CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Evaluating the Characteristics of Code Reviewers and the Code Reviewed by Them in Open Source Projects

Master of Science Thesis in the Programme Software Engineering

TAHIR YOUSAF

KASHIF HABIB KHAN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2016

Evaluating the Characteristics of Code Reviewers and the Code Reviewed by Them in Open Source Projects

TAHIR YOUSAF

KASHIF HABIB KHAN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2016

Evaluating the Characteristics of Code Reviewers and the Code Reviewed by Them in Open Source Projects

TAHIR YOUSAF
KASHIF HABIB KHAN

Examiner: RICHARD BERNTSSON SVENSSON
Supervisors: JAN SCHRÖDER, CHRISTIAN BERGER

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Göteborg, Sweden 2016

Evaluating the Characteristics of Code Reviewers and the Code Reviewed by Them in Open Source Projects

TAHIR YOUSAF
KASHIF HABIB KHAN
Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg

Abstract

Open Source Software (OSS) solutions play an important role in software industry. People all around the world use open source applications in their daily life. Development practices in OSS usually don't follow established industry standards, teams are often distributed, and experience among team members varies greatly. Nevertheless, OSS has to fulfill the same quality standards as conventional software.

Within OSS, gatekeeping is the process of controlling quality in a way that contribution goes through a formal review. OSS use high experienced people (during code reviews) to review and control the commits of less experienced people. But it is not evident, if committers with more experience actually produce higher quality code.

In this study we investigate how experience influences the quality of code contributions. This shall enable us to get a better understanding how quality assurance processes in OSS work. This study is carried out to evaluate the characteristics of code reviewers and their contribution efficiency. The study is comprised of six different Apache projects and exploring the facts by using source code characteristics. The results of this study present interesting information about characteristics of code reviewers and contributions made by them. We investigate the relationship between contributor's experience and contribution efficiency. According to our study results, there is no correlation between contributor's experience and contribution efficiency. A developer with less experience can also provide efficient contributions. Results of this study can be useful for software professionals, managers and IT researchers.

Keywords: Code Review, Code Inspection, Code Reviewer, Gatekeeper, Open Source Software, Apache Software Foundation, Source Code Repository

Acknowledgement

We would like to show our appreciation and gratitude to all those who have been involved in the whole process of this thesis. We are very grateful to our supervisors Jan Schröder and Christian Berger. They provided us great guidance, innovative ideas and constructive feedback to complete this work. We also acknowledge the Department of Computer Science and Engineering for providing educational resources and valuable knowledge during master study.

Finally, we would like to thank our families and friends for their support and motivation throughout the study.

Tahir Yousaf & Kashif Habib Khan, Gothenburg 2016-03-10

Contents

List of Abbreviations

OSS	Open Source Software
ASF	Apache Software Foundation
RTC	Review Then Commit
CTR	Commit Then Review
SQL	Structured Query Language
JQL	Jira Query Language
GQM	Goal Question Metric
LOC	Lines of Code
OSSD	Open Source Software Development
QA	Quality Assurance
RQ	Research Question
UoA	Units of Analysis

1. Introduction

Open Source Software (OSS) solutions play a key role in the software industry and provide mission critical services to organizations. Development practices in OSS usually don't follow established industry standards, teams are often distributed, and experience among team members varies greatly. Nevertheless, OSS has to fulfill the same quality standards as conventional software.

Peer code review activity is an important quality assurance technique in both industrial development and the open source software (OSS) community [2] [5]. This technique is used in a semi-formal way or partially in commercial software projects which is an effective but expensive approach. On the other hand, open source community has resolved the financial barrier because they are self-motivated volunteers [1]. When Peer review is performed as part of each Software development process activity, it identifies defects that can be fixed early in the software development life cycle [4].

The focus of this study is the phenomenon of 'Gatekeeper' within open source software development (OSSD). As the name suggests, this role work as a gatekeeper to the project code base and maintains its quality. Every addition in the project code base goes by review through these persons. They can either accept, modify or reject a change submitted by developers.Code reviewers are usually more experienced persons and provide efficient contributions [6].
We use the terms 'Code reviewer' and 'Gatekeeper' alternatively in the next sections to relate it to the other studies.

In this study, we investigate the characteristics of code reviewers, in regard of their experience and contribution efficiency. We are interested in finding a way to measure their experience and contribution efficiency. The experience and efficiency measured in other studies are conducted for the developer role[26] [28]. Moreover, the methods use very few metrics for the measurement (such as lines of code for measuring developer experience and use file commits error ratio for measuring contribution efficiency) [26] [28]. We are interested to know how the experience and contribution efficiency vary among code reviewers using series of mining based metrics, and to find out the relationship between experience and contribution efficiency for these persons.

The process of code reviews and role of committer is more clearly defined in Apache Software Foundation (ASF)[1]. It is a major OSS community with several successful projects. By a collaborative and meritocratic development process, Apache projects deliver enterprise level,

[1] http://www.apache.org/foundation/how-it-works.html#roles

freely available software solutions that attract large communities of users [3]. They have a defined project role for performing the reviews.

We perform a case study by mining data from ASF issue tracker systems. We select the closed and fixed issues and the code file commits made for the resolution of these issues. Then we find out the characteristics of these code files. This case study consists of multiple units of analysis. The units of analysis are six projects from ASF with differences such as application domain, project size, team size and commonalities such as similar project organization for tasks, issues and code using the same issue tracker systems and code repository systems.

1.1 Goal and Research Questions

The goal of this case study is to investigate the characteristics of the code reviewer and the code where reviews are performed within OSS projects. Moreover, we investigate how these characteristics can impact the success of code reviews.

Studies show that involving code reviews result in improved software evolvability by making the code easier to understand and modify [9]. In [9], the authors classify the code review findings as functional and evolvability (structure, documentation and visual representation) defects, where the evolvability defects ratio is higher than functional defects. We investigate the effectiveness of code reviews in regard of improvement or quality of code contributions from the code files where the fixes are made to solve evolvability defects. We measure the experience of a developer on a project and the characteristics of code contributions by the developer. We also analyze whether experienced developer’s contributions are effective in term of source code quality.

In this study, we collect data about already performed code reviews from the Apache open source project's code base. This data is used to analyze the code reviewer effect on software quality. We also investigate if there is a relationship between the experience of a code reviewer and the success of code reviews. Respectively, we identified the following research questions:

1. How can a code contributor be classified in terms of project experience?
2. How can code contributions to the projects be characterized by assessing code/file characteristics?
3. How does contributor’s project experience correlate with code contributions?

1.2 Contributions

This study can be used to understand how effective is the role and characteristics of the code reviewer in the development of software systems. This information can be used while making the key decisions in the development of software projects, for example, when to introduce reviews in the project life cycle. The results of the study show the impact of code review in terms of code quality and contributor experience.

In this study, we measure contributors experience purely based on the source control system and using file characteristics of the commits. This technique can be used to measure contributor’s experience in an automated way. We contribute by providing methods to measure efficiency and experience of contributors with the help of different metrics and formulas.

1.3 Scope

Our study focuses on OSS because it is challenging how quality is maintained. We have chosen Apache Software Foundation (ASF) projects for our case study. The ASF is a non-profit corporation that works as a major organization with over 140 software projects that are released under the Apache open source license [3]. We investigate six Apache projects and collect the data related to resolved and closed issues of the project and code files associated with these bug fixes. ASF uses a bug tracking system known as JIRA for managing the issues related to projects. The projects we have selected for this study are shown in Table 1.1.

The projects are mainly selected on the bases of their recent activities, a project should have an active code repository with a commit activity in the week when data imported. The recent activity tells us that the project is an active project with regular contributions from the committers. We also considered the number of Total Issues, the number of Total Committers and length of the project. Project should not be a new project or very small project. These criterionsensure that we get reliable data and a large-enough data set, to perform significant statistical analyses.

Table 1.1: Selected Apache projects for study

Projects	**URLs**
Struts 2	http://struts.apache.org/
OFBiz	https://ofbiz.apache.org/
Felix	https://felix.apache.org/
Maven	https://maven.apache.org/
ActiveMQ	http://activemq.apache.org/
Sling	https://sling.apache.org/

1.4 Structure of Document

Chapter 1 presents background, research questions and scope of the thesis. Chapter 2 discusses related work; how similar studies are performed by other researchers. Chapter 3 includes the methodology, data collection and data analysis approach. This chapter also provides information, how the research questions are addressed. Chapter 4 presents results in detail. Afterwards, in Chapter 5 Analysis and Discussion are presented based on the results collected during this study. Finally, Chapter 6 gives the conclusion of this study.

2. Related Work

We have identified related studies on code reviews and their effectiveness on software quality. To find these studies, we performed searches using the keywords ‘Peer review’, ‘Code review’, ‘Code inspection’, ‘Code reviewer’, ‘OSS’ and ‘Apache Software Foundation’. For searching, we have selected the IEEE digital library, ACM digital library and Science Direct because these libraries are considered good sources for computer science related studies. Search strings using the keywords were created and used in the libraries to find related work. We decided for a combination of these three libraries, as we expect them cover a significant part of recent computer science literature.

A study shows that 27% of incorrect bug fixes made by contributors who have never worked before on source code files related with the fix [15]. Other finding suggests that the quality control should preferably perform on changes made by a single developer with limited prior experience [16]. As the project grows more complex, only few developers who have been involved actively over a certain period of time can fully understand software architecture and effectively contribute to its development [17]. The studies [15][16][17] raise a point that a new developer can cause problems instead of benefit and experienced developer can contribute effectively on software projects. To solve this problem, when people contributing with no or little prior experience, code is reviewed first to maintain the quality by code reviewers. These are normally the people who have more experience on any specific OSS project [6]. Their duties are to review every addition before committing to the project base.

In their study, Wahyudin et al. discuss quality assurance (QA) activities in OSS projects to focus on the questions, ‘What are the QA practices used in OSS projects?’ and ‘how do they perform such activities?’ [1]. They carried out a case study based on Tomcat, Myface two OSS projects. Tomcat is a pure volunteer project, whereas Myface is hybrid. They build a performance hypothesis in relation to both types of projects. They illustrated QA practices may differ in the different type of projects and the involvement of project community effects on QA practices. Moreover, they proposed a framework based on stakeholder interviews in the QA process which can be implemented in any OSS project [1]. The Framework has three processes group those are defect detection, defect verification and solution verification. Code self-review and team review are performed in the solution verification process.

Another study by Rigby and German [2], provides a good understanding of the peer review mechanism in Apache Server. The study is conducted by using archival records of email discussion and data repositories. They provided a comparison between two Apache review techniques; *review-then-commits* (RTC) and *commit-then-review* (CTR) as well as a comparison of Apache review to an inspection in the commercial project. This study is based on the data of one project from ASF and focusing on characteristics of code reviews. Findings show that both techniques RTC and CTR are not perfect in all environments. An optimal review process may be designed by using formal code reviews frequently on critical sections of code before project releases and quick review in the early development process.

In order to assure software quality, early detection of defects is highly recommended and code review is one of effective approach for early detection of defects [6][7]. In a study by Kemerer and Paulk [8], the impact of design and code reviews on software quality is discussed. This research shows that the quality of products depends upon the quality control techniques such as reviews, and the defect removal effectiveness of reviews depend on review rate. Review quality decrease when the review rate exceeds the recommended maximum of 200 lines of code (LOC) per hour [8]. This study verifies through results that code inspection can produce good quality results in the software development process.

An empirical case study by Rigby and Store [7] investigates the procedure and behavior that developers used to find which code patch is to be reviewed. Data is collected for five OSS projects and interviews are conducted with nine core developers working on these selected projects for this study. They describe how the patch is selected to perform the review and who to review it. They show interesting facts after interviewing developers that experienced committer in this area is normally reviewer and select the patch for review on the basis of area of interest and expertise. Also discussed characteristics of the code reviewer, divided into two types of personas positive and negative. Positive persons: objective analyzer (a reviewer provides criticism as questions and encourage the discussion), expert adviser (expert reviewers provide advices to new developers), enthusiastic support (reviewers provide good solutions and take ownership of committing patch). Whereas negative persons: grumpy cynic (experienced the member can become cynical when new developers suggest fail solutions), frustratedly resigned (when a discussion on review has been carrying extended period of time, a reviewer may resign from the discussion).

A study by Khanjani and Sulaiman [13] focuses on quality of review process in open source software. The author briefly describes the concept of quality assurance under Open Source Software Development (OSSD) model in general; furthermore, discussed the advantages and disadvantages of the OSSD model in relation to close source software. They say OSSD model technique is safer and faster than traditional technique to improve software quality. They find the two factors: code review, data testing is important in software quality. Nevertheless, they highlight the importance of peer review as a technique to improve the software quality.

A comparative case study by Asundi and Jayant [14] examines the process of the code review of different types of projects. They collected data from five open source projects and performed an analysis. Results show that core members of projects have a high ratio (90%) in the involvement of patch submission and review process for three projects. This study shows that some patches are not reviewed due to the incorrect format of submission according to documentation of the project. Based on their results, they observe that four projects have at least one response to each patch submission on average. That means every patch is reviewed at least once.

The term code review, code inspection and code analysis are used in the software industry for checking the quality of the code. Code inspections are beneficial for an additional reason and they make the code easier to understand and change [9]. This study by Siy and Votta [9] shows that 60% of all issues appeared in code reviews are not problems, but they improve the maintainability of the code by following coding standards and decrease code redundancies. In an OSS community project, where a large group of people contributes to the project, it is very important that the code is easy to understand and modifiable. Another study by Mantyla and Lassenius [10] affirms that code reviews are good for identifying the code defects because in later phases these cannot be found as they do not have effect on software's visible functionality.

While each OSS project has a core group of developers (committers) with write access to the code repository, new developers without this privilege can also make their contributions, mainly by submitting patches to project mailing lists [11]. The patch submission and acceptance process are critical to OSS communities [11]. It is not always immediately clear to whom to assign a submitted patch for review [12]. It can be challenging to find a good reviewer for a patch [12].

The papers are similar to our studies. These studies are about the code review process, patch submission and how code review effects on software quality in OSS projects. We could not identify studies assessing the characteristics of actual code reviewers as an example experience of contributors and their impact on the software development process.

3. Methodology

This chapter describes the research method for the thesis project. After research method description, data collection process is illustrated in detail. In the end of this chapter, Goal Question Metric (GQM) approach is used to describe the research questions in detail.

3.1 Research Method

We investigate the characteristics of code reviewers and the code where reviews are performed. We select six projects as cases to perform analysis. This thesis project focuses on exploring the phenomenon of a code reviewer in environment of OSS development. Our research strategy is a case study, performed on six different Apache projects. Our definition of case study is based on Stol and Fitzgerald [21]. They describe it as any research conducted in real-world setting, that focus on the specific phenomenon without changing the real environment is considered to be a field study or case study. The research process is following the recommendation by P. Runeson and M. Höst [22]. The steps are described below.

Case Study Design: The goal of this study and its preliminary research questions are defined. A review on existing literature is performed. Six projects are explored in this study and limitations are defined. The ASF projects are chosen as a domain of this study. See more detail about projects in Section 1.2.

This is a case study with multiple units of analysis (UoA) as shown in Figure 3.1. The units of analysis are six projects from ASF with differences such as application domain, project size, team size and commonalities such as similar project organization for task and issues using the same issue tracker system and code repository system. Each project has issues with the same attributes of information.

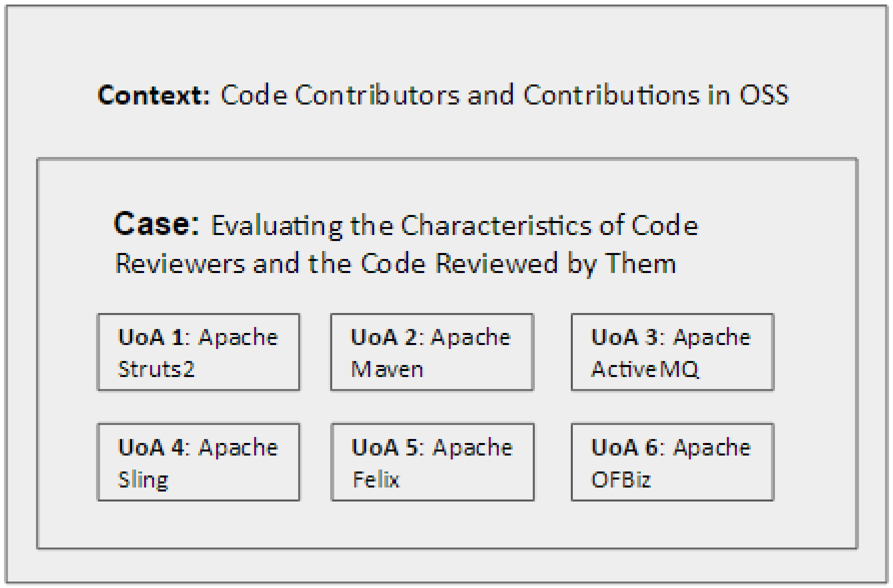

Context: Code Contributors and Contributions in OSS
Case: Evaluating the Characteristics of Code Reviewers and the Code Reviewed by Them
UoA 1: Apache Struts2
UoA 2: Apache Maven
UoA 3: Apache ActiveMQ
UoA 4: Apache Sling
UoA 5: Apache Felix
UoA 6: Apache OFBiz

Figure 3.1: Case study with multiple units of analysis (UoA)

Preparation of Data Collection: In this step, first data sources are investigated. The rules for selection of project are defined. The rules are included;

- A project should be active in recent time (should have commits in the week when the data import starts). This avoids investigating outdated and inactive projects.
- During preliminary investigations, we found that projects with less than 20 contributors are small projects with insufficient data. Hence, we have decided to limit ourselves to projects with more than 20 contributors. A project should have more than twenty contributors as the minimum limit.
- A project should have the minimum limit of 4, 000 issue records in issue tracking system.
- And project should have commit history over the couple of years.

Multiple sources and collection methods are defined for the study. Each project contains thousands of record, so we perform data collection in an automated way. We also perform manual data collection as a test data to build our automated data collection method and ensure the data reliability. Data collection processes are illustrated in Section 3.2.1 and Section 3.2.2.

Collecting Evidence: The data is collected from defined automated methods. Useful data is collected that help to address the research question of case study. Data can also be used to perform further analysis.

Analysis of Collected Data: Conclusion is derived from analysis and includes further possible research to enhance the case study. The threats of validity are analyzed. Data analyses include behavior or impact of contributor's experience on code contributions.

3.2 Data Collection

The first step in the data collection are investigations of available resources. The Apache Software Foundation (ASF) provides the following resources for possible contributions.

Mailing List: Each project has multiple mailing lists in ASF, where user can post a question, feedback, comment about configuration of project, issues and new feature suggestion. Contributors can post their messages in respective mailing list.

Issue and Bug Tracking: Each project uses their own Issue tracker [3] instance to record bugs or issue data. Apache typically use JIRA[2] and BugZilla as bug tracking systems, where contributors can find information regarding issues. These systems track information for issue data, for example Issue Date, Issue status, Issue Type, Update Date, Description, Assignee, Reported by etc.

Source Code Repositories: Apache uses SVN and Git repositories for source control. These repositories are accessible through a website Atlassian Fisheye6[3]. It is mirroring the subversion and Git repositories and it is integrated with issue tracker system JIRA. Realized as web service, Fisheye6 provides information about source code, fields of information include Issue ID, Commit Date, Committer ID, Number of files involved in particular commit, etc.

This study investigates the characteristics of code reviewer and assesses contributions on the project. We acquire the data from issue tracking systems and source code repositories. The data is collected for six Apache projects Struts2, OFBiz, Felix, Maven, ActiveMQ and Sling. Basic information about size and participation about the project is listed in Table 3.1. The data is collected using application scripts written in Microsoft C#.Net and stored in Microsoft SQL Server.

[2]https://issues.apache.org/jira/secure/BrowseProjects.jspa#all
[3]https://fisheye6.atlassian.com/browse

Table 3.1: Project facts

Project Name	Total No. of issues imported	Total No. of files commits	Total No. of contributors
Struts 2	4006	71357	25
OFBiz	5683	62119	33
Felix	4242	51851	42
Maven	4310	47321	44
ActiveMQ	5173	39497	64
Sling	4663	60832	32

3.2.1 Manual Approach for Data Collection

To prepare for an automated data collection process, first we gathered the required data by a manual process. The manual approach involves preparation of excel sheets manually from all possible data sources. Initially, we have collected data of issues from the Apache Struts2 project. We have prepared excel sheets from Apache resources to develop the understanding of data. It is also important to explore all possible or available fields of data by a manual process so that we know about these for automation. This approach is practiced for one project. Figure 3.2 depicts the process of the manual approach.

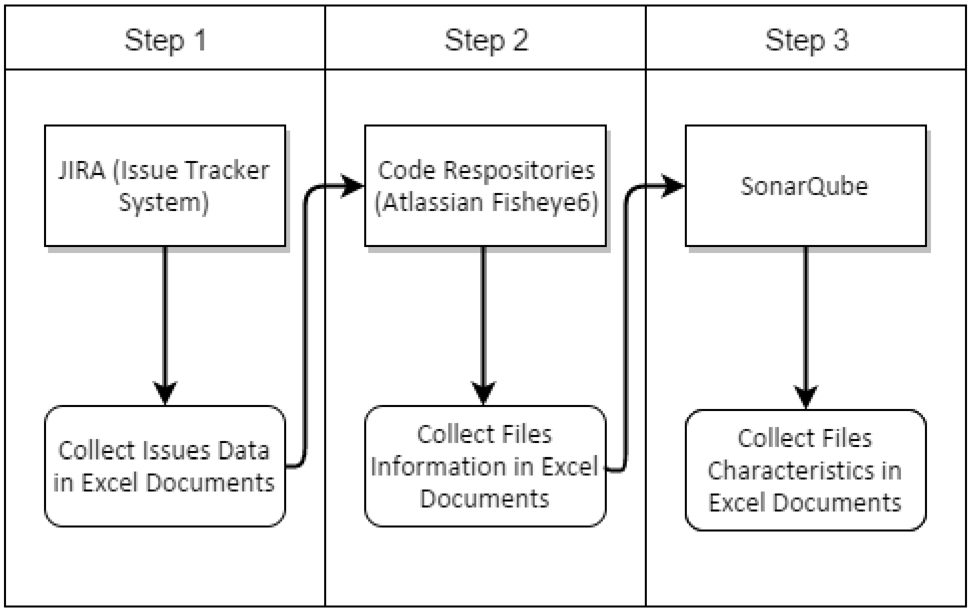

Step 1
Step 2
Step 3
JIRA (Issue Tracker System)
Code Respositories (Atlassian Fisheye6)
SonarQube
Collect Issues Data in Excel Documents
Collect Files Information in Excel Documents
Collect Files Characteristics in Excel Documents

Figure 3.2: Manual Data Collection Process

Step 1: Collection of Issues Data

The issue tracking system is used to track different kind of issues information depending upon how the tracking system is used in organizations. In ASF, an issue represents either a bug, an improvement, task, sub-task, new feature or a test.

We have made a query using JIRA Query Language (JQL) for extracting the issue's data and export them into excel files. There is a limit for export data into excel that is maximum 100 issues. So, we have designed JQL on yearly bases (keeping the records under 100) and exported issues into excel files. Excel files were made according to each year. In case yearly issues exceed from 100 records then we have divided into multiple files. We have prepared 52 files for the Struts2 project.

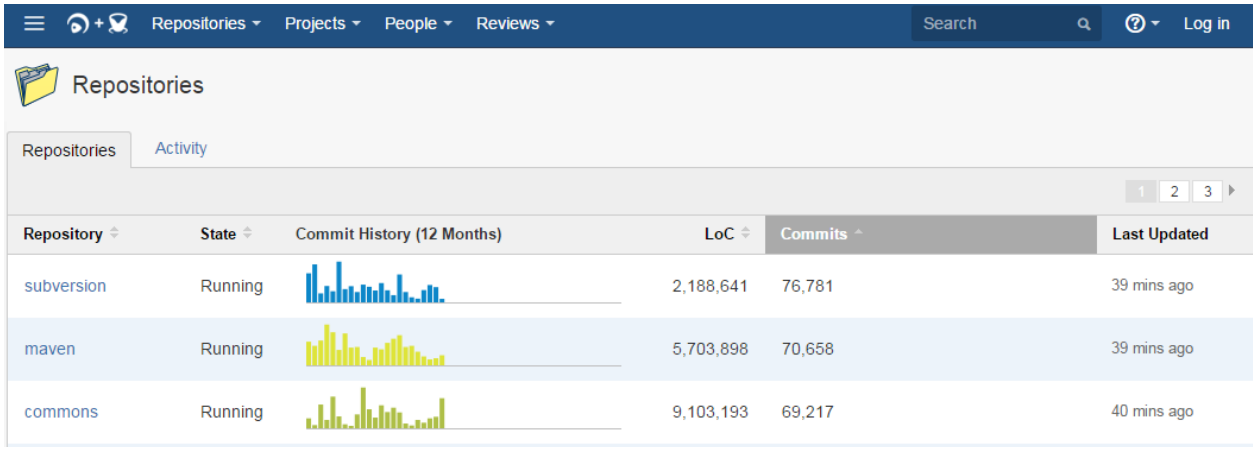
The excel files contains information about issues for example Project, Issue Key, Summary, Issue Type, Status, Priority, Resolution, Assignee, Reporter, Committer, Creator, Created Date, Updated Date, Resolved Date, Components, Linked Issues, Description, Date of First Response etc.

For retrieving the issues data, we have used the following source web url:

https://issues.apache.org/jira/browse/WW-4270?jql=project%20%3D%20WW%20AND%20status%20%3D%20Closed%20ORDER%20BY%20priority%20DESC

Step 2: Collection of File Commit Data

We can find commit information related to each issue obtained in step 1 through Atlassian Fisheye6. It is a web interface that provides information about commits for example Committer Username, Commit Date, Commit ID, Committed File Paths, etc. We have collected commit information against issues manually and saved them in excel files. Repositories in Atlassian Fisheye6 are linked with JIRA and its interface looks like as in Figure 3.2. We can see the time since last recent activity on the repository. For an example, Maven project have last update 39 minutes ago as shown in Figure 3.3, so it is an active project in the repository.



Repository	State	Commit History (12 Months)	LoC	Commits	Last Updated
subversion	Running		2,188,641	76,781	39 mins ago
maven	Running		5,703,898	70,658	39 mins ago
commons	Running		9,103,193	69,217	40 mins ago

Figure 3.3: Atlassian Fisheye6 Code Repositories with Commit History Information

For retrieving the commit information for each issue, we have used following source web url to explore the code files:

https://fisheye6.atlassian.com/browse/struts/core/src/main/java/org/apache/struts2/views/DefaultTagLibrary.java?r1=0aa0a69068c8dd7c61119f2a5baf8b9ab697c750&r2=9aedd857a4294a5091bce6abcdcb183f83833cb6

Step 3: Collection of Files Characteristics Data

Next step is collecting file characteristics for committed files through SonarQube [24]. It is a web tool to measure file complexity. Apache projects are configured with this tool to see the complexity of projects. We have manually collected the data from the SonarQube web links and saved it into excel files. The file characteristics include Lines of Code, Complexity, Number of Classes, Number of Functions and Complexity per Function.

3.2.2 Automated Approach for Data Collection

Using JIRA and Atlassian Fisheye6 web URLs, it is possible to automate the data collection process to gather the large amount of data for the research. We have executed web url's by writing web client applications to process the web requests and fill in a database for the analysis. Figure 3.4 shows the process of automated data collection in detail.

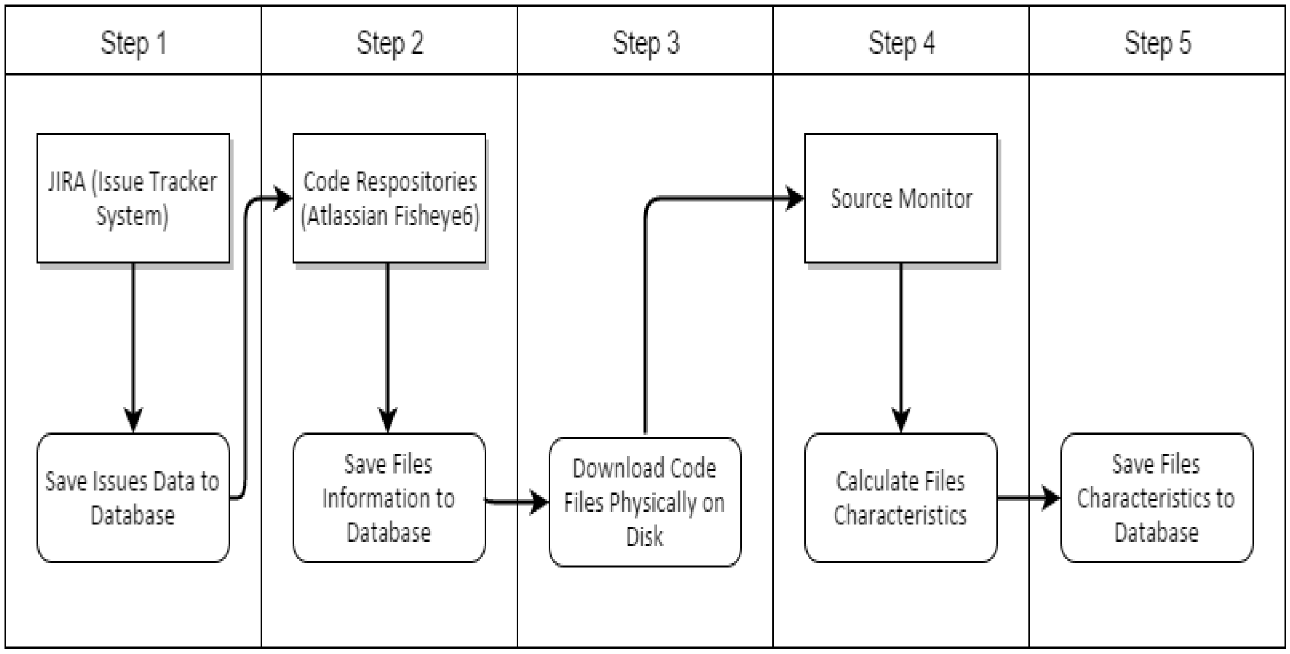

Step 1
Step 2
Step 3
Step 4
Step 5
JIRA (Issue Tracker System)
Save Issues Data to Database
Code Respositories (Atlassian Fisheye6)
Save Files Information to Database
Download Code Files Physically on Disk
Source Monitor
Calculate Files Characteristics
Save Files Characteristics to Database

Figure 3.4: Automated Data Collection Process

Step 1: Collection of Issues Data

We automated the collection of issue's data by generating web links dynamically for every month with the month start and end date, an example source link is below:

https://issues.apache.org/jira/sr/jira.issueviews:searchrequest-xml/temp/SearchRequest.xml?jqlQuery=project+%3D+"+apacheProject+"+AND+status+in+%28Resolved%2C+Closed%29+and+createdDate+%3E%3D+%27" + startDate + "%27+and+createdDate+%3C%3D+%27" + endOfMonth+ "%27&tempMax=200"

When we download XML from the above link, it provides us with the same information regarding issues' data as we have talked above (downloadable excel file) data in XML format. This required us to write a program which calls web request and download the XML file. Then parse the XML and find the related information. Next we save the information into our SQL

Server database. To process and filter all issues' data related to one project, we had to change these parameters to the above link for example Project Name, and date ranges on issue create date with the Max limit of records of 200 (from JIRA, it's only possible to export 100 records once but when we tried changing download link with a program to 200 records, it worked).We collected this data month-wise, because JIRA produces an error if more than 200 issues are requested at once. The collected data is saved in the SQL table for issue's data. Fields are shown in Table 3.2.

Table 3.2: Fields of information for Issues data

Issues Data Fields			
Project	Issue Key	Title	Issue Link
Summary	Issue Type	Status	Priority
Resolution	Assignee	Reporter	Created
Updated	Resolved	Affects Version	Fixed Version
Components	Linked Issues	Description	Labels
Flags	Date of First Response		

Step 2: Collection of File Commit Data

In the second step, we execute script for each issue key in an automated way with links like below,

https://fisheye6.atlassian.com/search/" + repository +
"/?ql=select%20revisions%20from%20dir%20%22%2F%22%20where%20comment%20matches%20%22
" + issueKey +
"%22%20order%20by%20date%20%20desc%20%20group%20by%20changeset%20return%20path%2C%20revision%2C%20author%2C%20date%2C%20csid%2C%20totalLines%2C%20linesAdded%2C%20linesRemoved&csv=true

The above link provides us with the file commit information for one issue in CSV format. This web request is made while saving the issue's information. Upon receiving the response from this web request, we parsed the CSV data to save it in the database for analysis. SQL table contains this information regarding file's data. Fields are shown in Table 3.3.

Table 3.3: Fields of information for files data

Files Data Fields				
Project	Issue Key	Revision	Author	File Path
Commit Date	Changeset ID	Total Lines	Lines Added	Lines Removed

Step 3: Downloading File Revisions Involved in Bugs Physically on Disk Storage

In this step, using a script we processed the saved file commits with file path's information in our database to download the actual files from the Fisheye6 database. It is the most time consuming process as it downloads thousands of files with multiple revisions.

We have downloaded all file versions involved in issues. We have downloaded the version of a file when an issue is introduced in a file and file version when the issue is resolved for that file. So we have a state of the file when an issue exists in the file and a state when it is fixed. Table 3.4 illustrates the before and after commit information with an example of a file 'Form.java'.

Table 3.4: An Example with File Before and After Commit Information

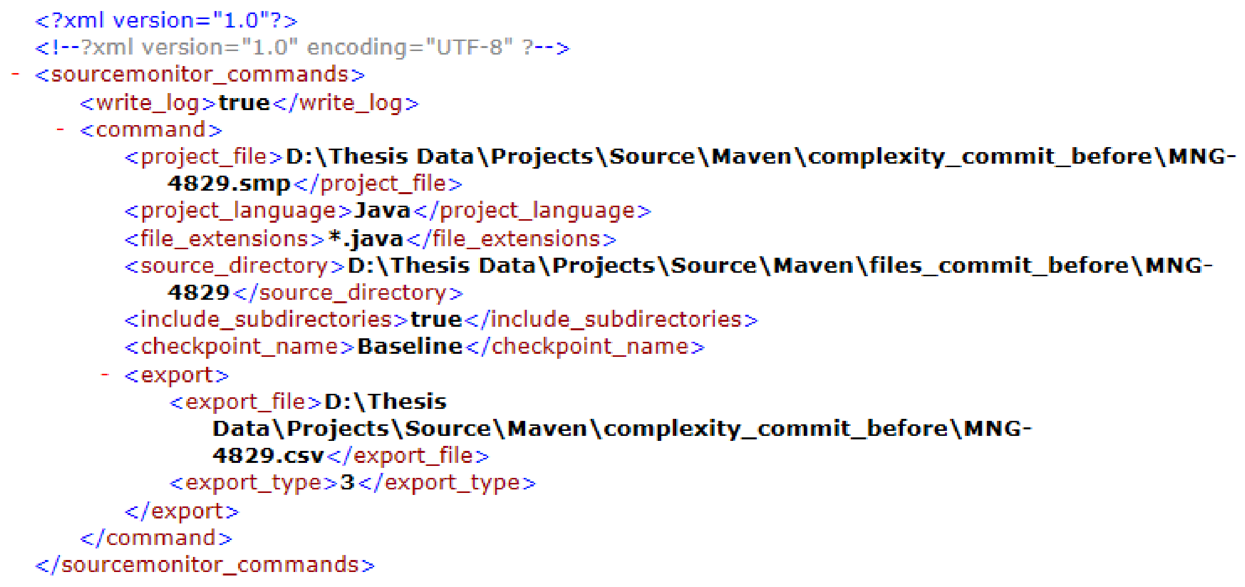
Commit Type	**Project**	**Revision**	**File Path**	**Commit Date**	**Total Lines**	**Lines Added**	**Lines Removed**
After commit	Struts 2	1485978	struts2/components /Form.java	2013-05-24 08:56	490	143	8
Before commit	Struts 2	1292705	struts2/components /Form.java	2012-02-23 08:40	355	1	1

Step 4: Calculating the Characteristics of Downloaded Files

After downloading the files involved in bug fixes, we have measured the characteristics of files using Source Monitor Tool [20]. We haven't used SonarQube [24] for the automated process, due to the reason it can only be configured with one version of a file, not for multiple versions of a single file, and second it requires to configure a complete project. For our study we need to check characteristics for specific files.

Source Monitor provides console interface for measuring characteristics of the large number of files programmatically. We have written a program to take each saved file as an input from the disk and calculate its complexity.

The console interface requires an XML configuration file (shown in Figure 3.5) as an input to generate the files with characteristic's data. This file takes the input of parameters like the path of the folder where the code files are placed, the path of the folder where it needs to save the characteristic's data file, the format of data (either XML or CSV), code file programming language. We have divided the process to measure characteristics by each issue. Source Monitor provides characteristic's information as output in CSV format.



```
  <?xml version="1.0"?>
  <!--?xml version="1.0" encoding="UTF-8" ?-->
- <sourcemonitor_commands>
     <write_log>true</write_log>
   - <command>
        <project_file>D:\Thesis Data\Projects\Source\Maven\complexity_commit_before\MNG-
           4829.smp</project_file>
        <project_language>Java</project_language>
        <file_extensions>*.java</file_extensions>
        <source_directory>D:\Thesis Data\Projects\Source\Maven\files_commit_before\MNG-
           4829</source_directory>
        <include_subdirectories>true</include_subdirectories>
        <checkpoint_name>Baseline</checkpoint_name>
      - <export>
           <export_file>D:\Thesis
              Data\Projects\Source\Maven\complexity_commit_before\MNG-
              4829.csv</export_file>
           <export_type>3</export_type>
        </export>
     </command>
  </sourcemonitor_commands>
```

Figure 3.5: Input configuration file for Source Monitor console application interface

Step 5: Saving Characteristics of Files into the Database

In this step, we parse the CSV data generated in step 4, and save it in the database using another console application written. We have created two tables with the same structure to add the characteristics of a single file. One table contains the characteristics when the issue was found while the other contains the characteristics when issue is solved. So that we can analyze the difference of characteristics before and after the fix. SQL tables contains the following information regarding characteristic's data. Fields are shown in Table 3.5.

Table 3.5: Fields of information for files complexity data

Files Characteristics Data Fields			
Project	Code Version	File Path	Number of Lines
Percentage Comments	Statements	Classes	Methods per Class
Max Complexity	Average Complexity	Max Depth	Average Depth
Average Statements per Method	Percentage Branch Statements		

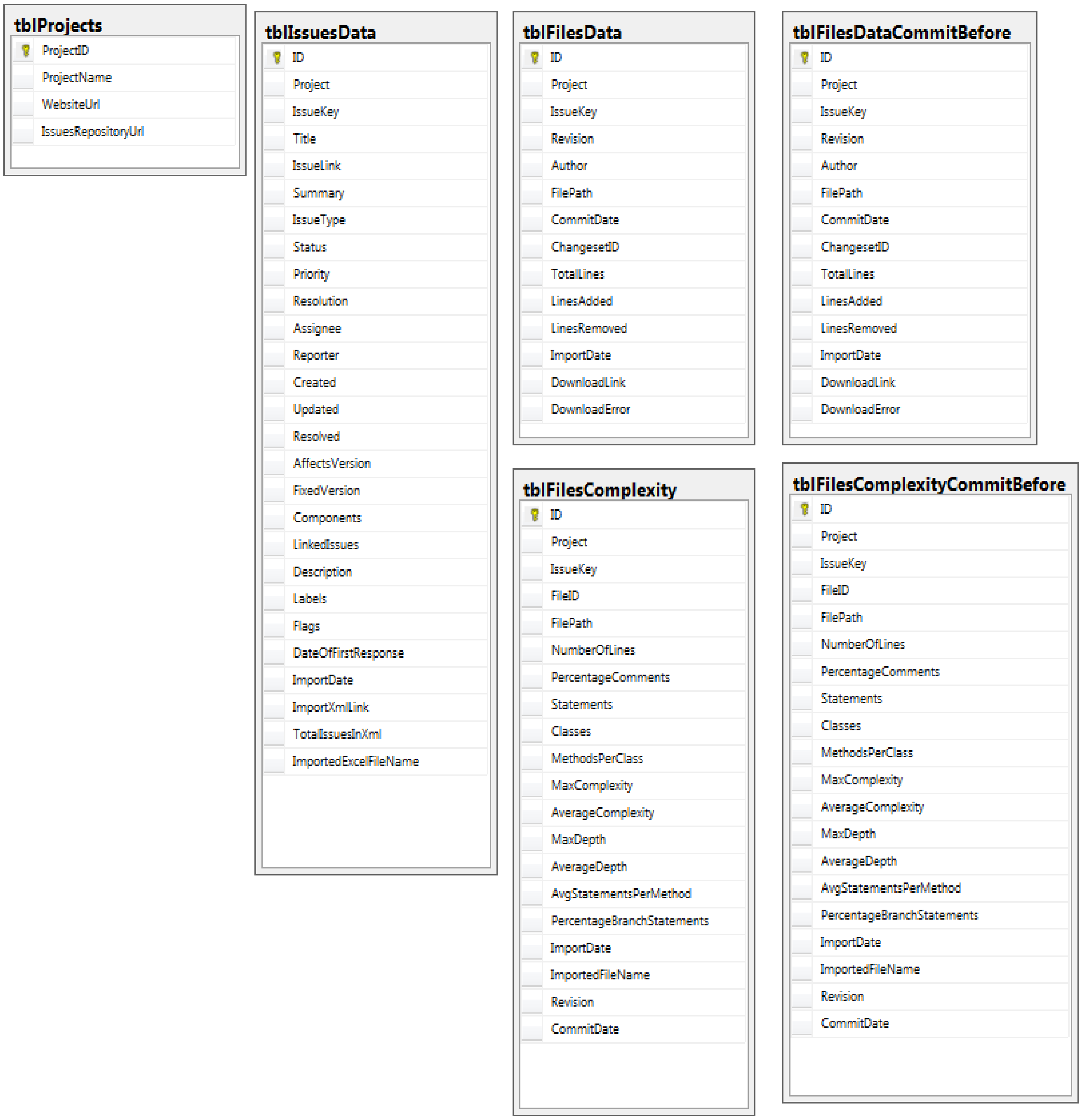
3.2.3 SQL Database Schema

The final database structure includes six tables for data and five tables for results generation from this data. Descriptions for the tables are summarized in Table 3.6.

Table 3.6: Description of database tables

Tables	Description
tblProjects	Contains the general information about apache projects
tblIssuesData	Issues data for each project
tblFilesData	Files commit of version when bug is fixed
tblFilesComplexity	Files characteristics of version when bug is fixed
tblFilesDataCommitBefore	Files commit of version when bug is introduced
tblFilesComplexityCommitBefore	Files characteristics of version when bug is introduced
tblAnonymizeAuthor	Anonymize the contributor names, anonymized contributor ID’s are used in the results
tblResultsRQ1	Metrics information involved in RQ1.
tblResultsRQ1Normalized	Normalized results for RQ1
tblResutlsRQ2	Metrics information involved inRQ 2
tblResultsRQ2Normalized	Normalized results for RQ2

The Figure 3.6 and Figure 3.7 represents that how the data is organized in SQL tables.



tblProjects
- ProjectID
- ProjectName
- WebsiteUrl
- IssuesRepositoryUrl

tblIssuesData
- ID
- Project
- IssueKey
- Title
- IssueLink
- Summary
- IssueType
- Status
- Priority
- Resolution
- Assignee
- Reporter
- Created
- Updated
- Resolved
- AffectsVersion
- FixedVersion
- Components
- LinkedIssues
- Description
- Labels
- Flags
- DateOfFirstResponse
- ImportDate
- ImportXmlLink
- TotalIssuesInXml
- ImportedExcelFileName

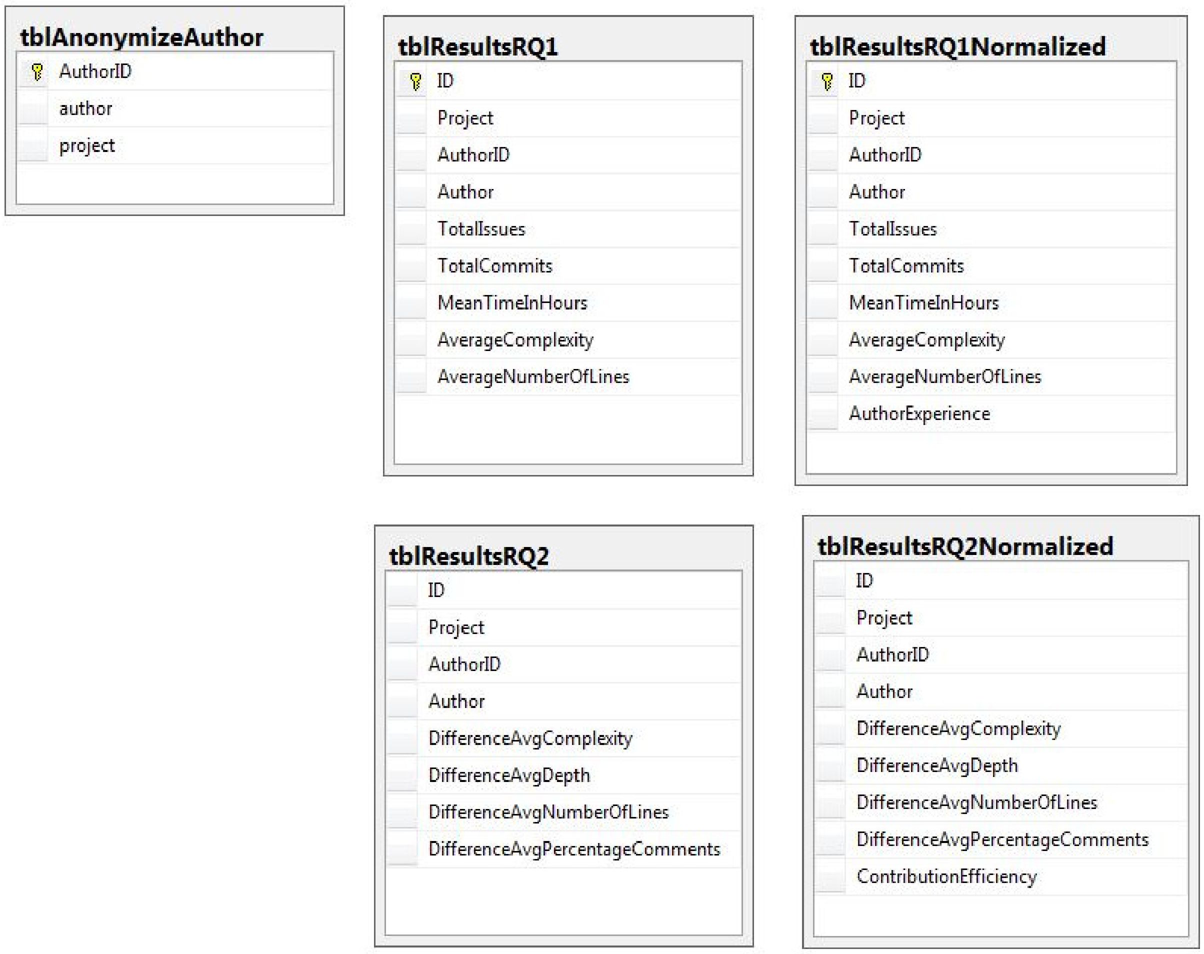
tblFilesData
- ID
- Project
- IssueKey
- Revision
- Author
- FilePath
- CommitDate
- ChangesetID
- TotalLines
- LinesAdded
- LinesRemoved
- ImportDate
- DownloadLink
- DownloadError

tblFilesDataCommitBefore
- ID
- Project
- IssueKey
- Revision
- Author
- FilePath
- CommitDate
- ChangesetID
- TotalLines
- LinesAdded
- LinesRemoved
- ImportDate
- DownloadLink
- DownloadError

tblFilesComplexity
- ID
- Project
- IssueKey
- FileID
- FilePath
- NumberOfLines
- PercentageComments
- Statements
- Classes
- MethodsPerClass
- MaxComplexity
- AverageComplexity
- MaxDepth
- AverageDepth
- AvgStatementsPerMethod
- PercentageBranchStatements
- ImportDate
- ImportedFileName
- Revision
- CommitDate

tblFilesComplexityCommitBefore
- ID
- Project
- IssueKey
- FileID
- FilePath
- NumberOfLines
- PercentageComments
- Statements
- Classes
- MethodsPerClass
- MaxComplexity
- AverageComplexity
- MaxDepth
- AverageDepth
- AvgStatementsPerMethod
- PercentageBranchStatements
- ImportDate
- ImportedFileName
- Revision
- CommitDate

Figure 3.6: Database table schema


tblAnonymizeAuthor
- AuthorID
- author
- project

tblResultsRQ1
- ID
- Project
- AuthorID
- Author
- TotalIssues
- TotalCommits
- MeanTimeInHours
- AverageComplexity
- AverageNumberOfLines

tblResultsRQ1Normalized
- ID
- Project
- AuthorID
- Author
- TotalIssues
- TotalCommits
- MeanTimeInHours
- AverageComplexity
- AverageNumberOfLines
- AuthorExperience

tblResultsRQ2
- ID
- Project
- AuthorID
- Author
- DifferenceAvgComplexity
- DifferenceAvgDepth
- DifferenceAvgNumberOfLines
- DifferenceAvgPercentageComments

tblResultsRQ2Normalized
- ID
- Project
- AuthorID
- Author
- DifferenceAvgComplexity
- DifferenceAvgDepth
- DifferenceAvgNumberOfLines
- DifferenceAvgPercentageComments
- ContributionEfficiency

Figure 3.7: Database table schema for keeping results of data for analysis

3.3 Measurement Metrics

This section describes the metrics, used to measure the artifacts of RQ1 and RQ2. The metrics are measured by using file characteristics. Each metric is defined in detail as follows.

Lines of Code (LOC): Total number of physical lines in a source code file is considered LOC or Number of Lines [20]. Empty and commented lines are also included in LOC.

Percentage Comments: The lines that contain comments are counted and then compared to the total number of lines in the file to compute this metric [20].

Number of Classes: Classes and Interfaces are counted on bases of their declarations in a source code file [20].

Methods per class: This metric count the number of methods in a class [20].

Code Complexity: The complexity metric is measured as defined by Steve McConnel's book [20], and using method is based on Tom McCabe's work in which complexity is computed by counting the number of decision points in routine [25]. Method or function is considered as routine. Each method or function has a complexity of one plus one for each branch statement like if, else, while, for, or foreach [20]. Each arithmetic if statements such as (MyBoolean ? ValueIfTrue : ValueIfFalse) add one to the total complexity as well. A complexity increases by one for each logical operator ('&&' and '||') in the logic within if, for, while or similar logic statement [20].

Average Complexity: It is a measure by taking average of overall complexity computed for each method or function in a file [20].

Block Depth: Nesting code can be used in most languages, nested blocks are almost always introduced with control statements like "if", "case" and "while" [20]. The code gets harder to read when depth of nested block grows, more conditions must be evaluated with each new nested depth level [20].Block level is zero at the start of each file and increases by one for each level of nested statements.

Average Block Depth: It is measured as weighted average of the block depth of all statements in a file [20].

3.4 Goal Question Metric Approach

Goal, Question, Metric (GQM) is an approach that is used to define the project goals in systematic and traceable way [18]. It specifies a measurement model with three levels. First is conceptual level where goal is defined for an object. Second is operational level where set of questions are used to define the model. Third is quantitative level where set of metrics are determined in order to answer the question in measurable way. GQM is a way to derive and choose a specific task in a top-down and goal-oriented fashion [19]. This approach minimizes the effort of data collection because only required data is to be recorded [19]. We use a GQM approach to address our thesis's research questions. We define goals, questions and metrics to answer the research questions based on collected data. This approach improved our method by clearly defining the goal, questions to achieve the goal and metrics to answer the questions.

3.4.1 GQM for Research Question 1

The goal of Research Question (RQ) 1 is measuring the contributor's experience. To achieve this goal, we formulate the question "How can a code contributor be classified in terms of project experience?". We decided on a set of metrics, which we expected to address contributors experience. The metrics are; 1) Total number of commits, 2) Total number of issues assigned, 3) number of lines/LOC, 4) Code complexity of file, 5) Mean time contribution in number of hours/minutes. Figure 3.8 shows the GQM for the RQ1 and also metrics description.

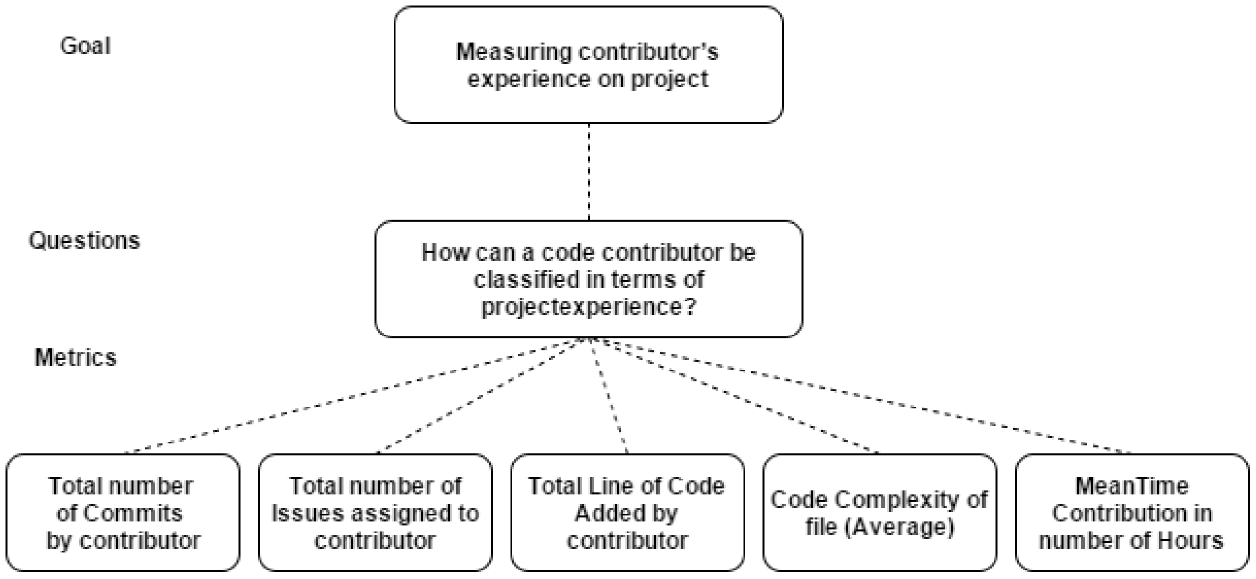

Goal
Measuring contributor's experience on project
Questions
How can a code contributor be classified in terms of projectexperience?
Metrics
Total number of Commits by contributor
Total number of Issues assigned to contributor
Total Line of Code Added by contributor
Code Complexity of file (Average)
MeanTime Contribution in number of Hours

Figure 3.8: Goal Question Metrics for RQ1

Total number of Commits by contributor: It measures the number of files of commits to the repository for a contributor. High values indicate more participation and low values represents less participation.

Total number of issues assigned to contributor: This is the measure of total number of issues assigned to a contributor on a specific project. High values indicate more participation and low values represents less participation.

Total Lines of Code added by contributor: This metric gives total number of lines added by a contributor in all his files commits. High values indicate more participation and low values represents less participation.

Code Complexity of files (in Average): Code complexity of files, we take this to measure the contributor's level of expertise in a sense if he worked on more complex files or not. If this number is high means contributor worked on complex files and less in number indicates that contributor worked on less complex files.

Mean time contribution in number of Hours: It indicates the time duration between two file commits. We are taking average of time duration among all commits by a contributor. High values indicate less participation and low values represents more participation.

3.4.2 GQM for Research Question 2

The goal of RQ2 is assessing the characteristics of contributions made to project. We formulate two sub questions to achieve this goal. First sub question is, what are the characteristics of code/file, where bugs are found and fixed? We find that the code file characteristics include Lines of Code, Average Complexity, Code Depth, Percentage Comments, Number of Classes, Number of Functions, Statements, Max Complexity, Max Depth etc.

Second sub question is; how can code/file characteristics be combined to assess contributions? To assess contributions using code file characteristics, we take difference of code characteristics when a commit is made with a version just before that commit (file version just before a commit represents the file state when a contributor started work on a file to fix an issue). Figure 3.9 shows the GQM for the RQ2 and evaluated difference metrics description.

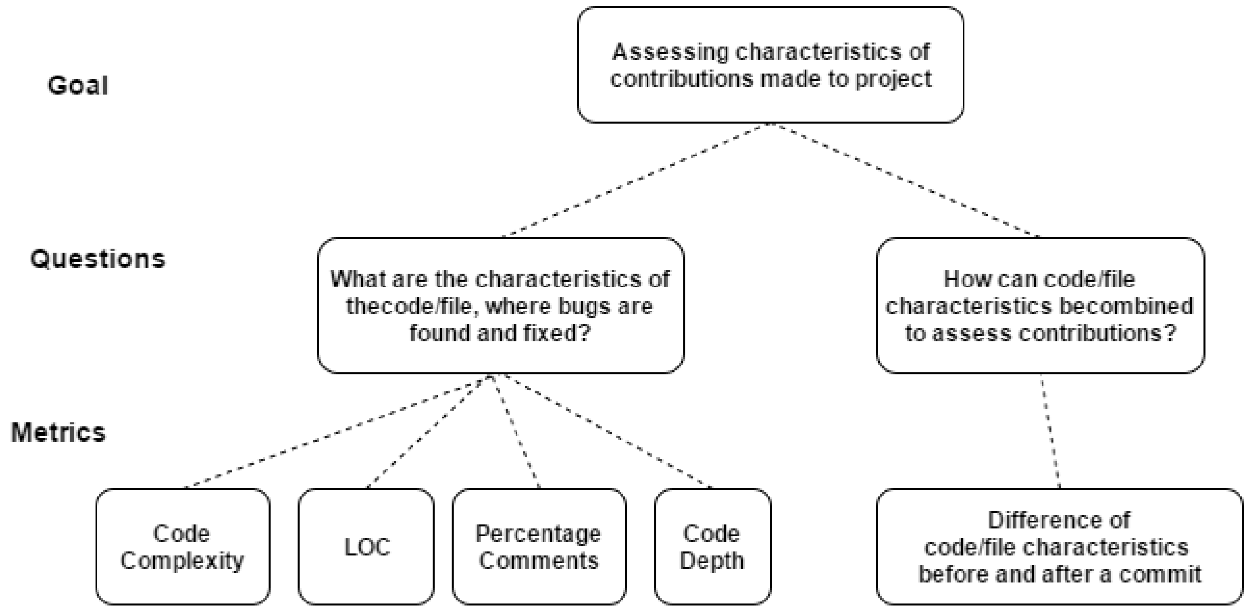

Goal
Assessing characteristics of contributions made to project
Questions
What are the characteristics of thecode/file, where bugs are found and fixed?
How can code/file characteristics becombined to assess contributions?
Metrics
Code Complexity
LOC
Percentage Comments
Code Depth
Difference of code/file characteristics before and after a commit

Figure 3.9: Goal Question Metrics for RQ2

We define contribution assessment in the following as contribution efficiency. We decided to use 4 file characteristics to assess the contributions efficiency. These four metrics include Average Code Complexity, Lines of Code, Percentage Comments and Average Code Depth. The definitions of these file characteristics are given in Section 3.3.

The selected 4 metrics further transformed into 4 manipulated metrics to assess the contributions. Manipulated metrics are the metrics that gives comparison results (difference values) of two different versions of the same code file to provide an evidence about the efficiency of a contribution. These are as follows:

Average Code Complexity Difference: It is calculated by subtracting average complexity of a file after commit from average complexity before commit. So in case it is decreased, then it gives a positive number value.

Lines of Code Difference: This metric is calculated by taking the difference of Lines of Code after and before commit. This value give the number of lines added in a commit. In case Lines of Code added, then this measure gives positive number value.

Percentage Comments Difference: The difference of Percentage Comments after and before commit. Positive difference values show us that there is addition of documentation in a code file.

Average Code Depth Difference: It is measured in similar way as Average Complexity. We subtract Average Code Depth of a file after commit from Average Code Depth before commit. So in case of decrease of Average Code Depth, the difference value is a positive number.

These metrics are manipulated in a form, so that positive values represent an increase in contribution efficiency while negative values represent decrease in contribution efficiency.

Our Strategy for measuring efficiency is described below with four points of view. With combining all four points, we expect to create a realistic assessment of contribution efficiency.

- A decrease in Average Code Complexity shows positive impact on contribution efficiency. The reason is, that the contributor reduced the overall complexity of the code file with his/her commit. The commit thereby contributed to understandability and maintainability of the code.

- An increase in LOC shows that contributor made a contribution by writing numbers of lines of code. This metric gives an idea that contributor understands well the code file so that he is able to contribute with addition of code lines.

- An increase in Percentage Comments gives a positive effect on efficiency by increasing the readability or maintainability of the code file.

- A decrease in Code Depth represents that code is made simpler and so easier to understand and modify.

Since every contributor have hundreds of records of evaluated metrics data, we have taken averages of these values to calculate contribution efficiency of a contributor.

3.4.3 GQM for Research Question 3

The goal of RQ3 is to calculate correlation values between contributor's experience and contribution's efficiency. Figure 3.10 shows the GQM diagram for RQ3. The results of RQ1 and RQ2 (contributors experience and contribution efficiency) are the metrics for RQ3.

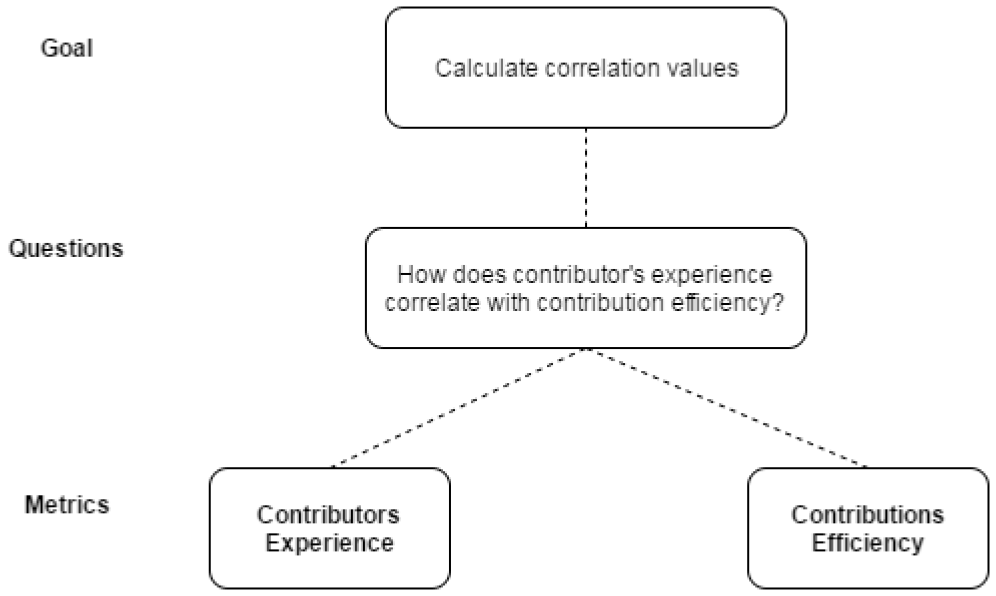

Goal
Calculate correlation values
Questions
How does contributor's experience correlate with contribution efficiency?
Metrics
Contributors Experience
Contributions Efficiency

Figure 3.10: Goal Question Metrics for RQ3

4. Results

This section presents detailed results of our case studies. These results are collected from the automated data collection process which is mentioned in Section 3.2.2. We illustrate the results received for all 6 projects using the results of Apache Struts2 project. The results are generated using data of 28, 077 issues and 332, 977 files of commits with their characteristics. The detailed analysis of these results follows in Chapter 5. For reader's interest, all other results are presented in the Appendix A.

4.1 Contributor's Experience(Results for RQ1)

The experience of contributors is measured by using the metrics; Total Issues Assigned, Total Files of Commits by Contributor, Mean Time between commits (in hours), Average Complexity of files committed, Total Lines Added by Contributor. Descriptions of these metrics are illustrated in Section 3.3.

Table 4.1: Cumulative Results for contributor's experience of Apache Struts2 project

Contributors ID	Total Issues Assigned	Total Commits by Contributor	Mean Time In Hours	Average Complexity	Total Lines Added by Contributor
1	22	3821	211.73	1.72	23136
2	6	6	787.08	6.03	34
3	0	0	0	0	0
4	4	5	3059.45	1.66	230
5	7	27	182.12	1.86	512
6	1	1	0	4.14	1
7	5	13	576.55	2.07	802
8	61	346	260.4	1.86	6552
9	78	33142	57.48	1.7	1653569
10	19	97	356	2.6	1942
11	18	52	402.78	2.42	793
12	41	251	105.98	2.33	1589
13	20	171	379.08	2.28	925
14	38	1181	456.85	2.12	122727
15	335	1921	82.17	2.55	27551
16	37	117	519.92	2.62	1628
17	242	8554	42.68	1.96	430957
18	228	4850	74.07	1.81	355627
19	1	22	0	1.2	440
20	45	251	287.97	2.41	14185
21	1	1	0	2.55	17
22	57	10842	465.3	1.8	554780
23	71	858	24.45	2.08	23926
24	9	23	447.27	2.56	312
25	45	214	523.42	2.94	1550
26	13	744	671.72	2	1028

Table 4.1 shows results for the contributor's experience of Apache Struts2 project. In the next step, each metric value is normalized using a min-max data normalization technique [23]. It is simplest method to rescale the value in range {0,1}. The Equation 4.1 is used to calculate rescale value. It calculates the relative value so contributor's experience is also calculated relatively with respect to the other contributor involved in project.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Equation: 4.1

Where x is the original value and x' is normalized value. For example, we rescale the Total number of issues assigned to a contributor's data, and issues assigned span {40, 150}. Where min. issues assigned value is 40 and max. value is 150. To rescale this data, we first subtract 40 from each Issue assigned value and divide the result by 110 (the difference between the max. and min. issue assigned value).

$$x'' = x' \times \frac{20}{100}$$

Equation: 4.2

The final contributors' experience is calculated by combining all five metrics with an equal weight of 20% for each metric value. Equation 4.2 is used for calculating weight of 20%.

Contributor's Experience = Total Issues Assigned + Total Files of Commits by Contributor + Mean Time (in hours) + Average Complexity + Total Lines Added by Contributor

Table 4.2: Normalized results for contributor’s experience of Struts2 project

Contributors ID	Total Issues Assigned	Total Commits by Contributor	Mean Time In Hours	Average Complexity	Total Lines Added by Contributor	Contributor's Experience
1	0.01257	0.02305	0.18766	0.17847	0.00280	**0.38150**
2	0.00299	0.00003	0.14974	0.00000	0.00000	**0.15274**
3	0.00000	0.00000	0.00000	0.00000	0.00000	**0.00000**
4	0.00180	0.00002	0.00000	0.18095	0.00003	**0.18278**
5	0.00359	0.00016	0.18961	0.17267	0.00006	**0.36594**
6	0.00000	0.00000	0.00000	0.07826	0.00000	**0.07826**
7	0.00240	0.00007	0.16362	0.16398	0.00010	**0.33009**
8	0.03593	0.00208	0.18445	0.17267	0.00079	**0.39384**
9	0.04611	0.20000	0.19782	0.17930	0.20000	**0.62323**
10	0.01078	0.00058	0.17815	0.14203	0.00023	**0.33119**
11	0.01018	0.00031	0.17507	0.14948	0.00010	**0.33483**
12	0.02395	0.00151	0.19463	0.15321	0.00019	**0.37198**
13	0.01138	0.00103	0.17663	0.15528	0.00011	**0.34340**
14	0.02216	0.00712	0.17151	0.16190	0.01484	**0.37041**
15	0.20000	0.01159	0.19620	0.14410	0.00333	**0.54363**
16	0.02156	0.00070	0.16735	0.14120	0.00020	**0.33030**
17	0.14431	0.05162	0.19880	0.16853	0.05212	**0.56376**
18	0.13593	0.02926	0.19673	0.17474	0.04301	**0.55041**
19	0.00000	0.00013	0.00000	0.20000	0.00005	**0.20005**
20	0.02635	0.00151	0.18263	0.14990	0.00172	**0.36059**
21	0.00000	0.00000	0.00000	0.14410	0.00000	**0.14410**
22	0.03353	0.06542	0.17095	0.17516	0.06710	**0.44674**
23	0.04192	0.00517	0.20000	0.16356	0.00289	**0.40837**
24	0.00479	0.00013	0.17214	0.14369	0.00004	**0.32065**
25	0.02635	0.00129	0.16712	0.12795	0.00019	**0.32160**
26	0.00719	0.00448	0.15735	0.16687	0.00012	**0.33153**

In Table 4.2, the rightmost column in yellow shows the normalized results for the contributor’s experience of Apache Struts2 project. Contributor 9 has the maximum experience that is 0.62323 and Contributor 6 has minimum experience with result value 0.07826. The table also contains the normalized measurement values.

The zero values for experience are unavoidable in the results. For an example, in Table 4.2 Contributor 3 have 0 experience. This is the case when we get Contributor name from the source systems, but we didn’t find any contribution related to him or her on these systems.

4.2 Contribution Efficiency (Results for RQ2)

The goal of RQ2 is assessing the contributions efficiency. A single contribution consists of a file commit and a file version just before that commit. We used file characteristics (Code Complexity, Code Depth, LOC and Percentage Comments) to measure the contribution efficiency. The metrics are; Difference of average Code Complexity, Difference of average Code Depth, Difference of average LOC, Difference of Percentage Comments. The intention behind these metrics is discussed in Section 3.4.2.

Table 4.3: Results for contribution efficiency of Struts2 project

Contributors ID	Difference Avg. Complexity	Difference Avg. Depth	Difference Avg. LOC	Difference Avg. Percentage Comments
1	-0.00980	-0.00049	3.49020	4.38333
2	-0.12500	-0.01250	0.75000	-0.07500
3	0.00000	0.00000	0.00000	0.00000
4	0.14800	0.10600	13.40000	0.52000
5	-0.04000	-0.05300	14.20000	-1.68000
6	0.00000	0.06000	-1.00000	0.30000
7	0.26167	-0.06000	30.33333	-2.11667
8	-0.01802	-0.03542	9.10417	-0.27813
9	-0.02966	-0.02610	10.79661	0.84661
10	-0.08913	-0.01978	10.97826	-0.27391
11	-0.17273	-0.04568	8.88636	-0.18636
12	-0.02024	-0.02084	9.87952	-0.63373
13	0.00089	-0.02768	-0.30357	0.18393
14	-0.21256	-0.05068	4.82051	-0.79658
15	0.00840	0.00359	6.47166	-0.55283
16	0.05329	-0.01630	5.45205	0.14658
17	0.05191	0.00057	6.27901	-0.37605
18	-0.00272	-0.00332	9.12816	-0.83187
19	0.00000	0.00000	20.00000	19.64286
20	0.12663	0.01257	3.82178	-0.19010
21	0.00000	0.00000	0.00000	0.00000
22	-0.10261	-0.01625	2.39674	-0.00761
23	-0.03730	-0.02230	2.27000	-0.38800
24	0.01111	-0.00111	6.72222	-0.38333
25	-0.03057	-0.01557	4.27273	-0.53636
26	0.00000	0.00000	0.37097	-0.02419

Table 4.3 shows results for the contribution efficiency of Apache Struts2 project. In the next step, these metric values are normalized as described in Section 4.1.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \times \frac{25}{100}$$

Equation: 4.3

The final contribution efficiency of contributor is calculated by combining all four metrics with an equal weight of 25%. Equation 4.3 is used for calculating weight of 25%.

Contribution Efficiency of Contributor = Difference of Average Code Complexity + Difference of Average Code Depth + Difference of Average LOC + Difference of Percentage Comments

Table 4.4: Normalized results for contribution efficiency of Apache Struts2 project

Contributors ID	Difference Avg. Complexity	Difference Avg. Depth	Difference Avg. LOC	Difference Avg. Percentage Comments	Contribution Efficiency
1	0.10689	0.08962	0.05732	0.07468	**0.32851**
2	0.04616	0.07154	0.02234	0.02346	**0.16349**
3	0.00000	0.00000	0.00000	0.00000	**0.00000**
4	0.19008	0.25000	0.18383	0.03029	**0.65420**
5	0.09097	0.01054	0.19404	0.00502	**0.30057**
6	0.11206	0.18072	0.00000	0.02777	**0.32055**
7	0.25000	0.00000	0.40000	0.00000	**0.65000**
8	0.10256	0.03702	0.12899	0.02112	**0.28969**
9	0.09642	0.05105	0.15060	0.03405	**0.33211**
10	0.06507	0.06057	0.15291	0.02117	**0.29972**
11	0.02100	0.02156	0.12621	0.02218	**0.19095**
12	0.10139	0.05897	0.13889	0.01704	**0.31628**
13	0.11253	0.04868	0.00889	0.02643	**0.19653**
14	0.00000	0.01403	0.07430	0.01517	**0.10350**
15	0.11648	0.09577	0.09538	0.01797	**0.32561**
16	0.14015	0.06581	0.08237	0.02600	**0.31433**
17	0.13942	0.09122	0.09292	0.02000	**0.34356**
18	0.11063	0.08536	0.12930	0.01476	**0.34004**
19	0.11206	0.09036	0.26809	0.25000	**0.72050**
20	0.17881	0.10930	0.06155	0.02213	**0.37180**
21	0.11206	0.09036	0.01277	0.02432	**0.23950**
22	0.05797	0.06589	0.04336	0.02423	**0.19145**
23	0.09239	0.05678	0.04174	0.01986	**0.21078**
24	0.11791	0.08869	0.09858	0.01991	**0.32510**
25	0.09594	0.06692	0.06731	0.01816	**0.24833**
26	0.11206	0.09036	0.01750	0.02404	**0.24396**

Table 4.4 shows the normalized results for the contribution efficiency of Apache Struts2 project. Contributor 19 has maximum contribution efficiency with value 0.72050. Whereas Contributor 14 has minimum contribution efficiency with result value 0.10350.

The zero values for contribution efficiency are unavoidable in the results. For an example, in Table 4.4 Contributor 3 have 0 efficiency. This is the case when we get Contributor name from the source systems, but we didn't find any contribution related to him or her on these systems.

4.3 Correlation between Experience and Efficiency (Results for RQ3)

The goal of RQ3 is to find out the correlation between contributor experience and contribution's efficiency. Correlation between sets of data is a measure of how well the data is related. The common measure of correlation in parametric statistics is by using Pearson Correlation. It can be measured by the Equation 4.4.

$$r = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}}$$

Equation: 4.4

We have performed statistical tests (using R with an environment RStudio [27]) for calculating the correlation between contributor experience and their contribution efficiency.

4.3.1 Interpretation of Pearson Correlation Coefficient

The Pearson Correlation Coefficient value ranges from -1 to 1. A value of 1 means a strong relationship between X & Y. It means that Y increases as X increases. A value of 0 means no correlation. A value of -1 means a strong correlation also but this value means Y decreases as X increases. Correlation does not make a statement about causalities. If X correlates with Y, it does not mean that X causes behavior of Y.

4.3.2 Correlation Coefficient Values for Six Projects

Our results are given in Table 4.5. The results show that there is no correlation between contributor experience and efficiency. In our statistical tests, normalized datasets give us Pearson Coefficient value ranging from -0.2669051 to 0.1229097.

We additionally provide the p-value, stating if the correlation coefficients are significantly different from 0. As none of the p-values indicates significance, we conclude that even the weak correlations may be received by chance.

Table 4.5: Correlation Coefficient value for each project

Project	Pearson's correlation coefficient	p-value
Struts 2	-0.1665629	0.4262
Maven	-0.2669051	0.1053
ActiveMQ	-0.07569234	0.5864
Sling	0.1229097	0.5253
Felix	-0.1393565	0.404
OFBiz	-0.05370417	0.782

5. Analysis and Discussions

This section presents the analysis of results and discuss them in detail. By the end of this chapter, threats to validity are assessed.

5.1 Contributor's Experience

RQ1: How can a code contributor be classified in terms of project experience?
According to our study, code contributor can be classified in terms of project experience by combining 5 metrics of information, which are the total number of issues or tasks, total files of commits, mean time between commits in hours, average file complexity and total lines of contribution added by a contributor.

It is a common practice that contributor's experience is measured by how long they contributed to a project (with the help of the contributor start date and end date) and by using the frequency of contributions [26]. In our study, we are examining the code files (where work is done by a contributor) to measure the experience. Two of our metrics, lines of code and bug fixing contributions are discussed for assessing developer contributions in an empirical study [29], it is argued in the study that these metrics can better perform when combined with other metrics of information. And that is the case with our study, we have combined these 2 metrics with other 3 metrics.

In our study, we don't use the difference between first and last contribution date as the work period of a contributor while calculating the experience. The reason is that, in OSS environment people don't contribute on the regular basis. So for an example a developer A with work period of 1 year may have more experience than a developer B with the work period of 2 years, because it is possible that developer B have one contribution 2 years ago and then few contributions later. Instead we use an approach to see a contributor activity with the help of the mean time difference in hours between contributions.

If we analyze the contributor's experience in Figure 5.1 for the Apache Struts2 project, top 4 contributors (contributor 9, 17, 18 and 15) have more contributions experience than other contributors. Hence, using our combined metrics we can identify differences among the contributors and identify key developers with high experience.

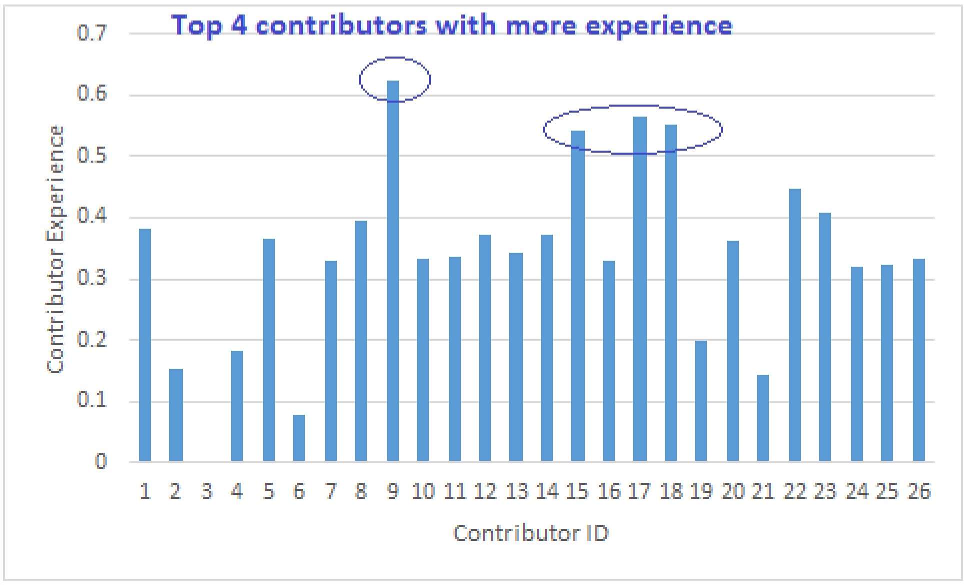


Figure 5.1: Apache Struts2 Contributor's Experience

This way it can be very useful to analyze contributor experience on a project while assigning tasks or issues to developers. We are able to answer RQ1, since we have classified the contributor's in terms of their experience.

5.2 Contributions Efficiency

RQ2: How can code contributions to the projects be characterized by assessing code/file characteristics?
Code contributions can be characterized by using 4 metrics of information according to our study, these are average file complexity, average code depth, lines of code and percentage comments. These code/file characteristics can be combined by taking difference of 'commit before' and 'commit after' values.

In other studies [26] [28], the quality of contribution is normally measured by one binary metric information, which is the rate of non-bug-introducing commits. It is calculated by taking a 'true' value if a project compiles without any error after commit and a 'false' in case it generates errors after the commit.

A recent study proposes a way to measure contribution efficiency by using complexity metrics. This study result suggests that the developers who unnecessarily increase the code complexity are less efficient [29]. This is an empirical study but it lacks the assessment of metrics of code depth and percentage comments that we used in our method with combination of code complexity and lines of code.

Another study [30], used a model to measure contributions by using lines of code plus a contribution factor. This study identified actions that can be classified as contributions factor and type of actions, such as add lines of code of good/ bad quality with type as positive/ negative impact. Another example of action is commit a file with type as positive impact. But this study is not using the code complexity metrics for measuring contributions.

For our study, we have chosen 4 file characteristics to include in the comparisons for measuring efficiency that can contribute in efficiency of a code file. These characteristics are average code complexity, average code depth, lines of code and percentage comments. By taking the difference of these file characteristics with before commit versions, we are able to measure contribution efficiency.

If we analyze the contribution's efficiency in Figure 5.2 for the Apache Struts2 project. The top 3 contributors (contributor 4, 7 and 19) have more efficient contributions than other contributors. This can be helpful to see the best contributors in regard of quality of contributions.

It is therefore possible to answer RQ2, using the metrics it is possible to assess efficiency by combining file assessments. For an example, contributor 4 has efficiency value of 0.65 (as shown in Table 4.4). In the histogram, we see that contributor 3 has 0 value for contribution efficiency because there is no information for this contributor to measure efficiency. This is an example of outlier in Struts 2.

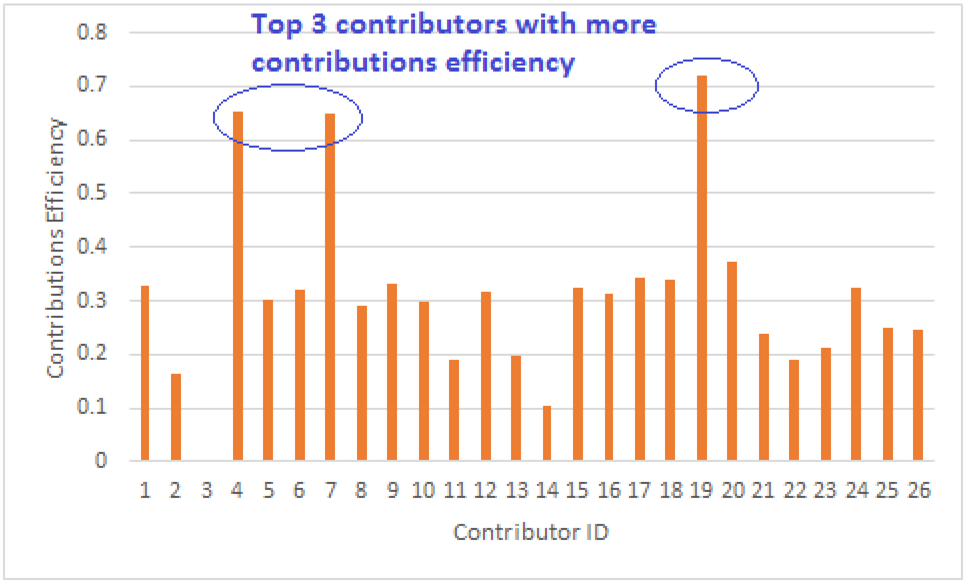


Figure 5.2: Apache Struts2 Contributions Efficiency

In our study, for measuring contributions efficiency we actually gathered all code file characteristics first and then analyzed every characteristic to include or exclude in the measurements. We have used average code complexity, average code depth, lines of code and percentage comments for measuring contribution efficiency. We have not used number of statements, number of functions, number of classes, max code complexity and max code depth in our measurements. The reasons for the characteristics that are not used in measurements are;

- Number of Statements, Functions and Classes: We have taken the lines of code metric that covers the total lines of code file (size attributes). The number of statements, number of functions and number of classes correlate with Lines of Code. For an example, it is repeating to use number of statements because the number of lines are directly proportional to number of statements.

- Max complexity and max code depth: These metrics show the complexity or depth of a part of the file but not for complete code file. So we have excluded these measures and included instead the averages for complexity and code depth.

5.3 Correlation between Contributor’s Experience and Contributions Efficiency

RQ3: How does contributor’s project experience correlate with code contributions?
According to statistical tests based on our results, there is no correlation between contributor’s experience and contribution's efficiency. Table 4.5 shows Pearson Correlation Coefficients for all six projects with their p-values.

To see the correlation visually, we have drawn scatter plots using the normalized data of our results with the help of RStudio [27]. Along x-axis there is the contributor’s experience and y-axis represents contribution's efficiency. Here we take an example of Apache ActiveMQ project for the discussion with help of visual diagram for the correlation in Figure 5.3. Apache ActiveMQ have more contributors than all other five projects, so the more data points. For all remaining projects; to see correlation diagrams, please refer to Appendix C (Scatter plots).

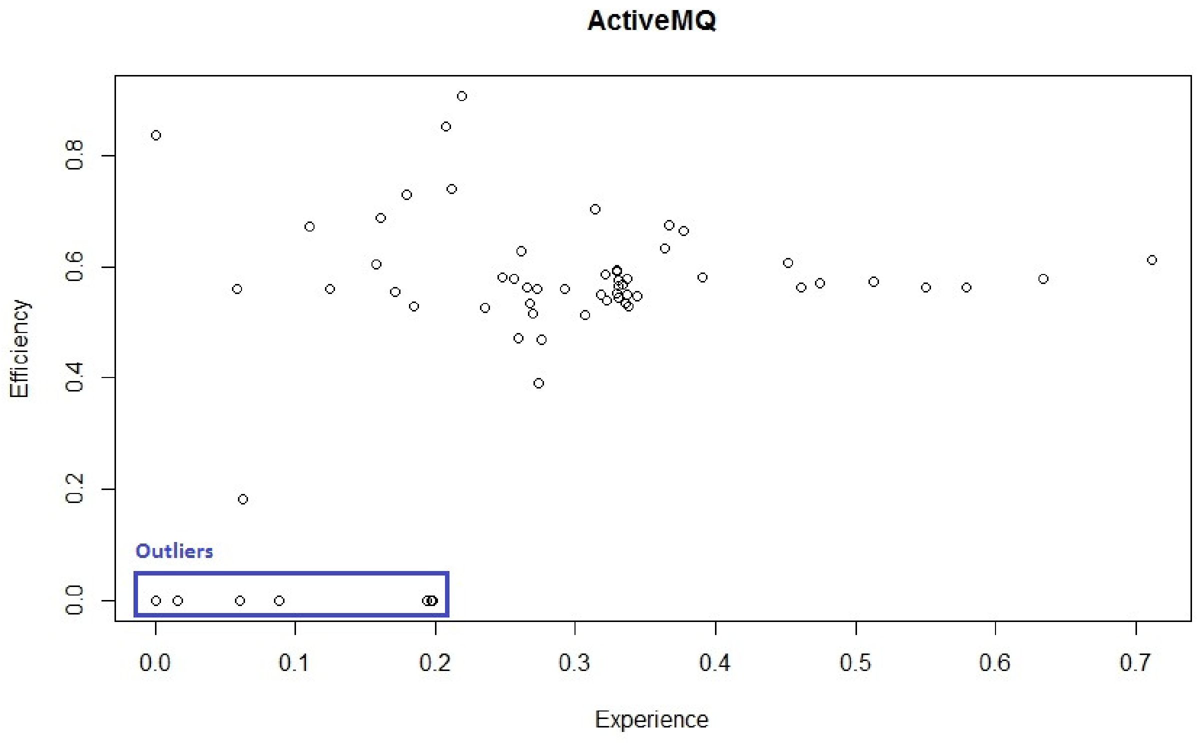


Figure 5.3: Scatter plot for ActiveMQ project

In Figure 5.3 for the Apache ActiveMQ project, we can again see no correlation. The data points are linearly distribution, in parallel to the x-axis. Hence, using the metrics as described in Section 3, we found that contribution efficiency does not increase with contributor experience. Furthermore, the highest peak of efficiency was even observed in the lowest third of contributor experience. There is a cluster of data points in the center, showing that more contributors have experience in the range around 0.35 and contribution efficiency around 0.6. This behavior shows that contributors with moderate experience have good contributions. We find similar clusters of data points in center for Felix and OFBiz projects. This observation can be seen in scatter plots given in Appendix C.

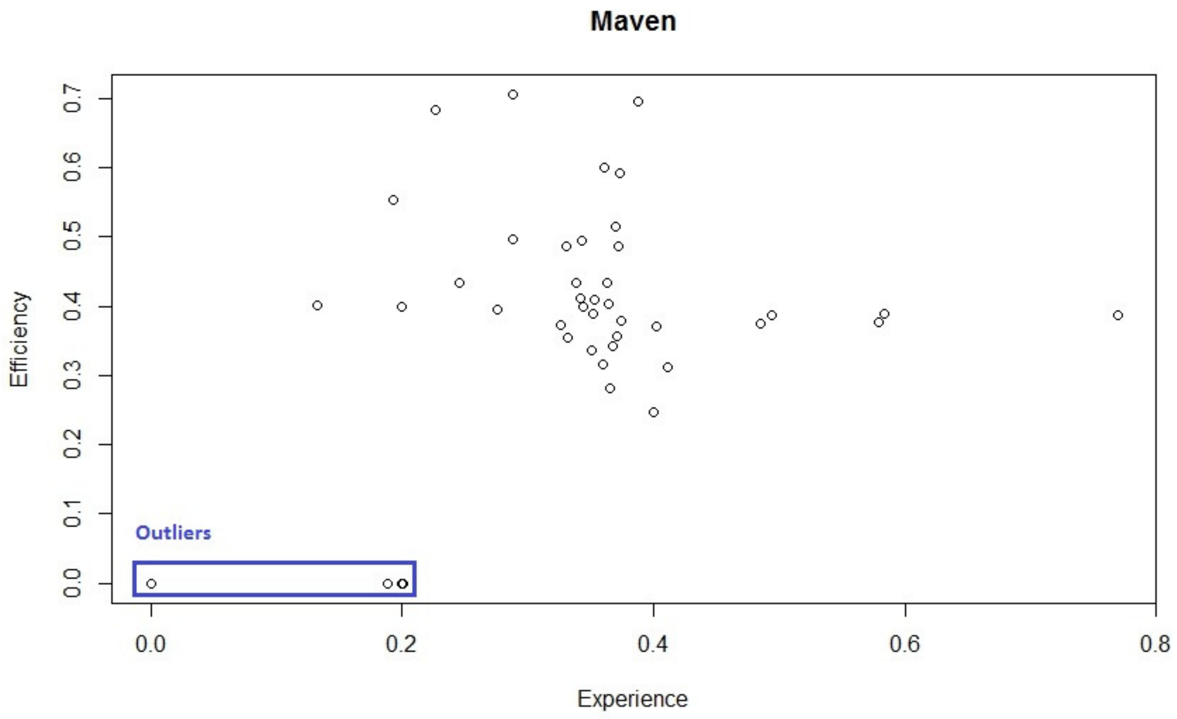


Figure 5.4: Scatter plot for Maven project

Similar behavior can be observed in Figure 5.4, in the scatter plot for Apache Maven. It shows that contributors with experience in the range around 0.38 are very different in regard of quality of contributions. The lowest value is 0.3 and the peak value for contribution efficiency is 0.7 in this region.

During the analysis, while looking at scatter plots and results we have decided to exclude the outliers for calculating correlation values. Outliers are those data points where either contributor efficiency or contributor experience has zero value or both measures have zero values. The zero values are unavoidable in the result's calculation process. These outliers have strong impact on the correlation values, even though they don't include any information. For an example, in case of Apache ActiveMQ, if we calculate correlation value including these outliers, it gives us somewhat moderate correlation value of 0.49 with confidence p-value 2.236e-05. Whereas if we see the actual correlation in the Figure 5.3, there is no correlation for ActiveMQ project. Hence, we observed that correlation including the outliers is strongly misleading. So we have excluded outliers for calculating the correlation values for all six projects.

Why there is no correlation between contributor experience and contributor efficiency? This question is hard to answer. One possible reason can be that contribution efficiency not only depend on contributor experience but also depend on other factors of personality, such as developer's IQ level, way of thinking, education, command in tools, etc. Another reason may be that; in this context we have examined the committers working inside Apache active code base. The committer role can have experience to some extent on a project with another role (such as a developer) before write access to the project.

We made an additional observation from the results that is not related to answering the research questions. If we analyze all six projects and relate it to the total number of committers on projects. Figure 5.5 shows us that, increasing the number of committers in a project repository not possibly increase the contribution's efficiency. For an example ActiveMQ have 65 contributors, the highest number of committers in this dataset, but the efficiency is lower than other two projects OFBiz and Sling. The same behavior can be seen for Apache Maven.

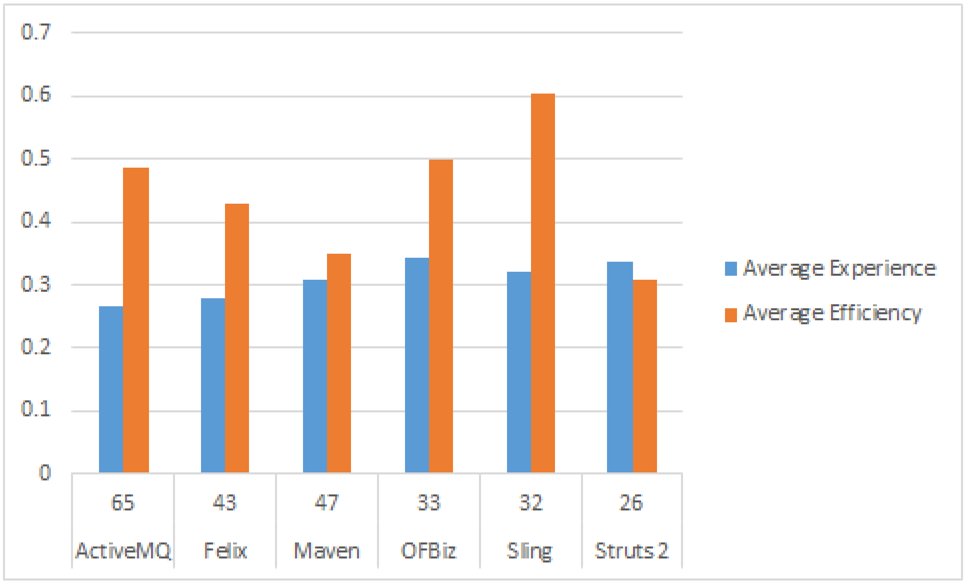


Figure 5.5: Behavior of Contributor's Experience and Efficiency with Total number of Committers (Team Size)

5.4 Threats to Validity

There exist different ways to classify aspects of validity and threats to validity. We have identified threats to internal and external validity according to P. Runeson and M. Höst [22]. These threats to validity are described in detail in this section.

Internal validity:

- Few commits contain large number of files, sometimes a complete project. This is possible when a committer moves the repository or create a new repository for a project. These affect the values taken for average mean time between commits. The average mean time metric tells about the activity of a developer, so we take the distinct commit date and time to measure the developer activity. It should not be dependent on how many files are committed in one commit. To mitigate the problem due to this reason we have considered multiple or large amount of files in one commit as a single activity date time when taking average mean time in hours between commits.

- We have collected data from online issue tracking system and file commits from active code repositories using automated scripts. It may be possible that a developer work on an issue that is not reported on these systems. The impact of this missing information is not significant due to the reason we evaluate the commits associated with an issue, and issues can be linked with file commits to active repositories in these systems for reviews.

- We have used an open source code complexity measurement tool named Source Monitor. Complexity can be measured in different ways. There are other software applications in the market for measuring complexity. It is possible that other tools would have resulted in different results. We use Source Monitor, as it provides a variety of measurements and it is widely used code analysis tool. The program supports multiple programming languages (C++, Java, C#, Visual Basic, Delphi or HTML) and provides advanced features like method and function level metrics for more detailed analysis and comparison. To mitigate the problem that different software's measure complexity in different ways, we have used the same tool for all projects. So that the results are consistent in this regard.

- We have made the study results by using data from the metrics. In order to avoid bias, we have given equal weight to these metrics. For an example, while calculating contribution efficiency each of 4 metrics have 25 % weight. Further investigation into the assessment of contributor experience might result in an adjustment of those weights. For an example, a recent study discussed about the importance of code complexity metric for the contributions efficiency [29].

External validity:

- We have chosen six medium scale projects for this study. Adding more data would increase the amount of data for the analysis and might affect the results. We cannot fully mitigate this generalizability threat. We have collected data for six different units of analysis and received similar results. This gives us an indication, that the results might be generalizable to other OSS projects as well.

6. Summary and Conclusion

This study analyzed six Apache projects data including 28,077 issues and 332,977 files of commits from Apache issue tracking system JIRA and Apache code repositories (Atlassian Fisheye). We considered not only bugs but also tasks, improvements, tests, documentation and new features.

The study evaluates the contributor’s experience and contribution efficiency in open source projects using issue's data and file commits made for resolving these issues. In this study we proposed different methods to measure contributor’s experience and contribution's efficiency with help of multiple data metrics. Contribution experience is measured using metrics that tell about the amount of contributions, frequency and complexity level of files on which developer worked on a project. Contribution efficiency is measured using file characteristics of code files (Code Complexity, Code Depth, LOC, Percentage Comments).

We present findings, which are similarly received for all six projects. We did not find correlation between experience and efficiency results. It means that a low experience contributor can also provide good contributions. In an Apache environment, a contributor with few contributions may also provide effective code. The reason could be that he or she gets write access to the code repository as a committer after having an experience on the project with another role such as a developer.

Another observation tells us that, larger the numbers of committers (or team size) on a project don’t have the positive impact on the efficiency of contributions.

6.1 Future Work

In this section, possible future work is discussed in detail.

- We have evaluated six Apache projects. Further studies can be extended to include more projects from the Apache domain or other open source communities like Source forge and Red Hat.
- We considered a set of most interesting file characteristics in this study. Other file characteristics can also be used to see the impact of these on results. Other file characteristics include; Number of Classes, Number of Methods, Methods per class, Statements and Method Calls.

- In further studies, contributors overall experience can be assessed to see the quality of contributions. For an example, if a contributor is working on more than one project, then it can be analyzed to see whether it’s making contributions more efficient or not.

7. References

[1] D. Wahyudin, A. Schatten, D. Winkler, and S. Biffl. Aspects of Software Quality Assurance in Open Source Software Projects: Two Case Studies from Apache Project. *In 33rd EUROMICRO Conference on Software Engineering and Advanced Applications,* pages 229 - 236, 2007.

[2] P. C. Rigby, D. M. German, and M. A. Storey. Open source software peer review practices:A Case Study of the Apache Server. *In 30th International Conference Software Engineering*, pages 541 - 550, 2008.

[3] The Apache Software Foundation, Available at: http://apache.org/foundation (visited on 2015-03-21).

[4] D. Huizinga, and A. Kolawa. Automated Defect Prevention: Best Practices in Software Management. Wiley-IEEE Press, page 261, 2007.

[5] P. C. Rigby, D. M. German, L. Cowen, and M. A.Storey. Peer Review on Open-Source Software Projects: Parameters, Statistical Models, and Theory, *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 23(4), Article No. 35, 2014.

[6] J. Liang, and O. Mizuno. Analyzing Involvements of Reviewers through Mining a Code Review Repository. *In Joint Conference of the 21st Int'l Workshop on Software Measurement and 6th Int'l Conference on Software Process and Product Measurement*, pages 126 - 132, 2011.

[7] P. C. Rigby, M. A. Storey. Understanding Broadcast Based Peer Review on Open Source Software Projects. *In 33rd International Conference on Software Engineering*, pages 541 - 550, 2011.

[8] C. F. Kemerer, M. C. Paulk. The Impact of Design and Code Reviews on Software Quality: An Empirical Study Based on PSP Data. *IEEE Transactions on Software Engineering,* 35(4):534 - 550, 2009.

[9] H. Siy, and L. Votta. Does The Modern Code Inspection Have Value?. *In IEEE International Conference on Software Maintenance*, pages 281 - 289, 2001.

[10] M. V. Mantyla, C. Lassenius. What Types of Defects Are Really Discovered in Code Reviews?, *IEEE Transactions on Software Engineering*, 35(3)430 - 448, 2009.

[11] C. Bird, A. Gourley, and P. Devanbu. Detecting Patch Submission and Acceptance in OSS Projects. *In Fourth International Workshop ICSE on Mining Software Repositories*, page 26, 2007.

[12] J. B. Lee, A. Ihara, A. Monden, and K. Matsumoto. Patch Reviewer Recommendation in OSS Projects. *In 20th Asia-Pacific APSEC on Software Engineering*, vol. 2 pages 1 - 6, 2013.

[13] A. Khanjani, and R. Sulaiman. The process of quality assurance under open source software development. (ISCI), *In IEEE Symposium on Computers & Informatics*, pages 548 - 552, 2011.

[14] J. Asundi, and R. Jayant. A patch review process in open source software development communities: A comparative case study, *In 40th Annual Hawaii International Conference on System Sciences,* page 166, 2007.

[15] Z. Yin, D. Yuan, Y. Zhou, S. Pasupathy, and L.Bairavasundaram. How do fixes become bugs?, *In Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering,* pages 26 - 36, 2011.

[16] F. Rahman, and P. Devanbu. Ownership, experience and defects: a fine-grained study of authorship, *In 33rd International Conference on Software Engineering,* pages 491 - 500, 2011.

[17] G. V. Krogh, S. Spaeth, and K. R. Lakhani. Community, joining, and specialization in open source software innovation: a case study, *Research Policy,* 32(7):1217 - 1241, 2003.

[18] V. R. Basili, and H. D. Rombach. The TAME project: towards improvement-oriented software environments, *IEEE Transactions on Software Engineering,* 14(6):758 - 773, 1988.

[19] I. Eusgeld, F.C. Freiling, and R. Reussner (Eds.). *Dependability Metrics*, LNCS 4909, page 39. Springer-Verlag Berlin Heidelberg, 2008.

[20] Source Monitor. Available at: http://www.campwoodsw.com/sourcemonitor.html (visited on 2015-08-17).

[21] K. Stol, and B. Fitzgerald. A Holistic Overview of Software Engineering Research Strategies. *In Proceedings of the 3rd International Workshop on Conducting Empirical Studies in Industry*, pages 47-54, 2015.

[22] P. Runeson, and M. Höst. Guidelines for conducting and reporting case study research in software engineering, *Empirical Software Engineering*, 14:131 - 164, 2009.

[23] A. Jain, K. Nandakumar, and A. Ross. Score normalization in multimodal biometric systems, *Pattern Recognition*, 38(12):2270 - 2285, 2005.

[24] Sonarqube. Available at: http://www.sonarqube.org/ (visited on 2015-03-16).

[25] S. McConnel, Code Complete, *Microsoft Press*, 2nd edition, page 458, 2004.

[26] J. Eyolfson, L. Tan, and P. Lam. Do Time of Day and Developer Experience Affect Commit Bugginess?, *In Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 153 - 162, 2011.

[27] RStudio Software Tool. Available at: https://www.rstudio.com/home/ (visited on 2016-01-04).

[28] Y. Qiu, W. Zhang, W. Zou, J. Liu, and Q. Liu. An Empirical Study of Developer Quality, *In IEEE International Conference on Software Quality, Reliability and Security - Companion*, pages 202 – 209, 2015.

[29] J. Lima, C. Treude, F. Figueira Filho, and U. Kulesza. Assessing Developer Contribution with Repository Mining-Based Metrics, *In IEEE International Conference on Software Maintenance and Evolution*, pages 536 – 540, 2015.

[30] G. Gousios, E. Kalliamvakou, and D. Spinellis. Measuring Developer Contribution from Software Repository Data, *In Proceedings of the international working conference on Mining software repositories*, pages 129 – 132, 2008.

Appendix A: Results of Contributor’s Experience

Table A.1: Results for contributor’s experience of Maven project

Contributors ID	Total Issues Assigned	Total Files of Commits by Contributor	Mean Time In Hours	Average Complexity	Total Lines Added by Contributor	Contributor's Experience
27	0.00035	0.00025	0.19983	0.00000	0.00002	**0.20020**
28	0.00000	0.00000	0.00000	0.00000	0.00000	**0.00000**
29	0.00243	0.00018	0.19074	0.00000	0.00012	**0.19330**
30	0.00070	0.00008	0.00000	0.13123	0.00003	**0.13195**
31	0.20000	0.20000	0.19961	0.16970	0.20000	**0.76931**
32	0.00278	0.00066	0.18666	0.15336	0.00046	**0.34326**
33	0.11200	0.11070	0.19878	0.14809	0.12497	**0.58385**
34	0.01774	0.00747	0.19784	0.15415	0.00208	**0.37181**
35	0.00904	0.00410	0.19534	0.16021	0.00475	**0.36935**
36	0.00000	0.00000	0.00000	0.00000	0.00000	**0.00000**
37	0.00591	0.00066	0.17030	0.16785	0.00009	**0.34415**
38	0.00035	0.00018	0.06484	0.16126	0.00027	**0.22672**
39	0.00174	0.00012	0.19450	0.16917	0.00004	**0.36545**
40	0.00765	0.00178	0.19087	0.16126	0.00166	**0.36144**
41	0.00035	0.00012	0.17199	0.15810	0.00003	**0.33047**
42	0.00000	0.00000	0.00000	0.20000	0.00000	**0.20000**
43	0.01391	0.04505	0.19124	0.15758	0.04853	**0.41126**
44	0.00313	0.00059	0.17822	0.15020	0.00020	**0.33175**
45	0.10365	0.10271	0.19886	0.14335	0.03955	**0.48541**
46	0.00070	0.00094	0.18505	0.18577	0.00125	**0.37276**
47	0.04765	0.14141	0.19500	0.16891	0.16781	**0.57937**
48	0.01322	0.00346	0.19464	0.14282	0.00156	**0.35224**
49	0.00000	0.00000	0.00000	0.00000	0.00000	**0.00000**
50	0.00278	0.00377	0.18518	0.16733	0.00433	**0.35962**
51	0.00139	0.08478	0.19872	0.16601	0.12851	**0.49463**
52	0.00000	0.00004	0.00000	0.00000	0.00000	**0.00000**
53	0.00000	0.00000	0.00000	0.00000	0.00000	**0.00000**
54	0.00313	0.00723	0.15967	0.17866	0.00935	**0.35081**
55	0.00035	0.05529	0.17392	0.16548	0.06246	**0.40221**
56	0.00035	0.01491	0.17846	0.16495	0.02704	**0.37081**
57	0.00070	0.00043	0.18334	0.16864	0.00017	**0.35285**
58	0.00000	0.00037	0.20000	0.16337	0.00015	**0.36352**
59	0.00278	0.00424	0.19902	0.15995	0.00226	**0.36400**
60	0.00626	0.03430	0.18434	0.16364	0.04618	**0.40042**
61	0.00278	0.00061	0.18224	0.14097	0.00037	**0.32636**
62	0.00000	0.00014	0.19274	0.08274	0.00010	**0.27557**
63	0.00174	0.00023	0.15533	0.13096	0.00010	**0.28813**
64	0.00000	0.01349	0.17825	0.16047	0.02835	**0.36707**
65	0.02330	0.00711	0.19903	0.14941	0.00286	**0.37459**
66	0.00174	0.00045	0.19836	0.08748	0.00043	**0.28801**
67	0.00035	0.00018	0.17726	0.16390	0.00026	**0.34176**
68	0.00035	0.00002	0.20000	0.00000	0.00000	**0.20035**
69	0.00000	0.00000	0.00000	0.00000	0.00000	**0.00000**
70	0.00348	0.00043	0.18262	0.05903	0.00031	**0.24544**
71	0.00209	0.00041	0.17121	0.16522	0.00026	**0.33877**
72	0.02713	0.00702	0.19606	0.15494	0.01009	**0.38823**
73	0.00035	0.00006	0.18827	0.00000	0.00006	**0.18867**

Table A.2: Results for contributor’s experience of Sling project

Contributor s ID	Total Issues Assigned	Total Files of Commits by Contributor	Mean Time In Hours	Average Complexity	Total Lines Added by Contributor	Contributor' s Experience
74	0.00173	0.00019	0.10675	0.09224	0.00005	**0.20076**
75	0.01123	0.00630	0.18681	0.14447	0.00373	**0.34623**
76	0.08687	0.12573	0.19600	0.16235	0.13760	**0.58282**
77	0.01036	0.00442	0.18937	0.15482	0.00253	**0.35709**
78	0.00017	0.00011	0.07707	0.00000	0.00009	**0.07733**
79	0.00000	0.00003	0.19985	0.12706	0.00002	**0.32693**
80	0.20000	0.20000	0.19757	0.12753	0.13767	**0.66277**
81	0.00259	0.00216	0.13909	0.14635	0.00090	**0.28894**
82	0.00898	0.00670	0.16529	0.06212	0.00574	**0.24212**
83	0.14439	0.15910	0.19660	0.13976	0.20000	**0.68075**
84	0.00000	0.00002	0.00000	0.00000	0.00000	**0.00000**
85	0.01123	0.01555	0.17613	0.13976	0.01134	**0.33846**
86	0.00000	0.00003	0.20000	0.19294	0.00001	**0.39296**
87	0.00121	0.00083	0.16220	0.20000	0.00007	**0.36348**
88	0.00052	0.00670	0.16688	0.12941	0.00002	**0.29682**
89	0.05976	0.02829	0.19288	0.12188	0.01302	**0.38754**
90	0.00225	0.00119	0.13784	0.00000	0.00127	**0.14136**
91	0.01036	0.00778	0.19487	0.14635	0.00235	**0.35394**
92	0.00138	0.00033	0.00000	0.16565	0.00004	**0.16707**
93	0.00829	0.01820	0.19309	0.16000	0.00441	**0.36579**
94	0.00656	0.00517	0.12676	0.15953	0.00462	**0.29748**
95	0.00881	0.00675	0.19011	0.14918	0.00366	**0.35175**
96	0.00950	0.01518	0.19467	0.16565	0.01085	**0.38067**
97	0.04473	0.05722	0.19714	0.14965	0.07698	**0.46849**
98	0.00121	0.00559	0.18492	0.10165	0.00272	**0.29050**
99	0.01572	0.02390	0.19681	0.18306	0.01961	**0.41520**
100	0.02383	0.01359	0.19544	0.10541	0.00804	**0.33273**
101	0.00000	0.00002	0.00000	0.14682	0.00001	**0.14683**
102	0.00017	0.00000	0.01245	0.03859	0.00000	**0.05121**
103	0.00190	0.00368	0.19171	0.15012	0.00087	**0.34459**
104	0.01209	0.04849	0.19422	0.16094	0.01492	**0.38216**
105	0.00155	0.00127	0.07624	0.17129	0.00216	**0.25125**

Table A.3: Results for contributor's experience of Felix project

Contributors ID	Total Issues Assigned	Total Files of Commits by Contributor	Mean Time In Hours	Average Complexity	Total Lines Added by Contributor	Contributor's Experience
106	0.00000	0.00019	0.00000	0.00000	0.00017	**0.00017**
107	0.00130	0.00128	0.16547	0.00000	0.00016	**0.16693**
108	0.00000	0.00002	0.00000	0.10705	0.00013	**0.10718**
109	0.00324	0.00101	0.19798	0.07821	0.00056	**0.27999**
110	0.00940	0.00237	0.19426	0.02244	0.00019	**0.22628**
111	0.00357	0.00208	0.19446	0.16923	0.00170	**0.36895**
112	0.00130	0.00007	0.19951	0.00577	0.00001	**0.20659**
113	0.00097	0.00381	0.19965	0.08077	0.00227	**0.28366**
114	0.10762	0.18291	0.19671	0.14167	0.05974	**0.50573**
115	0.12253	0.08390	0.19810	0.07500	0.01905	**0.41468**
116	0.01135	0.00461	0.19522	0.10385	0.00168	**0.31209**
117	0.01361	0.00835	0.18130	0.00128	0.00438	**0.20058**
118	0.03468	0.02807	0.19798	0.11282	0.00536	**0.35084**
119	0.00065	0.00005	0.00000	0.11090	0.00000	**0.11155**
120	0.03079	0.08820	0.19855	0.11987	0.02585	**0.37506**
121	0.00972	0.00570	0.19014	0.13205	0.00423	**0.33615**
122	0.00454	0.00065	0.14275	0.07756	0.00005	**0.22490**
123	0.00227	0.00027	0.19174	0.10577	0.00020	**0.29998**
124	0.20000	0.20000	0.19846	0.09038	0.20000	**0.68884**
125	0.00097	0.00558	0.16187	0.11154	0.00217	**0.27655**
126	0.00746	0.01651	0.17757	0.08910	0.00502	**0.27915**
127	0.12415	0.06428	0.19783	0.05962	0.02773	**0.40933**
128	0.00097	0.00017	0.18305	0.06282	0.00003	**0.24688**
129	0.01037	0.00804	0.19364	0.15641	0.00177	**0.36220**
130	0.02237	0.02450	0.19353	0.15128	0.00867	**0.37584**
131	0.00000	0.00031	0.00000	0.20000	0.00027	**0.20027**
132	0.00000	0.00000	0.00000	0.00000	0.00000	**0.00000**
133	0.00259	0.00270	0.18091	0.12821	0.00251	**0.31422**
134	0.00454	0.00251	0.14650	0.00000	0.00120	**0.15223**
135	0.02139	0.00763	0.18569	0.11859	0.00364	**0.32932**
136	0.04052	0.01224	0.19467	0.00769	0.00598	**0.24886**
137	0.03793	0.01453	0.18911	0.02821	0.00620	**0.26144**
138	0.02593	0.02308	0.19458	0.13974	0.00497	**0.36523**
139	0.00032	0.00000	0.00000	0.06154	0.00008	**0.06195**
140	0.18444	0.10806	0.19837	0.04808	0.04001	**0.47090**
141	0.00130	0.00007	0.00000	0.07564	0.00001	**0.07694**
142	0.00000	0.00005	0.20000	0.13141	0.00000	**0.33141**
143	0.00194	0.02696	0.16898	0.06154	0.00885	**0.24132**
144	0.00259	0.00227	0.19692	0.18462	0.00102	**0.38514**
145	0.00000	0.00048	0.19669	0.12628	0.00022	**0.32319**
146	0.00389	0.00191	0.15382	0.11218	0.00137	**0.27126**
147	0.00357	0.00111	0.13455	0.09808	0.00092	**0.23712**
148	0.12836	0.03379	0.19386	0.04231	0.00639	**0.37092**

Table A.4: Results for contributor's experience of OFBiz project

Contributors ID	Total Issues Assigned	Total Files of Commits by Contributor	Mean Time In Hours	Average Complexity	Total Lines Added by Contributor	Contributor's Experience
149	0.02241	0.06772	0.19604	0.18330	0.13234	**0.53409**
150	0.00964	0.00253	0.19333	0.07080	0.00062	**0.27439**
151	0.00000	0.00006	0.20000	0.19750	0.00000	**0.39750**
152	0.04821	0.02160	0.19878	0.13898	0.00345	**0.38941**
153	0.00717	0.00157	0.19335	0.14102	0.00029	**0.34183**
154	0.01042	0.00634	0.19822	0.17784	0.00237	**0.38885**
155	0.00195	0.00013	0.19968	0.20000	0.00001	**0.40164**
156	0.00378	0.00219	0.18376	0.16068	0.00049	**0.34871**
157	0.00443	0.00141	0.19962	0.15795	0.00007	**0.36207**
158	0.00026	0.00029	0.20000	0.00000	0.00004	**0.20030**
159	0.00938	0.00120	0.18972	0.18795	0.00011	**0.38717**
160	0.00000	0.00000	0.00000	0.07375	0.00000	**0.07375**
161	0.01759	0.00849	0.19813	0.18250	0.00673	**0.40495**
162	0.00925	0.00558	0.19036	0.17705	0.00145	**0.37811**
163	0.04495	0.01694	0.19824	0.14795	0.00437	**0.39552**
164	0.00235	0.00091	0.18635	0.17011	0.00028	**0.35909**
165	0.20000	0.20000	0.19971	0.16886	0.20000	**0.76857**
166	0.00795	0.00295	0.19551	0.15500	0.00225	**0.36070**
167	0.01577	0.00236	0.19483	0.14193	0.00045	**0.35298**
168	0.00182	0.00045	0.19961	0.14682	0.00011	**0.34837**
169	0.02072	0.00478	0.19722	0.14193	0.00129	**0.36116**
170	0.00039	0.00014	0.19849	0.00000	0.00001	**0.19889**
171	0.00977	0.01798	0.19938	0.16716	0.00747	**0.38378**
172	0.00469	0.00649	0.19865	0.16352	0.00081	**0.36767**
173	0.00026	0.00012	0.19847	0.00239	0.00000	**0.20112**
174	0.00039	0.00010	0.19888	0.17080	0.00000	**0.37008**
175	0.00065	0.00021	0.19418	0.13670	0.00002	**0.33156**
176	0.00000	0.00000	0.14392	0.00000	0.00000	**0.14392**
177	0.00847	0.00294	0.19826	0.16795	0.00234	**0.37702**
178	0.00013	0.00012	0.19005	0.00000	0.00001	**0.19019**
179	0.00104	0.00053	0.19951	0.12102	0.00023	**0.32180**
180	0.00534	0.00083	0.19545	0.14830	0.00014	**0.34923**
181	0.00000	0.00009	0.19981	0.00000	0.00028	**0.20009**

Table A.5: Results for contributor's experience of ActiveMQ project

Contributors ID	Total Issues Assigned	Total Files of Commits by Contributor	Mean Time In Hours	Average Complexity	Total Lines Added by Contributor	Contributor's Experience
182	0.01170	0.00570	0.18174	0.07126	0.00283	**0.26753**
183	0.00039	0.00013	0.01539	0.00000	0.00001	**0.01579**
184	0.00039	0.00035	0.18119	0.02915	0.00052	**0.21125**
185	0.00039	0.00004	0.19775	0.00000	0.00000	**0.19813**
186	0.00078	0.00057	0.19302	0.17247	0.00080	**0.36706**
187	0.00390	0.00172	0.17473	0.19757	0.00129	**0.37749**
188	0.15750	0.14446	0.19798	0.12470	0.06964	**0.54982**
189	0.00351	0.00146	0.15997	0.14899	0.00140	**0.31387**
190	0.00039	0.00009	0.15007	0.09717	0.00010	**0.24772**
191	0.00858	0.00305	0.19067	0.12955	0.00221	**0.33101**
192	0.00234	0.00137	0.13360	0.13765	0.00175	**0.27535**
193	0.00078	0.00031	0.15025	0.05668	0.00010	**0.20781**
194	0.02183	0.06102	0.19747	0.13846	0.03290	**0.39066**
195	0.00234	0.00071	0.17765	0.08907	0.00023	**0.26929**
196	0.02222	0.00724	0.19214	0.11498	0.00176	**0.33110**
197	0.02534	0.01007	0.19377	0.11579	0.00223	**0.33713**
198	0.00039	0.00022	0.16743	0.12389	0.00014	**0.29185**
199	0.00078	0.00115	0.16493	0.10688	0.00022	**0.27282**
200	0.00156	0.00260	0.14288	0.17652	0.00081	**0.32177**
201	0.00819	0.00389	0.18198	0.13846	0.00130	**0.32993**
202	0.03041	0.02265	0.19428	0.11012	0.00970	**0.34451**
203	0.00039	0.00009	0.19175	0.06397	0.00003	**0.25614**
204	0.00000	0.00071	0.19973	0.13360	0.00029	**0.33362**
205	0.00078	0.00124	0.19523	0.06478	0.00054	**0.26132**
206	0.00000	0.00000	0.00000	0.00000	0.00005	**0.00005**
207	0.00468	0.00163	0.17837	0.08178	0.00100	**0.26583**
208	0.20000	0.16600	0.19806	0.13441	0.10122	**0.63369**
209	0.00000	0.00004	0.00000	0.00000	0.00000	**0.00000**
210	0.07018	0.06194	0.19725	0.11174	0.20000	**0.57917**
211	0.00117	0.00115	0.20000	0.13522	0.00062	**0.33701**
212	0.00585	0.00839	0.13588	0.18138	0.00630	**0.32940**
213	0.00507	0.00508	0.04891	0.12955	0.00139	**0.18492**
214	0.00117	0.00247	0.19182	0.00000	0.00051	**0.19350**
215	0.00000	0.00022	0.00000	0.17976	0.00015	**0.17991**
216	0.00936	0.00623	0.18059	0.11417	0.00265	**0.30676**
217	0.09669	0.18327	0.19321	0.12713	0.05780	**0.47483**
218	0.00000	0.00004	0.00000	0.12470	0.00007	**0.12476**
219	0.00000	0.00000	0.00000	0.00000	0.00000	**0.00000**
220	0.00000	0.00084	0.00000	0.00000	0.00012	**0.00012**
221	0.05692	0.05024	0.18719	0.14332	0.06394	**0.45136**
222	0.00078	0.00022	0.08705	0.00000	0.00001	**0.08784**
223	0.00078	0.00185	0.12387	0.14899	0.00019	**0.27383**
224	0.01793	0.00737	0.18659	0.12874	0.00255	**0.33582**
225	0.00312	0.00053	0.03187	0.20000	0.00013	**0.23512**
226	0.00039	0.00018	0.05952	0.00000	0.00000	**0.05991**
227	0.00000	0.00022	0.00000	0.00000	0.00001	**0.00001**
228	0.01481	0.00693	0.19555	0.14818	0.00531	**0.36385**
229	0.00000	0.00000	0.00000	0.00000	0.00000	**0.00000**
230	0.00000	0.00256	0.00000	0.19190	0.00472	**0.19662**
231	0.00000	0.00018	0.00000	0.15789	0.00019	**0.15809**
232	0.00000	0.00013	0.00000	0.11012	0.00008	**0.11020**
233	0.00000	0.00009	0.00000	0.17085	0.00005	**0.17090**
234	0.00156	0.03430	0.19489	0.02915	0.03360	**0.25920**
235	0.00000	0.00004	0.00000	0.16032	0.00005	**0.16037**
236	0.00312	0.00508	0.19478	0.12227	0.00257	**0.32274**
237	0.00078	0.00225	0.15762	0.17895	0.00087	**0.33821**
238	0.12554	0.09978	0.19458	0.13603	0.05716	**0.51332**
239	0.00546	0.00296	0.18511	0.12632	0.00085	**0.31773**
240	0.17076	0.20000	0.19675	0.15061	0.19319	**0.71131**
241	0.02963	0.01770	0.19778	0.09474	0.00771	**0.32986**
242	0.07914	0.05625	0.19736	0.12713	0.05699	**0.46062**
243	0.00195	0.00062	0.15125	0.06478	0.00126	**0.21924**
244	0.00039	0.00044	0.00000	0.06154	0.00004	**0.06197**
245	0.00000	0.00013	0.00000	0.05749	0.00031	**0.05780**
246	0.00234	0.00110	0.13073	0.19757	0.00009	**0.33073**

Appendix B: Results of Contribution Efficiency

Table B.1: Results for contribution efficiency of Maven project

Contributors ID	Difference Avg. Complexity	Difference Avg. Depth	Difference Avg. LOC	Difference Avg. Percentage Comments	Contribution Efficiency
27	0.00000	0.00000	0.00000	0.00000	**0.00000**
28	0.00000	0.00000	0.00000	0.00000	**0.00000**
29	0.25000	0.15889	0.07190	0.07372	**0.55450**
30	0.12431	0.09262	0.11914	0.06539	**0.40146**
31	0.13904	0.07830	0.11298	0.05742	**0.38774**
32	0.12981	0.08158	0.15954	0.12332	**0.49424**
33	0.13151	0.08777	0.11481	0.05488	**0.38897**
34	0.14233	0.08032	0.16586	0.09788	**0.48638**
35	0.15228	0.08872	0.14862	0.12577	**0.51540**
36	0.00000	0.00000	0.00000	0.00000	**0.00000**
37	0.14432	0.09262	0.10065	0.06262	**0.40021**
38	0.15767	0.23251	0.29420	0.00000	**0.68438**
39	0.13952	0.07605	0.03903	0.02653	**0.28113**
40	0.10808	0.00000	0.40000	0.09176	**0.59984**
41	0.14432	0.08268	0.14585	0.11370	**0.48654**
42	0.14432	0.09262	0.10887	0.05429	**0.40010**
43	0.05518	0.09359	0.10022	0.06230	**0.31129**
44	0.12968	0.05594	0.14100	0.02752	**0.35414**
45	0.11406	0.05141	0.17095	0.03850	**0.37492**
46	0.18715	0.25000	0.00000	0.15561	**0.59277**
47	0.13029	0.07652	0.10086	0.06938	**0.37706**
48	0.15489	0.04447	0.14984	0.04011	**0.38931**
49	0.00000	0.00000	0.00000	0.00000	**0.00000**
50	0.15303	0.11200	0.01861	0.03281	**0.31645**
51	0.13128	0.09257	0.10019	0.06247	**0.38651**
52	0.00000	0.00000	0.00000	0.00000	**0.00000**
53	0.00000	0.00000	0.00000	0.00000	**0.00000**
54	0.12777	0.07907	0.07316	0.05664	**0.33665**
55	0.11493	0.09250	0.10074	0.06263	**0.37079**
56	0.09852	0.09173	0.09888	0.06712	**0.35626**
57	0.14549	0.08660	0.12045	0.05782	**0.41035**
58	0.13151	0.09262	0.15087	0.05891	**0.43392**
59	0.13630	0.05930	0.15459	0.05419	**0.40437**
60	0.00000	0.08874	0.10333	0.05597	**0.24804**
61	0.13167	0.07605	0.10620	0.05804	**0.37196**
62	0.15607	0.08710	0.08696	0.06539	**0.39552**
63	0.11110	0.22515	0.03595	0.12508	**0.49728**
64	0.09548	0.08602	0.09923	0.06244	**0.34317**
65	0.14402	0.03386	0.14454	0.05559	**0.37801**
66	0.14432	0.08434	0.22698	0.25000	**0.70565**
67	0.14192	0.07882	0.12873	0.06123	**0.41069**
68	0.00000	0.00000	0.00000	0.00000	**0.00000**
69	0.00000	0.00000	0.00000	0.00000	**0.00000**
70	0.07024	0.02746	0.21048	0.12647	**0.43465**
71	0.14130	0.08894	0.16776	0.03486	**0.43285**
72	0.12494	0.09186	0.27971	0.20001	**0.69652**
73	0.00000	0.00000	0.00000	0.00000	**0.00000**

Table B.2: Results for contribution efficiency of Sling project

Contributors ID	Difference Avg. Complexity	Difference Avg. Depth	Difference Avg. LOC	Difference Avg. Percentage Comments	Contribution Efficiency
74	0.12059	0.19475	0.15532	0.10094	**0.57159**
75	0.11542	0.20379	0.31193	0.10449	**0.73564**
76	0.12505	0.20213	0.15660	0.11803	**0.60181**
77	0.14649	0.21264	0.24547	0.16430	**0.76890**
78	0.00000	0.00000	0.00000	0.00000	**0.00000**
79	0.25000	0.23888	0.15797	0.19901	**0.84586**
80	0.12489	0.20832	0.13868	0.20033	**0.67222**
81	0.11643	0.17471	0.21772	0.18120	**0.69006**
82	0.19560	0.21766	0.30460	0.15561	**0.87348**
83	0.13800	0.21182	0.18817	0.21535	**0.75335**
84	0.00000	0.00000	0.00000	0.00000	**0.00000**
85	0.12974	0.20751	0.19774	0.25000	**0.78499**
86	0.15385	0.13067	0.18257	0.00000	**0.46709**
87	0.12650	0.22459	0.10384	0.19043	**0.64537**
88	0.00000	0.00000	0.00000	0.00000	**0.00000**
89	0.13648	0.22520	0.15877	0.15916	**0.67962**
90	0.21633	0.25000	0.17546	0.16135	**0.80314**
91	0.13003	0.23386	0.09696	0.16336	**0.62422**
92	0.13263	0.00000	0.13336	0.17496	**0.44095**
93	0.13345	0.22403	0.10558	0.17458	**0.63763**
94	0.12748	0.21830	0.12106	0.18432	**0.65116**
95	0.15999	0.24441	0.00000	0.21181	**0.61621**
96	0.12267	0.21155	0.12220	0.16667	**0.62308**
97	0.12732	0.18869	0.15762	0.18996	**0.66359**
98	0.06631	0.19601	0.08416	0.13996	**0.48644**
99	0.13849	0.19815	0.12653	0.16806	**0.63124**
100	0.11866	0.18876	0.40000	0.17393	**0.88135**
101	0.09416	0.19601	0.21210	0.17058	**0.67285**
102	0.00000	0.20417	0.14321	0.14871	**0.49609**
103	0.12066	0.22112	0.09976	0.16342	**0.60495**
104	0.13796	0.22442	0.08039	0.21008	**0.65285**
105	0.13329	0.16674	0.32937	0.06925	**0.69866**

Table B.3: Results for contribution efficiency of Felix project

Contributors ID	Difference Avg. Complexity	Difference Avg. Depth	Difference Avg. LOC	Difference Avg. Percentage Comments	Contribution Efficiency
106	0.00000	0.00000	0.00000	0.00000	**0.00000**
107	0.00000	0.00000	0.00000	0.00000	**0.00000**
108	0.00000	0.00000	0.00000	0.00000	**0.00000**
109	0.09039	0.02547	0.40000	0.01877	**0.53464**
110	0.09397	0.01912	0.26659	0.03991	**0.41959**
111	0.10826	0.03262	0.29688	0.05488	**0.49265**
112	0.09906	0.02803	0.28685	0.05475	**0.46868**
113	0.09526	0.03583	0.27729	0.07808	**0.48646**
114	0.10039	0.02912	0.31711	0.05061	**0.49723**
115	0.09645	0.02618	0.28827	0.04763	**0.45853**
116	0.10232	0.03826	0.31502	0.04094	**0.49655**
117	0.11246	0.03846	0.22963	0.05336	**0.43390**
118	0.10170	0.03231	0.27763	0.05323	**0.46487**
119	0.09867	0.02082	0.26636	0.05363	**0.43947**
120	0.10402	0.04070	0.26214	0.05634	**0.46321**
121	0.00000	0.00000	0.00000	0.00000	**0.00000**
122	0.10091	0.02975	0.26921	0.04403	**0.44390**
123	0.09889	0.02723	0.24587	0.06064	**0.43263**
124	0.11031	0.03108	0.32095	0.04957	**0.51191**
125	0.05950	0.00000	0.37062	0.08693	**0.51705**
126	0.10869	0.01486	0.35835	0.02268	**0.50457**
127	0.10079	0.04806	0.31147	0.05093	**0.51124**
128	0.09398	0.00601	0.31143	0.04784	**0.45926**
129	0.10273	0.03687	0.27009	0.04473	**0.45442**
130	0.10098	0.03499	0.29625	0.05517	**0.48739**
131	0.25000	0.25000	0.00000	0.24858	**0.74858**
132	0.00000	0.00000	0.00000	0.00000	**0.00000**
133	0.10071	0.02947	0.27169	0.05426	**0.45612**
134	0.10309	0.04817	0.24084	0.03664	**0.42874**
135	0.10421	0.03210	0.29678	0.05429	**0.48739**
136	0.11127	0.03736	0.28537	0.05243	**0.48643**
137	0.10292	0.03086	0.33142	0.04288	**0.50808**
138	0.09844	0.02145	0.31366	0.04327	**0.47683**
139	0.08676	0.01902	0.33602	0.25000	**0.69180**
140	0.11144	0.03122	0.29764	0.05589	**0.49618**
141	0.10252	0.03824	0.31348	0.05731	**0.51155**
142	0.11330	0.03343	0.24382	0.08123	**0.47178**
143	0.10176	0.03494	0.21803	0.00332	**0.35806**
144	0.09871	0.03137	0.26577	0.03279	**0.42865**
145	0.00000	0.01722	0.30734	0.00000	**0.32455**
146	0.09030	0.02860	0.35012	0.04278	**0.51179**
147	0.09118	0.01979	0.36910	0.05385	**0.53392**
148	0.14851	0.04011	0.28194	0.06409	**0.53465**

Table B.4: Results for contribution efficiency of OFBiz project

Contributors ID	Difference Avg. Complexity	Difference Avg. Depth	Difference Avg. LOC	Difference Avg. Percentage Comments	Contribution Efficiency
149	0.19189	0.10590	0.13206	0.14556	**0.57542**
150	0.18429	0.08229	0.22876	0.09129	**0.58663**
151	0.18570	0.09051	0.00000	0.13426	**0.41048**
152	0.18399	0.06601	0.20954	0.06553	**0.52507**
153	0.16933	0.03251	0.27988	0.08419	**0.56591**
154	0.15416	0.05153	0.38635	0.05854	**0.65058**
155	0.18570	0.08745	0.23673	0.12338	**0.63327**
156	0.18534	0.09289	0.13573	0.12015	**0.53411**
157	0.18453	0.09389	0.10914	0.11378	**0.50133**
158	0.18570	0.08745	0.37268	0.11923	**0.76507**
159	0.24826	0.11448	0.08313	0.15821	**0.60408**
160	0.18407	0.08745	0.13257	0.12338	**0.52747**
161	0.17118	0.04399	0.19641	0.08967	**0.50125**
162	0.19047	0.05373	0.29773	0.16711	**0.70904**
163	0.16890	0.05249	0.18297	0.13582	**0.54018**
164	0.17849	0.08849	0.24116	0.25000	**0.75815**
165	0.18979	0.08167	0.16131	0.10526	**0.53803**
166	0.17437	0.03522	0.28324	0.01558	**0.50842**
167	0.11127	0.09293	0.18091	0.13624	**0.52135**
168	0.17576	0.08313	0.13925	0.11996	**0.51810**
169	0.18017	0.06651	0.40000	0.03957	**0.68625**
170	0.00000	0.00000	0.00000	0.00000	**0.00000**
171	0.25000	0.00000	0.29887	0.05307	**0.60194**
172	0.19127	0.08793	0.32519	0.08289	**0.68729**
173	0.00000	0.25000	0.11019	0.12470	**0.48489**
174	0.18570	0.07520	0.14204	0.02177	**0.42472**
175	0.18570	0.08745	0.12310	0.12338	**0.51963**
176	0.00000	0.00000	0.00000	0.00000	**0.00000**
177	0.18540	0.07727	0.17832	0.12093	**0.56192**
178	0.00000	0.00000	0.00000	0.00000	**0.00000**
179	0.11094	0.12623	0.21779	0.00000	**0.45497**
180	0.18925	0.06682	0.26240	0.08403	**0.60251**
181	0.00000	0.00000	0.00000	0.00000	**0.00000**

Table B.5: Results for contribution efficiency of ActiveMQ project

Contributors ID	Difference Avg. Complexity	Difference Avg. Depth	Difference Avg. LOC	Difference Avg. Percentage Comments	Contribution Efficiency
182	0.12831	0.15818	0.08463	0.16330	**0.53441**
183	0.00000	0.00000	0.00000	0.00000	**0.00000**
184	0.13451	0.10705	0.40000	0.09738	**0.73895**
185	0.00000	0.00000	0.00000	0.00000	**0.00000**
186	0.14031	0.19263	0.19294	0.14786	**0.67374**
187	0.15862	0.16900	0.23146	0.10533	**0.66441**
188	0.13977	0.20089	0.04922	0.17418	**0.56406**
189	0.14103	0.19877	0.19747	0.16605	**0.70332**
190	0.12464	0.21597	0.03536	0.20500	**0.58097**
191	0.14016	0.18096	0.07818	0.14417	**0.54347**
192	0.13739	0.14075	0.13981	0.05150	**0.46945**
193	0.14744	0.25000	0.21121	0.24500	**0.85365**
194	0.14066	0.18486	0.07404	0.18212	**0.58169**
195	0.13740	0.19678	0.00000	0.18045	**0.51464**
196	0.13051	0.20972	0.03748	0.18861	**0.56631**
197	0.13972	0.19121	0.05415	0.19267	**0.57774**
198	0.15641	0.18329	0.05774	0.16233	**0.55978**
199	0.14042	0.20689	0.02352	0.18846	**0.55929**
200	0.13836	0.19841	0.03368	0.21693	**0.58739**
201	0.13894	0.21705	0.02655	0.21118	**0.59372**
202	0.14267	0.19345	0.03652	0.17566	**0.54831**
203	0.14103	0.20916	0.05135	0.17667	**0.57820**
204	0.13731	0.18436	0.08751	0.15762	**0.56680**
205	0.13818	0.20714	0.10345	0.17796	**0.62673**
206	0.14815	0.24319	0.24318	0.20167	**0.83619**
207	0.14031	0.21426	0.04869	0.15833	**0.56160**
208	0.14029	0.19093	0.07038	0.17558	**0.57718**
209	0.00000	0.00000	0.00000	0.00000	**0.00000**
210	0.14082	0.19040	0.05374	0.17790	**0.56286**
211	0.14223	0.15434	0.13324	0.11868	**0.54850**
212	0.14043	0.19582	0.05071	0.16647	**0.55342**
213	0.13811	0.19231	0.01791	0.17950	**0.52783**
214	0.00000	0.00000	0.00000	0.00000	**0.00000**
215	0.25000	0.23979	0.00872	0.23167	**0.73018**
216	0.13445	0.15341	0.06213	0.16341	**0.51340**
217	0.13836	0.19649	0.05330	0.18337	**0.57152**
218	0.14103	0.01856	0.21654	0.18500	**0.56113**
219	0.00000	0.00000	0.00000	0.00000	**0.00000**
220	0.00000	0.00000	0.00000	0.00000	**0.00000**
221	0.14008	0.19221	0.10045	0.17555	**0.60829**
222	0.00000	0.00000	0.00000	0.00000	**0.00000**
223	0.14447	0.04012	0.01405	0.19222	**0.39086**
224	0.13942	0.17413	0.05468	0.16505	**0.53328**
225	0.14055	0.16151	0.05490	0.17056	**0.52752**
226	0.00000	0.00000	0.00000	0.00000	**0.00000**
227	0.00000	0.00000	0.00000	0.00000	**0.00000**
228	0.13589	0.23276	0.06218	0.20159	**0.63242**
229	0.00000	0.00000	0.00000	0.00000	**0.00000**
230	0.00000	0.00000	0.00000	0.00000	**0.00000**
231	0.13651	0.16832	0.11707	0.18389	**0.60579**
232	0.14316	0.20916	0.13661	0.18333	**0.67226**
233	0.14103	0.20916	0.01405	0.19167	**0.55590**
234	0.00000	0.13428	0.33644	0.00000	**0.47072**
235	0.14209	0.20235	0.15792	0.18500	**0.68737**
236	0.13329	0.18505	0.08715	0.13351	**0.53900**
237	0.13821	0.18131	0.05062	0.15765	**0.52780**
238	0.14072	0.19601	0.05547	0.18003	**0.57222**
239	0.13919	0.20111	0.03352	0.17664	**0.55046**
240	0.13871	0.19476	0.09789	0.18038	**0.61175**
241	0.14108	0.21803	0.05378	0.17741	**0.59030**

242	0.13994	0.19378	0.05501	0.17473	**0.56347**
243	0.13853	0.12067	0.39772	0.25000	**0.90692**
244	0.09350	0.00000	0.03149	0.05712	**0.18211**
245	0.13818	0.20916	0.02471	0.18833	**0.56037**
246	0.14103	0.22277	0.02790	0.18500	**0.57670**

Appendix C: Scatter Plots

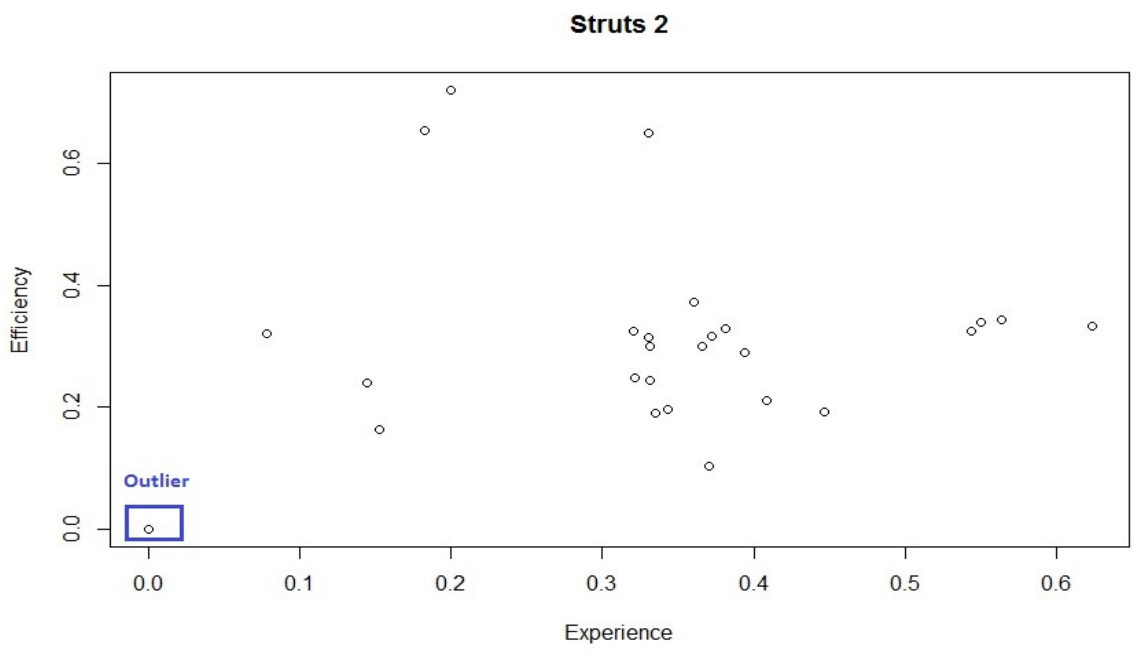


Figure 5.1: Scatter plot for Struts 2 project

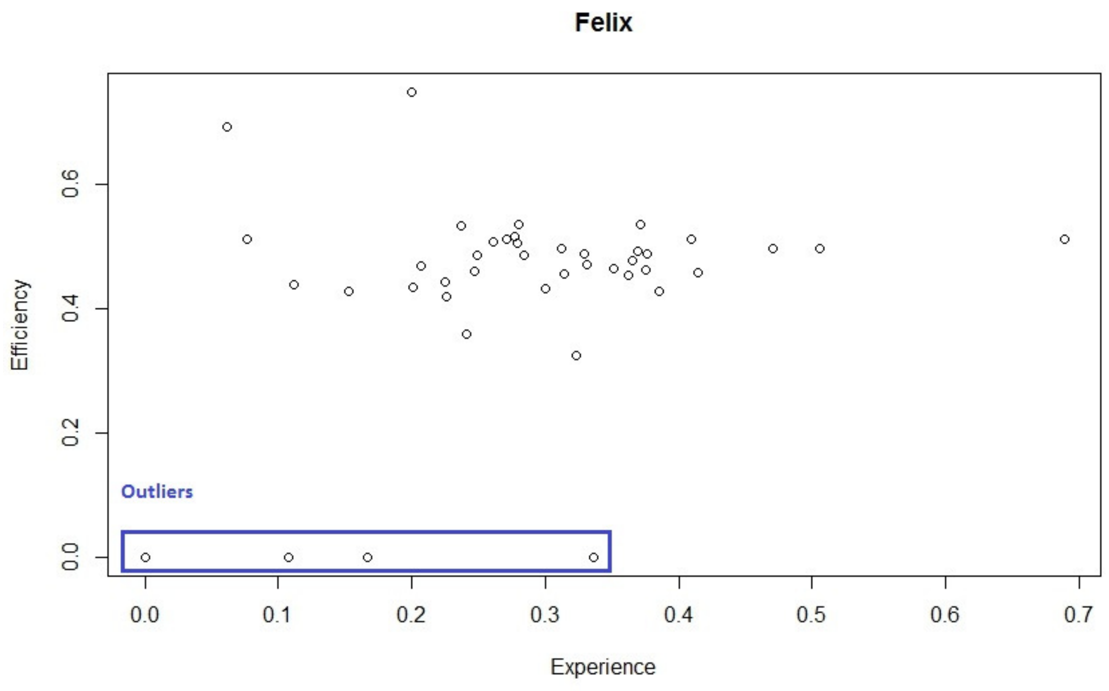


Figure 5.2: Scatter plot for Felix project

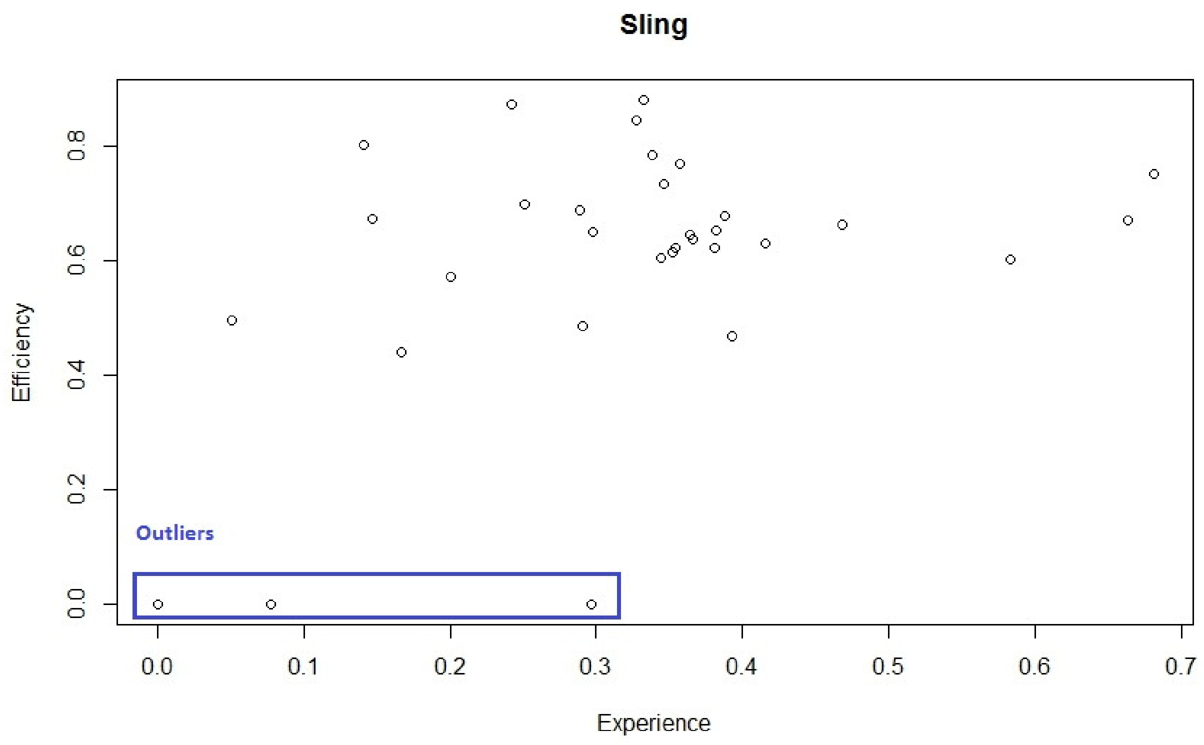


Figure 5.3: Scatter plot for Sling project

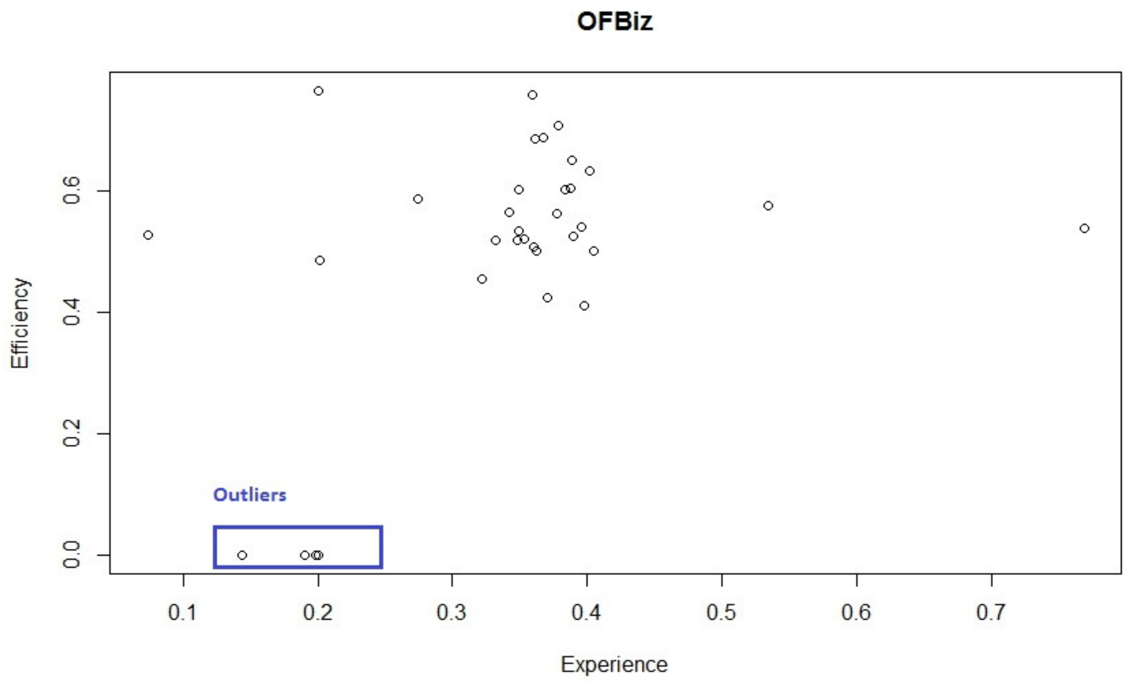


Figure 5.4: Scatter plot for OFBiz project