MASTER

Bug fixing in action
activities, process, and knowledge

Lin, B.

Award date:
2016

Disclaimer
This document contains a student thesis (bachelor’s or master’s), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
Bug Fixing in Action: Activities, Process, and Knowledge

Master’s Thesis

Bin Lin

Supervisor: dr. Alexander Serebrenik
Co-supervisors: Alexey Zagalsky
                prof. dr. Margaret-Anne Storey

Exam Committee: dr. Alexander Serebrenik
                dr. Michel Westenberg
                dr. Anton Wijs
                Alexey Zagalsky (non-voting)

Eindhoven, August 2016
Abstract

Bug fixing plays a vital role in software development. Research has shown that debugging occupies around half of the programming time and costs huge financial budgets as well as massive human resources. A case study of WordPress is conducted in this thesis to provide insights into the modern bug fixing practices. This study investigates bug fixing from three perspectives: bug fixing activities, the bug fixing process, and knowledge flow in bug fixing.

First, we examine the limitations of using the bug status to understand bug fixing activities and identify bug fixing activities by inspecting bug tickets and communication archives. Second, we propose an approach to extract a process model of bug fixing, analyze when the communication tools are used, and explore the task distribution patterns of community leads. Third, we study how knowledge flows between bug tickets and how external knowledge resources aggregate on the bug tracking system.

As a result, our study proposes a new way to explore bug fixing process and provides new insights into the bug fixing practice. Moreover, we give suggestions for improving the bug fixing tools and practice, and present typologies for bug fixing activities, reasons of bug references, and external resources on bug tracking systems. Our results can serve as the basis for future bug fixing studies.

Keywords: Bug Fixing, Process Mining, Knowledge Flow, WordPress
Acknowledgments

Foremost, I would like to express my deepest gratitude to my supervisor, Dr. Alexander Serebrenik, who is always there to provide timely and invaluable help. He always inspires me with his dedication and enthusiasm to research and steers me in the right direction. After taking his course “Software Evolution”, I realized that there were so many interesting things to explore in the field of software engineering and started to look into this area. He also offered me opportunities to work with different excellent researchers and attend premier conferences, which significantly broadened my horizon and enriched my experience. His assistance in my future career plan was also critical for me to figure out which way to go. I cannot thank him enough for his support and encouragement in both academic and social life. It is my great honor to have him as my supervisor.

Furthermore, I would also like to thank the CHISEL group, especially Prof. Dr. Margaret-Anne Storey and Alexey Zagalsky, for their continued support and patience during my internship at UVic and graduation project. Their experience, comments, suggestions and tips are a precious asset for not only my current study but my future academic life. I would also thank other CHISEL members for being so kind and friendly to me. They made my two-month stay in Victoria even more enjoyable.

Additionally, I would also like to acknowledge my exam committee members: Dr. Michel Westenberg and Dr. Anton Wijs for offering their time to review the thesis and attend the defense.

Moreover, I would like to thank all of my friends during my study in the Netherlands. Their companionship and support made this journey exciting and delightful.

Finally, I must express my profound gratitude to my parents. Without their unconditional love and support, I never would have been able to get this far. I have always been far away from home since my bachelor, but their care is always by my side.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>v</td>
</tr>
<tr>
<td>List of Figures</td>
<td>vii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>ix</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Research Statement and Scope .................................... 1
   1.1.1 Bug Fixing Activities ..................................... 1
   1.1.2 Bug Fixing Process ......................................... 2
   1.1.3 Knowledge Flow in Bug Fixing ............................... 2

1.2 Contributions .................................................. 2

1.3 Thesis Organization ............................................ 3

## 2 Case Selection

   2.1.1 What WordPress Is and Why We Choose It .................. 5
   2.1.2 WordPress in Academia ..................................... 6

2.2 How to Fix a Bug for WordPress ................................. 6
   2.2.1 Workflow and Tool Use during Bug Fixing in WordPress .... 6
   2.2.2 An Example of a Bug Ticket on Trac ........................ 8

## 3 Bug Fixing Activities

3.1 Introduction .................................................... 11

3.2 Related Work .................................................. 11

3.3 Case Study Design ............................................. 12
   3.3.1 Research Questions ......................................... 12
   3.3.2 Data Collection ............................................ 13
   3.3.3 Data Analysis Process ..................................... 13

3.4 Results ........................................................ 14
   3.4.1 What are the limitations of using the bug status to understand the bug fixing process ............................................... 14
   3.4.2 What activities are conducted during the bug fixing process? ................................................................. 15

3.5 Conclusions and Future Work ................................... 17
   3.5.1 Conclusions .................................................. 17
   3.5.2 Limitations .................................................. 17
   3.5.3 Future Work ................................................ 18
CONTENTS

4 Bug Fixing Process ........................................ 19
   4.1 Introduction ........................................... 19
   4.2 Related Work ........................................... 19
      4.2.1 Process Model Discovery ........................... 20
      4.2.2 Social Network Analysis ............................ 21
   4.3 Case Study Design ...................................... 22
      4.3.1 Research Questions .................................. 22
      4.3.2 Data Collection and Analysis ........................ 22
   4.4 Results .................................................. 27
      4.4.1 How can we identify the bug fixing process? What can we learn from the bug fixing process? .......... 27
      4.4.2 What are communication tools mainly used for during bug fixing? .................. 32
      4.4.3 How do community leads distribute tasks during bug fixing? .................. 33
   4.5 Conclusions and Future Work ............................ 35
      4.5.1 Conclusions .......................................... 35
      4.5.2 Limitations .......................................... 35
      4.5.3 Future Work ........................................ 35

5 Knowledge Flow in Bug Fixing .............................. 37
   5.1 Introduction ........................................... 37
   5.2 Related Work ........................................... 37
   5.3 Case Study Design ...................................... 38
      5.3.1 Research Questions .................................. 38
      5.3.2 Data Collection and Analysis ........................ 39
   5.4 Results .................................................. 39
      5.4.1 Why and How do bug tickets are connected on Trac? .................. 40
      5.4.2 What types of knowledge from external resources are shared on Trac? .......... 40
   5.5 Conclusions and Future Work ............................ 43
      5.5.1 Conclusions .......................................... 43
      5.5.2 Limitations .......................................... 43
      5.5.3 Future Work ........................................ 43

6 Conclusions .................................................. 45

Bibliography .................................................... 47

Appendix .......................................................... 55

A Process model based on automatically extracted activities 55
List of Figures

2.1 Bug fixing workflow based on the WordPress handbook .............................. 7
2.2 Bug status transitions .................................................................................. 8
2.3 An example of a bug ticket on Trac .............................................................. 9

4.1 A fragment of the event log based on new bug statuses ............................. 25
4.2 Examples of community leads ....................................................................... 27
4.3 Process model based on the bug status ....................................................... 28
4.4 Process model based on all automatically extracted activities .................... 29
4.5 The transitions between “keyword:needs-patch” and “patch:add” extracted from the process model ................................................................. 30
4.6 The transitions between “status:new” and “patch:add” extracted from the process model ................................................................. 30
4.7 Partial statistics table of case duration of the trimmed process .................... 30
4.8 Durations between bug opening and the first patch (only tickets with during less than one hour are presented) ..................................................... 31
4.9 The transitions between “commit:add” and “owner:self” extracted from the process model ................................................................. 31
4.10 Incoming and outgoing activities of the tools and their occurrences ............ 32
4.11 Social network extracted by the “Mine for a Working-Together Social Network” plugin ................................................................. 33
4.12 Social network extracted by the “Mine for a Similar-Task Social Network” plugin ................................................................. 34

5.1 Examples of external resources linked to Trac ............................................... 41
5.2 Distribution of different types of external resources ....................................... 42

A.1 Process model based on automatically extracted activities .......................... 56
List of Tables

2.1 Fields of a bug report ........................................ 10
3.1 Classification of contributors’ activities during bug fixing ........ 15
4.1 Activities extracted from Trac tickets .......................... 23
4.2 Activities automatically extracted from Trac and the related activities observed from labeling Trac and communication tools ............... 24
5.1 How many Trac tickets refer to those tickets which belong to the same component 40
5.2 Percentage distribution of each type of external resources .......... 42
Chapter 1

Introduction

Bug fixing is a vital activity in software development projects. According to a study conducted by Britton et al. [1], debugging occupies 49.9% of the programming time, which results in an estimated global cost of $312 billion per year including wages and overheads. Good bug fixing practices can help companies or development groups save substantial financial budgets and human resources. To enhance the bug fixing productivity, a comprehensive understanding of bug fixing is necessary. Researchers have conducted lots of studies to investigate specific activities of bug fixing, such as duplicate bug report detection [2, 3, 4, 5, 6] and bug assignment [7, 8, 9, 10, 11]. Existing research has also recognized the critical role played by tools for bug fixing, such as issue tracking and version control systems [12, 13]. All of these studies show bug fixing has been a critical part of modern software development.

1.1 Research Statement and Scope

The overarching goal of this research is to provide developers with deep insights into the modern bug fixing practices. We also expect that our research can inspire readers to reflect on how developers can improve the bug fixing process.

To achieve this goal, we conduct a case study of WordPress\textsuperscript{1}, which is one of the most popular open-source web software projects.

Bug fixing is a broad topic and it covers many aspects. To scope this research, the study has been limited to three areas: bug fixing activities, the bug fixing process and knowledge flow in bug fixing. We choose these three areas because they are fundamental elements of bug fixing: activities constitute the process, and the process is enriched by software engineering knowledge.

1.1.1 Bug Fixing Activities

Bug fixing activities are basic units of bug fixing process. During the bug fixing, developers use the bug status to announce what activities they are conducting and have conducted. However, the bug status might be incomplete and erroneous [14]. A better approach for identifying developers’ activities during bug fixing is necessary.

Our research regarding bug fixing activities is guided by the following two questions.

\textsuperscript{1}https://wordpress.org/
CHAPTER 1. INTRODUCTION

RQ 3.1 What are the limitations of using the bug status to understand the bug fixing activities?

RQ 3.2 What activities are conducted during the bug fixing process?

1.1.2 Bug Fixing Process

Since manually labeling bug fixing activities is time-consuming and unscalable, we need to propose a new way to automatically extract activities from bug tickets. By mining bug fixing activities we can obtain the process workflow, analyze how tools are used and how bug fixing tasks are distributed to developers.

Our research regarding the bug fixing process is motivated by the following three questions.

RQ 4.1 How can we identify the bug fixing process? What can we learn from the bug fixing process?

RQ 4.2 What are communication tools mainly used for during bug fixing?

RQ 4.3 How do community leads distribute tasks during bug fixing?

1.1.3 Knowledge Flow in Bug Fixing

During the bug fixing process, knowledge from both inside and outside the bug tracking system are invaluable assets. Knowledge is *the information, understanding, or skill that you get from experience or education*[^1]. In our study, we mainly consider the knowledge about repositories and tools and we are interested in the knowledge flow between bug tickets, and the knowledge flow between the bug tracking system and external resources.

The research regarding knowledge flow in bug fixing is motivated by the following two questions.

RQ 5.1 Why and how are bug tickets connected on Trac?

RQ 5.2 What types of knowledge from external resources are shared on Trac?

1.2 Contributions

The main contributions of this work are as follows.

A new way to explore bug fixing process. Our findings suggest that several defects exist when using the bug status to analyze the bug fixing practice. We have proposed a new way to extract the bug fixing activities, yielding valid results. Our experience can serve as the basis for studies on bug fixing process.

New insights into the bug fixing practice and suggestions for improvement. We have demonstrated how to use process mining techniques for investigating the bug fixing process from different perspectives, including process workflows, tool use, and developers’ collaboration. The results obtained from the process are good supplements for current bug fixing studies. We also provide several suggestions for improving the bug fixing practice and tools.

CHAPTER 1. INTRODUCTION

**Typologies.** We have proposed several typologies in our study, which include typologies for bug fixing activities, reasons of bug references, and external resources on bug tracking systems. These typology models are the basis for further software engineering studies.

1.3 Thesis Organization

The remainder of this thesis is structured as follows. In Chapter 2, we present our study case, which includes the background information of WordPress, the reasons for choosing WordPress and the official procedures of bug fixing. The research is divided into three topics: bug fixing activities, the bug fixing process, and knowledge flow in bug fixing. Each topic is presented in a separate chapter: bug fixing activities is studied in Chapter 3, the bug fixing process is studied in Chapter 4, and the knowledge flow in bug fixing is studied in Chapter 5.

For each chapter, we structure the content as five sections. The first section, which is an introduction, provides an overview of the study conducted in the chapter. The second section describes background information about the topic as well as related academic studies. In the third section, we present the research questions and the methodology regarding data collection and analysis. The results are explained in the fourth section. The fifth section draws the conclusion and discusses limitations and the future work of the study.

At the end of this thesis, Chapter 6 summarizes the whole thesis and sheds light on potential extension of this research.
Chapter 2

Case Selection

In this chapter, we mainly describe our study case. We begin with an introduction of WordPress, a world leading open-source content management system (CMS), and why we choose it for the case study. We then elaborate how to fix a bug for WordPress, including tools the community uses and formal bug fixing workflow.

2.1 WordPress: World Leading Open-Source Software

In this section, we first introduce the WordPress software and why we choose it as our study case. Then we list some studies on WordPress.

2.1.1 What WordPress Is and Why We Choose It

WordPress\(^1\), initially released as a blogging system in 2003, is open-source web software. It has evolved to a full content management system with a great number of themes, widgets and plugins.

We choose WordPress as our study case based on the following reasons.

- **WordPress is world leading web software.** According to a market report by W3Techs\(^2\), a company which collects information about the technology used for websites, by July 30, 2016, “WordPress is used by 59.6% of all the websites whose content management system we know. This is 26.5% of all websites.”\(^3\) Studying how such successful software works may provide quotable experiences for other software.

- **WordPress has a large community.** In addition to massive users online, there are also a large number of offline meetups organized by the WordPress Community. By July 30, 2016\(^4\), WordPress has already gathered 346,988 active members and 1,073 meetups. The community is also very diverse, as WordPress has been translated to 162 languages and dialects\(^5\).

\(^1\)https://wordpress.org/about/
\(^2\)https://w3techs.com/
\(^3\)https://w3techs.com/technologies/details/cm-wordpress/all/all
\(^4\)http://www.meetup.com/topics/wordpress/
\(^5\)https://make.wordpress.org/polyglots/teams/
• **The WordPress community adopts of modern tools.** Tools such as SVN\(^6\), Trac\(^7\) and Slack\(^8\) are used for coding and communication generate abundant rich information for our study.

• **WordPress provides easy access to archives.** All the archives including bug reports, chatting logs, code change histories are available online for free, which allows us to collect the data easily.

• **WordPress is widely studied in academia.** In recent years, there has been an increasing amount of literature on WordPress. Section 2.1.2 lists some relevant studies.

Our study focuses on the analysis of the bug reports and communication tools the WordPress community uses. We choose them because they are the main channels the WordPress community use to provide support for bug fixing.

### 2.1.2 WordPress in Academia

WordPress has been extensively studied by the research community and the research has investigated both technical and social aspects.

A number of researchers have studied the interference effects of different units in WordPress. Koskinen \textit{et al.} \cite{15} investigated the interference between the resources declared and used by different WordPress plugins, and proposed an approach to detect and locate such interference. Sayagh \textit{et al.} \cite{16} further examined the interference caused by multi-layer configuration options in WordPress, WordPress plugins and the PHP system.

Security issues have also been discussed. Koskinen \textit{et al.} \cite{17} studied the vulnerability of WordPress plugins against well known security exploits and found no correlation between the plugin ratings and the number of vulnerabilities.

Norrie \textit{et al.} \cite{18} surveyed WordPress developers to understand modern web development practices and claimed that it is necessary to have a stronger focus on interface-driven development and better support for reusing elements of design and implementation from other projects.

Vasilescu \textit{et al.} \cite{19} investigated technical support networks around CMS including WordPress and revealed the presence and participation of women in online communities.

### 2.2 How to Fix a Bug for WordPress

In this section, we first introduce the bug fixing workflow based on the WordPress guidelines for developers and the tools the WordPress community uses for bug fixing. We then illustrate an example of a bug report.

#### 2.2.1 Workflow and Tool Use during Bug Fixing in WordPress

The WordPress community maintains a \textit{Core Contributor Handbook}\(^9\), which serves as a guidebook for developers who want to contribute to WordPress.

---

\(^6\)https://wordpress.org/plugins/about/svn/
\(^7\)https://core.trac.wordpress.org/
\(^8\)https://make.wordpress.org/chat/
\(^9\)https://make.wordpress.org/core/handbook/
CHAPTER 2. CASE SELECTION

According to the handbook, a workflow is followed and several tools are involved to successfully fix a bug (Figure 2.1).

![Workflow diagram](image)

Figure 2.1: Bug fixing workflow based on the WordPress handbook

We can summarize the workflow as follows.

1. **Report the bug on Trac.** When WordPress developers or users find a bug, they are supposed to report it on Trac\(^{10}\), the issue tracking system WordPress adopted for bug management.

2. **Determine which bug to fix.** Developers need to determine which bug to work on. Besides finding a bug to fix themselves, developers can also be assigned to a bug by others.

3. **Work for a patch.** Patches, simple text files showing the code changes, are the only way that a regular developer can contribute to the source code of WordPress. In other words, patches present how developers want to fix the bug. To create a patch, developers need to get a working copy of WordPress from SVN\(^{11}\), the version control system that WordPress uses to maintain the source code. Developers are then supposed to upload the patch they implement to Trac as required in the handbook.

4. **Commit to the repository.** After reviewing team reviews the patch, developers with commit rights commit the patch to the code repository and close the bug ticket.

During the bug fixing, developers can use communication tools for discussion. The communication tools they use have evolved from mailing lists\(^{12}\), to IRC\(^{13}\) and then to Slack\(^{14}\).

In the workflow, developers use the bug status to represent which bug fixing stage they are at. There are six bug statuses in total, which are “new”, “accepted”, “assigned”, “reviewing”, “closed”, and “reopened”. The transitions between these statuses can be seen in Figure 2.2.

---

\(^{10}\) [https://trac.edgewall.org/](https://trac.edgewall.org/)

\(^{11}\) [https://subversion.apache.org/](https://subversion.apache.org/)

\(^{12}\) [https://codex.wordpress.org/Mailing_Lists](https://codex.wordpress.org/Mailing_Lists)

\(^{13}\) [https://codex.wordpress.org/IRC](https://codex.wordpress.org/IRC)

\(^{14}\) [https://make.wordpress.org/chat/](https://make.wordpress.org/chat/)
2.2.2 An Example of a Bug Ticket on Trac

Since Trac is the main bug management system WordPress adopts, here we give an example of a bug ticket on Trac.

A bug ticket consists of two parts: the bug report and the comments. Bug reports contain basic information about a bug, an example of which can be seen in Figure 2.3a. In the bug report, information is organized as different fields as shown in Table 2.1. This information is important for getting a rough idea of the bug. However, not all the fields are compulsory. When a bug is resolved, we can expect that the following fields are filled: “title”, “description”, “type”, “resolution”, “status”, “reported by”, “opened” and “closed”. Comments follow the bug report. In the comments, developers can attach files, upload patches, and discuss the bug. Meanwhile, several tools such as SVN and Slack are integrated into Trac and their activities are presented as a comment as well. As can be seen from the example Figure 2.3b, when someone mentions the bug on Slack, the link to the chat log will be displayed on Trac as a bot comment.
CHAPTER 2. CASE SELECTION

(a) An example of a bug report

Doesn’t work browse media library on Frontend

Reported by: purevsooj  Owned by: adamsilverstein

Milestone: 4.5.3  Priority: normal

Severity: normal  Version: 4.5

Component: Media  Keywords: has-patch fixed-major

Focuses: javascript

Description

I’m using browse media library dialog in frontend.
It was working on wordpress 4.4 and previous all version.
I don’t understand that issue. It’s only work on firefox but it doesn’t work on chrome and safari.
Please see the attached images.

(b) Comments to the example bug report

Figure 2.3: An example of a bug ticket on Trac
### Table 2.1: Fields of a bug report

<table>
<thead>
<tr>
<th>field</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>The title summarizes the bug.</td>
</tr>
<tr>
<td>Type</td>
<td>On Trac, not all tickets are bugs. A ticket can be one of the following types: defect (bug), enhancement, feature request, and task (blessed).</td>
</tr>
<tr>
<td>Resolution</td>
<td>A ticket can be closed as fixed, duplicate, invalid, worksforme, wontfix or maybelater.</td>
</tr>
<tr>
<td>Status</td>
<td>Status indicates what stage the bug is in, which can be new, assigned, reviewing, closed, reopened.</td>
</tr>
<tr>
<td>Reported by</td>
<td>“Reported by” indicates who reported the issue.</td>
</tr>
<tr>
<td>Owned by</td>
<td>The owner is used by committers and trusted core contributors to accept and assign tickets among themselves.</td>
</tr>
<tr>
<td>Opened</td>
<td>“Opened” is followed by the time when the ticket is created.</td>
</tr>
<tr>
<td>Closed</td>
<td>“Closed” is followed by the time when the ticket is closed.</td>
</tr>
<tr>
<td>Milestone</td>
<td>The milestone indicates the release where the ticket will be addressed.</td>
</tr>
<tr>
<td>Priority</td>
<td>The priority describes how serious the ticket is from the perspective of the WordPress project.</td>
</tr>
<tr>
<td>Severity</td>
<td>The severity describes how serious the ticket is from the perspective of the reporter.</td>
</tr>
<tr>
<td>Version</td>
<td>The version indicates when the issue arises.</td>
</tr>
<tr>
<td>Component</td>
<td>Components are well-defined, functional areas of WordPress.</td>
</tr>
<tr>
<td>Keywords</td>
<td>The keywords from pre-defined list describe the current status or required actions for a ticket.</td>
</tr>
<tr>
<td>Focuses</td>
<td>Focuses are specific aspects of components, such as UI, docs, performance, etc.</td>
</tr>
<tr>
<td>Description</td>
<td>Description explains the detail of the ticket, which might include how to reproduce, actual versus expected results, etc.</td>
</tr>
</tbody>
</table>
Chapter 3

Bug Fixing Activities

3.1 Introduction

Developers’ activities are basic elements of the bug fixing process. Studying developers’ activities during bug fixing can help us gain insights into modern bug fixing practice and lay a foundation for further studies of developers’ collaboration patterns. In the WordPress community, Trac serves as the main bug management system, integrating several tools such as SVN and Slack. Since the archives of these tools are publicly accessible, developers’ activities embedded in these tools are transparent to a certain extent.

In this chapter, we apply a qualitative exploratory case study methodology to answer the following research questions:

RQ 3.1 What are the limitations of using the bug status to understand the bug fixing activities?

RQ 3.2 What activities are conducted during the bug fixing process?

By mining Trac tickets, we identify the following limitations of the bug status: 1) the bug status is not always completely recorded; 2) the bug status sometimes is not recorded in the correct order or in time; 3) the wrong bug status might be used in the bug report; 4) the bug status is not always clear for people to understand its meaning. We also analyze the potential impacts of these limitations.

After analyzing archives of Trac and the communication tools, we identify activities conducted by developers during bug fixing and classify them into eight categories. We also compare our results with previous studies and our results show higher emphasis on developers’ awareness, social and organizational activities.

In the end of this chapter, we conclude our results, state the limitations of our research and propose the corresponding future work. Our concerns mainly focus on the threats to validity regarding the impacts of research sample selection, time change and differences between communities.

3.2 Related Work

A number of studies have been conducted to understand developers’ activities during bug fixing.
A bug’s life cycle consists of transitions between bug statuses, including self-loops. The bug status is frequently used as one of the important information sources to study developers’ activities. For example, Zhang et al. [20] combined information of the bug status and developers’ actions (including responding to tickets and editing source code) to analyze delays when fixing bugs. Weiss et al. [21] predicted the time spent on fixing bugs by investigating the average lifespan (i.e., duration between the first and the last bug statuses.) of similar, order bugs.

Moreover, a number of researchers have explored the particular activities based on the bug status. For instance, Jeong et al. [22] focused on the reassignment activity (i.e., the transition of the bug status from “assigned” to “assigned”), and Jongyindee et al. [23] focused on the reopening activity (i.e., the transition of the bug status from “closed” to “reopened”).

However, the bug status might not always be reliable. Aranda et al. [14] found the bug report fields sometimes were filled incorrectly. This finding indicates that bias can be introduced if we only use the bug status to study developers’ activities.

Researchers have also investigated developers’ activities at a more detailed level instead of checking the bug status. D’Ambros et al. [24] regarded developers’ activities as modifications of bug fields. They visualized activities with black bars positioned based on when activities happened, such that people could easily understand the relationship between the activities and the entire bug lifespan. Crowston and Scozzi [25] identified developers’ activities by examining the text of the bug report and comments and classified these activities into six categories: submit, assign, analyze, fix, test&post and close. They also discovered how these categories depend on each other according to the order of occurrence.

### 3.3 Case Study Design

#### 3.3.1 Research Questions

To understand the bug fixing activities, bug reports are an important source of information since they record bug statuses, bug reporter and developers’ communication, as well as links to various software artifacts.

Researchers have used information from bug reports to mine the bug fixing process. However, Aranda et al. [14] claimed bug reports were not always correctly recorded by investigating bug reports of Microsoft projects. Since developers of open-source software might be located around the globe and the bug reports are accessible by the public, they might need more detailed bug reports to collaborate and assist newcomers onboard. Therefore, we wonder if the bug reports for open source software can precisely reflect the bug fixing process. Here, we propose our research question

**RQ 3.1 What are the limitations of using the bug status to understand the bug fixing activities?**

From the results of RQ 3.1 we know bug statuses have many limitations when used for understanding the bug fixing activities. Additionally, the statuses are limited and it is impossible to interpret developers’ activities during transitions of these statuses. Currently, most bug fixing process studies focus on the bug statuses, which provide very limited insights into human activities. Although Crowston et al. [26] has investigated the tasks involved in bug fixing, their study is only based on bug reports. Since some activities are conducted in other channels such as communication tools [14], we can assume that tasks extracted from only bug reports are incomplete and cannot comprehensively present the entire bug fixing
process. Thus, we want to combine different information sources (i.e., Trac, mailing list, IRC and Slack) and study the process from a deeper perspective. We are interested in the following question:

**RQ 3.2** What activities are conducted during the bug fixing process?

### 3.3.2 Data Collection

As mentioned, the main goal of this chapter is to offer important insights into bug fixing activities. Qualitative exploratory case study methodology \cite{27} is employed to answer the questions presented in Section 3.3.1.

We mined data from the public archives of Trac, mailing list, IRC and Slack of the WordPress Project. To obtain the data, we designed a crawler to scrape the whole database of the Trac system (until May 30, 2016) and downloaded mailing list archives\(^1\). We did not crawl the IRC and Slack chatting logs since crawling is prohibited according to their robots exclusion protocol, namely robots.txt. Therefore, we manually stored the related data from IRC and Slack when necessary.

In our study, we used the bugs reported in recent 10 years (from May 30, 2006 to until May 30, 2016). In total, there were 21,178 bug tickets which fell into this range.

### 3.3.3 Data Analysis Process

#### Limitation of the Bug Status

To answer RQ 3.1, we selected 150 bug tickets as our sample data. To examine the limitations of the bug status for understanding bug fixing activities, we needed to ensure that the bug ticket contained enough information. Thus, we focused on bugs with abundant comments. Since the median value of the comment number of all closed bug tickets was 5.0, we decided to only study those bugs with more than five comments. The selected bug tickets were also latest as of the date when the database was collected (i.e., May 30, 2016).

The data analysis process of RQ 3.1 consists of two phases.

1. For each of the 150 selected tickets, we carefully compared the tickets with status changes. Our goal was to examine whether the activities embedded in the bug tickets could be precisely presented by the bug status. To achieve this goal, we read all comments to understand the context surrounding the bug, when we found the status should change at a certain point, we checked if the change was correctly recorded in the tickets. If not, we wrote down the ticket number and the type of mistake. After finishing examining all tickets, we summarized those mistakes.

2. We wrote a script to list all the possible status change paths of all of the 19031 closed bug tickets. We then checked if there were any abnormal patterns in the paths when compared with the official workflow (Figure 2.1). Our aim is to find problems with the bug status from a high level.

#### Activities during Bug Fixing

Since activities in different channels might be different, to get a comprehensive understanding of what developers do during bug fixing, we need to consider not only the bug tracking system

\(^1\) https://codex.wordpress.org/Mailing_Lists
Trac, but also other tools for communication, including mailing list, IRC and Slack. These tools are adopted by the WordPress community for bug fixing and their archives are publicly accessible. We divided Trac tickets into four categories: 1) Trac tickets with links to Slack discussion; 2) tickets with links to IRC discussion; 3) Trac tickets with links to mailing list discussion; and 4) Trac tickets without links to any communication tools.

To answer RQ 3.2, we selected 30 tickets from each category with the same selection criteria as RQ 3.1. We adopted an inductive approach to analyze the data. We read the comments on Trac and messages on communication channels, extracted the activities from the text and wrote these activities down when they first emerged. After collecting all activities, we checked if they could be merged and obtained a new full list of activities.

After getting all the activities, we categorized them to get a more clear structure for the activities. Since Aranda et al. already proposed a list of goals based on the contributors’ activities, we reused them and classified the activities we observed into their categories. The categories include:

- **Discovery.** Detect the unexpected behavior and record it as a bug.
- **Diagnosis.** Understand the nature, cause and impact of the bug and plan the actions to take for fixing the bug.
- **Assignment.** Determine the person who will be responsible for the bug.
- **Search.** Find the related knowledge, resources, skills, experts for the bug.
- **Correction.** Correct code and change relevant artifacts such as documentation.
- **Closure** Determine to close the ticket.
- **Awareness.** Communicate status to relevant participants.

During the analysis process, we classified activities, which did not fall into any of the categories, into “others”.

### 3.4 Results

#### 3.4.1 What are the limitations of using the bug status to understand the bug fixing process

To answer RQ 3.1, we sampled 150 bug tickets and compared the bug status with ticket comments. We also checked the status change paths of all bugs. We summarized the issues in the bug status and categorized them into the four main types: 1) Completeness, 2) Synchrony, 3) Correctness, and 4) Explicitness. Below we present our typology of issues regarding the bug status, meanwhile, some typical examples are given to support the typology.

**Completeness.** We examined whether all statuses were recorded. We found that incomplete status records were common in practice. For example, in ticket #36637, the owner of the bug was assigned but the status never changed to “assigned”. Another example is ticket #36906: some bug fixing activities were conducted after the ticket status was set to “closed”. That is, a “reopened” status is missing. We also found the statuses “accepted” and “reviewing” were not widely used, appearing in only 27 and 22 out of 150 tickets, respectively while accepting and reviewing the tickets were common actions during bug fixing.
Synchrony. We examined whether the status was recorded in the correct order in time. Two patterns against synchrony were found in our study: 1) the status was not assigned in time after the corresponding activity was executed. For example, ticket #36296 was not closed after committing and stayed inactive for several weeks. This ticket was finally closed after a reminder of another developer. 2) The status was assigned before the corresponding activity was finished. For example, the status of ticket #36055 was set to “assigned” when the developer was asking if another developer could take charge of it.

Correctness. We examined whether the status recorded was correct. In our study, two patterns against correctness were identified: 1) the status was recorded even if it did not happen. For example, the status of ticket #36660 was set to assigned although it was not assigned to anyone. 2) the developer selected the wrong status instead of a correct one. For instance, the status of ticket #8570 was changed to “new” from “reopened” after assigning an owner to the ticket.

Explicitness. We examined whether the status recorded was explicit and unambiguous. The results show that a status might be used to record different activities. For example, in ticket #36096, status “accepted” represented that developers approved the validity of the ticket and agreed to implement a patch, while in ticket #35956, status “accepted” represented that developers accepted the patch and agreed to commit the patch.

The validity of understanding bug fixing process with bug statuses can be negatively impacted by the issues presented above. When trying to discover the bug fixing model, incomplete records of bug status will lead to incomplete process workflow. Besides, asynchronous bug statuses can introduce bias into time-based process analysis such as bottleneck detection. Additionally, incorrect and implicit statuses may also result in misinterpretation of the bug fixing process. Moreover, the bug status can only provide very shallow insights into bug fixing process since merely six statuses are in use. These statuses do not cover all of the developers’ activities and what happens between different statuses remains unknown.

3.4.2 What activities are conducted during the bug fixing process?

To answer RQ 3.2, we sampled 120 bug tickets and identified eight main types of activities that contributors conduct during bug fixing: (1) Discovery (2) Diagnosis (3) Assignment (4) Search (5) Correction (6) Closure (7) Awareness (8) Others. Table 3.1 lists these activities and explains the classification criteria.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discovery</td>
<td></td>
</tr>
<tr>
<td>present the issue</td>
<td>Use text/screenshots to describe the issue encountered as well as its environment.</td>
</tr>
<tr>
<td>request for further explanation of the issue</td>
<td>Request bug reporter to provide more details about the issue.</td>
</tr>
<tr>
<td>refine the issue description/summary</td>
<td>Modify the description or summary of the ticket.</td>
</tr>
<tr>
<td>discover new problems</td>
<td>Discover new bugs during the bug fix process.</td>
</tr>
<tr>
<td>discover remaining problems</td>
<td>Discover what problems still exist after applying the attempted fix.</td>
</tr>
<tr>
<td>Diagnosis</td>
<td></td>
</tr>
<tr>
<td>discuss the nature/cause</td>
<td>Discuss what causes the issue, when the issue starts to appear and what type (defect, enhancement, etc.) and component (template, multisite, etc.) the issue belongs to.</td>
</tr>
</tbody>
</table>

Table 3.1: Classification of contributors’ activities during bug fixing

Continued on next page
### Table 3.1 – continued from previous page

<table>
<thead>
<tr>
<th>Activity</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>discuss how to duplicate the issue</td>
<td>Discuss how to reproduce the issue, including environment settings and duplication steps.</td>
</tr>
<tr>
<td>accept the ticket</td>
<td>Accept the ticket and prepare to work on it.</td>
</tr>
<tr>
<td>discuss the solution</td>
<td>Discuss how to solve the issue; the solution can also include planned actions, the impact of the solution and guideline for the solution.</td>
</tr>
<tr>
<td>discuss the impact of the issue</td>
<td>Discuss how severe and important the issue is, how difficult to solve the issue, what problems will arise due to the issue.</td>
</tr>
<tr>
<td>discuss the scope of the new problems</td>
<td>Decide if the new issue found during the bug fixing is related to the current issue.</td>
</tr>
<tr>
<td>link to related/duplicated issues</td>
<td>Discover related/duplicated issues.</td>
</tr>
<tr>
<td>discuss when to fix</td>
<td>Discuss by which version or which date the issue should be fixed.</td>
</tr>
<tr>
<td>Q&amp;A related technical details</td>
<td>Discuss, introduce, or learn related techniques and specific skills.</td>
</tr>
<tr>
<td>check the issue status</td>
<td>Check the current status of the issue.</td>
</tr>
<tr>
<td>reopen the ticket</td>
<td>Reopen closed tickets.</td>
</tr>
<tr>
<td><strong>Assignment</strong></td>
<td></td>
</tr>
<tr>
<td>request somebody to execute a task</td>
<td>Ask someone if he/she could deal with an issue or test/review the patch.</td>
</tr>
<tr>
<td>claim a task for themselves</td>
<td>A contributor claims he/she will/is doing some task, or accepts a request.</td>
</tr>
<tr>
<td>assign a task to somebody</td>
<td>A contributor assigns a task to someone else.</td>
</tr>
<tr>
<td>discuss contributors responsibility</td>
<td>Discuss which contributor takes charge of which part of the issue.</td>
</tr>
<tr>
<td><strong>Search</strong></td>
<td></td>
</tr>
<tr>
<td>look for related information</td>
<td>Discover information which help solve the problem, including online tutorials, documents, previous experience, etc.</td>
</tr>
<tr>
<td>look for/refer to a relevant person</td>
<td>Discover people who are familiar with the issue.</td>
</tr>
<tr>
<td><strong>Correction</strong></td>
<td></td>
</tr>
<tr>
<td>add a patch</td>
<td>Upload a patch to Trac.</td>
</tr>
<tr>
<td>explain a patch</td>
<td>Use text/screenshots to explain the patch (including what the patch does and its impact.)</td>
</tr>
<tr>
<td>request a new/updated patch</td>
<td>Request patch author to update the patch.</td>
</tr>
<tr>
<td>commit to fix this bug</td>
<td>Commit the patch to the source code repository.</td>
</tr>
<tr>
<td>link to relevant commit</td>
<td>Refer to the commit which fixes a relevant bug.</td>
</tr>
<tr>
<td>decide which patch to commit</td>
<td>Discuss which patch to use for fixing the bug.</td>
</tr>
<tr>
<td>request for further explanation of the patch</td>
<td>Request patch author to give more information about the patch.</td>
</tr>
<tr>
<td>request for a review/test review/test a patch</td>
<td>Request contributors to review or test the patch.</td>
</tr>
<tr>
<td>update documentation</td>
<td>Give opinions/suggestions/feedback to a patch/commit.</td>
</tr>
<tr>
<td><strong>Closure</strong></td>
<td></td>
</tr>
<tr>
<td>decide whether to close</td>
<td>Discuss if the issue should be closed, or suggest to close the ticket.</td>
</tr>
<tr>
<td>close the ticket</td>
<td>Close the ticket and set the resolution.</td>
</tr>
<tr>
<td>link to follow-up tickets</td>
<td>Refer to new tickets for remaining/newly discovered problem.</td>
</tr>
<tr>
<td><strong>Awareness</strong></td>
<td></td>
</tr>
<tr>
<td>announce the bug status</td>
<td>Announce what activities has been finished, e.g., announce a patch has been uploaded.</td>
</tr>
<tr>
<td>announce the contributor’s status</td>
<td>Announce if the contributor is available.</td>
</tr>
<tr>
<td>introduce the ticket into discussion group</td>
<td>A contributor post ticket number or the link of the issue to start a discussion, and a bot displays the summary of the ticket.</td>
</tr>
<tr>
<td>remind others to do something</td>
<td>Remind other contributors to execute some tasks.</td>
</tr>
<tr>
<td><strong>Others</strong></td>
<td></td>
</tr>
<tr>
<td>organize a talk/meeting</td>
<td>Organize a talk or a meeting to discuss relevant problems or make the plan for bug fixing.</td>
</tr>
<tr>
<td>express emotion/appreciation</td>
<td>Express personal feelings or thank others for their help/work.</td>
</tr>
<tr>
<td>welcome new members</td>
<td>Welcome people who open the first ticket and introduce Trac and the WordPress community.</td>
</tr>
</tbody>
</table>

Continued on next page
CHAPTER 3. BUG FIXING ACTIVITIES

In our study, we used the categories from Aranda et al. [14] to classify developers’ activities during bug fixing. Our results indicate that their categories did not cover all the activities. These activities are more related to social and organizational aspects, e.g., “organize a talk/meeting” and “express emotion/appreciation”. Open-source software projects usually have a higher developer turnover rate than closed-source commercial software projects, and helping newcomers onboard is an important activity for keeping the community prosperous. Therefore, we could expect some new activities regarding helping newcomers onboard, such as “select fist bugs for new members” and “welcome new members”.

Compared to typology proposed by Crowston and Scozzi [25], we considered activities about developers’ awareness while they did not. Maintaining group awareness is vital for smooth and effective collaboration [28], thus, it is worthwhile to pay attention to developers’ awareness-related activities. Moreover, similar to the categories from Aranda et al., Crowston and Scozzi did not include social and organizational activities. Furthermore, we also found several activities were conducted more on the communication tools, for example, they were more likely to “look for/refer to a relevant person” and “request somebody to execute a task” on the communication tools. Communication tools also allow developers to “remind others to do something”, which did not happen on Trac.

3.5 Conclusions and Future Work

3.5.1 Conclusions

In this study, we first investigate the limitation of bug statuses for understanding developers’ activities and we find four types of limitations in terms of completeness, synchrony, correctness and explicitness. We then analyze what activities are conducted by developers during bug fixing and compare our results with other research. Our new typology can serve as a necessary supplement for previous studies.

3.5.2 Limitations

Although our findings are insightful, some limitations still exist. First, the number of data samples for analysis is not large, which might introduce bias into our results. Second, we did not consider the effect of evolution of bug fixing practices when we were studying developers’ activities. However, the way developers collaborate to fix bugs might be different from 10 years ago. Since we only studied latest bugs that met our selection criteria, some activities only commonly conducted long time ago might not be disclosed. Third, different communities might have different regulations and standard procedures for bug fixing, the results would not be exactly the same if we apply our methods into other communities.
3.5.3 Future Work

Although we have found the limitations of the bug status for studying developers’ activities, we did not propose a solution to address the problem. We expect future work on automatic identification and correction of wrong statuses.

We would also be interested to study whether developers’ activities evolve over time and whether the adoption of new tools changed their bug fixing practice.

As a continuation of this study, we also anticipate that we could apply our methods to other open-source software communities and examine the generalization ability of our results.
Chapter 4

Bug Fixing Process

4.1 Introduction

Understanding the process of bug fixing can assist to learn the modern software development practices followed. Mining the process can help us validate the designed process model [29], and discover the inefficiencies and inconsistencies during bug-fixing [30].

In this chapter, we investigate the following research questions:

RQ 4.1 How can we identify the bug fixing process? What can we learn from the bug fixing process?

RQ 4.2 What are communication tools mainly used for during bug fixing?

RQ 4.3 How do community leads distribute tasks during bug fixing?

To identify the bug fixing process, we first extract activities automatically from bug tickets and convert these activities into an event log. We feed the event log to process mining tools and obtain the process model of bug fixing. We analyze the process model and identify the bottlenecks as well as other notable phenomena.

To understand what communication tools are mainly used for, we analyze the incoming and outgoing activities of the tools and select the top five activities. We find Slack and IRC are used in a similar way but Slack outperforms in bug assignment.

To understand how community leads distribute tasks during bug fixing. We use social network analysis to analyze the whether community leads are divided into several small groups to fix same bugs or execute similar tasks. The results indicate community leads do not follow a clear task distribution policy regarding who to work with and what tasks to execute.

In the end of this chapter, we summarize our results, present the limitations of our research and describe potential future work.

4.2 Related Work

Process mining is a technique “to discover, monitor and improve real process by extracting knowledge from event logs readily available in today’s (information) systems” [31]. The discovered model of process mining can be either a process model (e.g., a Petri net [32], BPMN [33]) or models that describe other perspectives (e.g., a social network [34]).
CHAPTER 4. BUG FIXING PROCESS

In this section, we present the theory, related academic studies, and tools of two commonly used techniques in process mining: process model discovery and social network analysis.

4.2.1 Process Model Discovery

Theory

Process model discovery techniques can extract process model by automatically analyzing the imported event logs. Each item in the event logs is supposed to contain four elements: 1) case (i.e., a process instance), 2) activity (i.e., a well-defined step), 3) performer (i.e., the person who initiates or executes the activity), 4) timestamp. Several process model discovery algorithms have been proposed for mining event logs, including $\alpha$-algorithm, heuristics miner and genetic miner. However, since real-life processes are less structured and more complex, the discovered models are often “spaghetti-like”. Therefore, a new process mining approach was proposed to adaptively simplify the process, namely, fuzzy miner.

Process Model Discovery in Software Engineering

Process mining can help software engineers better understand, analyze and optimize their software processes.

Poncin et al. mined bug status changes to discover the bug life cycle. The results provided an initial view on how bugs were handled in practice and suggested the actual bug life cycle is different from the life cycle prescribed by an official guide.

Gupta et al. also employed process mining techniques to discover the bug fixing process. They compared the designed process with the real process and identified the inconsistencies between them. Moreover, they identified self-loops, back-forth transitions, and bottlenecks. They also applied process mining to the event logs extracted from multiple software repositories including bug tracking system, peer code review system and version control system. Similarly, bottlenecks and some interesting bug fixing patterns were discovered.

Tools

Several tools have been developed to discover the process model.

One of the most popular process mining tools is ProM. ProM is an open-source framework for integrating process mining techniques, which are implemented as plugins. It provides a standard environment such that users can easily install or update the plugins, operate with clear GUI, import data, and export results. Regarding the process model discovery, ProM provides plugins for most state-of-the-art process model discovery algorithms including $\alpha$-algorithm, heuristic miner, genetic miner and fuzzy miner.

Disco is another popular process mining tool. The core functionality of Disco is that Disco can discover process models automatically after importing event logs. The process discovery algorithm is based on the optimized Fuzzy miner. With Disco, users can easily check the process statistics, filter the event logs, analyze the performance based on duration, and visualize the process flow over time.
4.2.2 Social Network Analysis

Theory

The first step of social network analysis is to extract the social network from event logs. There are several metrics for discovering the social network \([45]\), for example, reassignment metric (i.e., if an activity is frequently reassigned from one individual to another), working-together metric (i.e., if performers frequently work on the same cases), and similar task metric (i.e., if performers frequently work on the same activities).

After obtaining the social network, there are many metrics available to measure the network \([46]\). Different metrics should be used for different purposes. For example, degree, which represents the number of vertices connected to a node, can be used to understand how popular a vertex is. Distance centrality, which measures the proximity of a vertex to the rest, can be used to understand the ability of the vertex to influence other nodes.

Social Network Analysis in Software Engineering

Software development can be modeled as self-organizing, collaboration, social networks rather than random network \([47]\).

Social networks are commonly used to represent developers’ collaboration, which is often related to the software success. Meneely et al. \([48]\) used social network analysis to understand the relationship between developers’ collaboration and software failure and found a strong correlation exists between the social network measures and failures. Wolf et al. \([49]\) converted team communication structures into a social network and predicted whether an integration would fail based on relevant network structure metrics. Sureka et al. \([50]\) demonstrated how social network analysis could be used to discover potential risks and vulnerabilities in the software.

Social networks are also used to analyze the structure of software project. Crowston et al. \([51]\) analyzed the social network of interactions represented in 62,110 bug reports and a negative relationship was found between the level of centralization and the project size, which indicated larger projects were more modular. Howison et al. \([52]\) inquired into the social structure of development teams over time. They found change of the distribution of centralization is uncommon and large project teams do not change more frequently.

Tools

Several tools has been developed to extract and analyze social networks.

The process mining tool ProM \([43]\) can also be used to extract the social network from the event logs.

There are some other tools that are widely used for social network analysis. Unlike ProM, these tools cannot analyze the event logs directly. That is, users need to manually generate the social network and these tools can assist visualizing and analyzing the network. NetworkX \([53]\) is a Python language package for social network analysis, which provides data structures for representing networks and can be used to calculate network properties and structure measures such as shortest paths and betweenness centrality. Gephi \([54]\) is another popular tool for social network analysis. Gephi provides abundant modules for social network analysis. The most important advantage to NetworkX is that it provides a user friendly GUI, which makes the analysis process more interactive and convenient.
4.3 Case Study Design

4.3.1 Research Questions

Process mining, a technique to discover the process, aims to automatically construct the models explaining the behavior observed in the event log \[55\]. Understanding bug fixing process can help us discover the abnormal behaviors, optimize bug fixing workflow and improve working productivity. Therefore, we propose the following questions:

**RQ 4.1 How can we identify the bug fixing process? What can we learn from the bug fixing process?**

Another aspect for investigating bug fixing process is the tools developers use. Vyas et al. \[56\] carried out fieldwork in a multinational software team to understand the use of bug tracking system by interviews and in-situ observations. However, the use of other tools for bug fixing has received scant attention. Currently, there are many studies on tools for software development \[57, 58, 59\], but few of the studies have investigated the tool use for bug fixing. Communication tools such as the mailing list, IRC and Slack are widely used for discussing bugs in the WordPress Community. We are curious what role these communication tools play during bug fixing. Thus, we propose the question:

**RQ 4.2 What are communication tools mainly used for during bug fixing?**

The WordPress community is led by two main avenues: the community leads and the community volunteers. The community leads consist of WordPress co-founder, lead developers and core developers with permanent commit access \[1\]. These community leads are the driving force of the community, and their management ensures that the community runs smoothly and successfully. We are curious about their managerial coordination, that is, we have the following question:

**RQ 4.3 How do community leads distribute tasks during bug fixing?**

More specifically, we would like to investigate 1) whether they are divided into several small groups which deal with bugs together; and 2) whether they are assigned to specific tasks during the bug fixing process.

4.3.2 Data Collection and Analysis

To study the bug fixing process, we used the data of 10-year Trac tickets collected in Section 3.3.2. Our approaches to analyzing data and answering research questions are as follows.

**How can we identify the bug fixing process? What can we learn from the bug fixing process?**

To answer RQ 4.1, we used process mining techniques to discover the process model of bug fixing. From RQ 3.1 we learned that using only bug statuses as activities to discover the bug fixing process had many limitations, thus, we intended to create new events to provide deeper insights into bug fixing and diminish the impact of bias in bug statuses. Since producing a generalized process model required a large number of cases, it is inefficient and unscalable to manually label all the tickets with activities. Therefore, we automatically extracted developers’ actions and changes of ticket fields and then converted them into activities.

For developers’ actions, we formatted the activity as “object:action”. For example, when a developer uploaded a patch, we extracted this activity and recorded it as “patch:add”. For

---

1https://make.wordpress.org/core/handbook/about/organization/#the-wordpress-core-team
ticket field changes, we formatted the activity as “field name:field content”. For example, when a developer changed the “resolution” field into “fixed”, we recorded this activity as “resolution:fixed”. However, for some fields, we did not care what content they set the field to because we only considered developers’ activities. For example, when a user set the milestone for a Trac ticket, we cared more about the developer’s action instead of the exact version he/she set. Thus, we recorded this activity as “milestone:set”. Another reason for such a manipulation was that the process model would be extremely complex if we created a new activity for each milestone. We also divided the changes of the “owner” field into “owner:self” and “owner:other” based on whether the developer assigned the Trac ticket to himself/herself.

The complete list of activities extracted can be found in Table 4.1.

Table 4.1: Activities extracted from Trac tickets

<table>
<thead>
<tr>
<th>Activity</th>
<th>Meaning</th>
<th>Frequency</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>commit:add</td>
<td>A patch has been committed to code repository</td>
<td>22,756</td>
<td>11.71%</td>
</tr>
<tr>
<td>component:set</td>
<td>The component of the ticket has been set</td>
<td>4,679</td>
<td>2.41%</td>
</tr>
<tr>
<td>focus:set</td>
<td>The focus of the ticket has been set</td>
<td>1,111</td>
<td>0.57%</td>
</tr>
<tr>
<td>keyword:2nd-opinion</td>
<td>An opinion from another person is needed</td>
<td>618</td>
<td>0.32%</td>
</tr>
<tr>
<td>keyword:close</td>
<td>The ticket is ready for closure</td>
<td>1,109</td>
<td>0.57%</td>
</tr>
<tr>
<td>keyword:commit</td>
<td>The patch has been reviewed and tested by a trusted member and is ready to commit</td>
<td>2,761</td>
<td>1.42%</td>
</tr>
<tr>
<td>keyword:dev-feedback</td>
<td>Feedback from a core developer or trusted members is needed</td>
<td>997</td>
<td>0.51%</td>
</tr>
<tr>
<td>keyword:early</td>
<td>The ticket is a priority, and should be handled early in the next release cycle</td>
<td>548</td>
<td>0.28%</td>
</tr>
<tr>
<td>keyword:good-first-bug</td>
<td>A good starting point for new contributors before tackling complicated bugs</td>
<td>134</td>
<td>0.07%</td>
</tr>
<tr>
<td>keyword:has-patch</td>
<td>A patch has been uploaded to Trac</td>
<td>7,730</td>
<td>3.98%</td>
</tr>
<tr>
<td>keyword:has-unit-tests</td>
<td>Unit tests have been added to the patch (corresponding to keyword:needs-unit-tests)</td>
<td>80</td>
<td>0.04%</td>
</tr>
<tr>
<td>keyword:i18n-change</td>
<td>String has been changed and translators need to be notified</td>
<td>50</td>
<td>0.03%</td>
</tr>
<tr>
<td>keyword:needs-codex</td>
<td>Documentation in the Codex needs updating or expanding</td>
<td>67</td>
<td>0.03%</td>
</tr>
<tr>
<td>keyword:needs-docs</td>
<td>Inline documentation for the code is needed</td>
<td>64</td>
<td>0.03%</td>
</tr>
<tr>
<td>keyword:needs-patch</td>
<td>A patch is needed.</td>
<td>2,468</td>
<td>1.27%</td>
</tr>
<tr>
<td>keyword:needs-refresh</td>
<td>The patch needs to be merged and re-submitted because it becomes invalid since the change of source code.</td>
<td>371</td>
<td>0.19%</td>
</tr>
<tr>
<td>keyword:needs-screenshots</td>
<td>Screenshots are needed because the UI has been changed by a commit/patch</td>
<td>25</td>
<td>0.01%</td>
</tr>
<tr>
<td>keyword:needs-testing</td>
<td>The patch is needed to be tested</td>
<td>1,274</td>
<td>0.66%</td>
</tr>
<tr>
<td>keyword:needs-unit-tests</td>
<td>The ticket has been reviewed, found to be desirable to solve but needs more unit test due to high risks of causing other issues</td>
<td>418</td>
<td>0.22%</td>
</tr>
<tr>
<td>keyword:reporter-feedback</td>
<td>Feedback from the bug reporter is needed</td>
<td>2,111</td>
<td>1.09%</td>
</tr>
<tr>
<td>keyword:ui-feedback</td>
<td>Feedback regarding user interface from the core team is needed</td>
<td>136</td>
<td>0.07%</td>
</tr>
<tr>
<td>keyword:ux-feedback</td>
<td>Feedback regarding user experience from the core team is needed</td>
<td>107</td>
<td>0.06%</td>
</tr>
<tr>
<td>milestone:set</td>
<td>The milestone of the field was set</td>
<td>23,625</td>
<td>12.16%</td>
</tr>
</tbody>
</table>

Continued on next page
## Chapter 4. Bug Fixing Process

### Table 4.1 – continued from previous page

<table>
<thead>
<tr>
<th>Activity</th>
<th>Meaning</th>
<th>Frequency</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner:other</td>
<td>A contributor has assigned the ticket to others</td>
<td>2,722</td>
<td>1.4 %</td>
</tr>
<tr>
<td>owner:self</td>
<td>A contributor has assigned the ticket to himself/herself</td>
<td>6,449</td>
<td>3.32 %</td>
</tr>
<tr>
<td>patch:add</td>
<td>A patch has been uploaded by a contributor</td>
<td>23,091</td>
<td>11.89 %</td>
</tr>
<tr>
<td>priority:set</td>
<td>The priority of the ticket is set</td>
<td>1,575</td>
<td>0.81 %</td>
</tr>
<tr>
<td>resolution:duplicate</td>
<td>The resolution of the ticket is set to duplicate</td>
<td>16,725</td>
<td>8.61 %</td>
</tr>
<tr>
<td>resolution:invalid</td>
<td>The resolution of the ticket is set to invalid</td>
<td>3,314</td>
<td>1.71 %</td>
</tr>
<tr>
<td>resolution:maybelater</td>
<td>The resolution of the ticket is set to maybelater</td>
<td>3,314</td>
<td>1.71 %</td>
</tr>
<tr>
<td>resolution:wontfix</td>
<td>The resolution of the ticket is set to wontfix</td>
<td>2,026</td>
<td>1.04 %</td>
</tr>
<tr>
<td>status:accepted</td>
<td>The ticket is accepted</td>
<td>909</td>
<td>0.47 %</td>
</tr>
<tr>
<td>status:closed</td>
<td>The ticket is closed</td>
<td>1,986</td>
<td>1.02 %</td>
</tr>
<tr>
<td>status:new</td>
<td>The ticket is created</td>
<td>24,647</td>
<td>12.69 %</td>
</tr>
<tr>
<td>status:reopened</td>
<td>The ticket is reopened</td>
<td>3,469</td>
<td>1.79 %</td>
</tr>
<tr>
<td>status:reviewing</td>
<td>The ticket is under reviewing</td>
<td>620</td>
<td>0.32 %</td>
</tr>
<tr>
<td>summary:change</td>
<td>The summary of the ticket has been changed</td>
<td>1,990</td>
<td>1.02 %</td>
</tr>
<tr>
<td>version:set</td>
<td>The version of the ticket has been set</td>
<td>3,820</td>
<td>1.97 %</td>
</tr>
</tbody>
</table>

Meanwhile, we mapped the activities extracted from the Trac into the activities we observed in Section 3.1 such that we could know what activities could be revealed and what activities were still hidden when using an automatic method to extract activities. The mapping scheme can be seen as Table 4.2. Although the automatically extracted activities do not cover all activities we labeled, it has greatly enriched the activities represented by only bug statuses. We extracted process models based on these automatically extracted activities.

### Table 4.2: Activities automatically extracted from Trac and the related activities observed from labeling Trac and communication tools

<table>
<thead>
<tr>
<th>Automatically generated activity</th>
<th>Corresponding manually labeled activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>commit:add</td>
<td>commit to fix this bug</td>
</tr>
<tr>
<td>component:set</td>
<td>discuss the nature/cause</td>
</tr>
<tr>
<td>focus:set</td>
<td>discuss the nature/cause</td>
</tr>
<tr>
<td>keyword:2nd-opinion</td>
<td>request for a review/test</td>
</tr>
<tr>
<td>keyword:close</td>
<td>decide whether to close</td>
</tr>
<tr>
<td>keyword:commit</td>
<td>decide which patch to commit</td>
</tr>
<tr>
<td>keyword:dev-feedback</td>
<td>request for a review/test</td>
</tr>
<tr>
<td>keyword:early</td>
<td>discuss when to fix</td>
</tr>
<tr>
<td>keyword:good-first-bug</td>
<td>select first bugs for new members</td>
</tr>
<tr>
<td>keyword:has-patch</td>
<td>add a patch</td>
</tr>
<tr>
<td>keyword:has-screenshots</td>
<td>explain a patch</td>
</tr>
<tr>
<td>keyword:has-unit-tests</td>
<td>review/test a patch</td>
</tr>
<tr>
<td>keyword:i18n-change</td>
<td>announce the bug status</td>
</tr>
<tr>
<td>keyword:needs-codex</td>
<td>update documentation</td>
</tr>
<tr>
<td>keyword:needs-docs</td>
<td>Inline documentation for the code is needed</td>
</tr>
<tr>
<td>keyword:needs-patch</td>
<td>request a new/updated patch</td>
</tr>
<tr>
<td>keyword:needs-refresh</td>
<td>request a new/updated patch</td>
</tr>
<tr>
<td>keyword:needs-screenshots</td>
<td>request for further explanation of the patch</td>
</tr>
<tr>
<td>keyword:needs-testing</td>
<td>request for a review/test</td>
</tr>
<tr>
<td>keyword:needs-unit-tests</td>
<td>request for a review/test</td>
</tr>
</tbody>
</table>

Continued on next page
In real practice, several actions might be taken at the same time, e.g., the resolution was set at the same time as ticket closure and process mining techniques might give them a random order. That is, we might get a process model that setting the bug resolution follows ticket closure. Since we regard the bug opening as the first phase of a bug’s lifecycle and bug closure as the last phase, we need to manually adjust the time of bug opening and closure to ensure that they were in the right places we expected. To achieve this goal, We performed a pre-processing step before generating the event log: one second was subtracted from the time of ticket opening and one second was added to the time of ticket closure.

Figure 4.1 shows a small fragment of the event log.

```
<table>
<thead>
<tr>
<th>case</th>
<th>activity</th>
<th>timestamp (complete)</th>
<th>performer</th>
</tr>
</thead>
<tbody>
<tr>
<td>13315</td>
<td>status:new</td>
<td>2010-05-10 2:33</td>
<td>shdus</td>
</tr>
<tr>
<td>16652</td>
<td>status:new</td>
<td>2011-02-24 19:52</td>
<td>ryan</td>
</tr>
<tr>
<td>11666</td>
<td>status:new</td>
<td>2009-12-29 19:54</td>
<td>miqrogrove</td>
</tr>
<tr>
<td>13316</td>
<td>status:new</td>
<td>2010-05-10 6:01</td>
<td>filosofo</td>
</tr>
<tr>
<td>16653</td>
<td>status:new</td>
<td>2011-02-24 22:23</td>
<td>techotronic</td>
</tr>
<tr>
<td>11667</td>
<td>status:new</td>
<td>2009-12-29 20:47</td>
<td>bitplane</td>
</tr>
<tr>
<td>14985</td>
<td>status:new</td>
<td>2010-09-28 12:58</td>
<td>doktorbro</td>
</tr>
<tr>
<td>16654</td>
<td>status:new</td>
<td>2011-02-25 2:13</td>
<td>sayontan</td>
</tr>
<tr>
<td>13317</td>
<td>status:new</td>
<td>2010-05-10 9:29</td>
<td>hakre</td>
</tr>
<tr>
<td>16655</td>
<td>status:new</td>
<td>2011-02-25 2:33</td>
<td>pmenard</td>
</tr>
</tbody>
</table>
```

Figure 4.1: A fragment of the event log based on new bug statuses

We used Disco\(^2\) to discover the process model of bug fixing. We chose Disco rather than

\(^2\)https://fluxicon.com/disco/
other process mining tools because Disco could provide detailed statistics. It also provided powerful log filters which allow users to easily tailor the log to needs based on case performance, timeframe, variation, attributes, event relationships, or endpoints.

Our process analysis consists of two steps as follows. First, we prepare two event logs: one with only bug statuses, the other with automatically extracted statuses. We then feed them into Disco and generate the process model with the default settings. We compared these models with official Trac workflow and recorded discrepancies. Second, we inspected the median durations between connected activities and analyzed whether they were efficient based on our experience.

What are communication tools mainly used for during bug fixing?

To understand what communication tools are mainly used for during bug fixing, we summarized incoming activities (i.e., activities right before the tool use) and outgoing activities (i.e., activities right after the tool use) of the tools and counted their occurrences. We then selected top five incoming and top five outgoing activities based on the occurrences. We next analyzed these activities and made assumptions about the main use of tools.

We identified the use of the tools by the links to their chatting archives. During data analysis, we found mailing lists only connected to 142 activities while Slack and IRC connected 3895 and 1434 activities respectively. When the WordPress community used mailing lists as the main communication tool to discuss bugs, there was no integration for automatically posting the link to the mail archive on Trac. Therefore, we could identify the use of mailing lists only if developers manually posted the link on Trac. Thus, many information from mailing lists might be missing. Moreover, the small number of mailing list occurrences made it incomparable with IRC and Slack. Thus, we only discuss Slack and IRC for this question.

How do community leads distribute tasks during bug fixing?

When a group of people executes tasks, two approaches for task distribution are often used: 1) people are divided into small groups to execute the whole process of the task; 2) people are divided into groups that execute specialized sub-tasks, namely flow production [60]. To understand whether it is the case for the bug fixing, we decomposed the research question RQ 4.3 into two sub-questions: 1) Are community leads divided into groups to fix same bugs together? 2) Are community leads divided into groups to conduct similar bug fixing activities?

We used social network analysis to answer these questions. In detail, we used the working together metric to understand how frequently community leads are performing activities for the same tickets; and used the similar task metric to understand whether different groups of community leads are performing different clusters of activities.

The first step is to identify community leads. On Trac, community leads are presented with a special label such as “Core Committer” and “4.6 Release Lead” (Figure 4.2). When checking the source code of any ticket page, we can find the community leads are listed in the variable “wpTracContributorLabels”. Thus, we identified 40 community leads.
We first used ProM to generate the social network. To study whether community leads were divided into several groups to work on bug fixing, we used “Mine for a Working-Together Social Network” plugin, which constructed social network based on if performers frequently worked on the same cases. To study whether community leads focused on specific tasks, we used “Mine for a Similar-Task Social Network” plugin, which constructed social network based on if performers frequently worked on the same activities. The plugins could also group clusters. The idea behind this step is that if community leads were divided into several groups or focused on specific tasks, the obtained social network would show clusters. For example, if the social network obtained from “Mine for a Working-Together Social Network” plugin contained five clusters, we could indicate that developers were divided into five groups and members in the group were more connected and had more chances to work with other members within the group.

By observing the social network diagram, we could only get an initial hypothesis and we would need to verify our observation. We need a metric to quantitatively assess the possibility of dividing the network into clusters. We used “modularity” metric to evaluate the social network decomposition [61]. “Modularity” can be either positive or negative. When “modularity” is negative or zero, the social network is unable to divide. Positive values indicate the possible presence of the clusters. “Modularity” is always less than 1, and high “modularity” value indicates a strong cluster structure. In practice, the “modularity” value of networks with a strong cluster structure is typically between 0.3 and 0.7 [62].

Since ProM did not support exporting the social networks and it did not provide any mechanisms to analyze the structure of social networks, we had to use another tool for the network analysis. We wrote a script to construct the same networks as ProM and then fed it into Gephi. We used Gephi because it provided a convenient and interactive interface such that we could easily perform calculation and read the results. Since Gephi already integrated “modularity” module, we could easily get the modularity value by running “modularity” module.

4.4 Results

4.4.1 How can we identify the bug fixing process? What can we learn from the bug fixing process?

To answer RQ 4.1, we extracted activities from the bug status, contributors’ actions and change of Trac fields. We fed the event log with only bug status and the event log with all
activities into Disco respectively and obtained corresponding process models. We compared these process models with the official Trac workflow (Figure 2.2).

**Process model based on the bug status**

Figure 4.3 shows the process model based on the bug status. In the diagram, the triangle symbol represents the start of the process and the stop symbol illustrates the end of the process. The events are represented by boxes and the number stands for how many times these events occurred. The arrows stand for the transition from one event to another event. The two values near the arrow mean how many times these transitions happen and how long these transitions take (median value).

![Process model based on the bug status](image)

Figure 4.3: Process model based on the bug status

When we compare this process model and the official Trac workflow, we can find following phenomena.

- The status of most bug tickets (86.2%) is changed to “closed” directly after “new”. However, it does not mean that no activities happen during two statuses, as we discovered in Section 3.4.1 that bug status is not completely recorded.

- Most of the time, the actual workflow follows the official workflow. However, some minor differences still exist. For example, in the real workflow model, we could not find the pattern of loop “accepted” albeit it exists in the official workflow.
• Mistakes occur when changing the bug status. The status of “new” is not always served as starting point: the status of one Trac ticket changed to “new” from “new”, 85 tickets changed to “new” from “reopened” and 75 changed to “new” from “assigned”. This fact backs our finding in Section 3.4.1 that sometimes the bug status is recorded incorrectly. However, consider these 161 wrong transition to the status “new” only count for 0.75% ((1+85+75)/21339) of all occurrences of status “new”.

Process model based on all automatically extracted activities

Although the process model based on the bug status can bring us some insights into the lifecycle of a bug, we are still not clear about what exactly happen between two statuses. Thus, to get a deeper understanding of contributors’ actions during bug fixing, we obtained a new process model based on activities automatically extracted as Figure 4.4. We also attached the file to the appendix as Figure A.1 for easy viewing.

In the process model, the sum of the incoming arrows is not necessarily equal to the number in the state or the sum of the outgoing arrows. Because the model is abstraction of the real process, it only presents frequent transitions and some uncommon transitions are omitted. We can imagine if the image display all the transitions, the process model will be “spaghetti-like” due to the huge number of cases and it will be hard for us to read the process model.

![Figure 4.4: Process model based on all automatically extracted activities](image)

What we can learn from the process based on all automatically extracted activities

By observing the process model carefully, we have found some notable facts.

• When the need for a patch is announced, developers respond and create the patch fast. As can be seen from Figure 4.5, the median duration between “keyword:needs-patch”
and “patch:add” is 13.2 hours, which means that more than half of the bugs can be fixed in less than one day when developers get notified of the bug.

• For at least one third of the bugs, developers upload the patch right after they report the bugs without any other activities on Trac and the median duration is less than one minute (Figure 4.6). The fact indicates many developers have already created a patch before reporting the bugs. To further verify this fact, we trimmed the event logs in Disco and got the process model between “status:new” and the first “patch:add”. We inspected the statistics of case duration of the trimmed process. The screenshot of partial statistics table can be seen in Figure 4.7.
We exported the table to Excel, and performed simple statistics. We found that 7159 out of the 11722 bug tickets with at least a patch uploaded their first patch within one hour after they opened the tickets. We then obtained the Pareto chart [63] for the durations of these 7159 tickets, which can be seen as Figure 4.8.

![Figure 4.8: Durations between bug opening and the first patch (only tickets with during less than one hour are presented)](image)

We can see that more than 5000 tickets, 42.7% of the bug tickets with at least a patch, get their first patch in three minutes. The result suggests that a large percentage of bugs’ lives are still hidden from the bug tickets.

- As can be seen in Figure 4.9, 4719 of 64449 self assignment come at the same time as committing to the code repository, which indicates the timestamp of “owner:self” cannot be used as the time when developers accept the task and start working on the bug. Self assignment is more like a remark about who commit the patch.

![Figure 4.9: The transitions between “commit:add” and “owner:self” extracted from the process model](image)

- The bottleneck of the bug fixing process is collecting feedback. We can find three longest durations between connected activities. The median durations between three activit-
ies “reporter-feedback”, “keyword:2nd-opinion”, “keyword:dev-feedback” and their next activities are 12.5 days, 10.2 days, and 4.8 days respectively. One potential reason of this problem is that developers are not aware that it is their turn to take actions. A better communication and task assignment mechanism might help unblock the bottleneck.

4.4.2 What are communication tools mainly used for during bug fixing?

To answer RQ 4.2, we calculated the top 5 incoming and outgoing activities for the tool use. The results can be seen as Figure 4.10.

From Figure 4.10a, we can see that Slack is often used before the patch is created, and possible activities are diagnosing the cause and discussing the solution for the bug. Moreover, based on these incoming and outgoing activities we can infer Slack is often used for discussing when to fix the bug, who the bug should be assigned to, and which patch to commit.

From Figure 4.10b, we can see four of the five incoming activities and three of the five outgoing activities of IRC are same as Slack. We can thus claim that Slack and IRC are used in a similar way.

However, some differences still exist. For example, Slack is widely used to discuss the bug before the bug is assigned. One potential reason for these differences is that it is easier to search for people who can fix the bug on Slack. Because IRC is synchronous communication tool, developers need to be online to discuss the bug. Otherwise, they have to access the
separate site which stores chatting logs and search for the related discussion. Slack breaks through this limitation due to its asynchronous characteristic and the messages on Slack has much more chances to approach most of the developers who use the tool. Thus, we might assume Slack makes bug assignment easier. This assumption could be further validated by surveying or interviewing developers from the WordPress community.

4.4.3 How do community leads distribute tasks during bug fixing?

To understand whether community leads are divided into several groups to work on bug fixing, we extract the social network with “working together metric” as Figure 4.11.

![Figure 4.11: Social network extracted by the “Mine for a Working-Together Social Network” plugin](image)

In the social network, the presence of clusters is invisible. Thus, we could infer community leads are not divided into groups which fix bugs together. They are likely to fix bugs with any community lead. To verify our assumption, we calculated “modularity” in Gephi. As a result, the network was divided into two clusters with a “modularity” value 0.096, which
suggests the cluster structure is weak. Thus, we can back our assumption and state that there is no apparent evidence to support the clustering based on working-together metric.

To understand whether community leads are divided into groups to conduct similar activities, we extracted the social network with “similar task metric” as Figure 4.12.

![Figure 4.12: Social network extracted by the “Mine for a Similar-Task Social Network” plugin](image)

Similarly, the presence of clusters is also invisible in this network. Thus, we could infer community leads are not divided into groups which conduct similar tasks. They do not follow the practice of flow production. To further verify this assumption, we also calculated “modularity” and we found the network was divided into two clusters with a “modularity” value 0.006. As it is very close to 0, we can state that the network is almost unable to divide based on similar-task metric.

In summary, community leads do not have a clear task distribution policy regarding who to work with and what tasks to execute.
4.5 Conclusions and Future Work

4.5.1 Conclusions

In this chapter, we propose a new approach for extracting activities from bug tickets and retrieve the process model of bug fixing from the activities. We analyze the process model and identify notable phenomena such as big portions of developers have already created patches before they report the bugs. We also identify the process often gets stuck for a while when requesting others’ feedback.

We also analyze the popular incoming and outgoing activities of the tool use and find a similar use between Slack and IRC, however, Slack is widely used before bug assignment and it might have advantages in finding developers to fix the bug.

We inspect how community leads collaborate during the bug fixing process and find they do not follow a specific task distribution policy about who to work with and what tasks to execute.

4.5.2 Limitations

Limitations exist in our study. When extracting activities from the bug tickets, some of the activities we identify are not covered although we have tried to retrieve as many activities as possible. Furthermore, when analyzing the use of tools, we only get some assumptions but do not have a chance to verify them. However, our assumptions can serve as a confirmatory question of further study. When inspecting community leads’ task distribution patterns, we do not consider the impact of lead team personnel changes.

4.5.3 Future Work

As an extension of our work, future work include:

- Surveying and interviewing WordPress community members to get more insights at a deeper level. We would like to further validate our findings with the surveys and interviews. We would also like to explore more aspects of bug fixing, such as the rationales of choosing and switching tools to work with.

- Improving activity extraction techniques. Text processing techniques might be used to automatically recognize the content of bug report comments and retrieve the relevant activities.
Chapter 5

Knowledge Flow in Bug Fixing

5.1 Introduction

In the field of software engineering, the main asset a software organization has is the knowledge held by employees [64]. Development teams use different media channels including emails and instant messaging (IM) to support the transfer of knowledge and the channels are constantly evolving over time [65].

Since knowledge is a broad concept, in our study, we mainly focus on explicit individual knowledge, including private tools and repositories. In this chapter, we investigate the following research questions:

RQ 5.1 Why and how are bug tickets connected on Trac?
RQ 5.2 What types of knowledge from external resources are shared on Trac?

We identify reasons why developers refer to one ticket in another and categorize the reasons into seven types: 1) similar issues, 2) duplicate bugs, 3) beneficial patches, 4) follow-ups, 5) cause of the bug, 6) clustering, 7) suggestions for investigation. We also check whether most tickets are linked to other tickets from the same components. The results were negative.

We identify external links other than WordPress Trac and categorized their types into six types: 1) communication, 2) bug related demonstration, 3) community maintenance, 4) code maintenance, 5) bug related technical information, and 6) tool reference. Based on the distribution of external resource types, we propose two approaches regarding tools and onboarding newcomers to improve the bug fixing practice.

In the end of this chapter, we summarize our results, present the limitations of our research regarding tool evolution and describe potential future work.

5.2 Related Work

Knowledge can be either explicit (i.e., formal, systematic and can be easily shared and communicated) or implicit (i.e., less formal, not well structured, and cannot be easily articulated). Based on where the knowledge resides, knowledge can also be either individual or collective (i.e., shared). In the field of software engineering, knowledge can be classified into four categories [66]:

Bug Fixing in Action: Activities, Process, and Knowledge 37
CHAPTER 5. KNOWLEDGE FLOW IN BUG FIXING

- **explicit individual knowledge** includes private tools and repositories.
- **explicit collective knowledge** includes development methodologies/techniques, IEEE standards, tailored inspection process and software quality models.
- **implicit individual knowledge** includes subjective software development experiences (e.g., debugging, design, testing).
- **implicit collective knowledge** includes success/failure stories of project teams and team culture.

Knowledge is commonly formed, shared, manipulated and captured on media (i.e., development tools and communication channels) [67]. A case study on R community indicates that the community uses multiple channels efficiently at the same time to share and curate knowledge as each channel has its own advantages and disadvantages [68].

To understand how knowledge is shared during the software development process, the knowledge flow model was proposed to present peer-to-peer knowledge sharing and management in cooperative teams [69]. Grasping how knowledge flows in a software organization helps to locate the areas which can be improved and inspire ideas for approaches to the improvements [70]. A few efforts have been made on extracting knowledge flows in software engineering. Gendreau et al. [71] proposed a model to present the knowledge flow in software project development, which linked knowledge sources to five basic cognitive factors: acquisition, validation, synchronization, realization and crystallization. Mitchell et al. [72] presented a knowledge mapping technique to model software project-level knowledge flows.

5.3 Case Study Design

In this section, we introduce the research questions regarding knowledge flow in bug fixing, and explain how we collected and analyzed the data.

5.3.1 Research Questions

Our main research goal is to understand how knowledge flows during bug fixing process. However, since the whole bug fixing process is full of different types of knowledge, to make our study more organized and logical, we focus on “explicit individual knowledge”, mainly involving private tools and repositories.

We investigate the knowledge flow from two perspectives: 1) knowledge flow between Trac tickets, and 2) knowledge flow between Trac and external resources.

Regarding knowledge flow between Trac tickets, we propose the following research question:

**RQ 5.1 Why and how are bug tickets connected on Trac?**

Bug tickets are not isolated islands. We can see that other tickets are frequently referred to during the bug fixing process. We would like to study for what purposes developers refer to other Trac tickets. Since components divide the tickets into several groups based on the functionality, we might assume those tickets within the same component are more connected. We would like to examine whether most Trac tickets refer to those tickets which belong to the same component.
Regarding knowledge flow between Trac and external resources, we propose the following research question:

**RQ 5.2** What types of knowledge from external resources are shared on Trac?

Trac, which integrates all kinds of external information, is the main hub for bug fixing in the WordPress community. We wonder what types of external knowledge artifacts are shared on Trac.

### 5.3.2 Data Collection and Analysis

To study the knowledge flow in bug fixing, we used the data of Trac tickets produced in the last 10 years and collected in Section 3.3.2. Our approaches to analyzing data and answering research questions are as follows.

**Why and how are bug tickets connected on Trac?**

To answer RQ 5.1, we conducted two studies.

We first collected latest tickets which contain links to other tickets. During the data collection, we removed duplicate ticket references caused by quotation. To analyze the why developers refer to other tickets in Trac, we followed a qualitative analysis method: we started with 30 tickets and manually inspected the context of the ticket reference and wrote down the reason, and categorized it. We then analyzed more tickets and iteratively formed emergent reasons until no new reasons emerged.

We then used all the bug tickets which contained links to other bug tickets to examine whether most Trac tickets were referred by those which belonged to the same component. This goal could be easily achieved by comparing the components of the tickets which referred to other tickets and tickets which were referred.

**What types of knowledge from external resources are shared on Trac?**

To answer RQ 5.2, we collected 100 tickets which contained links to external resources other than Trac system of WordPress. During the data collection, we removed duplicated external links caused by quotation. Moreover, links in the error messages were abandoned, since it usually contained server host information and bug reporters did not expect developers to access those links to obtain information. Furthermore, dummy links, which are made up to explain the bug (e.g., example.com), were also removed. We followed a qualitative analysis method used for the previous research question: we manually inspected the content of the external resources and categorized these resources based on their functionality.

### 5.4 Results

To understand how knowledge flows in bug fixing, we first identified why developers referred to other Trac tickets, and examined whether most of the Trac tickets referred to other tickets which belonged to the same component. We then investigated and categorized what external resources other than Trac were presented by developers. In this section, we present our findings.
5.4.1 Why and How do bug tickets are connected on Trac?

We identified several typical reasons for ticket references on Trac.

- **Similar issues.** The two tickets discuss similar but not exactly the same issues, or they share part of the problems.

- **Duplicate bugs.** One ticket is the duplicate of another ticket, i.e., they discuss the same issue.

- **Beneficial patches.** The patch of one bug can help fix the other bug to a certain extent.

- **Follow-ups.** One ticket is proposed as the follow-up of the other ticket because some remaining issues have not been resolved.

- **Cause of the bug.** The bug in one ticket is caused when fixing the bug in the other ticket.

- **Clustering.** Sometimes, Trac tickets are created to discuss several similar/related existing tickets.

- **Suggestions for investigation.** Developers sometimes refer to tickets with a specific attribute and suggest another developer to inspect those tickets. For example, developers refer to tickets with keyword “good-first-bug” for newcomers such that newcomers can start with not complex tickets first.

We identified 2165 ticket references and checked whether the two tickets (referring and being referred to) belonged to the same component.

Table 5.1: How many Trac tickets refer to those tickets which belong to the same component

<table>
<thead>
<tr>
<th>Component</th>
<th>Number of cases</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same</td>
<td>1108</td>
<td>51.2%</td>
</tr>
<tr>
<td>Different</td>
<td>1057</td>
<td>48.8%</td>
</tr>
</tbody>
</table>

From Table 5.1 we can know that around half of the tickets are linked to the tickets which belong to the different components. The result contradicts our original assumption and suggests that Trac tickets from different components also play an important role when fixing bugs although the tickets have different functionality.

5.4.2 What types of knowledge from external resources are shared on Trac?

We identified 262 external links from the selected 100 Trac tickets. WordPress SVN is excluded in the links because it has been completely integrated to Trac system and become a part of Trac. We accessed these external links, inspected the content of the webpages and classified them into six categories based on their functionality.

- **Communication.** These links direct to chat logs of the communication tools. Since we used the latest tickets for analysis, Slack is the only tool the WordPress community used for discussing bug fixing. However, as we know, mailing lists and IRC were used before.
• **Bug related demonstration.** These links direct to resources used to demonstrate the bug before or after a patch. We identified three types of demonstration:

  – *Screenshots.* Developers upload screenshots to file or image sharing services (*e.g.*, Dropbox and Imgur).
  
  – *Screencasts.* Developers record the running scenario of the software when the bug occurs and upload it to video sharing website, such as Youtube.
  
  – *Demo websites.* Developers set up running website of the software and demonstrate the bug.

• **Community maintenance.** These links direct to the community or the support forum. They are used in two cases: 1) developers post links to the core contributor guide to inform newcomers how to contribute to the community; 2) developers ask reporters to post on support forum when they open an invalid ticket (*i.e.*, the ticket is a technical support request instead of a bug).

• **Code maintenance.** These links direct to the code, commits or code change logs related to the bug.

• **Bug related technical information.** These links direct to tutorials, API references, software manuals, discussions (on forums, Q&A sites or other issue tracking sites) or articles (often posted on personal websites or blogs) which are useful for fixing the bug.

• **Tool reference.** These links direct to the software, WordPress plugins, WordPress themes, third-party libraries or services.

Figure 5.1 shows some of the external resources linked to WordPress. Trac, which serves as a hub, connects different kinds of information.

![Figure 5.1: Examples of external resources linked to Trac](image)

We calculated the ratio of each type of external resources and the results can be seen in Table 5.2. We also visualized the results with a sunburst chart [73] as Figure 5.2.
Table 5.2: Percentage distribution of each type of external resources

<table>
<thead>
<tr>
<th>First-level category</th>
<th>Second-level category</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication</td>
<td>Screenshots</td>
<td>24.3%</td>
</tr>
<tr>
<td>Bug related demonstration</td>
<td>Screenshots</td>
<td>18.0%</td>
</tr>
<tr>
<td></td>
<td>Screencasts</td>
<td>0.4%</td>
</tr>
<tr>
<td></td>
<td>Demo sites</td>
<td>7.1%</td>
</tr>
<tr>
<td>Community maintenance</td>
<td>Core contributor handbook</td>
<td>6.7%</td>
</tr>
<tr>
<td></td>
<td>Support forum</td>
<td>4.7%</td>
</tr>
<tr>
<td>Code maintenance</td>
<td>Other issue tracking system</td>
<td>10.6%</td>
</tr>
<tr>
<td>Bug related technical information</td>
<td>Other issue tracking system</td>
<td>3.5%</td>
</tr>
<tr>
<td></td>
<td>Forum</td>
<td>3.9%</td>
</tr>
<tr>
<td></td>
<td>Blogs and articles</td>
<td>4.7%</td>
</tr>
<tr>
<td></td>
<td>Q&amp;A sites</td>
<td>1.2%</td>
</tr>
<tr>
<td></td>
<td>API references and software manuals</td>
<td>6.3%</td>
</tr>
<tr>
<td>Tool reference</td>
<td></td>
<td>8.6%</td>
</tr>
</tbody>
</table>

Figure 5.2: Distribution of different types of external resources

Based on these results, we suggest two approaches in which bug tracking systems and practices can be improved.
Better tool support for screenshots. Although bug reporters can upload screenshots to Trac, many of them still prefer to upload screenshots to third-party image sharing services. One possible cause of this behavior is that Trac only allows users to upload one attachment each time, which is inconvenient for users with multiple screenshots. Once the third-party image sharing services are down, the images on Trac will be broken. Thus, it is necessary to encourage users to upload screenshots to Trac directly.

Better support for onboarding newcomers. As we can see, over 10 percent of the external links are for community maintenance. If there is a mechanism on the Trac which guides users to learn how to use the system and report valid bugs, developers might save a lot of energy on dealing with inexperienced newcomers.

5.5 Conclusions and Future Work

5.5.1 Conclusions

In this study, we mainly focus on how knowledge flows between Trac tickets and what types of external knowledge aggregate on Trac. We first categorize reasons why developers refer to other Trac tickets and then prove tickets from different components were also often connected. Moreover, we classify external resources based on their functionality and propose two suggestions for improving the bug tracking system and practice.

5.5.2 Limitations

Like any empirical study, sample selection might introduce bias to our results. In our case, when studying external resources, we selected latest tickets as our study samples. However, since tools and services evolve fast, external resources popular now might not even exist several years ago. However, the results we observed can reflect the modern bug fixing practice.

5.5.3 Future Work

Since our samples are only a small part of the whole bug database, when studying external resources, we would expect heuristic approaches for identifying the types of external resources.

In our work, we mainly study explicit individual knowledge, however, knowledge in bug fixing covers much more than that. We would also like to research other types of knowledge such as implicit collective knowledge.
Chapter 6

Conclusions

In this thesis, we investigate bug fixing from three perspectives: bug fixing activities, the bug fixing process and knowledge flow in bug fixing.

Regarding bug fixing activities, we identify the limitations of using the bug status to understand the bug fixing activities and categorize them into four types: 1) incomplete statuses, 2) asynchronous statuses, 3) incorrect statuses and 4) implicit statuses. The impacts of these limitations are presented. We also identify developers’ activities during bug fixing and classified them into eight categories: 1) discovery, 2) diagnosis, 3) assignment, 4) search, 5) correction, 6) closure, 7) awareness and 8) others. Compared to other studies, the activities we identify highlight the awareness category and also represent activities from communication tools.

Regarding the bug fixing process, we first propose an approach to automatically extract activities from bug tickets and we then convert the activities into an event log and use Disco to retrieve the process model of bug fixing. By analyzing the process model, we identify the bottlenecks of the process and several notable facts about the process. We also analyze what Slack and IRC are used for and compare their use. At last, we explore the community leads’ collaboration pattern during the bug fixing process and find they do not have a clear task distribution policy in terms of who to work with and what tasks to execute.

Regarding knowledge flow in bug fixing, we first analyze the purposes that developers refer to other Trac tickets and identify seven main categories: 1) similar issues, 2) duplicate bugs, 3) beneficial patches, 4) follow-ups, 5) cause of the bug, 6) clustering and 7) suggestions for investigation. We also prove only half of the tickets refer to other tickets from the same components. We then classify external resources based on their functionality and propose two suggestions regarding tool support and onboarding newcomers.

Our study can serve as the basis for future bug fixing research. Our results shed lights on the modern bug fixing practices and provide suggestions for improving bug fixing process.
Bibliography


Bug Fixing in Action: Activities, Process, and Knowledge 49
BIBLIOGRAPHY


Appendix A

Process model based on automatically extracted activities