Refinement and Resolution of Just-in-Time Requirements in Open Source Software: A Case Study

Tanmay Bhowmik & Anh Quoc Do

Department of Computer Science and Engineering
Mississippi State University, USA

Abstract

Just-in-time (JIT) requirements are characterized as not following the traditional requirement engineering approach, instead focusing on elaboration when the implementation begins. In this experience report, we analyze both functional and non-functional JIT requirements from three successful open source software (OSS) projects, including Firefox, Lucene, and Mylyn, to explore the common activities that shaped those requirements. We identify a novel refinement and resolution process that all studied requirements followed from requirement inception to their complete realization and subsequent release. This research provides new insights into how OSS project teams create quality features from simple initial descriptions of JIT requirements. Our study also initiates three captivating questions regarding JIT requirements and opens new avenues for further research in this emerging field.

Keywords: just-in-time requirements; refinement; resolution; functional requirements; non-functional requirements
2010 MSC: 00-01, 99-00

1. Introduction

According to traditional requirements engineering (RE), the requirements should be fully specified upfront so that the developers can proceed with the implementation activities. Recent RE research, however, has determined that there is no significant correlation between successful OSS projects and definitive software documents [1] and such a traditional RE approach is often found to be rather ineffective in OSS development paradigm [2]. Consequently, the notion of "just-in-time" (JIT) requirements, characterized by lightweight representation and continuous refinement of requirements, has recently emerged to bridge the gap between theory and practice [3].

Recent research has provided valuable insights about the characteristics and dominance of JIT requirements in successful OSS systems [2, 4, 5, 6, 7, 8]. Ernst and Murphy [2] emphasized the tightly coupled nature of elaboration and implementation of JIT requirements and indicated that the requirements are informally captured in a lightweight form, e.g., user stories, and further refined during implementation. Heck and Zaidman [5] pointed out the importance of JIT trait specially when requirements are likely to change and can not be documented in details before implementation. With a few exceptions, such as [2], much of recent JIT RE research has mostly studied functional requirements [9] whereas non-functional requirements (NRFs) [9] have remained largely unexplored. Furthermore, little is known about the individual activities that shape the functional and non-functional requirements in JIT RE process.

In this experience report, we present a case study involving three successful large-scale OSS systems—Firefox, Lucene, and Mylyn—and detail the common trend we observe in their JIT RE process. In particular, we pick one functional and one non-functional JIT requirement from each system, analyze relevant comments and artifacts recorded over the issue tracking systems, identify individual activities performed by the stakeholders during the resolution process, and further present a detailed workflow associated with JIT RE. In what follows, Section II highlights recent studies on JIT requirements. Section III presents our methodology and study setup. Section IV details the refinement and resolution process we observe, some additional discussion is presented in Section V, Section VI highlights the limitations and our future work.

Table 1: Selected Just-in-time Requirements

ID	Requirement	Category**	Partic
FIREFOX-480148	Restore visible tabs first when restoring session	Functional	1
FIREFOX-767676	Implement Security UI Telemetry	Non-functional (Security)	1
LUCENE-2507	Automaton spellchecker	Functional	3
LUCENE-2127* LUCENE-2215*	Improved large result handling	Non-functional (Efficiency)	1
MYLYN-256699	Show description in preview mode for existing tasks with editable descriptions	Functional	7
MYLYN-116487	Create performance test harness	Non-functional (Testability)	6

*The reporter first opened issue 2127 to discuss the requirement and eventually liked an idea proposed by a contributor named Aaron.
allow Aaron upload his patch. This implementation was then examined for its feasibility. Other developers also contributed to further

**We follow Boehm's NFR list [10] in selecting and classifying our NFRs.

2. Background and Related Work

The term "just-in-time" (JIT) was popularized by Toyota and other Japanese firms in the 1950s referring to meeting customer demands at the right time and in the exact amount with minimized inventory cost [11]. Inspired by the idea of JIT in the manufacturing industry, RE researchers have utilized the notion of JIT requirements to stress the efficient implementation achieved through reduced documentation effort. To that end, Michael Lee [7] detailed the analysis process of an agile methodology and outlined two fundamental principles of JIT requirements analysis (JITRA): "identify when they are needed" and "only at the level of detail required".

Research related to JIT requirements has recently emerged. Studying four different OSS communities, Scacchi [4] initially discovered eight kinds of "software informalisms" playing critical role in OSS RE. Scacchi *et al.* [1] further explored that a new OSS requirement is often a functionality or feature informally captured through a story telling or a user experience at the initial stage. In a seminal work on JITRA, Ernst and Murphy [2] studied three requirements from three large-scale OSS systems and indicated the fact that OSS requirements are initially captured in an informal manner and later elaborated during implementation. In a recent work, Heck and Zaidman [5] identified a set of quality criteria and proposed a framework to assess the quality of JIT requirements. Analyzing the resolution time of JIT requirements from two OSS systems, Bhowmik and Reddivari [8] recently reported two interesting observations about JIT RE activities and product release time. Despite these research initiatives, knowledge in JIT RE is fairly limited. To further expand the literature, in this experience report, we highlight the individual activities involved in the refinement and resolution of JIT requirements by analyzing both functional and non-functional requirements.

3. Study Setup

In this study, we investigate requirements from three large-scale OSS projects: Firefox, Lucene, and Mylyn. There are several key criteria that lead to this selection. First, these are stable and successful projects. Therefore, studying their features can uncover some good latent practice. Second, they were previously studied in JIT RE research [2, 8], thus providing valuable insights about the JIT aspects of their requirements. Finally, their requirements and associated data required to conduct our study are publicly available on the issue tracking systems such as Jira (for Lucene) or Bugzilla (for Firefox and Mylyn). In what follows, we provide a brief overview of the subject systems and detail the selection criteria for candidate requirements.

3.1. Mozilla Firefox

Mozilla Firefox[1], first released in November 2004, is a popular open source web browser developed by the Mozilla Foundation. Firefox is well-known for its extensibility with many useful add-ons and security enhancements compared to its main rival at that time, Internet Explorer 6 from Microsoft.

3.2. Lucene

Apache Lucene[2] is a Java-based open-source software library to support full text indexing and searching the content for cross-platform applications. Lucene aims to provide scalable high-performance indexing mechanisms supporting powerful and highly efficient searching algorithms.

3.3. Mylyn

Mylyn[3] is an open-source *task-focused interface* for Eclipse that concentrates on reducing information overload and increasing productivity for the users. To that end, Mylyn provides relevant task views and monitors user's activity to make multitasking more convenient in a timely manner.

3.4. Selecting Candidate Requirements

Each project contains a tremendous number of requirements making it impractical to study all of them within the scope of this paper. Therefore, we apply the critical case sampling technique [12], i.e., selecting a single case or small number of cases that can yield important information about the phenomenon of interest. Using this technique, we collect prospective requirements that are likely to reveal a concrete picture about JIT requirement refinement and resolution process. The selection criteria used to pick the sample requirements include: 1) requirements should start with a lightweight representation (i.e., user stories), and 2) requirements should trigger a significant discussion among

[1]https://www.mozilla.org/en-US/firefox
[2]https://lucene.apache.org
[3]http://www.eclipse.org/mylyn

stakeholders (i.e., number of participants and comments associated with the requirements). As such criteria still generates a long list of candidate requirements for each system, we randomly select a functional and a non-functional one from each list for further analysis.

Table I provides an overview of the requirements chosen by applying the aforementioned criteria. In order to obtain a general insight into the JIT RE process, we select one functional and one non-function requirement from each project. In the next section, we detail the individual activities we notice during the refinement process for the selected JIT requirements. We also report on the observed relationships among those activities and the collaboration among stakeholders that shapes a JIT requirement and its corresponding implementation.

4. Results and Analysis

This section presents the JIT RE refinement and resolution process we have explored in our study. We observe that all the studied requirements follow an iterative and evolutionary approach. Figure 1 presents the general refinement and resolution process that we find common among the requirements. Subsequently, we specify each activity in the process and then include further discussions based on our investigation.

4.1. Refinement and Resolution Steps

Initialize Requirement. As an essential part in RE activities, requirement initialization marks the inception of JIT requirements. For each requirement, one authorized developer (called reporter) creates a high-level feature request in the issue tracking system with a short and clear title. In addition, the reporter provides a brief description to guarantee that all other developers have an original idea about the requirement specification before moving forward to the review activity.

Review. The review takes place after the requirement is reported. This activity includes two courses of action.

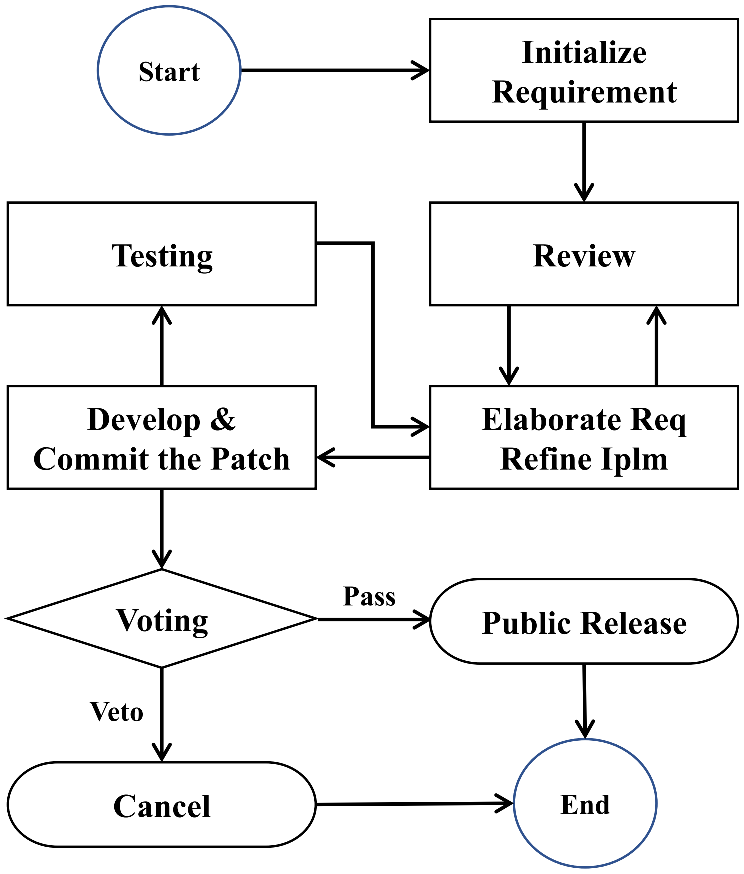

Figure 1: Refinement and Resolution of just-in-time Requirements.

Requirement Review: When the requirement is proposed, there is an informal review to verify if the requirement is eligible for implementation. In FIREFOX-480148, for example, one participant argues that this feature had been implemented before and should not be integrated into the current system (this claim, however, is later disproved). If the requirement passes through verification, others can contribute to further elaboration, making it fit for the project roadmap.

Implementation Review: Whenever a developer commits a patch, he/she notifies others by posting a comment over the issue tracking system. The contributors download committed patch to their local machines, examine and run the code for further testing and review. By doing so, other developers gain crucial insights into the committed patch and provide constructive feedback for improvement, if necessary.

Beside above reviews, each project might have exclusive standards to examine the qualification of the requirement and its associated implementation. In Firefox, for instance, patches that are module-crossed, have a high impact on the existing system, or are likely to involve security aspects, will have to undergo an additional super-review that consists of crucial expert-reviews for critical standards.

Elaborate Requirement / Refine Implementation. We notice in our study that JIT requirements and their associated implementation are highly coupled. Usually, JIT requirements are not well-defined upfront before implementation begins. Therefore, some misconceptions naturally exist during implementation. The following are some typical problems the developers addresses in the JIT requirements we examine.

Requirement Elaboration: At first, a JIT requirement is stated as a compact title with a short description. Unsurprisingly, reporters sometimes have difficulties anticipating all suitable uses for the users. Hence, there might be some unpleasant user experiences when the feature is developed at the beginning. For instance, in MYLYN-256699, users can only trigger editing description by double-clicking on the text field. This approach seemed inappropriate where several developers pointed out that the common practice should be one-click when setting cursor whereas double-click is used for selecting text. As a result, the developer changed to one-click editing and everyone came to an agreement. We identify that the refined requirement specification gradually reaches consensus and fulfills project roadmap in this manner.

Implementation Refinement: Implementation plays an important role in allowing a JIT requirement to be integrated into the system. For all three projects, quality code is found to be absolutely crucial. For each committed patch, developers examine the code, identify potential flaws, if any, and provide suggestions to improve the current implementation. The main aspects that developers focus on to refine the implementation are: (1) **Correctness**: the implementation works the way specification states (e.g., LUCENE-2127, MYLYN-256699); (2) **Performance**: the implementation operates efficiently (e.g., LUCENE-2127, MYLYN-116487); and (3) **Coding Style**: the implementation follows the coding rules specified for a certain project (e.g., FIREFOX-767676, FIREFOX-480148).

As we notice in our study, this activity has a strong relationship with other activities. A development loop among Testing, Review, Elaboration, and Development activities is established to construct an effective and well-defined JIT requirement. During the interaction among these activities, JIT requirements and their implementation progress gradually.

Develop and Commit the Patch. In this activity, the developer basically transforms his understanding about the requirement specification, which is shaped through his own experience and others' feedback, to code and subsequently pushes the changes to the remote repository. The original reporter plays a vital role in developing the code. However, following the nature of OSS projects, other developers are also considerably involved in the activity. During the discussion, remaining developers can either give valuable suggestions (e.g., LUCENE-2127) or develop their own implementation and commit to the development repository (e.g., MYLYN-256699). As the requirement becomes more mature, the implementation also progresses to adapt with constructive feedback from reviewers.

Testing. The testing activity takes place after a developer commits the code in order to examine the implementation and provide helpful feedback. It includes but not limited to:

- Functionality Testing - justify that the new feature operates in an expected manner (all the studied requirements)
- GUI Testing - make sure that the GUI is consistent with the existing system and the design elements display in an expected manner (e.g., MYLYN-

256699)

- Performance Testing - verify the efficiency of the system with the new implementation (e.g., LUCENE-2127)
- System Consistency Testing - verify that the newly implemented feature follows and utilizes existing system designs (all the studied requirements)

Voting. After finalizing the requirement specification and implementation, one last step is conducted to decide if the patch should be released to public or not. The voting process can take place over IRC, on mailing lists, or even in issue comments. For example, in LUCENE, an email is sent with a subject starting with [**VOTE**] to the mailing list calling for a vote. If all members vote for it, the patch will be integrated into the released version. Otherwise, if any member objects the patch, he/she needs to deliver convincing reasons and must provide an alternative implementation (as stated in the Apache Foundation Software website). The substitution will then be reviewed and examined for its feasibility. If passed, a new issue will be created triggering the lifecycle of a new requirement.

We, however, did not observe any replacement for the JIT requirements we studied. One underlying rationale we observe is that most of the developers involved in the feature already participated in the discussion over the issue tracking systems. They examined the feasibility of the feature and provided feedback. That resulted in a mature fully-developed specification, making consensus in the voting process inevitable.

5. Discussion

While conducting this preliminary research, we have discovered encouraging insights about the activities that drive the development of JIT requirements, thereby leading to the refinement process we have presented in this paper. This is a generalized process intended to be applicable beyond the six functional and non-functional JIT requirements we have studied. We notice a significant association among activities that gradually shapes a complete JIT requirement.

Research on JIT requirements is at the early of its dawn and this study aims to further expand understanding about the JIT aspects. Nonetheless, due to limited scope of this paper, the number of JIT requirements studied is yet very small. The critical sampling technique [12] has been used to seek for interesting patterns of functional and non-function JIT requirements. However, our number is constrained compared to the substantial amount of requirements our studied projects possess. In addition, with the studied non-functional JIT requirements, we notice that the developers mainly focus on implementation refinement rather than requirement elaboration (e.g., FIREFOX-767676, LUCENE-2127, MYLYN-116487). It should be noted that we have chosen non-functional requirements with a variety of focus, including security, performance and testability, so that the observation is broadly generalized. Such observation inspires us to formulate following questions regarding JIT requirements and our refinement process.

- *Q1*: How general the refinement process is? Is it possible to apply this process to a variety of JIT requirements?
- *Q2*: Do the quality of JIT requirements really improved if they follow the refinement process?
- *Q3*: Is there any distinction between the elaboration processes of functional and non-functional JIT requirements?

6. Limitation and future work

In this case study, we have analyzed the refinement and resolution of both functional and non-functional JIT requirements in three large OSS systems. We showed how JIT requirements evolve through a set of activities with the supportive collaboration from developers in OSS communities. In addition, we formulated three questions that could open subsequent research avenues in JIT

RE. Our findings, however, are based on two requirements from each of three subject systems, which is a major limitation. We plan further comprehensive studies to strengthen our results and address the aforementioned research questions.

References

[1] W. Scacchi, C. Jensen, J. Noll, M. Elliott, Multi-modal modeling, analysis and validation of open source software requirements processes, in: International conference on Open Source Systems, Vol. 1, Genoa, Italy, 2005, pp. 49–63.

[2] N. A. Ernst, G. C. Murphy, Case studies in just-in-time requirements analysis, in: 2012 Second IEEE International Workshop on Empirical Requirements Engineering (EmpiRE), 2012, pp. 25–32. doi:10.1109/EmpiRE.2012.6347678.

[3] T. A. Alspaugh, W. Scacchi, Ongoing software development without classical requirements, in: Requirements Engineering Conference (RE), 2013 21st IEEE International, IEEE, 2013, pp. 165–174.

[4] W. Scacchi, Understanding the requirements for developing open source software systems, IEEE Proceedings - Software 149 (1) (2002) 24–39. doi:10.1049/ip-sen:20020202.

[5] P. Heck, A. Zaidman, Quality criteria for just-in-time requirements: just enough, just-in-time?, in: 2015 IEEE Workshop on Just-In-Time Requirements Engineering (JITRE), 2015, pp. 1–4. doi:10.1109/JITRE.2015.7330170.

[6] N. Niu, T. Bhowmik, H. Liu, Z. Niu, Traceability-enabled refactoring for managing just-in-time requirements, in: 2014 IEEE 22nd International Requirements Engineering Conference (RE), 2014, pp. 133–142. doi:10.1109/RE.2014.6912255.

[7] M. Lee, Just-in-time requirements analysis-the engine that drives the planning game, Agile Alliance, 2002, pp. 25–32.

[8] T. Bhowmik, S. Reddivari, Resolution trend of just-in-time requirements in open source software development, in: 2015 IEEE Workshop on Just-In-Time Requirements Engineering (JITRE), 2015, pp. 17–20. doi:10.1109/JITRE.2015.7330214.

[9] L. M. Cysneiros, E. Yu, Non-Functional Requirements Elicitation, Springer US, Boston, MA, 2004, pp. 115–138.

[10] B. W. Boehm, J. R. Brown, M. Lipow, Quantitative evaluation of software quality, in: Proceedings of the 2nd International Conference on Software Engineering, IEEE Computer Society Press, 1976, pp. 592–605.

[11] T. Ohno, Toyota production system: beyond large-scale production, Productivity Press, 1988.

[12] M. Q. Patton, Qualitative Research and Evaluation Methods, SAGE Publications, 2001.