineering practices for OSS development and the problems and challenges that could occur during requirements discovery in OSS development. The survey provided quantitative evidence for informality of requirements generation in OSS development and revealed several problems and challenges that could occur during requirements discovery in OSS development (e.g., incompleteness of requirements information generated in OSS issue repositories). The quantitative analysis further revealed that most of the formal requirements engineering practices (from CSS development) were perceived as beneficial for OSS development in spite of having significantly low usage, suggesting a significant gap in perceptions and practice when it comes to OSS requirements discovery. This indicates potential design deficiencies in web-based OSS development environments as well as potential directions for improvement for requirements discovery in OSS development. For example, incorporating feature support for some of the formal RE practices perceived as beneficial within existing web-based OSS development environments can be a potential improvement for the design of web-based OSS development environments as well as an improvement in the OSS development methodology itself.

One of the potential improvement directions that emerged from the survey (specifically requirements reuse) was selected for empirical investigation because it can be easily incorporated into existing OSS development environments and also there is some evidence for the positive attitude of OSS contributors towards reusing software development artifacts (e.g., Von Krogh et al., 2005). A web-based experiment was designed using Google Code, an OSS development environment in order to empirically investigate the potential benefits from the availability of a library of reusable requirements within a web-based OSS development environment for requirement discovery during OSS development. What emerged from the analysis of experimental data is that the availability of a library of reusable requirements can be useful during OSS requirements discovery. For example, it can help individuals with low levels of technical experience in constructing more detailed requirements messages. Examples of such individuals could be non-developer users of OSS projects and newcomers to OSS projects. Also, the analysis revealed potential antecedents of requirements size, quantity, and completeness, namely availability of relevant information, technical background, and crowdsourcing experience.

Finally, the research focused on exploring a specific problem in the context of requirements discovery in OSS development, namely misclassification of OSS requirements artifacts. A qualitative approach was used to collect data from OSS developers about how the two main types of issue reporting interfaces used in OSS issue repositories may contribute to the misclassification problem and what could be done at the interface level for tackling the misclassification problem. Many issue reporting interface related design suggestions emerged for tackling the misclassification problem. For example, a simple interface that does not require users to perform any classification at the time of submission of issues may be more efficient in tackling the misclassification problem.

Different types of design knowledge contributions are possible. For example, invention (new knowledge/solutions for new problems), improvement (new knowledge/solutions for known problems), adaptation (non-trivial or innovative adaptation of known knowledge/solutions for new problems) and routine design (applying known knowledge/solutions to known problems) (Vaishnavi and Kuechler, 2015). The knowledge contributions from this research are potentially of type improvement. The fact that OSS requirements discovery is problematic and challenging have been noted by researchers in past (e.g., Dietze, 2005) and further explored and expanded in the first phase of this research (chapter three). New knowledge emerged about several potential directions of improvement for requirements discovery in OSS development; specifically, it was found that many formal requirements engineering practices if adapted efficiently within existing OSS development environments, could potentially be beneficial for OSS requirements discovery. One of improvement directions that emerged was empirically evaluated in the second phase of the research, providing evidence for actual benefits. The analysis also revealed how individual difference variables such as technical background could be important factors that may need to be taken into account while planning for or undertaking activities for improving OSS requirements discovery (new knowledge). In the final phase of the research, the misclassification problem for OSS requirements artifacts was explored. This is a newly identified problem that has not been investigated much. This problem was explored by involving OSS developers, an authentic source for determining whether misclassification has occurred or not. New knowledge emerged about how issue reporting interfaces may contribute to the misclassification problem and some of the things that could be done at the interface level to potentially reduce misclassification problem (design suggestions).

Design knowledge can be manifested in the form of material artifacts (instantiation) or abstract artifacts (constructs, models, frameworks, architectures, design principles, methods, design theories) (Vaishnavi and Kuechler, 2015). Models can be viewed as descriptions/representations about how things are/should be, often representing the connection between problem and solution components which could then enable further exploration of the potential effects of design decisions; a key focus of models is often utility rather than truth (Vaishnavi and Kuechler, 2015; Hevner et al., 2004; March and Smith, 1995). The knowledge obtained from this research is largely in the form of models, describing possible connections/associations between problem components (general informality/adhocness of OSS requirements discovery, requirements description, requirements quantity, requirements completeness, misclassification of OSS requirements artifacts) and solution components (formal requirements engineering practices, availability of a library of reusable requirements artifacts within OSS development environments, issue reporting interfaces).

6.2 Limitations

This study is based on the assumption/view that formal developmental practices from CSS development are beneficial for OSS development, in line with others such as Michlmayr (2005) and Ciolkowski and Soto (2008). This can be constrained by practical challenges in OSS development domain, including lack of resources (e.g., financial resources, volunteers) for many OSS projects, time constraints of existing volunteers, the geographically distributed nature of OSS development, the ad hoc nature of OSS evolution, and the design limitations of web-based OSS development environments. Thus, benefits from formal requirements engineering practices for OSS development would be constrained by the extent to which these practices can be successfully incorporated into existing OSS development workflows.

This research used survey and experiment methodologies and quantitative and qualitative data analysis methodologies. The limitations of these methodologies apply to this research. For example, in experimental research, the artificiality of the setting can be a limiting factor and when there is less control over extraneous variables that could be another limiting factor (McLeod, 2012). The experimental setting for phase two of this research was Google Code, a specific web-based OSS development environment which could be potentially limiting. Future research could address this by replicating the experiment in other web-based OSS development environments such as GitHub and Sourceforge. The cross-sectional nature of the study is a potential limitation, and future research could address this by incorporating some of the formal RE practices as part of a longitudinal study involving the development of some real-world OSS projects and analyze the outcomes.

6.3 Implications for Research

The first phase of this research (chapter three) indicated several potential directions of improvement for requirements discovery in OSS development only one of which was empirically evaluated in the second phase of the research. The other potential directions provide a large number of possibilities of exploration for future research. For example, formal requirements documentation practices, how could they be incorporated into existing workflows of OSS projects and if incorporated, would they be able to yield any actual benefit? One way in which many of the formal RE practices could be incorporated in web-based OSS development environments is in the form of features within those environments. One advantage of having such features within OSS development environments is that they will not obstruct the freedom and flexibility of OSS developers - developers can use such features if they wish to, but are not required to do so. Investigating the design of such features and how the OSS characteristics (e.g., scope, size) could influence their usage is a potential area for future research. For example, many of the requirements modeling practices were perceived as beneficial for OSS development and the practice “model the software architecture” was rated highest on the usefulness scale. Kazman et al. (2016) found that the OSS community engages in sufficient architectural discussion to support architectural thinking. Thus, a feature supporting architectural modeling (e.g., UML diagrams such as component diagrams and deployment diagrams that support architectural modeling (see Maksimchuk and Naiburg, 2004)) could be a potentially useful enhancement for web based OSS development environments as well as a feature that could potentially have a higher likelihood of usage by the OSS community. The survey identified several problems and challenges that could occur during requirements discovery in OSS development. Future research could analyze effective solutions for mitigating these problems, either in the form of some features or tools or some other ways.

The second phase of the research indicated some actual benefits from the availability of a library of reusable requirements (specifically requirements patterns) within OSS development environments. Also, technical experience and crowdsourcing experience were found to be important factors in the context of usage of the library of reusable requirements artifacts. Many possibilities of exploration for future research emerge. For example, whether and how, the requirements information existing in OSS issue repositories, discussion forums, and mailing lists could be analyzed for finding useful patterns. How could OSS projects plan for efficient usage of their reusable requirements artifacts by their existing contributors and interested potential contributors based on their experience and knowledge?

The third phase of the research provided many potential design suggestions for OSS issue reporting interfaces for tackling misclassification problem. Each of the design ideas that emerged could be empirically evaluated in future research. For example, one suggestion that emerged was that a simple issue reporting interface (like that of GitHub) would facilitate much greater participation of non-developer users while mitigating misclassification of requirements information. This could be empirically evaluated to see if that is the case, in the context of some real-world OSS project.

6.4 Implications for Practice

Requirements discovery is a challenging aspect of OSS development. Recent complaints by practitioners (developers of many OSS projects on GitHub) to the GitHub management provides evidence for this[5], with incomplete and missing information as major problems with OSS issue data (a major source of requirements for OSS projects). Well-structured external aids, such as a library of reusable requirements, can potentially

[5] https://github.com/dear-github/dear-github

assist existing and potential OSS contributors during requirements generation for improving the quantity and completeness of their requirements information. Efforts could be made to improve the reusability of informal requirements information that gets generated during the evolution of OSS projects (e.g., messages in discussion forums). For example, they could be made more structured, and better search facilities could be provided.

The adoption of OSS by organizations is increasing (e.g., Spinellis and Giannikas, 2012), but there are several challenges including integration problems (lack of ability to integrate with existing software systems), lack of technical and business knowledge to implement, customize and use OSS software, lack of OSS products with needed functionalities, questionable quality of existing OSS software, and hidden costs (e.g., time consuming to evaluate existing OSS products, need for user training) (Nagy et al., 2010; Hauge et al., 2010). Improving the quality of requirements generated during the evolution of OSS projects can potentially be of help while trying to address some of the above challenges. For example, if some of the requirements generated could match the functionalities needed by organizations that are potential future adopters, this could help reduce instances of missing functionalities. There is often need to customize OSS software to fit them into the organizational context, but often, there is a challenge of the lack of necessary knowledge within the organization to do so (Nagy et al., 2010; Hauge et al., 2010). Taking steps to ensure that some of the requirements that get generated during the evolution of OSS projects match with the needs of organizations that are potential adopters (e.g., organizational employees could submit feature requests in OSS issue repositories) can potentially help reduce customization and integration challenges and associated costs. For example, if the requirements submitted by organizational employees gets incorporated into the OSS software, this can potentially mean a lower burden on the organization in trying to make the OSS product usable in the organizational context, and lower costs arising from attempting to modify OSS software to fit the organizational needs, etc. In fact, Purcell (2015) found some evidence that the involvement of organizations in OSS development can be an efficient way to widen the representation of the interests of non-developer users during OSS development. This is important, given the valid concern that OSS development is largely represented by interests of developers (Purcell, 2015). The preceding discussion indicates the practical usefulness of incorporating external aids, such as a library of reusable requirements, that can potentially assist in generating more meaningful and complete requirements. It also indicates the usefulness of incorporating some of the formal requirements engineering practices into OSS development since some of these practices can potentially help improve the quality of OSS requirements.

Findings described in chapter five provide insights into improving the usability of OSS requirements gathering interfaces. Enforcing classification at the time of construction and submission of requirements can potentially reduce ease of use for contributors and deter participation, especially of individuals from a non-technical background. Such individuals could be employees of organizations (potential adopters) wanting to communicate their needs to the OSS project development team. Insights described in chapter five can shed light on the design of OSS requirements gathering interfaces that are more usable from the contributors' point of view.

Recently, there have been reports of usage of agile (development method targeting shorter development cycle to support complex, fast moving, competitive markets) requirements engineering practices in the industry (e.g., Ramesh et al., 2010). Ramesh et al. identified six agile requirements engineering practices, namely face to face communication over written specifications, iterative requirements engineering, greater importance to requirements prioritization, management of requirements change through constant planning, prototyping, and usage of review meetings and acceptance tests (Ramesh et al., 2010). As can be seen, there is overlap between agile RE practices identified by Ramesh et al. and the RE practices listed by Somerville and Swayer, including prototyping, requirements prioritization, requirements management through planning, review meetings and testing. These practices were perceived as beneficial for OSS development in the initial phase of this research. The idea of agility or flexibility, characteristic of the agile methodology, fits well with the flexibility that is characteristic of OSS development. Hence, OSS practitioners can potentially find some of these agile RE practices easy to incorporate in to their workflows. For example, in prototyping, a piece of software containing some functionalities (the prototype) is created by programmers and used as a way to communicate with users and to validate and refine requirements (Ramesh et al., 2010). This practice can align well with the implementation oriented focus of OSS developers and hence, potentially easy to incorporate in to their developmental activities. Another agile RE practice is testing in which tests are written (often for requirements validation, can be part of requirements specification too) to check if a piece of code is behaving as expected (Ramesh et al., 2010). OSS developers can potentially find this practice easy to incorporate in to their workflows since it is directly associated with coding.

REFERENCES

Agerfalk, P, J., Fitzgerald, B., Holmstrom, H., Lings, B., Lundell, B., Conchuir, E, O. (2005). "A framework for considering opportunities and threats in distributed software development". *International workshop on distributed software engineering*

Aggarwal, K., Hindle, A., Stroulia, E. (2014). "Co evolution of project documentation and popularity within GitHub". *Proceedings of the 11th working conference on mining software repositories*. Pp.360-363

Aksulu, A., Wade, M. (2010). "A comprehensive review and synthesis of open source research". *Journal of the association for information systems*, 11, pp.576 – 656

Alspaugh, T, A., Scacchi, W. (2013). "Ongoing software development without classical requirements" *In Proc. Of IEEE RE* pp. 165-174

Angst, C, M., Agarwal, R. (2009). "Adoption of electronic health records in the presence of privacy concerns: the elaboration likelihood model and individual persuasion". *MIS Quarterly,* 33(2), pp.339-370

Antonellis, D, V., Vandoni, L. (1993). "Temporal aspects in reuse of requirements specification". *Conference on advanced information systems engineering*. Berlin.

Anvik, J., Hiew, L., Murphy, G, C. (2005). "Coping with an open bug library". *Proceedings of the 2005 OOPSLA workshop on Eclipse technology*. pp. 35-39

Appan, R., Browne, G, J. (2010). "Investigating retrieval induced forgetting during information requirements determination". *Journal of the association for information systems*. 11(5). Pp.250-275

Arguello, J., Butler, B, S., Joyce, L., Kraut, R, E., Ling, K, S., Rosé, C, P., Wang, X. (2006). "Talk to Me: Foundations for Successful Individual-Group Interactions in Online Communities," *Proceedings of the 2006 ACM Conference on Human Factors in Computing Systems*, pp. 959-968.

Arthur, J, D., Groner, M, K. (2005). "An operational model for structuring the requirements generation process", *Requirements engineering*, 10, pp.45-62

Auld G, W., Diker, A., Bock, A., Boushey, C, J., Bruhn, C, M., Cluskey, M. (2007). "Development of a decision tree to determine appropriateness of NVivo in analyzing qualitative data sets". *Journal of Nutrition Education and Behaviour,* 39(1), pp.37- 47

Ayala, C., Franch, X., Conradi, R., Li, J., Cruzes, D. (2013). "Developing software with open source software components". *Finding source code on the web for remix and reuse*, pp.167-186

Badreddin, O., Lethbridge, T, C., Elassar, M. (2013). Modeling practices in open source software development. In *Open source software: Quality verification*, E. Petrinja et al. (Eds.) Springer, Berlin, 127-139

Baker, J., Parasuraman, A., Grewal, D., Voss, G, B. (2002). "The influence of multiple store environment cues on perceived merchandise value and patronage intentions". *Journal of marketing*. 66(2). Pp.120-141

Bano, M., Zowghi, D. (2013). "User involvement in software development and system success: a systematic literature review". *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, pp.125 – 130

Baytiyeh, H., Pfaffman, J. (2010). "Open source software: a community of altruists". *Computers in human behavior,* 26, pp.1345 - 1354

Begel, A., Bosch, J., Storey, M, A. (2013). "Social networking meets software development: Perspectives from GitHub, MSDN, Stack Exchange and TopCoder". *IEEE Software,* 30(1), pp.52-66

Benbasat, I., Lim, J. (2000). "Information technology support for debiasing group judgments: an empirical evaluation". *Organizational behavior and human decision processes,* 83(1), pp.167-183

Bettenburg, N., Just, S., Schroter, A., Weiss, C., Premraj, R., Zimmermann, T. (2008). "What makes a good bug report". *SIGSOFT*. Atlanta, Georgia, USA.

Blischak, J, D., Davenport, E, R., Wilson, G, A. (2016). "A quick introduction to version control with Git and GitHub". *PLoS Computational Biology,* 12(1), pp.1-18

Borgatti, S, P., Mehra, A., Brass, D., Labianca, G. (2009). "Network Analysis in the social sciences". *Science,* 323(5916), pp.892-895

Bradley, E, H., Curry, L, A., Devers, K, J. (2007). "Qualitative data analysis for health services research: Developing taxonomy, themes and theory". *Health services research,* 42(4), pp.1758-1772

Breaux, T, D., Hibshi, H., Rao, A., Lehker, J, M. (2012). "Towards a framework for pattern experimentation: Understanding empirical validity in requirements engineering patterns". *IEEE second international workshop on requirements patterns,* pp.41-47

Breu, S., Premraj, R., Sillito, J., Zimmermann, T. (2010). "Information needs in bug reports: improving cooperation between developers and users". *CSCW*. Savannah, Georgia, USA, pp.301-310

Browne, G, J., Ramesh, V. (2002). "Improving Information Requirements Determination: A Cognitive Perspective." *Information & Management,* 39, pp. 625-645.

Browne, G. J., Rogich, M. B. (2001). "An empirical investigation of user requirements elicitation: Comparing the effectiveness of prompting techniques". *Journal of Management Information Systems,* 17(4), 223

Bruce, B., Ritchie, J. (1997). "Nurses' practices and perceptions of family centered care". *Journal of pediatric nursing*. 12(4): pp.214-222.

Burton-Jones, A., Meso, P. (2008). "The effects of decomposition quality and multiple forms of information on Novices understanding of a domain from a conceptual model". *Journal of the association for Information Systems*, 9(12), pp.748-802

Cabot, J., Izquierdo, J, L, C., Cosentino, V., Rolandi, B., AtlanMod team. (2015). “Exploring the use of labels to categorize issues in open source software projects”. *SANER*.

Canfora, G., Cerulo, L. (2005). “Impact analysis by mining software and change request repositories”. *Software metrics, IEEE international symposium*.

Cavalcanti, Y, C., Silveira Neto, P, A., Almeida, E, S., Lucretius, D., Cunha, C, E, A., Meira, S, R. (2010). “One step more to understand the bug report duplication problem”. *Brazilian symposium on software engineering, IEEE*. Pp.148-157

Cavalcanti, Y, C., Silveira Neto, P, A., Machado, I, C., Vale, T, F., Almeida, E, S., Lemos Meira, S, R. (2014). “Challenges and opportunities for software change request repositories: a systematic mapping study”. *Journal of Software: evolution and process*, 26(7), pp.620-653

Chakraborty, S., Sarker, S., Sarker, S. (2010). “An exploration into the process of requirements elicitation: a grounded approach”. *Journal of the association for Information systems*. 11(4), pp.212-249

Chatzoglou P.D., Macaulay, L.A., 1997, “The Importance of Human Factors in Planning the Requirements Capture Stage of a Project”. *International Journal of Project Management*, 15 (1), pp. 39-53

Chengalur-Smith, I, N., Sidorova, A., Daniel, S. (2010). “Sustainability of open source projects: A longitudinal study”. *Journal of the association for Information Systems*. 11(11).

Chernak, Y. (2012). “Requirements reuse: the state of the practice”. *IEEE international conference on software science, technology and engineering*. Herzlia.

Choi, N., Chengalur-Smith, I. (2009). “An exploratory study on the two new trends in open source software: end-users and service”. *Hawaii International conference on system sciences*

Chu, R, K, S., Choi, T. (2000). “An importance performance analysis of hotel selection factors in the Hong Kong hotel industry: a comparison of business and leisure travellers”. *Tourism management*, 21(4): pp.363-377.

Ciolkowski, M., Soto, M. (2008). “Towards a comprehensive approach for assessing open source projects”. *Proceedings of Software Process and Product Measurement*, pp.316-330

Cleland-Huang, J., Dumitru, H., Duan, C., Castro-Herrera, C. (2009). “Automated support for managing feature requests in open forums”. *Communications of the ACM*. 52(10). Pp.68-74

Coleman, E, B. (1992). “Facilitating conceptual understanding in science: a collaborative explanation based approach”. *PhD dissertation*, University of Toronto

Condori-Fernandez, N., Pastor, O., Daneva, M., Abran, A., Castro, J. (2008). “Quantifying reuse from object oriented requirements specifications”. *Workshop on requirements engineering*.

Connolly, T., Routhieaux, R, L., Schneider, S, K. (1993). "On the effectiveness of group brainstorming: Test of one underlying cognitive mechanism". *Small group research*. 24(4). Pp.490-503

Contractor, N, S., Wasserman, S., Faust, K. (2006). "Testing multi-theoretical multilevel hypothesis about organizational networks: an analytical framework and empirical example". *Academy of Management review*. 31(3), pp.681-703

Cox, K., Niazi, M., Verner, J. (2009). "Empirical Study of Sommerville and Sawyer's Requirements Engineering Practices". *IET Software*, 3(5): pp. 339–355

Crowston, K., Annabi, H., Howison, J. (2003). "Defining open source software project success". *Proceedings of the 24th International conference on information systems*

Crowston, K., Howison, J (2005). "The social structure of free and open source software development". *First Monday,* 10(2)

Crowston, K., Scozzi, B. (2004). "Coordination practices for bug fixing within FLOSS development teams". *6th International conference on enterprise information systems*. Porto, Portugal.

Crowston, K., Wei, K., Howison, J., Wiggins, A. (2012). "Free/Libre open source software development: what we know and what we do not know". *ACM computing surveys*. 44(2), pp.1-35

Curley, S, P., Browne, G, J., Smith, G, F., Benson, P, G. (1995). "Arguments in the practical reasoning underlying constructed probability responses". *Journal of Behavioural decision making,* 8(1), pp.1- 20

Dabbish, L., Stuart, C., Tsay, J., Herbsleb. J. (2012). "Social coding in GitHub: transparency and collaboration in an open software repository". In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pp.1277–1286

Damian, D, E., Zowghi, D. (2003). "RE challenges in multi-site software development organizations". *Requirements engineering*. 8, pp.149-160

Daneva, M. (2000). "Establishing reuse measurement practices in SAP requirements engineering". *International conference on requirements engineering, IEEE*. Los Alamitos, CA.

Darke, P., Shanks, G. (1996). "Stakeholder viewpoints in requirements definition: a framework for understanding viewpoint development approaches". *Requirements engineering*, 1(2), pp.88-105

Davidson, J, L., Mohan, N., Jensen, C. (2011). "Coping with duplicate bug reports in free/open source software projects". *IEEE symposium on visual languages and human centric computing*. Pittsburg. Pp.101-108

Davis, F, D. (1989). "Perceived usefulness, perceived ease of use and user acceptance of information technology". *MIS Quarterly,* 13(3), pp.319-340

Davis, G, B. (1982). "Strategies for information requirements determination". *IBM Systems Journal*. 21(1). pp. 4-30

Dennis, A, R., Valacich, J, S., Connolly, T., Wynne, B, E. (1996). "Process structuring in electronic brainstorming". *Information systems research*. 7(2). Pp.268-278

Dennis, A., Wixom, B, A., Roth, R, A. (2012). "System analysis and design". John Wiley and sons, Inc. USA

Detienne, F. (1990). "Expert programming knowledge: a schema based approach". In J, M, Hoc, T, R, G, Green, R, Samurcay, D, Gilmore. (Eds.). *Psychology of programming, academic press, people and computer series*. Pp.205-222

Detienne, F. (1995). "Design strategies and knowledge in object oriented programming: effects of experience". *Human computer interaction*, 10(2), pp.129-169

Detienne, F., Burkhardt, J, M., Barcellini, F. (2006). "Open source software communities: current issues". *PPIG Newsletter*, pp.1-7

Diamant, E, I., Daniel, S. (2010). "Learning in open source software development: how organizational and national culture impact developers' learning". *Thirty first international conference on information systems*. St. Louis

Diehl, M., Stroebe, W. (1987). "Productivity loss in brain storming groups: towards the solution of a riddle". *Journal of personality and social psychology*. 53(3). Pp.497-509

Dietze. (2005). "Collaborative requirements definition processes in Open Source Software Development". *Requirements engineering for sociotechnical systems*.

Dixon, L, A. (1997). "An anchoring and adjustment strategy for re-design". *PhD thesis*. Georgia institute of technology

Doan, A., Ramakrishnan, R., Halevy, A, Y. (2011). "Crowdsourcing systems on the world wide web", *Communications of the ACM*, 54(4), pp.86-96

Dou, W., Lim, K, H., Su, C., Zhou, N., Cui, N. (2010). "Brand positioning strategy using search engine marketing". *MIS Quarterly*, 34(2), pp.261-279

Enns, H, G., Huff, S, L., Golden, B, R. (2003). "CIO influence behaviors: the impact of technical background". *Information and management*, 40(5), pp.467-485

Ernst, N., Murphy, G. (2012) Case studies in just-in-time requirements analysis. *IEEE second International Workshop on Empirical Requirements Engineering*. Chicago

Evermann, J., Wand, Y. (2005). "Ontology based Object-oriented Domain Modelling: Fundamental Concepts." *Requirements Engineering*, 10(2), pp.146-160

Falessi, D., Kidwell, B., Hayes, J, F., Shull, F. (2014). "On failure classification: The impact of "getting it wrong". *ICSE companion*

Feller, J., Fitzgerald, B. (2000). "A framework analysis of the open source development paradigm" *In Proc. of the 21st International Conference on Information Systems*
Fernandez, D, M., Penzenstadler, B. (2015). "Artefact based requirements engineering: the AMDiRE approach". *Requirements Engineering*, 20, pp.405-434

Fernandez, D, M., Wieringa, R. (2013). "Improving requirements engineering by artefact orientation". *14th international conference on product focused software development and process improvement*

Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L., Goedicke, M. (1992). “Viewpoints: a framework for integrating multiple perspectives in system development”. *International Journal of Software Engineering and Knowledge Engineering,* 2(1), pp.31-58

Firesmith, D. (2004). “Prioritizing requirements”. *Journal of Object Technology,* 3(8)

Fitzgerald, B. (2006). “The transformation of open source software”. *MIS Quarterly,* 30(3), pp.587-598

Fitzgerald, C., Letier, E., Finkelstein, A. (2011). “Early failure prediction in feature request management systems”. *IEEE 19th international requirements engineering conference*

Foushee, B, D. (2013). “Prevalence of reflexivity and its impact on success in open source software development”. *MS Thesis*. Brigham Young University

Fraser, H, S, F., Biondich, P., Moodley, D., Sharon, C., Mamlin, B, W., Szolovits, P. (2005). “Implementing electronic medical record systems in developing countries”. *Informatics in primary care*. 13(2), pp.83-96

Fritz, M, S., Mackinnon, D, P. (2007). “Required sample size to detect the mediated effect”. *Psychological science,* 18(3), pp.233-239

G. Kulk, G, P., Verhoef, C. (2008). “Quantifying requirements volatility effects”. *Science of computer programming*. 72. pp. 136-175

G. Robles., J. M. Gonzalez-Barahona, (2012). “A comprehensive study of software forks: Dates, reasons and outcomes,” in ´ *Open Source Systems: Long-Term Sustainability*. Springer, pp. 1–14

Gettys, C, F., Pliske, R, M., Manning, C., Casey, J, F. (1987). “An evaluation of human act generation performance”. *Organizational behavior and human decision processes,* 39, pp.23-51

Godfrey, M, W., Tu, Q. (2000). “Evolution in open source software: a case study”. *IEEE international conference on software maintenance*. Pp.131-142

Guo, P, J., Zimmermann, T., Nagappan, N., Murphy, B. (2010). “Characterizing and predicting which bugs get fixed: an empirical study of Microsoft windows”. *ICSE*. Cape Town, South Africa.

Guzzi, A., Bacchelli, A., Lanza, M., Pinzger, M., Van Deursen, A. (2013). “Communication in open source software development mailing lists”. *10th IEEE working conference on mining software repositories*. San Francisco. Pp.277-289

Hars, A., Ou, S. (2002). “Working for free? Motivations for participating in open source projects”. *International journal of electronic commerce*. 6(3), pp.25-39

Hauge, O., Cruzes, D, S., Conradi, R., Velle, K, S., Skarpenes, T, A. (2010). “Risks and risk mitigation in open source software adoption: Bridging the gap between literature and practice”. *Proceedings of 6th international IFIP WG 2.13 conference on open source systems, open source software: new horizons*.

Hauge, Q., Sorensen, C, F., Conradi, R. (2008). “Adoption of open source in the software industry”. *International conference on open source systems,* pp.211-222

Hawkins, D. (2014). “Biomeasurement: a student’s guide to biostatistics”. Oxford

Heck, P., Zaidman, A. (2013). “An analysis of requirements evolution in open source projects: recommendations for issue trackers”. *Proceedings of the 2013 International workshop on principles of software evolution*. NY, USA.

Herzig, K., Just, S., Zeller, A. (2013). “It’s not a bug, it’s a feature: how misclassification impacts bug prediction”. *Proceedings of the international conference on software engineering*. USA. Pp.392-401

Hevner, A, R., March, S, T., Park, J., Ram, S. (2004). “Design science in information systems research”. *MIS Quarterly,* 28(1), pp.75-105

Heppler, L., Eckert, R., Stuermer, M. (2016). “Who cares about my Feature Request”. *Open Source Systems: Integrating Communities: 12th IFIP WG 2.13 International Conference,* pp.85 - 96

Hoepfl, M, C. (1997). “Choosing qualitative research: A primer for technology education researchers”. *Journal of technology education,* 9(1), pp.47-63

Hoffmann, A., Janzen, A., Hoffmann, H., Leimeister, J. M. (2013) “Success Factors for Requirement Patterns Approaches - Exploring Requirements Analysts' Opinions and Wishes”. *Sozio-technisches Systemdesign im Zeitalter des Ubiquitous Computing (SUBICO).* Germany.

Hofmann, H, F., Lehner, F. (2001). “Requirements engineering as a success factor in software projects”. *IEEE Software,* 18(4), pp.58-66

Iannacci, F. (2005). “Beyond Markets and Firms: The Emergence of Open Source Networks”, *First Monday,* 10(5).

Janesick, V. (2000). “The choreography of qualitative research: Minuets, improvisations and crystallizations”. In *Handbook of qualitative research,* Denzin, N., Lincoln, Y, S (Eds.), pp. 379 – 399. Sage publications

Jensen, C., King, S., Kuechler, V. (2011). “Joining free/open source software communities: an analysis of newbies’ first interactions on project mailing lists”. *Hawaii International conference on system sciences*

Jiang, Y., Adams, B., German, D, M. (2013). “Will My Patch Make It? And How Fast? Case Study on the Linux Kernel,” *Proceedings of the tenth working conference on mining software repositories,* pp. 101–110.

Joode, V, D., Wendel, R. (2004). “Managing conflicts in open source communities”. *Electronic Markets*. 14(2), pp.104-113

Joyce, E., Kraut, R, E. (2006). “Predicting continued participation in news groups*”.* *Journal of computer-mediated communication,* 11, pp.723-747

Kabaale, E., Nabukenya, J. (2011). "A systematic approach to requirements engineering process improvement in small and medium enterprises: an exploratory study". *Product focused software process improvement*, pp.262-275

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D, M., Damian, D. (2014). "The promises and perils of mining GitHub". *Proceedings of the 11th working conference on mining software repositories.* NY, USA, pp. 92-101

Kamata, M, I., Tamai, T. (2007). "How does requirements quality relate to project success or failure", *15th IEEE International Requirements engineering conference*. Delhi, pp.69-78

Kang, Y., Harring, J, R. (under review). "Reexamining the impact of non-normality in two group comparison procedures". *Journal of experimental education*.

Katsamakas, E., Georgantzas, N, C. (2007). "Open source software development: a system dynamics model". *Conference of the system dynamics*

Kazman, R., Goldenson, D., Monarch, I., Nichols, W., Valetto, G. (2016). "Evaluating the effects of architectural documentation: A case study of a large scale open source project". *IEEE Transactions on Software Engineering*, 42(3), pp.222 - 247

Khondu, J., Capiluppi, A., Stol, K. (2013). "Is it all lost: a study of inactive open source projects", *9th International Conference on open source systems*.

Ko, A.J., Chilana, P. (2010). "How Power Users Help and Hinder Open Bug Reporting". *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI*), New York, pp. 1665-1674

Kochhar, P, S., Le, B, T., Lo, D. (2014). "It's not a bug, it's a feature. Does misclassification affect bug localization". *Mining software repositories*.

Krishnamurthy, S. (2002). "Cave or community? An empirical examination of 100 mature open source projects". *First Monday*. 7(6)

Kuk, G. (2006). "Strategic interaction and knowledge sharing in the KDE developer mailing list". *Management science*, 52, pp.1031-1042

Kuriakose, J., Parsons, J. (2015) [a]. How do open source software (OSS) developers practice and perceive requirements engineering: an empirical study. *5th IEEE international workshop on empirical requirements engineering*.

Kuriakose, J., Parsons, J. (2015) [b]. An enhanced requirements gathering interface for open source software development environments. *IEEE RE*.

Lam, W., McDermid, J, A., Vickers, A, J. (1997). "Ten steps towards systematic requirements reuse". *Requirements engineering*, 2(2), pp.102-113

Lamsweerde, A, V. (2000). "Requirements engineering in the year 00: A research perspective". *International Conference on software engineering*, pp.5-19

Laurent, P., Cleland-Huang, J. (2009). "Lessons learned from open source projects for facilitating online requirements processes". *15th International working conference REFSQ*, Amsterdam

Lee, S, T., Kim, H., Gupta, S. (2009). "Measuring open source software success". *Omega*, 37, Pp.426-438.

Li, Z., Wang, Z, Yang, Y., Wu, Y., Liu, Y. (2007). "Towards a multiple ontology framework for requirements elicitation and reuse". *International computer software and applications conference, IEEE*. Los Alamitos, CA.

Lintula, H., Koponen, T., Hotti, V. (2006). "Exploring the Maintenance Process through the Defect Management in the Open Source Projects -Four Case Studies". *IEEE International conference on software engineering advances*.

Liu. B. (2012). "Sentiment analysis and opinion mining". Morgan and Claypool.

Llanos, J, W, C., Castillo, S, T, A. (2012). Differences between traditional and open source development activities. In *Product-Focused Software Process Improvement*, O Dieste et al. (Eds.) Springer, Berlin, 131-144.

Lopez, T, R., Garrido, J, L., Supakkul, S., Chung, L. (2013). "A pattern approach to dealing with NFRs in ubiquitous systems". *CEUR workshop proceedings*, 998, pp.25-32

Lukyanenko, R., Parsons, J. (2015). "Principles for modeling user generated content". *Conceptual modeling*, pp.432-440

Lukyanenko, R., Parsons, J., Wiersma, Y, F. (2014). "The impact of conceptual modeling on dataset completeness: a field experiment". *Twenty fifth international conference on Information Systems*.

Lukyanenko, R., Parsons, J., Wiersma, Y, F. (2014)^{b}. "The IQ of the crowd: Understanding and improving information quality in structured user generated content". *Information systems research*, 25(4), Pp.669-689

Lumley, T., Diehr, P., Emerson, S., Chen, L. (2002). "The importance of the normality assumption in large public health data sets". *Annual review public health*. 23, pp.151-169

Maiden, N, A, M., Sutcliffe, A, G. (1991). "Reuse of analogous specifications during requirements analysis". *Proceedings of the sixth international workshop on software specification and design*. Pp.220-223

Mamlin, B, W., Biondich, P., Wolfe, B, A., Fraser, H, S, F., Jazayeri, D., Allen, C., Miranda, J., Tierney, W, M. (2006). "Cooking up an open source EMR for developing countries: openMRS- a recipe for successful collaboration". *AMIA annual symposium proceedings*. Pp.529-533

March, S, T., Smith, G, F. (1995). "Design and natural science research on information technology". *Decision support systems*, 15, pp.251-266

Mason, W., Suri, S. (2012). "Conducting behavioral research on Amazon's mechanical Turk". *Behavioral research*, 44, pp.1-23

Massey, B. (2002). "Where do open source requirements come from (And what we should do about it)", *Second ICSE workshop on open source software engineering*. Orlando, Florida.

Maksimchuk, R., Naiburg, E. (2004). "UML for mere mortals". Addison-Wesley

McLeod, P, L., Lobel, S, A. (1992). “The effects of ethnic diversity on idea generation in small groups”. *Academy of management proceedings*. Pp.227-231

McLeod, S, A. (2012). “Experimental method”. Retrieved from http://www.simplypsychology.org/experimental-method.html

Michlmayr, M. (2005). ”Software Process Maturity and the Success of Free Software Projects” In: Zieliński, K., Szmuc, T. (Eds.), *Software Engineering: Evolution and Emerging Technologies.* pp. 3–14

Miller, A, R., Tucker, C. (2009). “Privacy protection and technology diffusion: The case of electronic medical records”. *Management science*, 55(7), pp.1077-1093

Mockus, A., Fielding, R, T., Herbsleb, J, D. (2002). “Two case studies of open source software development: Apache and Mozilla”. *ACM Transactions on software engineering and methodology*. 11(3). Pp.309-346

Monge, P, R., Contractor, N, S. (2003). “Theories of communication networks”. Oxford University Press. New York

Morse, J, M. (1994). “Designing funded qualitative research”. In *Handbook of qualitative research,* Denzin, N., Lincoln, Y, S (Eds.), pp. 220 – 235, Sage publications

Morse, J, M., Richards, L. (2002). “Read me first for a user’s guide to qualitative methods”. Sage publications

Nagasundaram, M., Dennis, A, R. (1993). “When a group is not a group: the cognitive foundation of group idea generation”. *Small group research*. 24(4). Pp.463-489

Nagy, D., Yassin, A, M., Bhattacherjee, A. (2010). “Organizational adoption of open source software: Barriers and remedies”. *Communications of the ACM*, 53(3), pp.148-151

Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., Ye, Y. (2002). “Evolution patterns of open source software systems and communities”. *Proceedings of the international workshop on principles of software evolution*. Pp.76-85

Nevo, D., Benbasat, I., Wand, Y. (2012). “The knowledge demand of expertise seekers in two different contexts: knowledge allocation versus knowledge retrieval”. *Decision support systems*. 53, pp.482-489

Nichols, D., Twidale, M. (2003). “The usability of open source software”. *First Monday*. 8(1)

Nikula, U., Jantunen, S. (2005). “Quantifying the interest in open source systems: Case South-East Finland”. *International conference on open source systems*, pp.192-195

Noll, J., Liu, W. (2010). “Requirements elicitation in open source software development: a case study”. *Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, Cape town

Noll, John. (2008). “Requirements acquisition in open source development: Firefox2.0”. In *Open source development, communities and quality.* B Russo et al. (Eds.), Springer, Boston, 275, 69-79

Offen, R. (2002). “Domain understanding is the key to successful system development”. *Requirements engineering,* 7(3), pp.172-175

Palomares, C., Franch, X., Quer, C. (2014). “Requirements reuse and patterns: a survey”. *Requirements engineering: foundation for software quality,* pp.301-308

Parsons, J., Saunders, C. (2004). “Cognitive heuristics in software engineering: Applying and extending anchoring and adjustment to artifact reuse”. *IEEE Transactions on software engineering*. 30(12). Pp.873-888

Parsons, J., Wand Y. (2008). “Using cognitive principles to guide classification in information systems modeling”. *MIS Quarterly,* pp.839-868

Parsons, J., Wand, Y. (2014). “A foundation for open information environments”. *Proceedings of the European Conference on Information Systems*. Tel Aviv

Passos, L., Czarnecki, K. (2014). “A dataset of feature additions and feature removals from the Linux Kernel”. *11th working conference on mining software repositories*. pp. 376-379

Pacheco, C., Garcia, I. (2012). “A systematic literature review of stakeholder identification methods in requirements elicitation”. *The Journal of Systems and Software,* 85, pp.2171 - 2181

Perens, B. (1999). “The open source definition”. In: *Open sources: voices from the open source revolution,* Sebastopol, O’Reilly and Associates, pp.171 - 188

Pitts, M, G., Browne, G, J. (2007). “Improving Requirements Elicitation: An Empirical Investigation of Procedural Prompts.” *Information Systems Journal,* 17, pp. 89-110

Prikladniciki, R., Audy, J, L. N., Evaristo, R. (2003). “Global software development in practice lessons learned”. *Software process, improvement and practice,* 8(4), pp.267-281

Purcell, M, W. (2015). “On the role of FOSS business models and participation architectures in supporting open innovation”. *11th International symposium on Open Collaboration,* San Francisco, USA

Rantalainen, A., Hedberg, H., Iivari, N. (2011). A review of tool support for user-related communication in FLOSS development. In *Open source systems: grounding research,* S A Hissam et al. (Eds.) Springer, Berlin, 90-105

Ramesh, B., Cao, L., Baskerville, R. (2010). “Agile requirements engineering practices and challenges: an empirical study”. *Information Systems Journal,* 20(5), pp.449 - 480

Ratiu, D. (2009). “Reverse engineering domain models from source code”. *International workshop on reverse engineering models from software artifacts*. , pp.13-16

Raza, A., Capretz, L, F., Ahmed, F. (2012). “Users’ perception of open source usability: an empirical study”. *Engineering with computers,* 28, pp.109-121

Reips, U. (2002). “Theory and techniques of web experimenting”. In B, Batinic et al. (Eds.), *online social sciences*. Seattle, Hogrefe and Huber.

Renault, S., Mendez-Bonilla, O., Franch, X., Quer, C. (2009). "A pattern based method for building requirements documents in call-for-tender processes". *International journal of computer science and applications*. 6(5). pp. 175-202

Rietzschel, E, F., Nijstad, B, A., Stroebe, W. (2007). "Relative accessibility of domain knowledge and creativity: the effects of knowledge activation on the quantity and originality of generated ideas". *Journal of experimental social psychology*. 43(6). Pp.933-946

Robbins, J, E. (2003). "Adopting open source software engineering practices (OSSE) practices by adopting OSSE tools". In J. Feller., B. Fitzgerald., S. Hissam., K. Lakham (Eds.), *Making Sense of the Bazaar: Perspectives on Open Source and Free Software*. Sebastopol, CA. O'Reilly & Associates

Robillard, P, N. (1999). "The role of knowledge in software development". *Communications of the ACM*. 42(1). Pp.87-92

Rousseau, D, M. (2001). "Schema, promise and mutuality: the building blocks of the psychological contract". *Journal of occupational and organizational psychology*. 74. Pp.511-541

Rutherford, A. (2012). "ANOVA and ANCOVA: A GLM approach", 2nd edition. Wiley

Sawilowsky, S, S., Blair, R, C. (1992). "A more realistic look at the robustness and type 2 error properties of the t-test to departures from population normality". *Psychological bulletin*. 111(2). Pp.352-360

Scacchi, W. (2002). Understanding the requirements for developing open source software systems. *IEEE Software,* 149(1), pp.24–39.

Scacchi, W. (2004). "Free and open source development practices in the game community". *IEEE software,* 21(1), pp.59-66

Scacchi, W. (2007). "Free/Open source software development: Recent research results and emerging opportunities". *Proceeding ESEC-FSE Companion '07*. Pp.459-468

Scacchi, W. (2009). "Understanding requirements for open source software". In K. Lyytinen et al. (Eds.) *Design Requirements workshop. LNBIP*. Pp.467-494. Springer

Scacchi. W. 2006. "Understanding Free/Open Source Software Evolution," in *Software Evolution and Feedback: Theory and Practice*. N. H. Madhavji, J. F. Ramil and D. Perry (Eds.), John Wiley and Sons Inc., New York, pp.181-206

Schackmann, H., Lichter, H. (2009). "Evaluating process quality in GNOME based on based on change request data". *6th IEEE international working conference on mining software repositories*. Pp.95-98

Schmid, K. (2014). "Challenges and solutions in global requirements engineering- a literature survey". *Software quality, Model based approaches for advanced software and systems engineering,* pp.85-99

Schmider, E., Ziegler, M., Danay, E., Beyer, L., Buhner, M. (2010). "Is it really robust? Reinvestigating the robustness of ANOVA against violations of the normal distribution assumption". *Methodology: European journal of research methods for the behavioral and social sciences*. 6(4). Pp.147-151

Schoop, M., Jertilla, A., List, T. (2003). "Negoist: a negotiation support system for electronic business-to-business negotiations in e-commerce". *Data & Knowledge Engineering*, 47(3), pp.371-402

Shah, S, K. (2006). "Motivation, governance and the viability of hybrid forms in open source software development". *Management science*. 52(7). Pp.1000-1014

Sharif, K, Y., English, M., Ali, N., Exton, C., Collins, J, J., Buckley, J. (2015). "An empirically-based characterization and quantification of information seeking through mailing lists during Open Source developers' software evolution". *Information and Software Technology*, 57, pp.77 - 94

Siau, K., Long, Y. (2009). "Factors impacting E-Government development". *Journal of computer information systems*. 50(1), Pp.98-107

Simon, H, A. (1996). "The sciences of the artificial". 3rd ed.

Singh, P, V., Fan, M., Tan, Y. (2007). "An empirical investigation of code contribution, communication participation and release strategy in open source software development: a conditional hazard model approach". Retrieved from http://ifipwg213.org/system/files/singh_fan_tan.pdf

Singh, P, V., Tan, Y., Mookerjee, V. (2011). "Network effects: the influence of structural social capital on open source project success". *MIS Quarterly*, 35(4), pp.813-829

Singh, V., Twidale, M, B., Nichols, D, M. (2009). "Users of open source software-how do they get help". *Hawaii International conference on system sciences*

Sohn, S, Y., Mok, M, S. (2008). "A strategic analysis for successful open source software utilization based on a structural equation model". *Journal of systems and software*. 81(6), Pp.1014-1024

Sojer, M., Henkel, J. (2010). "Code reuse in open source software development: quantitative evidence, drivers and impediments". *Journal of the association for information systems*. 11(12). Pp.868-901

Sommerville, I., Sawyer, P. (1997). "Requirements engineering: a good practice guide". Wiley.

Sommerville, I., Sawyer, P. (1997b). "Viewpoints: principles, problems and a practical approach to requirements engineering". *Annals of software engineering*, 3(1), pp.101-130

Spinellis, D., Giannikas, V. (2012). "Organizational adoption of open source software"., *Journal of systems and software*, 85(3), pp.666-682

Spiro, R, J., Tirre, W, C. (1980). "Individual differences in schema utilization during discourse processing". *Journal of educational psychology*. 72(2), pp.204-208

Standish Group. (1995). "CHAOS report". Retrieved from https://www4.in.tum.de/lehre/vorlesungen/sw/SS2004/files/1995_Standish_Chaos.pdf

Standish Group. (1999). “CHAOS report”. Retrieved from https://www4.informatik.tu-muenchen.de/lehre/vorlesungen/vse/WS2004/1999_Standish_Chaos.pdf

Steinmacher, I., Conte, T, U., Gerosa, M, A., Redmiles, D, F. (2015). “Social barriers faced by newcomers placing their first contribution in open source software projects”. *Proceedings of the 18th ACM conference on Computer supported cooperative work and social computing*, pp.1-13

Steinmacher, I., Conte, T., Gerosa, M, A., Redmiles, D., Wiese, I, S. (2014). “The hard life of open source software project newcomers”. *International workshop on cooperative and human aspects of software engineering*.

Stewart, K, J., Ammeter, A, P., Maruping, L, M. (2006). “Impact of license choice and organizational sponsorship on user interest and development activity in open source software projects”. *Information systems research*. 17(2). Pp.126-144

Stull, D, E. (2008). “Analyzing growth and change: latent variable growth curve modeling with an application to clinical trials”. *Qual Life Res*, 17, pp.47-59

Subramaniam, C., Sen, R., Nelson, M, L. (2009). “Determinants of open source software project success: A longitudinal study”. *Decision support systems*. 46(2). Pp.576-585

Teeni, D. (2001). “A cognitive-affective model of organizational communication for designing IT”. *MIS Quarterly*. 25(2). Pp.251-312

Terry, M., Kay, M., Lafreniere, B. (2010). “Perceptions and practices of usability in the Free/Open source software community”. *CHI 2010*, Atlanta, USA.

Vaishnavi, V., Kuechler, W. (2015). “Design science research in information systems”. Retrieved from http://desrist.org/desrist/content/design-science-research-in-information-systems.pdf

Verner, J., Cox, K., Bleistein, S., Cerpa, N. (2005). “Requirements engineering and software project success: an industrial survey in Australia and US”. *Australasian journal of Information systems*. 13(1).

Vitalari, N, P. (1985). “Knowledge as a basis for expertise in systems analysis: an empirical study”. *MIS Quarterly*, 9(3), pp.221-241

Vlas, R, E., Robinson, W, N. (2012). Two rule based natural language strategies for requirements discovery and classification in open source software development projects. *Journal of management information systems*. 28(4), pp.11-38

Vlas, R. (2012). “A requirements based exploration of open source software development projects: Towards a natural language processing software analysis framework”. *PhD dissertation*. Georgia State University

Vlas, R., Robinson, W, N. (2013). “Applying a rule based natural language classifier to open source requirements: a demonstration of theory exploration”. *Hawaii International conference on system sciences*.

Vlas, R., Robinson, W. (2011). “A Rule-Based Natural Language Technique for Requirements Discovery and Classification in Open-Source Software Development Projects”. *Proceedings of the 44th Hawaii International Conference on Systems Science*

Vlas, R., Vlas, C. (2011). A requirements based analysis of success in open source software development projects. *Proceedings AMCIS,* Detroit

Von Krogh, G., Spaeth, S., Haefliger, S. (2005). “Knowledge reuse in open source software: an exploratory study of 15 open source projects”. *Proceedings of the 38th annual Hawaii international conference on system sciences*. Big Island, HI, USA

Walls, J, G., Widmeyer, G, R., Sawy, O, A, E. (1992). “Building an Information system design theory for vigilant EIS”. *Information systems research*. 3(1). Pp.36-59

Wang, D., Wang, Q., Yang, Y., Li, Q., Wang, H., Yuan, F. (2011). “Is it really a defect? An empirical study on measuring and improving the process of software defect reporting”. *International symposium on empirical software engineering and measurement, IEEE*. Pp.434-443

West, S.G., Finch, J.F., Curran, P.J. (1995). “Structural equation models with non-normal variables. Problems and remedies”. In R.H. Hoyle (Ed.). *Structural equation modeling: Concepts, issues and applications* (pp. 56-75). Newbury Park, CA: Sage.

Winkler, S., Pilgrim, J. (2010). “A survey of traceability in requirements engineering and model driven development”. *Software and systems modeling*. 9(4), pp.529-565

Withall, S. (2007). “Software Requirements Patterns”. Microsoft Press. Washington

Wixom, B, H., Todd, P, A. (2005). “A theoretical integration of user satisfaction and technology acceptance”. *Information systems research*. 16(1). Pp.85-102

Franch, X., Guerlain, C., Palomares, C., Quer, V., Renault, S (2011). "Interested in Improving Your Requirements Engineering Process? Try Requirement Patterns!" *Empirical Fair Track at REFSQ*.

Ye, Y., Kishida, K. (2003). “Towards an understanding of the motivation of open source software developers”. *International conference on software engineering*

Zahran, S. (2000). “Software process improvement: practical guidelines for business success”. Addison-Wesley. New York

Zhao, L., Elbaum, S. (2003). “Software quality assurance under the open source model”. *Journal of systems and software*. 66(1), pp.65-75

Zielczynski, P. (2007). “Requirements management using IBM Rational RequisitePro”. IBM Press

Zimmermann, T., Premraj, R., Bettenburg, N., Just, S., Schroter, A., Weiss, C. (2010). “What makes a good bug report”. *IEEE transactions on software engineering*. 36(5). Pp.618-643

APPENDICES

APPENDIX 1: Survey questionnaire (requirements engineering practices in OSS development)

Based on your experience, please indicate whether/how the requirements engineering practices listed in this page were used in the development of open source software project(s) you have worked on.

If any of the listed practice is unclear, please place the cursor on the * symbol next to it. A pop up will come up providing detailed explanation about the practice.

1) Please indicate whether/how the following requirements documentation practices were used in the development of open source software project(s) that you have worked on *

	always used	**mostly used**	**sometimes used**	**rarely used**	**never used**	**not applicable**	**I do not know**
define a standard document structure *	( )	( )	( )	( )	( )	( )	( )
explain how to use the document *	( )	( )	( )	( )	( )	( )	( )
include a summary of the requirements *	( )	( )	( )	( )	( )	( )	( )
Make a business case for the software*	( )	( )	( )	( )	( )	( )	( )
define specialized terms *	( )	( )	( )	( )	( )	( )	( )

make document layout readable*	( )	( )	( )	( )	( )	( )	( )
help readers find information *	( )	( )	( )	( )	( )	( )	( )
make the document easy to change*	( )	( )	( )	( )	( )	( )	( )

Comments:

2) Please indicate whether/how the following requirements elicitation practices were used in the development of open source software project(s) that you have worked on *

	always used	**mostly used**	**sometimes used**	**rarely used**	**never used**	**not applicable**	**I do not know**
assess software feasibility *	( )	( )	( )	( )	( )	( )	( )
be sensitive to political and organizational consideration *	( )	( )	( )	( )	( )	( )	( )
identify and consult	( )	( )	( )	( )	( )	( )	( )

software users *							
record requirement sources *	( )	( )	( )	( )	( )	( )	( )
define the software's operating environment *	( )	( )	( )	( )	( )	( )	( )
use business concerns to drive requirement elicitation *	( )	( )	( )	( )	( )	( )	( )
look for domain constraints *	( )	( )	( )	( )	( )	( )	( )
record requirements rationale *	( )	( )	( )	( )	( )	( )	( )
collect requirements from multiple viewpoints *	( )	( )	( )	( )	( )	( )	( )
prototype poorly understood requirements *	( )	( )	( )	( )	( )	( )	( )
use scenarios to elicit requirements *	( )	( )	( )	( )	( )	( )	( )

define operational processes *	( )	( )	( )	( )	( )	( )	( )
reuse requirements *	( )	( )	( )	( )	( )	( )	( )

Comments:

3) Please indicate whether/how the following requirements analysis and negotiation practices were used in the development of open source software project(s) that you have worked on *

	alway s used	**mostl y used**	**sometim es used**	**rarel y used**	**neve r used**	**not applicab le**	**I do not kno w**
Define software boundaries *	( )	( )	( )	( )	( )	( )	( )
Use checklists for requirements analysis *	( )	( )	( )	( )	( )	( )	( )
Provide software to support negotiations *	( )	( )	( )	( )	( )	( )	( )
Plan for conflict and conflict resolution *	( )	( )	( )	( )	( )	( )	( )

Prioritize requirements*	( )	( )	( )	( )	( )	( )	( )
Classify requirements using a multidimensional approach *	( )	( )	( )	( )	( )	( )	( )
Use interaction matrices to find conflicts and overlaps *	( )	( )	( )	( )	( )	( )	( )
Assess requirements risk *	( )	( )	( )	( )	( )	( )	( )

Comments:

4) Please indicate whether/how the following requirements describing practices were used in the development of open source software project(s) that you have worked on *

	always used	**mostly used**	**sometimes used**	**rarely used**	**never used**	**not applicable**	**I do not know**
Define standard templates for defining requirements *	( )	( )	( )	( )	( )	( )	( )
Use languages simply and concisely*	( )	( )	( )	( )	( )	( )	( )

Use diagrams appropriately *	( )	( )	( )	( )	( )	( )	( )
Supplement natural language with descriptions of requirement *	( )	( )	( )	( )	( )	( )	( )
specify requirements quantitatively *	( )	( )	( )	( )	( )	( )	( )

Comments:

5) Please indicate whether/how the following requirements modeling practices were used in the development of open source software project(s) that you have worked on *

	always used	**mostly used**	**sometimes used**	**rarely used**	**never used**	**not applicable**	**i do not know**
Develop complementary models of the proposed software *	( )	( )	( )	( )	( )	( )	( )
Model the software's environment *	( )	( )	( )	( )	( )	( )	( )

Model the software's architecture *	( )	( )	( )	( )	( )	( )	( )
Use structured methods for software modeling *	( )	( )	( )	( )	( )	( )	( )
Use a data dictionary *	( )	( )	( )	( )	( )	( )	( )
Document the links between user requirements and models *	( )	( )	( )	( )	( )	( )	( )

Comments:

6) Please indicate whether/how the following requirements validation practices were used in the development of open source software project(s) that you have worked on *

	alway s used	**mostl y used**	**sometim es used**	**rarel y used**	**neve r used**	**not applicabl e**	**I do not kno w**
Check that the requirements document meets your standards*	( )	( )	( )	( )	( )	( )	( )
Organize formal requirements inspection *	( )	( )	( )	( )	( )	( )	( )

Use multidisciplinary teams to review requirements *	( )	( )	( )	( )	( )	( )	( )
Define validation checklists*	( )	( )	( )	( )	( )	( )	( )
Use prototyping to animate requirements *	( )	( )	( )	( )	( )	( )	( )
Write a draft user manual*	( )	( )	( )	( )	( )	( )	( )
Propose requirements test cases *	( )	( )	( )	( )	( )	( )	( )
Paraphrase models *	( )	( )	( )	( )	( )	( )	( )

Comments:

7) Please indicate whether/how the following requirements management practices were used in the development of open source software project(s) that you have worked on *

	always used	**mostly used**	**sometimes used**	**rarely used**	**never used**	**not applicable**	**I do not know**
Uniquely identify each requirement*	( )	( )	( )	( )	( )	( )	( )

Define policies for requirements management *	( )	( )	( )	( )	( )	( )	( )
Define traceability policies *	( )	( )	( )	( )	( )	( )	( )
Maintain a traceability manual *	( )	( )	( )	( )	( )	( )	( )
Use a database to manage requirements *	( )	( )	( )	( )	( )	( )	( )
Define change management policies *	( )	( )	( )	( )	( )	( )	( )
Identify global software requirements *	( )	( )	( )	( )	( )	( )	( )
Identify volatile requirements *	( )	( )	( )	( )	( )	( )	( )
Record rejected requirements *	( )	( )	( )	( )	( )	( )	( )

Comments:

8) Based on your experience, please indicate whether the following activities were used for generating requirements during the development of the open source software project(s) that you have worked on

	always used	**mostly used**	**sometimes used**	**rarely used**	**never used**	**not applicable**	**I do not know**
requirements are asserted by an open source software developer based on his or her personal experience	( )	( )	( )	( )	( )	( )	( )
requirements are asserted by an open source software developer based on his or her personal knowledge of user needs	( )	( )	( )	( )	( )	( )	( )
requirements are contributed by users through bug reports *	( )	( )	( )	( )	( )	( )	( )
requirements are contributed	( )	( )	( )	( )	( )	( )	( )

by users through feature requests *							
requirements are derived from features found in some other software	( )	( )	( )	( )	( )	( )	( )

Comments:

For each of the requirements engineering practice listed on this page, based on your experience, please indicate how useful do you think, adopting each practice in the development of open source software projects would be.

If any of the listed practice is unclear, please place the cursor on the * symbol next to it.

9) Please indicate what you think about the usefulness of adopting the following requirements documentation practices in the development of open source software projects would be *

	extremely useful	**very useful**	**useful**	**not useful**	**harmful**	**not applicable**	**I do not know**
define a standard document structure *	( )	( )	( )	( )	( )	( )	( )
explain how to use the document *	( )	( )	( )	( )	( )	( )	( )
include a summary of the	( )	( )	( )	( )	( )	( )	( )

requirements *							
Make a business case for the software*	( )	( )	( )	( )	( )	( )	( )
define specialized terms *	( )	( )	( )	( )	( )	( )	( )
make document layout readable*	( )	( )	( )	( )	( )	( )	( )
help readers find information *	( )	( )	( )	( )	( )	( )	( )
make the document easy to change*	( )	( )	( )	( )	( )	( )	( )

Comments:

10) Please indicate what you think about the usefulness of adopting the following requirements elicitation practices in the development of open source software projects would be*

	extremely useful	**very useful**	**useful**	**not useful**	**harmful**	**not applicable**	**I do not know**
assess software feasibility *	( )	( )	( )	( )	( )	( )	( )

be sensitive to political and organizational consideration *	( )	( )	( )	( )	( )	( )	( )
identify and consult software users *	( )	( )	( )	( )	( )	( )	( )
record requirement sources *	( )	( )	( )	( )	( )	( )	( )
define the software's operating environment *	( )	( )	( )	( )	( )	( )	( )
use business concerns to drive requirement elicitation *	( )	( )	( )	( )	( )	( )	( )
look for domain constraints *	( )	( )	( )	( )	( )	( )	( )
record requirements rationale *	( )	( )	( )	( )	( )	( )	( )
collect requirements from multiple viewpoints *	( )	( )	( )	( )	( )	( )	( )

prototype poorly understood requirements *	( )	( )	( )	( )	( )	( )	( )
use scenarios to elicit requirements *	( )	( )	( )	( )	( )	( )	( )
define operational processes *	( )	( )	( )	( )	( )	( )	( )
reuse requirements *	( )	( )	( )	( )	( )	( )	( )

Comments:

11) Please indicate what you think about the usefulness of adopting the following requirements analysis and negotiation practices in the development of open source software projects would be*

	extremely useful	**very useful**	**useful**	**not useful**	**harmful**	**not applicable**	**I do not know**
Define software boundaries *	( )	( )	( )	( )	( )	( )	( )
Use checklists for requirements analysis *	( )	( )	( )	( )	( )	( )	( )

Provide software to support negotiations *	( )	( )	( )	( )	( )	( )	( )
Plan for conflict and conflict resolution *	( )	( )	( )	( )	( )	( )	( )
Prioritize requirements*	( )	( )	( )	( )	( )	( )	( )
Classify requirements using a multidimension al approach *	( )	( )	( )	( )	( )	( )	( )
Use interaction matrices to find conflicts and overlaps *	( )	( )	( )	( )	( )	( )	( )
Assess requirements risk *	( )	( )	( )	( )	( )	( )	( )

Comments:

12) Please indicate what you think about the usefulness of adopting the following requirements describing practices in the development of open source software projects would be *

	extremel y useful	**very usefu l**	**usefu l**	**not usefu l**	**harmfu l**	**not applicabl e**	**I do not kno w**

Define standard templates for defining requirements *	( )	( )	( )	( )	( )	( )	( )
Use languages simply and concisely*	( )	( )	( )	( )	( )	( )	( )
Use diagrams appropriately *	( )	( )	( )	( )	( )	( )	( )
Supplement natural language with descriptions of requirement *	( )	( )	( )	( )	( )	( )	( )
specify requirements quantitatively *	( )	( )	( )	( )	( )	( )	( )

Comments:

13) Please indicate what you think about the usefulness of adopting the following requirements modeling practices in the development of open source software projects would be*

	extremely useful	**very useful**	**useful**	**not useful**	**harmful**	**not applicable**	**I do not**

							kno w
Develop complementary models of the proposed software *	( )	( )	( )	( )	( )	( )	( )
Model the software's environment *	( )	( )	( )	( )	( )	( )	( )
Model the software's architecture *	( )	( )	( )	( )	( )	( )	( )
Use structured methods for software modeling *	( )	( )	( )	( )	( )	( )	( )
Use a data dictionary *	( )	( )	( )	( )	( )	( )	( )
Document the links between user requirements and models *	( )	( )	( )	( )	( )	( )	( )

Comments:

14) Please indicate what you think about the usefulness of adopting the following requirements validation practices in the development of open source software projects would be*

	extremely useful	**very useful**	**useful**	**not useful**	**harmful**	**not applicable**	**I do not know**
Check that the requirements document meets your standards*	( )	( )	( )	( )	( )	( )	( )
Organize formal requirements inspection *	( )	( )	( )	( )	( )	( )	( )
Use multidisciplinary teams to review requirements *	( )	( )	( )	( )	( )	( )	( )
Define validation checklists*	( )	( )	( )	( )	( )	( )	( )
Use prototyping to animate requirements *	( )	( )	( )	( )	( )	( )	( )
Write a draft user manual*	( )	( )	( )	( )	( )	( )	( )
Propose requirements test cases *	( )	( )	( )	( )	( )	( )	( )
Paraphrase models *	( )	( )	( )	( )	( )	( )	( )

Comments:

15) Please indicate what you think about the usefulness of adopting the following requirements management practices in the development of open source software projects would be*

	extremely useful	**very useful**	**useful**	**not useful**	**harmful**	**not applicable**	**I do not know**
Uniquely identify each requirement*	( )	( )	( )	( )	( )	( )	( )
Define policies for requirements management *	( )	( )	( )	( )	( )	( )	( )
Define traceability policies *	( )	( )	( )	( )	( )	( )	( )
Maintain a traceability manual *	( )	( )	( )	( )	( )	( )	( )
Use a database to manage requirements *	( )	( )	( )	( )	( )	( )	( )
Define change management policies *	( )	( )	( )	( )	( )	( )	( )
Identify global	( )	( )	( )	( )	( )	( )	( )

software requirements *							
Identify volatile requirements *	( )	( )	( )	( )	( )	( )	( )
Record rejected requirements *	( )	( )	( )	( )	( )	( )	( )

Comments:

16) Please indicate which of the following challenges and problems have you faced or expect to face during requirements discovery in open source software development? (Please check all that apply)

[ ] language barriers (for example, members with English as second language may have difficulty expressing, clarifying or understanding requirements in English)

[ ] members from different language and cultural backgrounds may demand different types of user interfaces

[ ] differences in national culture (requirements may be influenced by cultural beliefs, for example, members from some countries may prefer stability and prefer requirements from prior releases whereas members from some other countries may prefer continuous progress and hence new features; members from different cultural backgrounds may have different expectations about functionality, behavior and design)

[ ] differences in corporate culture (different members may be from different corporate environments that have different system usage characteristics)

[ ] differences in educational backgrounds (for example, low skilled members and high skilled members may have different expectations about how a system works)

[ ] requirements being expressed using diverse terminologies and diverse levels of details by members, making it difficult to analyze the requirements for discovering conflicts and redundancies

[ ] geographical distance and time difference between countries as barriers in interaction between members

[ ] diminished understanding of the working context of other members (members do not have enough familiarity with/knowledge of activities of remote group members and other background information thus leading to diminished understanding of the work context of remote members; this in turn leads to unwanted outcomes such as members in one region not being able to have adequate understanding of the requirements of the members in other region)

[ ] Different members may use different standards

[ ] difficult to achieve cohesion/form coalitions with other members

[ ] difficulty in trusting others

[ ] difficulty in managing conflict and having open discussions of interests (difficulty in managing conflicting interests of members during development because geographical distance makes it difficult to openly discuss about interests of members)

[ ] difficulty in achieving a common understanding of requirements

[ ] ineffective decision making meetings (e.g., ineffective meetings because of use of poor communication technologies; sometimes it is not even possible to know who have joined the meeting late)

[ ] delay (e.g., delayed email response)

[ ] Different laws and regulations in different countries

[ ] It is difficult to identify and include relevant members in the communication process for gathering requirements

[ ] Difficulty in negotiating requirements and prioritizing requirements

[ ] other: ____

Comments:

17) Which of the following issues have you faced while trying to use the information provided by users through bug reports and feature requests?

[ ] duplicate information *

[ ] invalid information *

[ ] incomplete or missing information

[ ] the bug could not be fixed or the requested feature could not be implemented

[ ] Other; please specify: ____

Comments:

18) Please indicate the age group that you belong to

	1-19	20-29	30-39	40-49	50+
Age	( )	( )	( )	( )	( )

19) Please indicate the continent you reside in

	North America	South America	Europe	Asia and rest of the world
Residence	( )	( )	( )	( )

20) Please indicate your highest education level

	Non-university education	Undergraduate or equivalent	Graduate or equivalent	PhD and higher
Highest education level	( )	( )	( )	( )

21) Please indicate your role in the open source software projects that you have worked on

	includes writing code	does not include writing code
Task profile in open source software projects	( )	( )

22) Please indicate the number of open source software projects that you have worked on

	1-4	5-9	10-14	15+

Number of open source software projects ever involved in	( )	( )	( )	( )

23) Please indicate the approximate amount of hours that you spent per week working on open source software projects

	1-4	**5-9**	**10-19**	**20+**
Hours spent working on open source software projects per week	( )	( )	( )	( )

APPENDIX 2: Holm-Bonferroni Correction

p-values from smallest to largest	Position in sequence	Adjusted alpha	Significant or not?
2.2154e-10	1	0.05/57=0.000877	significant
2.0674e-09	2	0.05/56=0.000893	significant
2.1781e-09	3	0.000909	significant
2.6085e-09	4	0.000926	significant
5.5024e-09	5	0.000943	significant
7.4062e-09	6	0.000962	significant
1.0159e-08	7	0.00098	significant
1.3709e-08	8	0.001	significant
1.5702e-08	9	0.00102	significant
2.7425e-08	10	0.001042	significant
3.2451e-08	11	0.001064	significant
4.0662e-08	12	0.001087	significant
7.3122e-08	13	0.001111	significant
1.2994e-07	14	0.001136	significant
2.2122e-07	15	0.001163	significant
2.3611e-07	16	0.00119	significant
4.0957e-07	17	0.00122	significant
4.6286e-07	18	0.00125	significant
4.7171e-07	19	0.001282	significant
8.6603e-07	20	0.001316	significant
1e-06	21	0.001351	significant
2e-06	22	0.001389	significant
2e-06	23	0.001429	significant
3e-06	24	0.001471	significant
7e-06	25	0.001515	significant

8e-06	26	0.001563	significant
1e-05	27	0.001613	significant
1.2e-05	28	0.001667	significant
1.3e-05	29	0.001724	significant
1.5e-05	30	0.001786	significant
1.7e-05	31	0.001852	significant
1.7e-05	32	0.001923	significant
2.6e-05	33	0.002	significant
2.6e-05	34	0.002083	significant
3.2e-05	35	0.002174	significant
7.3e-05	36	0.002273	significant
9.2e-05	37	0.002381	significant
0.000111	38	0.0025	significant
0.000127	39	0.002632	significant
0.000165	40	0.002778	significant
0.000256	41	0.002941	significant
0.000266	42	0.003125	significant
0.001	43	0.003333	significant
0.001	44	0.003571	significant
0.001	45	0.003846	significant
0.002	46	0.004167	significant
0.002	47	0.004545	significant
0.003	48	0.005	significant
0.005	49	0.005556	significant
0.005	50	0.00625	significant
0.005	51	0.007143	significant

0.005	52	0.008333	significant
0.042	53	0.01	no
0.066	54	0.0125	no
0.081	55	0.016667	no
0.491	56	0.025	no
0.63	57	0.05	no

APPENDIX 3: Experimental task description along with references [In the actual experiment, the references were omitted and only the description was provided to participants]

{An electronic medical record (EMR) system enables users to create, store and manage patient data electronically. It stores information such as lab tests, medication and diagnosis. It may have features such as allowing medical practitioners to analyze and update patient data and generating graphs of patient's lab tests.} [**http://www.google.com/patents/US5924074**]. There are both commercial electronic medical record software as well as open source electronic medical record software [**Webster, 2011**]. Open source electronic medical record software can be freely downloaded, used and modified. Some of the currently available open source electronic medical record software are listed below:

VistA (http://www.ehealth.va.gov/VistA.asp)

OSCAR (http://oscarcanada.org/)

OpenEMR (http://www.open-emr.org/)

OpenMRS (http://openmrs.org/)

FreeMED (http://freemedsoftware.org/)

The video below give an overview of an open source electronic medical record (specifically OSCAR EMR).

<Wiki: video url="http://www.youtube.com/watch?v=DGqDUcCNww0"/>

The short videos below further describe electronic medical records.

<Wiki: video url="https://www.youtube.com/watch?v=MOwML1N3TpM&index=1&list=TLYoJAfwXzPNG8ltGuqn6o6fm6gvdWBpWH"/>

<Wiki: video url="https://www.youtube.com/watch?v=TiQ8c11dkU0&index=2&list=TLYoJAfwXzPNG8ltGuqn6o6fm6gvdWBpWH"/>

The short video below describes how electronic medical record software may be used in real world settings.

<Wiki: video url="https://www.youtube.com/watch?v=97v5p9Nk2_I&list=PL48A96ACCA9486FC8&index=15"/>

{The currently existing electronic medical record software are difficult to interact with and inputting data into them is a difficult task. They are often incapable of providing suggestions supported with strong evidence and they are incapable of guiding practitioners about what to do next to help patients. They are incapable of identifying any important missing information. Often their design and implementation are not based on the latest medical knowledge available and they are often difficult to be updated. A major demand of users of electronic medical record software such as physicians is ease of use. For example, there should be minimal input requirements. Practitioners should be able to ask direct natural language questions to the software.} **[Ferrucci et al., 2013]**. {For achieving this, the software system must have a deep understanding of natural language and its nuances. They need to be able to analyze and make sense of very large quantities of different types of information sources. They should also be able to use contextual information to make inferences about the intent of the individual seeking the information. They should also be capable of providing evidence to support the information that they are providing to the information seeker.} **[Sudarsan, 2013]**

{IBM's Watson technology is a cognitive computing technology that can comprehend the subtle nuances of human language, navigate through vast amounts of textual content, and provide evidence-based answers to the questions of users. Watson technology processes a question given to it in a way that is similar to how humans would do it. It does an in depth analysis of the input question to determine what is being asked and then generate many possible candidate answers, by analyzing large volumes of available textual content (available in natural language form). Then Watson technology analyzes the available textual content to find evidence for supporting or refuting each answer. Watson technology has several reasoning algorithms embedded within it and by using these algorithms, it analyzes the available evidence for each answer along different dimensions such as time and geography. Watson technology finally produces a ranked list of candidate answers. For each candidate answer, Watson technology produces confidence scores indicating the degree of correctness of the answer and also displays links to supporting evidence that Watson technology has found.} [**Ferrucci et al., 2013; Sudarsan, 2013]**

{As an example, suppose that a user asks a question "In May 1898, Portugal celebrated the 400th anniversary of explorer A's arrival in India. Who is this explorer?" and a document available for analysis to the Watson technology has two paragraph, first "In May, Gary arrived in India after he celebrated his anniversary in Portugal" and second "On the 27th of May, 1498, Vasco da Gama landed in Kappad beach". A traditional keyword based search technology would return the incorrect answer Gary since there are many matching words in both paragraphs such as celebrated and anniversary. On the other hand, for the same question, Watson technology runs multiple algorithms to conduct a deeper analysis of both paragraphs. For example, Watson technology would run a temporal reasoning algorithm to match dates and would run a geospatial reasoning algorithm to determine that Kappad beach is in India and thus ultimately return the correct answer Vasco da Gama.}[https://www.youtube.com/watch?v=1z2FX7FfHR4]

{Watson technology is capable of reading raw human writing, which is unstructured data. For example, if a textual biography of a president is provided as input to Watson technology, it will subsequently break down the language in the text to parts and attempt to infer different information within that large body of textual content. Watson technology does not need the input information to be have a structured format to answer questions from the input biography data. Once Watson technology is trained to answer questions on one biography source, it can process other biographies effectively and answer questions. The ability of Watson technology is evident from the fact that in 2011, it was able to defeat two Jeopardy champions by using its natural language processing capabilities to process more than a million pages of stored unstructured textual content.} **[Sudarsan, 2013]**

The videos below describe Watson technology

<Wiki: video url="https://www.youtube.com/watch?v=DywO4zksfXw"/>

Below is a more descriptive video explaining the beginnings and underlying technology of IBM Watson.

<Wiki: video url="https://www.youtube.com/watch?v=3G2H3DZ8rNc"/>

The capabilities of Watson technology can empower real world applications used in different domains which can help users to perform tasks more efficiently. [**Sudarsan, 2013]**

The short video below describes perspectives on how Watson technology may be useful to health care domain.

<Wiki: video url="https://www.youtube.com/watch?v=vwDdyxj6S0U"/>

{Electronic medical record software application can also be powered by Watson technology. For example, Watson technology can utilize its ability to process and understand knowledge available in natural language to effectively process information available in electronic medical records. This can reduce the practitioners' burden associated with manually reading, synthesizing and making inferences from large amounts of information available in electronic medical records. As another example, Watson technology can analyze data in electronic medical record and generate diagnosis and treatment related recommendations along with supporting evidence. Medical practitioners often make cognitive errors in diagnosis. This is often because of erroneous synthesis or erroneous processing of information available to practitioners. Practitioners often make decisions quickly (e.g., quick diagnosis) and fail to consider plausible alternatives. A major challenge for medical practitioners is to process the large volume of information available in electronic medical records and the ever growing (and rapidly changing) medical knowledge (e.g., medical journals). Watson technology can assist medical practitioners in meeting these challenges through its different capabilities such as doing automatic extraction and presentation of relevant information from electronic medical records and providing a large variety of diagnosis related suggestions along with associated confidence and evidence.} **[Ferrucci et al., 2013]**

The short video below provides an illustration of an electronic medical record system integrated with Watson technology.

<Wiki: video url="http://www.youtube.com/watch?v=HZsPc0h_mtM"/>

{The Watson technology is available from IBM as a platform with all capabilities built into it. The open source electronic medical record software can be developed as an app that can be embedded with the Watson technology platform using application programming interfaces provided by IBM. }. [**Sudarsan, 2013]**

The figure below illustrates the idea of embedding an application with Watson technology.

http://i1376.photobucket.com/albums/ah10/surya1234/imagewatson_zps95704a50.jpg [**Sudarsan, 2013, p.5]**

{By embedding an application with Watson technology, the application gains cognitive capabilities. The developers have flexibility in the degree to which they embed Watson capabilities with the application and this can be decided based on domain specific needs. The application along with its cognitive capabilities could then be delivered to the end users through different channels such as mobile, tablet and desktop.} [**Sudarsan, 2013]**

An open source electronic medical record software powered by Watson technology would be highly beneficial for developing countries. {Developing countries have limited resources. There is need for better management of clinical data in developing countries and often there is need for rapid delivery of medical services while coping with limited technical resources. An open source electronic medical record software powered by Watson technology would be cheap and affordable to health care organizations in developing countries**.}[Fraser et al., 2005; Mamlin et al., 2006]**. {For example, VistA was the only EMR software that could be afforded by health care organizations in Mexico and it cost them 40 times less than what a commercial EMR software implementation would have cost.} [**Webster, 2011**]. {Open source EMR software powered by Watson technology can assist in better management of clinical and health data in developing countries (e.g., detecting errors) and also help practitioners make better decisions (e.g., more accurate diagnosis).} **[Fraser et al., 2005; Mamlin et al., 2006]**

The goal of this experimental task is to develop an open source electronic medical record software embedded with Watson technology. At this initial stage, the requirements (what the software must do; what characteristics it should have) **[Dennis et al., 2012]** for the proposed open source EMR software embedded with Watson technology is being collected.

By considering yourself as a potential user of the to-be developed electronic medical record software embedded with Watson technology, please think about the following:

{1. What are the business needs/organizational needs for the proposed software, if any?

2. What are the needs that the users of the proposed software may have?

3. What features/functionality should the proposed software have? (For example, it should display graphs of lab tests)

4. What characteristics/non-functional requirements does the proposed software needs to have? (For example, security requirements)

5. How should the proposed software be built? (For example, hardware and software that may be used in development)} **[Dennis et al., 2012]**

Please submit the requirements that you thought about for the proposed software in the textbox in crowdflower. Also, please submit any other information that you think is relevant for the development of the proposed software.

REFERENCES (Experimental task description)

Webster, P, C. (2011). "The rise of open source electronic health records". *The Lancet*. 377(9778). Pp.1641-1642

Ferrucci, D., Levas, A., Bagchi, S., Gondek, D., Mueller, E, T. (2013). "Watson: beyond jeopardy". *Artificial Intelligence*, 199, pp.93-105

Sudarsan, S. (2013). "An ecosystem of innovation: creating cognitive applications powered by Watson". *IBM Software white paper*

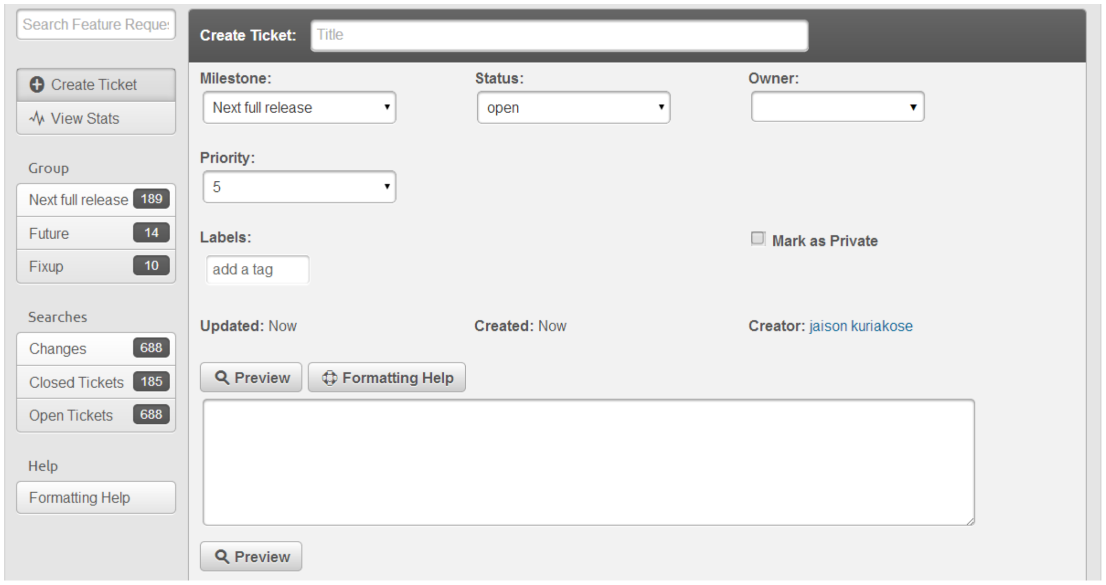
Fraser, H, S, F., Biondich, P., Moodley, D., Sharon, C., Mamlin, B, W., Szolovits, P. (2005). "Implementing electronic medical record systems in developing countries". *Informatics in primary care*. 13(2), pp.83-96

Mamlin, B, W., Biondich, P., Wolfe, B, A., Fraser, H, S, F., Jazayeri, D., Allen, C., Miranda, J., Tierney, W, M. (2006). "Cooking up an open source EMR for developing countries: openMRS- a recipe for successful collaboration". *AMIA annual symposium proceedings*. Pp.529-533

Dennis, A., Wixom, B., Roth, R. (2012). "Systems analysis and design". Wiley, USA

APPENDIX 4. Screenshots of Issue reporting interfaces in OSS development environments

Issue reporting interface used in Sourceforge (https://sourceforge.net)



Create Ticket: Title

Milestone: Next full release

Status: open

Owner:

Priority: 5

Labels: add a tag

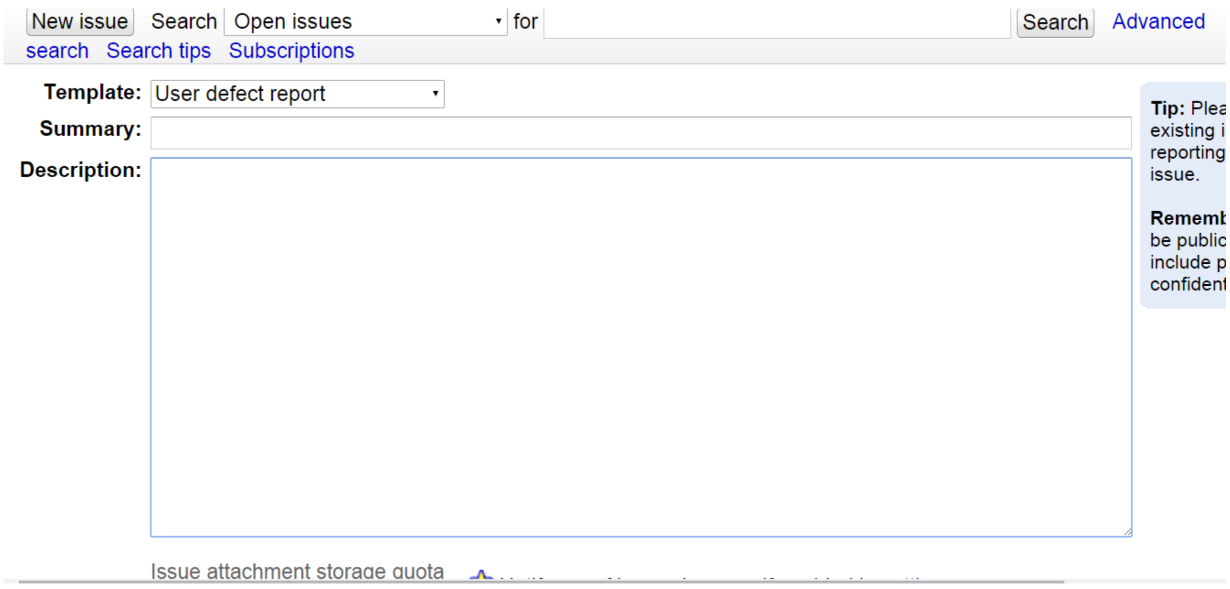
Mark as Private

Updated: Now

Created: Now

Creator: jaison kuriakose

Issue reporting interface used in Google Code (https://code.google.com)



New issue Search Open issues for Search Advanced search Search tips Subscriptions

Template: User defect report

Summary:

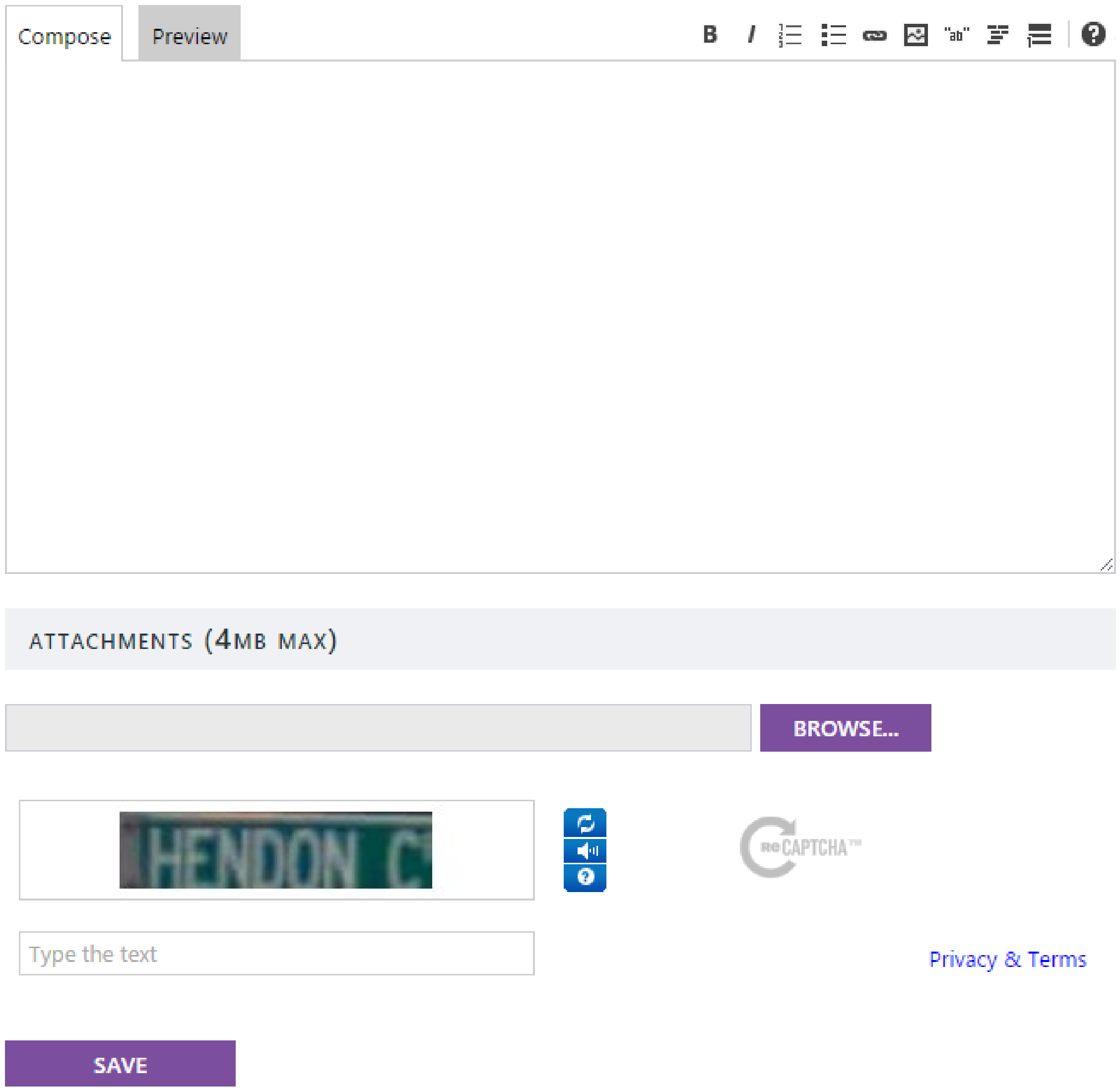
Description:

Tip: Plea existing i reporting issue.

Rememb be public include p confident

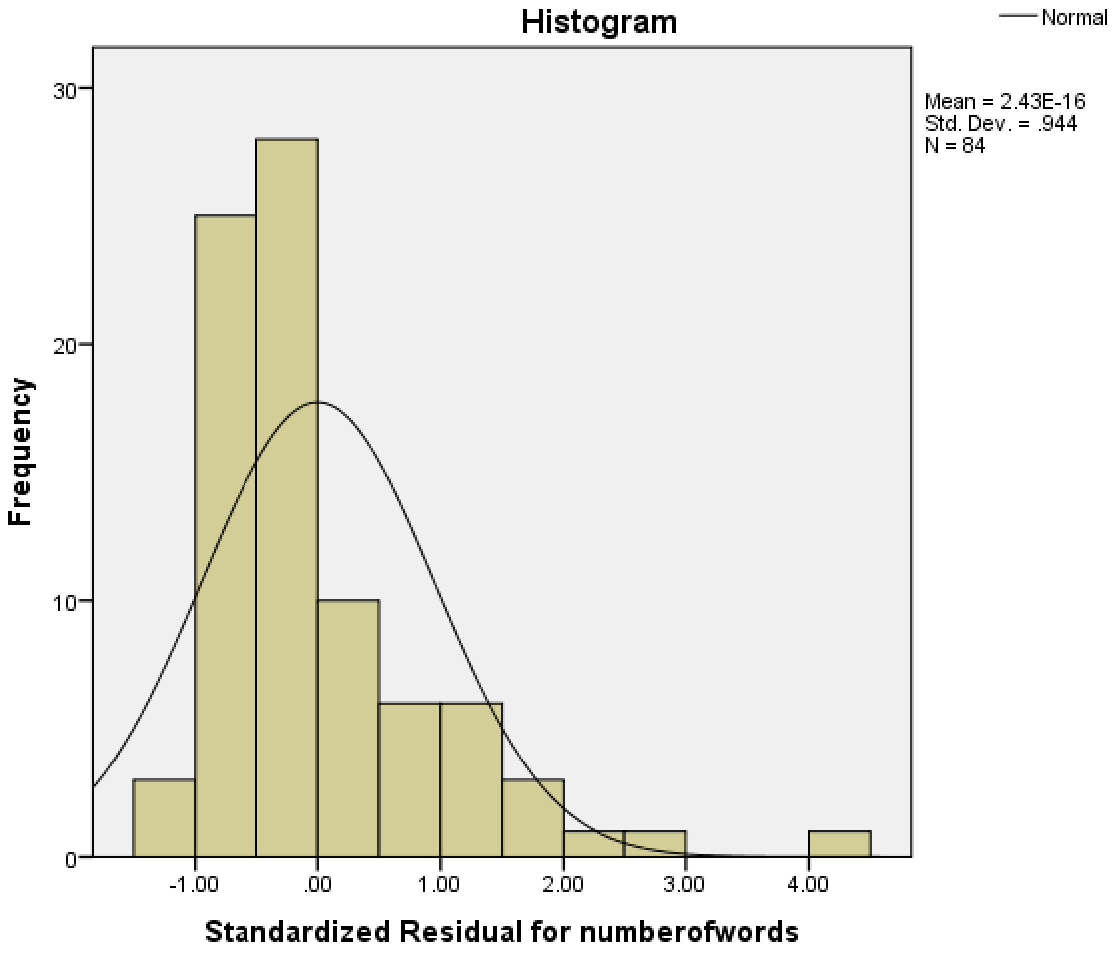
Issue attachment storage quota

Issue reporting interface used in CodePlex (https://www.codeplex.com/)



APPENDIX 5. Normality plot and Leven’s test

Normality plot of standardized residuals: (Dependent variable: Number of words)



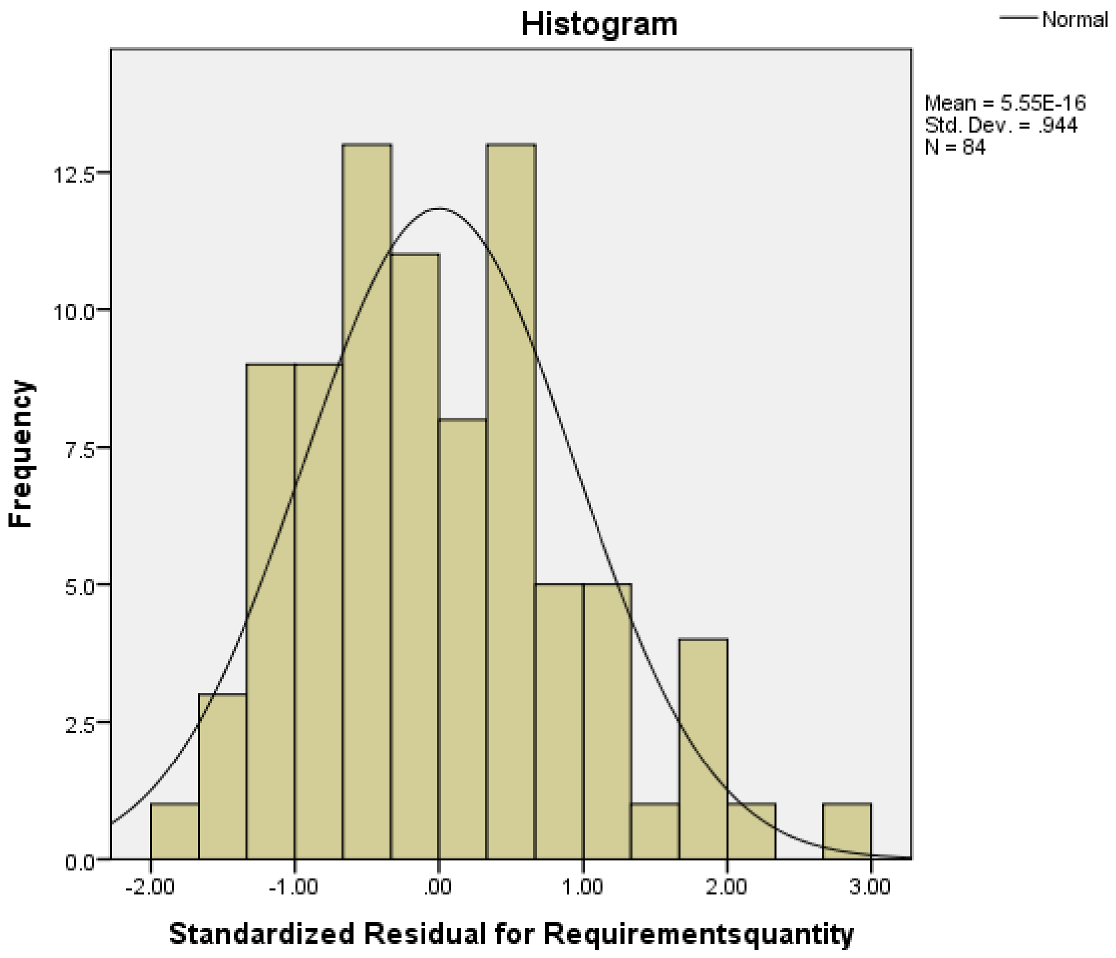
Skewness and Kurtosis (Dependent variable: number of words)

	Statistic	Standard error
Skewness	1.838	.263
Kurtosis	5.105	.520

Levene’s test of equality of error variances: (Dependent variable: number of words)

F	Df1	Df2	Sig.
3.667	1	82	.059

Normality plot of standardized residuals: (Dependent variable: requirements quantity)



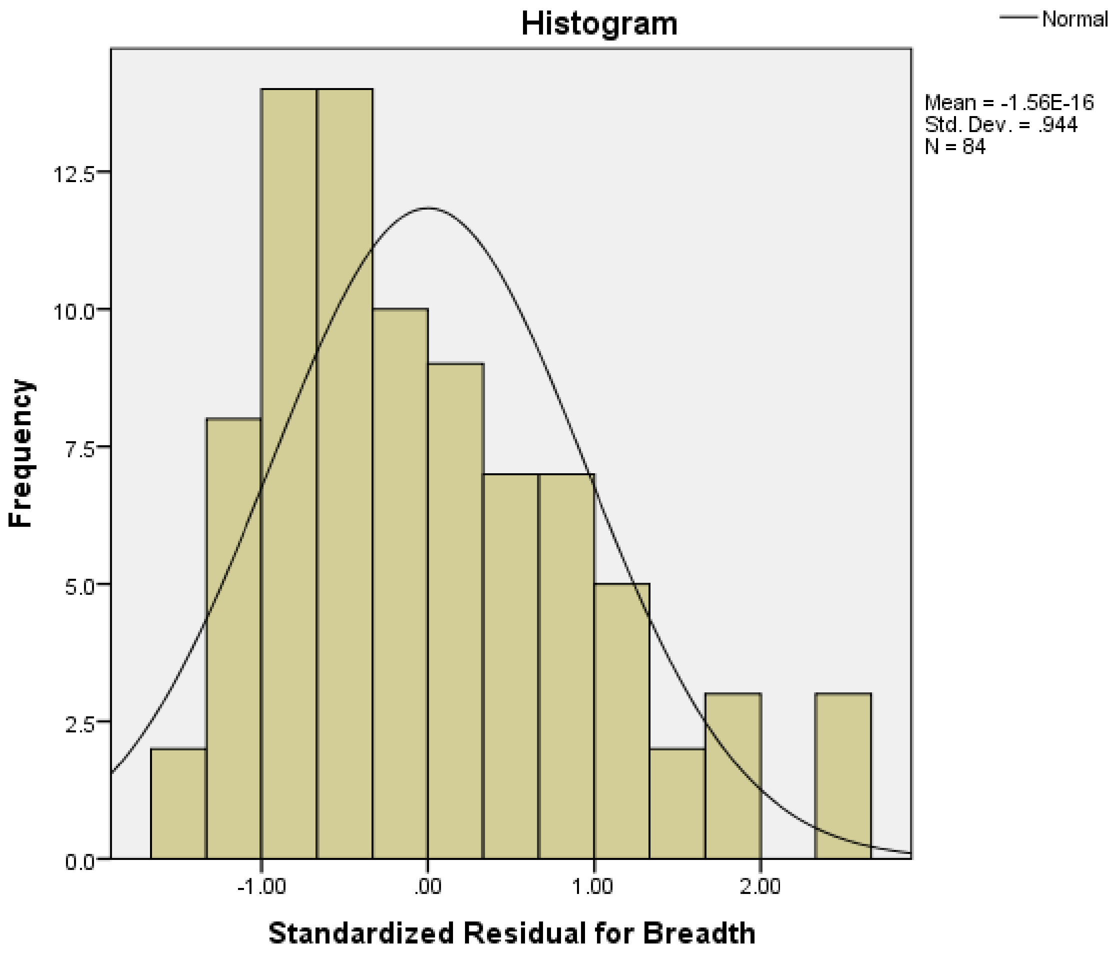
Skewness and Kurtosis (Dependent variable: Requirements quantity)

	Statistic	Standard error
Skewness	.541	.263
Kurtosis	.005	.520

Levene's test of equality of error variances: (Dependent variable: requirements quantity)

F	Df1	Df2	Sig.
.040	1	82	.843

Normality plot of standardized residuals: (Dependent variable: Requirements Breadth)



Skewness and Kurtosis (Dependent variable: Requirements Breadth)

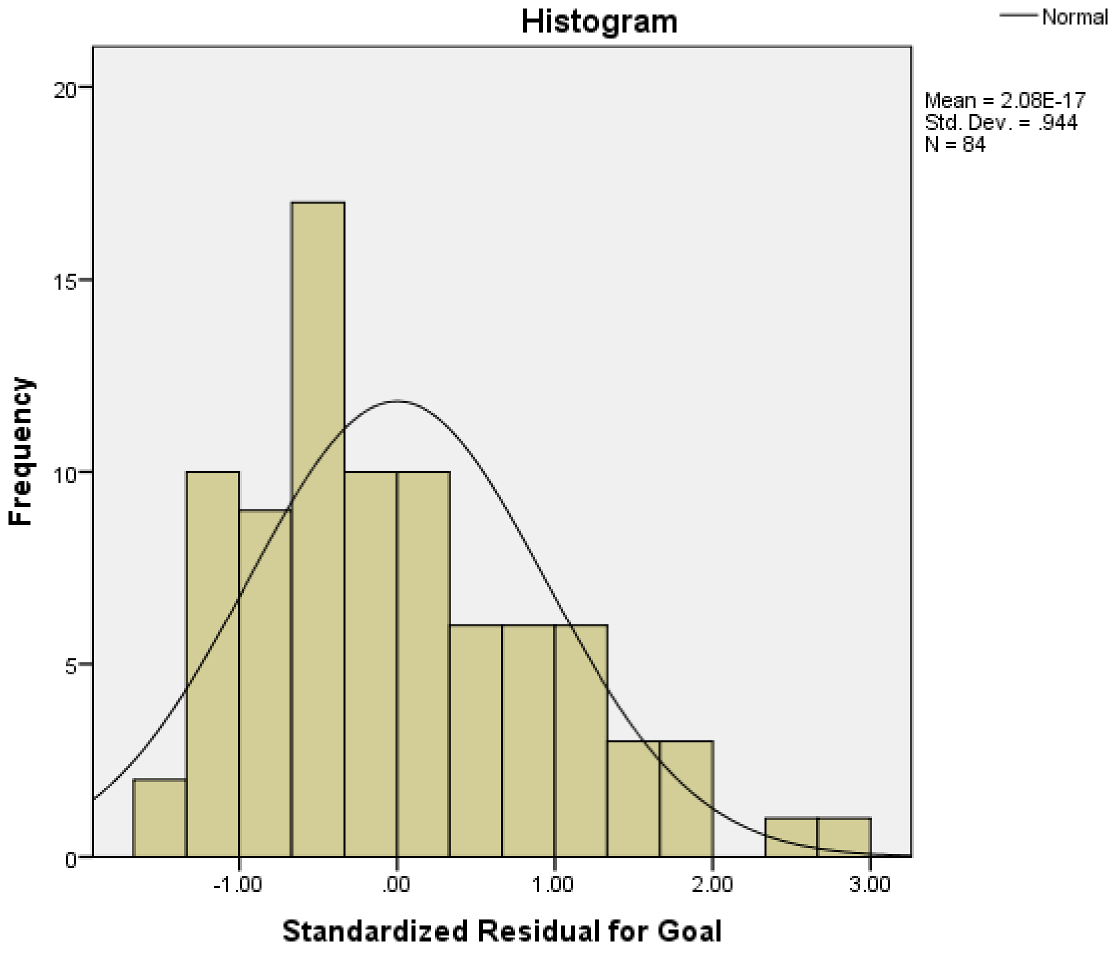
	Statistic	Standard error
Skewness	.817	.263

Kurtosis	.088	.520

Levene's test of equality of error variances: (Dependent variable: requirements breadth)

F	Df1	Df2	Sig.
.736	1	82	.393

Normality plot of standardized residuals: (Dependent variable: Requirements depth: goal category)



Skewness and Kurtosis (Dependent variable: Requirements depth: goal category)

	Statistic	Standard error
Skewness	.825	.263
Kurtosis	.203	.520

Levene’s test of equality of error variances: (Dependent variable: requirements depth: goal category)

F	Df1	Df2	Sig.
1.548	1	82	.217

APPENDIX 6: Skewness and kurtosis (differences of paired observations; paired samples t-test)

Skewness		Kurtosis	
Statistic	Std. Error	Statistic	Std. Error
-1.151	.263	2.601	.520
.230	.263	1.797	.520
-.802	.264	.615	.523
.101	.264	2.003	.523
.065	.266	3.041	.526
-.955	.264	1.008	.523
.124	.264	2.567	.523
-.017	.264	2.139	.523
-.590	.263	.966	.520
-.499	.263	.746	.520
-.700	.264	1.473	.523
.201	.263	2.314	.520
-.502	.264	2.192	.523
-.321	.264	-.136	.523
-.541	.263	1.347	.520
-.306	.264	-.230	.523
-.348	.264	1.128	.523
-.633	.266	-.069	.526
.017	.263	.598	.520
-.563	.266	.665	.526
-.325	.266	1.595	.526
-.714	.263	.757	.520
-.721	.263	1.715	.520
.003	.263	1.269	.520
-.356	.264	.701	.523
.100	.264	.952	.523
.153	.263	.204	.520

.150	.263	-.308	.520
-.015	.263	.546	.520
.063	.264	.859	.523
-.798	.264	1.537	.523
-.627	.266	1.547	.526
-.597	.266	2.507	.526
-.105	.264	1.603	.523
.418	.264	.764	.523
-1.217	.264	3.333	.523
-.421	.266	6.677	.526
-.040	.264	1.871	.523
.371	.264	1.698	.523
-.449	.264	.600	.523
-.492	.267	1.336	.529
-.649	.267	1.317	.529
-.309	.269	.843	.532
.081	.269	-.196	.532
-.034	.272	.183	.538
.132	.267	.488	.529
.019	.269	.098	.532
-.094	.267	-.440	.529
-.902	.263	.808	.520
-.720	.264	1.223	.523
-.340	.264	1.341	.523
-.288	.264	.832	.523
-.594	.263	.270	.520
.589	.264	1.464	.523
-.291	.263	.386	.520
-.477	.263	.465	.520
-.310	.263	1.110	.520

APPENDIX 7: OSS Issue reporting interfaces and misclassification questionnaire

A major problem with issues reported in the issue repositories of open source software projects is mis-classification (assigning incorrect label to an issue). For example, an issue labelled as a bug or a defect may not actually be a bug but rather an enhancement request, a feature request or a request for some documentation. Another example of mis-classification is when an issue labelled as a feature request may not actually be a feature request but rather a bug. Mis-classification of issues can result in problems and undesirable situations, for example, a valid bug that got mis-classified as a feature may not get fixed.

Different types of issue reporting interfaces can be found in the issue repositories of open source software projects. The goal of this survey is to find which interfaces are better at reducing mis-classification/incorrect labeling of issues. There is one question on the next page that shows two interfaces and asks which one is better at reducing misclassification/incorrect labelling of issues. The survey would need only about two minutes to complete. If you are interested in participating, please answer the question on the next page.

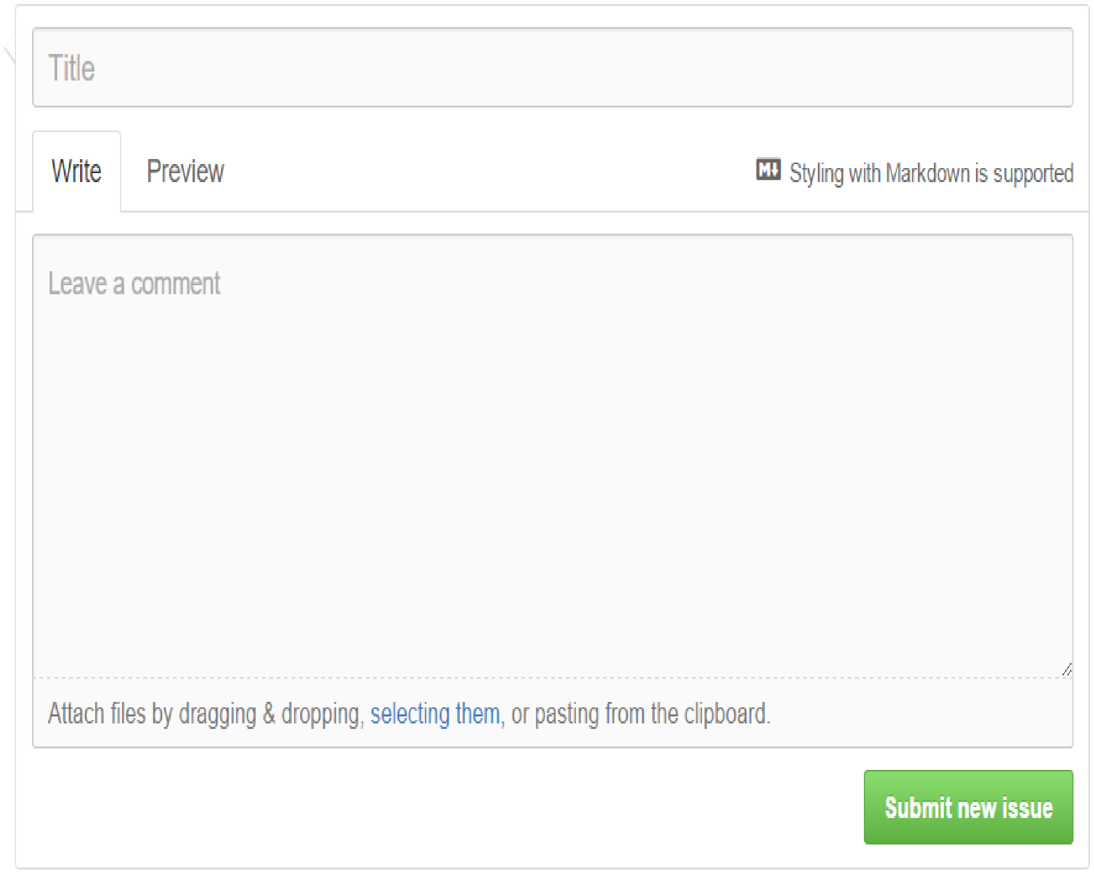
This survey is part of a PhD dissertation research on requirements gathering in open source software development. The survey is anonymous and no identifying information is asked. The proposal for this research has been reviewed by the Interdisciplinary Committee on Ethics in Human Research and found to be in compliance with Memorial University's ethics policy. If you have ethical concerns about the research, such as the way you have been treated or your rights as a participant, you may contact the Chairperson of the ICEHR at icehr@mun.ca or by telephone at 709-864-2861. Participants can send an email to jk5573@mun.ca for any questions or for results of the survey.

A major problem with issues reported in the issue repositories of open source software projects is mis-classification (assigning incorrect label to an issue). For example, an issue labelled as a bug or a defect may not actually be a bug but rather an enhancement request, a feature request or a request for some documentation. Another example of mis-classification is when an issue labelled as a feature request may not actually be a feature request but rather a bug. Mis-classification of issues can result in problems and undesirable situations, for example, a valid bug that got mis-classified as a feature may not get fixed.

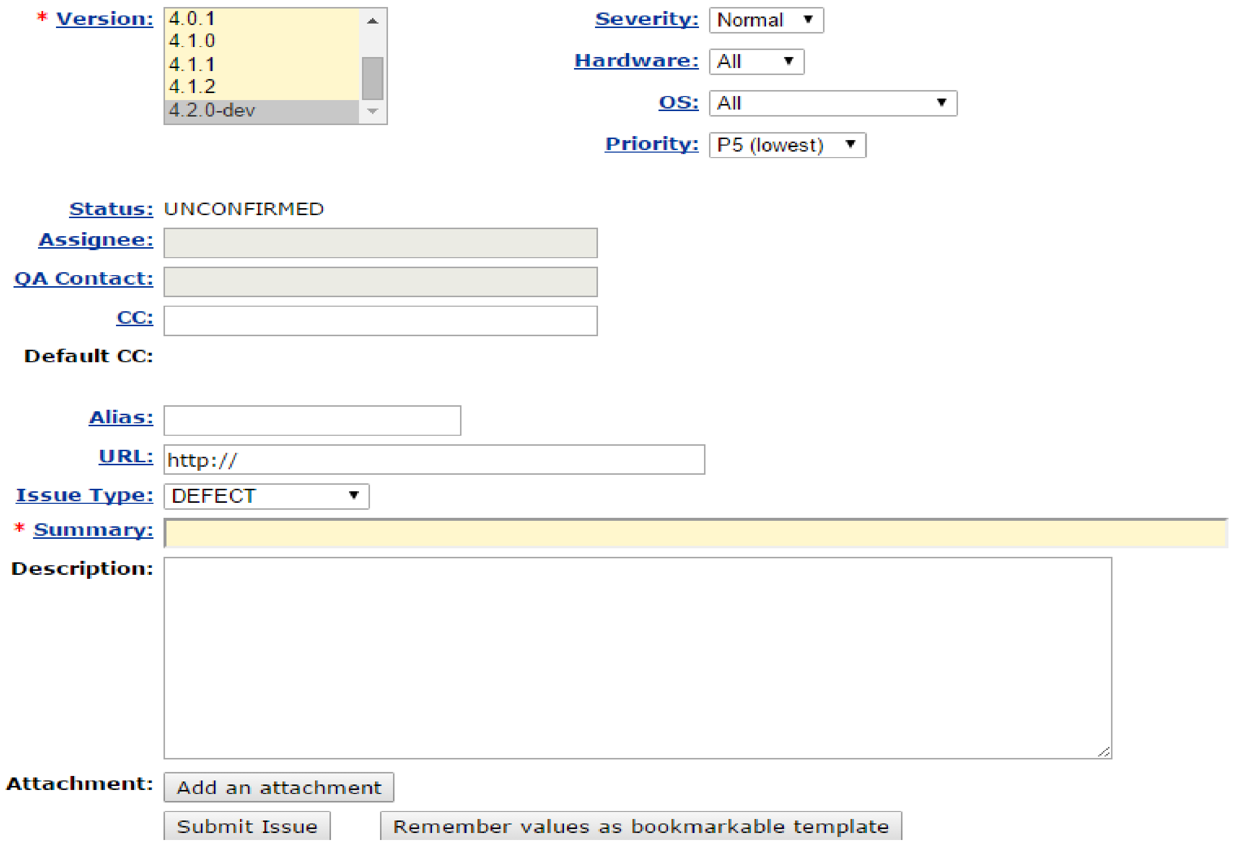
Different types of issue reporting interfaces can be found in the issue repositories of open source software projects. Two of them are shown below. In the first interface, issue reporters cannot assign any label to the issue that they are reporting and it is the developers/project administrators that assign the labels after they have been submitted.

In the second one, issue reporters can assign labels to the issues that they are reporting. Please carefully analyze both of them. Based on your experience and knowledge, please indicate which interface is more capable of reducing mis-classification/incorrect labelling.

Issue interface one is shown below:


Title
Write
Preview
Styling with Markdown is supported
Leave a comment
Attach files by dragging & dropping, selecting them, or pasting from the clipboard.
Submit new issue

Issue interface two is shown below:



* **Version:** 4.0.1, 4.1.0, 4.1.1, 4.1.2, 4.2.0-dev

Severity: Normal

Hardware: All

OS: All

Priority: P5 (lowest)

Status: UNCONFIRMED

Assignee:

QA Contact:

CC:

Default CC:

Alias:

URL: http://

Issue Type: DEFECT

* **Summary:**

Description:

Attachment:

*

Issue interface one

[✓ Issue interface two]

None of them

Both of them

2) Please provide any comments that you may have below.

APPENDIX 8. Demographics of participants in the experiment in phase two

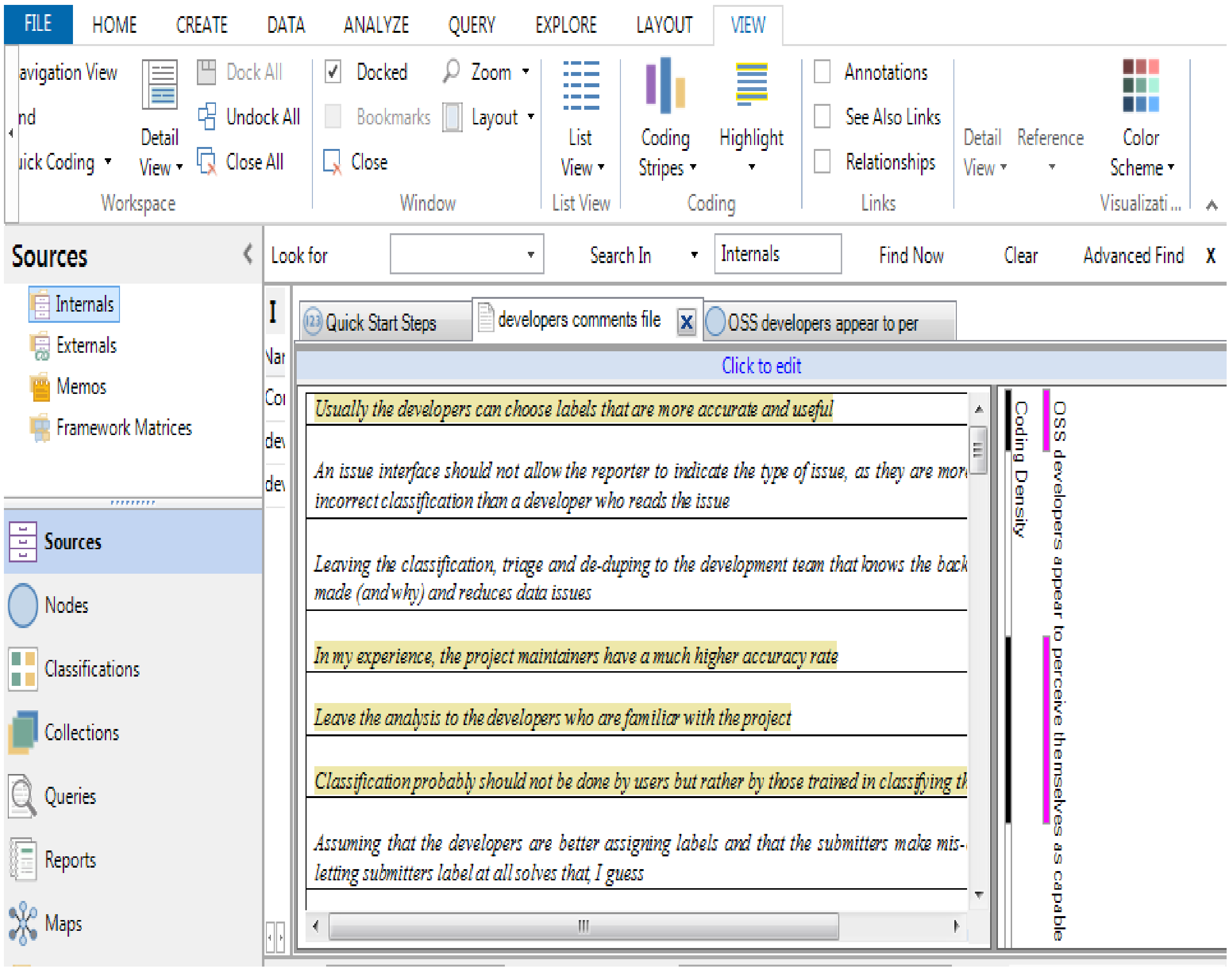
Experimental participants’ experience with open source software development

No experience or familiarity with open source software projects	General interest in open source software projects but never participated in the development of any	Participated in open source software development by reporting issues and/or participating in discussions	Participated in open source software development by writing code
58%	24%	12%	6%

Geographical locations of experimental participants

Geographical location	Percentage of participants
USA	21%
Canada	20%
Italy	8%
Germany	7%
Portugal	7%
Netherlands	7%
UK	6%
Spain	4%
Poland	4%
Belgium	2%
Ireland	2%
Denmark	2%
Estonia	2%
Sweden	1%
France	1%
Finland	1%
Australia	1%
New Zealand	1%

APPENDIX 9. Examples of coding of developers' comments



Usually the developers can choose labels that are more accurate and useful

An issue interface should not allow the reporter to indicate the type of issue, as they are mor
incorrect classification than a developer who reads the issue

Leaving the classification, triage and de-duping to the development team that knows the back
made (and why) and reduces data issues

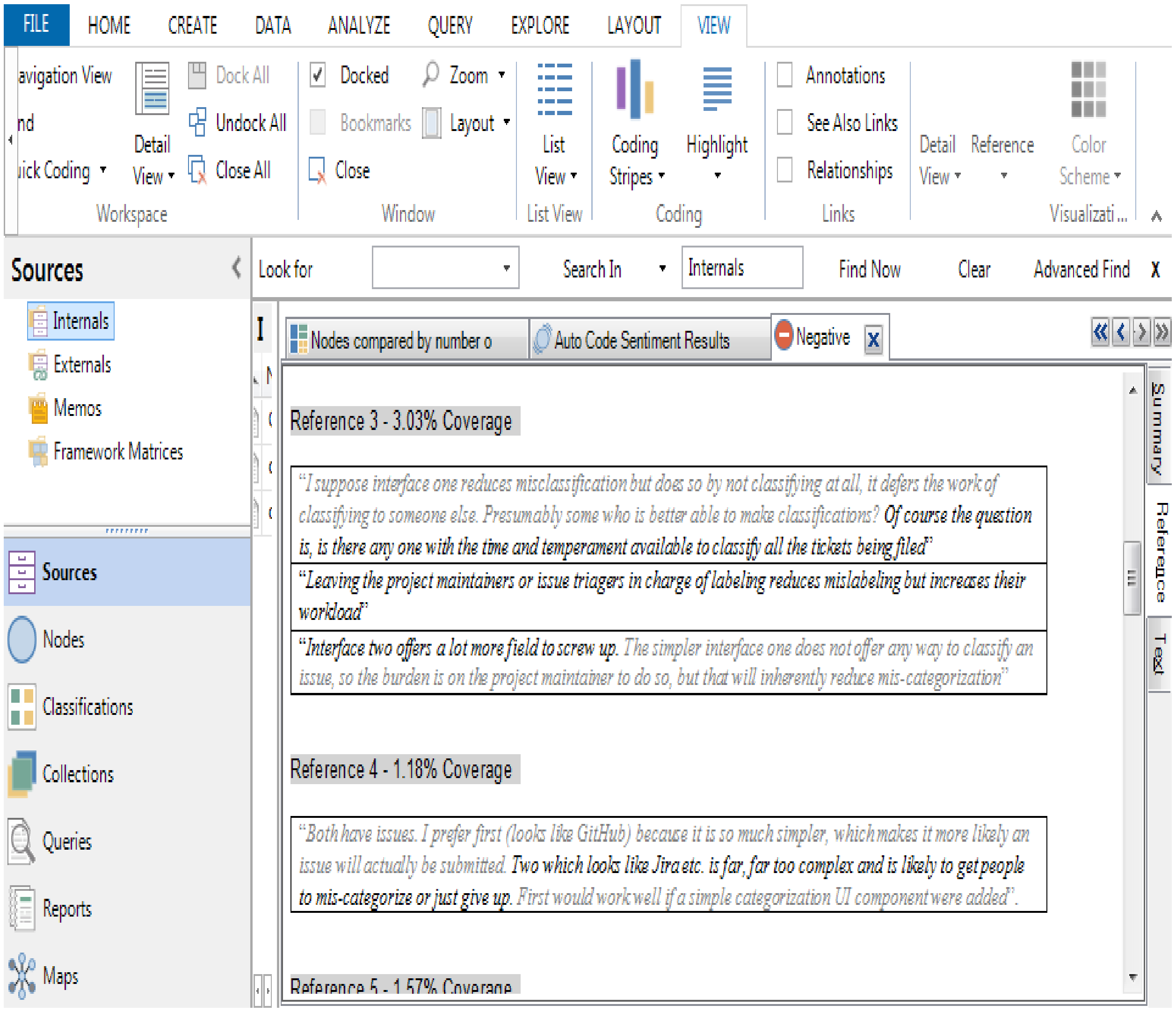
In my experience, the project maintainers have a much higher accuracy rate

Leave the analysis to the developers who are familiar with the project

Classification probably should not be done by users but rather by those trained in classifying th

Assuming that the developers are better assigning labels and that the submitters make mis-
letting submitters label at all solves that, I guess

Examples: sentiment analysis



Reference 3 - 3.03% Coverage

"I suppose interface one reduces misclassification but does so by not classifying at all, it defers the work of classifying to someone else. Presumably some who is better able to make classifications? Of course the question is, is there any one with the time and temperament available to classify all the tickets being filed"

"Leaving the project maintainers or issue triagers in charge of labeling reduces mislabeling but increases their workload"

"Interface two offers a lot more field to screw up. The simpler interface one does not offer any way to classify an issue, so the burden is on the project maintainer to do so, but that will inherently reduce mis-categorization"

Reference 4 - 1.18% Coverage

"Both have issues. I prefer first (looks like GitHub) because it is so much simpler, which makes it more likely an issue will actually be submitted. Two which looks like Jira etc. is far, far too complex and is likely to get people to mis-categorize or just give up. First would work well if a simple categorization UI component were added".

Reference 5 - 1.57% Coverage