An empirical study on developer related factors introducing bugs in commits

Vytautas Jankauskas
vytautas.jankauskas@stud-inf.unibz.it

***Abstract*—the purpose of this paper is to analyze developer related factors that can influence the likelihood on a commit to induce bug. Since every software system faces bugs during the development process, it is very important to find out all possible reasons which can cause an issue in a software system. This empirical study focuses on the analysis of relations between bug inducing commits and specific factors: (i) number of days, (ii) number of commits, (iii) number of lines of code (LOC) changed and (iv) number of developers who modified the same file between a bug-inducing commit and the commit before it made by the same developer. 7 Git Apache projects, which had been developed using Jira issue tracker were chosen at random and analyzed. The results of this study show that the defined factors have an influence on a possibility to induce bugs in some of analyzed projects.**

***Key words*—Commit, Statistical analysis, Issue, Software Repository.**

I. INTRODUCTION

Open source software systems are often developed by many developers from all around the world. Those systems go through a long path of development during the years. Open source projects were selected as a data source for this research because all their data (including a source code, changes in a source code, issues etc.) is available publicly. The input to the development of a software project by each developer can be seen and tracked in software repositories using version control system (VCS). VCS is a tool that helps to manage and track different versions of the software [2]. One of the most popular VCS nowadays is Git Version Control System. Therefore, projects which were developed using it were selected for this empirical study. The method of investigating and extracting these data from software repositories is called *Mining Software Repositories* (MSR) [1].

During the evolution of any software project developers have to deal with simple and complex tasks and that leads to the higher possibility of inducing a bug in a software system. A tool for managing and tracking bugs inside a software system is called Issue Tracker Software, or issue tracker. In open source projects issue trackers help not only to manage the lifecycle of the issue, showing the data when the issue was created or assigned to a developer, started being resolved and closed. As open source software systems use open source issue trackers, they also are a place where users and testers of a software system can see the progress of solving issues, suggest additional features or introduce a discussion on possible ways to solve the particular issue. This is the advantage of the public availability of the data in open source projects. Jira issue tracker (https://issues.apache.org/jira) was selected for this empirical study because it contains software projects various in size, complexity, duration of the development, number of developers who put their effort into the software development.

The data which can be extracted from software repositories include information about any changes in the code, the time of the change, the author of that change, the purpose of that change etc. These data can be used for various investigations. One of possible investigations was selected as a topic for this empirical study. It covers the possible influence of developer related factors to the induction of a bug inside a software system during the development phase.

All the data for this research were extracted from 2 sources:

1. Open source Java systems, using Apache codebase from Git VCS, available at: https://git.apache.org/.
2. Issue information of these Apache projects tracked using Jira issue tracker, available at: https://issues.apache.org/jira/.

As Git VCS tracks all the information about any changes done in the software project, every submitted change is called a commit. It contains data about the author of the commit, the message of a commit written by its author, the date of the commit, a list of files he/she has added, modified, deleted and the modifications done inside those files. Changes can be tracked by comparing different commits, files modified, commit messages etc.

In this paper, a term “bug-inducing commit” is referred to the commit extracted from Git VCS and Jira issue tracker using a Git SZZ algorithm [4]. Other commits, named as “clean commits” are all the rest commits extracted from Git VCS data.

Other data needed for this research is collected from Git software repositories and includes:

1. A Number of days (referred as “DAYS” in this empirical study) between 2 commits where the same file was modified by the same developer.
2. A Number of commits (referred as “COMMITS”) between 2 commits where the same file was modified by the same developer.
3. A Number of lines of code modified (referred as “LOC”) between 2 commits where the same file was modified.
4. A Number of developers who have modified the same file between 2 commits where the same file was

V. Jankauskas is bachelor student with the Free University of Bolzano, E-mail: Vytautas.jankauskas@stud-inf.unibz.it.
G. Bavota is Assistant Professor with the Free University of Bolzano, E-mail: Gabriele.Bavota@unibz.it.

modified by the same developer (referred as "DEVELOPERS").
5. The email address of the owner (developer) who submitted 2 commits where the same file was modified.
6. The date of the first commit where the same file was modified.
7. The date of the second commit where the same file was modified.
8. The id of the first commit where the same file was modified.
9. The id of the second commit where the same file was modified.
10. The name of the file which was modified in both commits (referred as "file name").

After extracting and collecting all the necessary information from software repositories and bug-report repositories, the descriptive statistics and statistical tests were performed. The results of this study leads to the conclusion that all analyzed factors influence a possibility of an introduction of a bug in some software systems.

***Structure of this paper*—**the Chapter II is used to define our empirical study design and its research questions as well as to provide specific details about the context selection, data extraction process, Git commands used, the algorithm and the structure of the Java program used to perform this empirical study. Also, the details about the replication package to perform this empirical study are provided as well as the data sets extracted and used to run statistical tests in this research.

The Chapter 3 reviews the results extracted from software repositories. The Chapter 4 for describes the process of the analysis of results in detail. Similar studies are reviewed in the Chapter 5. Threats to the validity of this research are described in the Chapter 6. Conclusion and future work is detailed in the Chapter 7.

Appendix 1 contains the box plots of the data sets which were considered as less representing the factors analyzed.

II. Study Design

The goal of this study is to analyze developer related factors to influence an induction of bugs inside a software system. This requires analysis of developers' submitted commits as well as analysis of bug reports.

The purpose of this study is to investigate which selected developer related factors influence the induction of a bug in a software project.

Research question formulated for this empirical study is:

- **RQ1:** *Which are the factors influencing the likelihood for a commit to induce bugs inside a software system?*
  *However, **RQ_1** contains 4 different factors which can impact the bug introduction in a project.*

These factors are:

1. Number of days between 2 specific commits done by the same developer where the same file was modified.
   During the development of a software system developers participate in this process very differently. Some of them are modifying files very frequently, others not so often. The aim of the analysis of this factor is to check if the time interval (in days) between the modifications of code files (specifically Java code files in this empirical study) can influence the induction of a bug inside a software project. Also, the analysis of this factor is chosen due to assumption that a long period of time and a short period of time may cause a different influence for an induction of a bug inside software project because the developer may remember more or less details about a code file as well as the focus on a code file modification process may differ regarding to the time since the last modification of the code file done by the same developer.
2. Total number of commits between 2 commits where the same file was modified by the same developer.
   The usage of a version control system in software development provides a lot of useful data. One type of data to extract from software repositories is the total number of commits made between any 2 commits during the project development phase. The path of a project development is not necessarily regularly distributed within its timeline. Therefore, the amount of days between 2 commits was not the only selected factor which may influence the induction of a bug. A number of commits may differ between 2 commits even if the number of days between those 2 commits is equal.
3. Number of lines of code (LOC) changed in a file between 2 specified commits done by the same developer.
   This factor was chosen assuming that a different number of LOC changed in the same file between commits can differently influence the possibility to induce a bug. As the higher number of LOC changed may cause developer to focus less on the specific details of a module of a system or the lower number of LOC changed may be caused because the developer was doing quick fixes and not paying enough attention to the quality of code etc.
4. Number of developers who modified the same file between 2 specified commits submitted by a single developer.
   A number of developers was chosen as one more factor to analyze because the amount of developers who modified the same file in a given interval of time between 2 commits may influence the possibility of a bug induction in a project. If no other developers modified the code file during some time it may be easier or harder for a developer to modify the same code file without introduction of a bug inside it.

All the factors reviewed lead to the following null hypotheses:

- ***Null hypothesis 1 ($H1_0$):*** The possibility of introducing a bug inside a software system is not related with the duration (in days) between a

bug-introducing commit and a previous commit where the bug-introducing file was modified by the same developer.

- ***Null hypothesis 2 ($H2_0$):*** The possibility of introducing a bug inside a software system is not related with a number of commits between a bug-introducing commit and a previous commit where a bug-introducing file was modified by the same developer.
- ***Null hypothesis 3 ($H3_0$):*** The possibility of introducing a bug inside a software system is not related with a number of LOC modified in a file in the bug-introducing commit and a previous commit done by the same developer.
- ***Null hypothesis 4 ($H4_0$):*** The possibility of introducing a bug inside a software system is not related with a number of developers who have modified the file between a bug-introducing commit and a previous commit done by the same developer.

A. Context Selection

All data for this project is extracted from Git VCS software repositories and Jira issue tracking software. A content of a single Git repository used for this research is a full project development history of an open source Java Apache software system. Furthermore, there is a relation between a single Git software repository and a Jira bug report repository as both of them refer to the same project and that helps us to track all the lifecycle of a software system.

For this empirical study 7 projects developed in Java programming language and using Apache codebase were selected at random from The Apache Software Foundation collection of read-only Git mirrors of Apache codebases. The contents of every project and their full version histories were extracted from the GitHub - a Web-based Git repository hosting service.

B. Data Extraction Process

This chapter describes how the data for this empirical study were collected and extracted. The data for each of the selected projects were extracted separately. The main steps for collecting the data were:

1. To clone Git repositories of selected projects to the local machine.
2. To extract all issues of related repositories from Jira issue tracking software.
3. To extract general information of every commit of a project.
4. To implement and use Git SZZ algorithm.
5. To sort and put into a single CSV file all the necessary data of the commit.

C. Git commands.

Some git command were necessary to use to extract the data from the repositories.

In order to clone a Git repository to a local machine, the *git clone* command was used:

- *'git clone <repository>'* clones a repository into a new directory. Parameter <repository> is a url to the git repository on GitHub (e.g. https://github.com/apache/ant-ivy.git).

The *git log* command was used to extract information about all the commits from the repository:

- *'git log --first-parent --name-status --date=iso --stat HEAD --pretty=format:\"<commit-id>%h</commit-id><author-email>%ae</author-email><date>%ad</date><message>%s</message>\" > outputPath'.*
  Parameter *--first-parent* is used to follow only the first parent commit upon seeing a merge commit. Parameter *--name-status* is used to reduce the information about the commit to show only names and status of changed files.
  Parameter *--date-iso* sets up the format of the date of the commit to show timestamps in ISO 8601-like format.
  Parameter *--stat* generates a diffstat.
  Parameter *HEAD* shows only commits in the specified revision range. HEAD is a reference to the currently checked out commit.
  Parameter *--pretty=format:* pretty-prints the contents of the commit logs in a given format. In the case of this project the content is print into a *log.txt* file for every project.
  Parameter *outputPath* is used to set a name for the output file.

The *git diff* command was used to show changes between commits:

- *'git diff <commit id1> <commit id2> --fileName > outputPath*'.
  Parameters *<commit id1>* and *<commit id2>* refer to the ID's of commits between which we want to see the changes.
  Parameter *--fileName* is used to restrict the changes between 2 commits only to the changes done inside the specified file.
  Parameter *outputPath* is used to specify the name of the output file where the data is stored after *git diff* command is executed.
- '*git diff <commit id1>^ <commit id2> > outputPath'.*
  The parameters are the same as in the *git diff* command above. The only difference is that the command has no specific file name set and it outputs the changes in all the files between 2 commits.

For the implementation of the Git SZZ algorithm the *git blame* command was used. It annotates each line in the given file with information from the revision which last modified the line:

- *'git blame -w -c <file name> > outputPath'.*
  Parameter *<file name>* is used to specify the name of the file which is to be revised.

The *git checkout* command switches branches or restores working tree files:

- *'git checkout <commit id>'.*
  Parameter *<commit id>* is set to specify the past version of the repository.

The *git rev-list* command lists commit objects in reverse chronological order.

- *'git rev-list <commit id1> ... <commit id2> > outputPath'.*
  Parameters *commit id1* and *commit id2* are used to restrict the list of commit objects only to the ones submitted between those 2 specified commits.

D. The algorithm of the Java program.

A Java program was created to extract all the data necessary for this research. The main parameters to extract data from the each project repository are:

1. The URL of the project repository on a GitHub.
2. The local repository path. It tells what will be the location of the cloned repository in a local machine.
3. The code of a selected project in a Jira issue tracking system. It is necessary to extract all the issues from the issue tracker using Jira REST API [5].

The execution time of the Java program for every analyzed project differs depending on the size of the project (in a number of total commits, in a number of issues) from 15 minutes to more than 7 hours.

The algorithm of the Java program written for this empirical study works as following:

1. It clones the repository of a selected software project to the local machine from the GitHub using *'git clone'* command.
2. It checkouts to the main ("master") branch using *'git checkout'* command.
3. It extracts the information about all commits from Git repository to the xml-formatted text file using the *'git log'* command.
4. It parses the text file with all the commits data and puts it to the *CommitBean* vector.
5. It extracts all the issues of the project from the Jira issue tracking system.
6. The *CommitBean* vector, which contains the data of all the commits is being parsed and the data of commits which were submitted to fix specific issues are being put into another *CommitBean* vector.
7. The *'git diff'* command is being executed to get the list of LOC changed on every bug-fixing commit.
8. The repository of the project is being checked out to the needed version by using the *'git checkout'* command.
9. The checked out version of a repository is being blamed using the *'git blame'* command (for each file in the bug-fixing commit separately) and it outputs the list of every line of code of that file with additional information. The additional information includes: the id of the commit in which a line of code was modified, the date when that commit was done, the name of the author of that commit. The output is stored into a text file.
10. The file with the output from the *'git blame'* command is being parsed and only the most recently changed lines of code (which have the latest commit date out of all commits in the blamed file) are considered as the ones which introduced a bug. Such a commit is being marked as bug-inducing (buggy) then. All the other commits are being marked as clean.

Steps 5-10 sums up the implementation of the Git SZZ algorithm for this empirical study.

11. The *CommitBean* vector, containing all the data about every commit in the repository is being parsed. The id's of the commits, containing the same modified Java code file are being assigned to the component of a *FinalBean* vector as well as the name of the file, the email of the developer who made those commits and the dates of both commits. The time in days between the 2 specified commits is then being calculated and assigned to the same component of a vector.
12. The *FinalBean* vector is being parsed and the *'git rev-list'* command for every component of a vector is being executed. This command outputs the list of all the commits made between 2 specified commits. The size of the list (a number of commits) is assigned to a parsed component of a vector. Also, the number of developers who modified the same file between 2 commits is calculated and assigned.
13. The number of LOC changed between 2 specified commits is calculated then using the *'git diff'* command. It is also added to the component of a *FinalBean* vector which is being parsed.
14. The values of a *FinalBean* vector are transferred to the *<project name>.csv* file as one component of a vector is represented by a single line in a CSV file.

All the steps mentioned above ensures the extraction of all the data necessary to further analysis of the factors selected for this research (number of days between 2 specified commits, number of commits between 2 specified commits, number of LOC changed between 2 specified commits and number of developers who modified the same file between 2 specified commits). Every line of the final CSV file also contains some additional data (the email of every developer, the dates of both commits, the id's of the both commits and the name of the Java file which is modified in both commits) which is not directly used in further analysis of this empirical study but might be useful for other type of research or a visualization of the data.

E. The structure of the Java program.

The structure of the Java program used in this empirical study consists of 2 groups of packages: *issueTracker* and *versioningSystem*.

Versioning system group consists of 3 packages:

1. *Git* package. The package consists of 4 classes:
    - The *GitUtilities* class executes all the necessary git commands for this project. Their output (if it exists) is then represented in a text file. A *cloneGitRepositoryUnix* method clones Git repository to the specified directory on a local machine. A *getLogFromGitRepository* method extracts the information about all the commits done within the repository. Lastly, a *getRevList* method extracts the list of commits in a timeline between 2 specified commits.
    - The *GitSZZ* class name refers to the Git SZZ algorithm, so all the Git commands needed for this algorithm are placed there.
      A *getBlameHistory* method extracts the history of the blamed file specified by its name.
      A *getDiff* method is used to get and output to the text file the changes of a specified file between two specified commits.
      Method *getDiffBuggy* outputs the changes between a bug-fixing commit and a commit before it in a timeline of a project. Method *checkoutCommit* is used to check out the version of a repository to the needed version before executing a *'git blame'* command.
      A *restore* method restores the version of a repository to the most recent one.
    - The *GitRead* class implements all the data reading from files which were generated using Git commands. E.g. the *readBlame* method parses the generated *blame.txt* file and extracts the necessary data of it.
    - The *Csv* class contains a single *generateCsvFile* method and it puts the data needed for further statistical analysis to the CSV file.
2. *Bean* package contains all the vector classes into which the data extracted from a project Git repository is put. E.g. all the necessary data extracted from the *'git blame'* command output is put into *BlameBean* vector class.
3. *General* package contains a single *General* class. The class contains the *main* method which controls the execution of all other methods.

Issue tracker group also consists of 3 packages:

1. *Bean* package. It contains all the vector classes into which the data are extracted from a Jira bug report repository.
2. *Miner* package contains the implementation class of the Jira REST API needed to extract all the issues of the project. It formats the Jira REST API input query and sends it to the Jira server. Also, it extracts the received data and puts it to the *IssueBean* vector.
3. *IssueMining* package contains a single *JiraMIning* class. It is used to ensure that all the issues would be extracted from a software project issue tracking repository since the response from the Jira server retrieves not more than 100 bug reports at a time.

F. Replication package

Replication package for this empirical study is available at:
https://drive.google.com/folderview?id=0B_DUK6KMx-XPNTlQdEw1NU16c1E&usp=sharing

It contains the Java program used to extract the data from Git repositories in *analysisJava.zip* file. It also contains the extracted data used for this empirical study in the *replicationData.zip* file. The R Statistical Framework commands used to perform statistical tests are included to *RCommands.txt* file. All components together are grouped in a *fullAnalysis.zip* file.

III. Results

All retrieved results from every project are stored inside separate CSV files. They contain 4 factors which influence we want to measure and analyze for this empirical study. A number in days between commits can have value between 0 and the biggest time interval between 2 commits where the same file was modified by the same developer (referred as a column with a title "DAYS" in any of the results CSV file). A number of commits between commits can also have a value between 0 and the highest number of commits between 2 commits done by the same developer where the same file was modified (referred as a column with a title "COMMITS"). A number of LOC changed between 2 commits can have value from 0 to the total amount of LOC in that specific Java code file (referred as a column with a title "LOC"). The last factor, a number of developers who modified the same file between 2 commits done by the same developer where the same file was modified can have a value between 0 (if file was not modified by any other developer during the given time) and a total number of developers taking a part in a project (referred as a column with a title ("DEVELOPERS").

IV. Analysis of the results

The results of this empirical study were analyzed using the R statistical framework. A full list of R commands used to analyze the results can be found in the replication package of this study.

The factors were divided to the bug-inducing (referred as BUGGY) and clean (referred as CLEAN) data sets to preform further analysis.

Firstly, descriptive statistics were computed by creating box plots separately for every factor analyzed in every

project. They display the full range of variation, the likely range of variation and a typical value (the median) of the data set [6].

Secondly, statistical tests were performed for every factor separately in every project. To compute if the data in selected data set is normally distributed, the Anderson – Darling normality test was run. This test is used to accept or reject the following null hypothesis **H_0**: The data are normally distributed. The **H_0** is accepted if the resulting p-value is >=0.05. If the p-value is <0.05, then the **H_0** is rejected and it means that the data in the selected data set is not normally distributed.

The results of all the analysis performed for every set of data show that the data are not normally distributed in all the analyzed projects and all the data sets. This caused the need to perform non-parametric tests. The Mann-Whitney test was performed to test every null hypotheses formulated for this empirical study in the (**$H1_0$**, **$H2_0$**, **$H3_0$**, **$H4_0$**). The test is used to analyze if there is any significant difference between clean and bug-inducing commits. The p-value as a result of the test has the same meaning: if the p-value is >=0.05, then the null hypothesis is accepted. If the p-value is <0.05, then the null hypothesis is rejected.

Last part of statistical tests performed for every project separately was to compute the effect size of measured differences. The Cliff's Delta is used as a measure of the effect size on non-parametric data [7]. The value of the effect size is interpret as:

- Negligible: d < 0.148 (the same for negative values).
- Small: 0.148 <= d < 0.33.
- Medium: 0.33 <= d < 0.474.
- Large: d >= 0.474.

Apache ActiveMQ Apollo project

Figure 1 represents a box plot of values of DAYS factor between bug-inducing and clean commits. Even though clean commits have more high values, but the mean of both buggy and clean commits is almost the same, so the majority of data is distributed in the same range of values. So, we can assume that probably the DAYS factor is not influencing or has a very little influence to the introduction of bugs on this project.

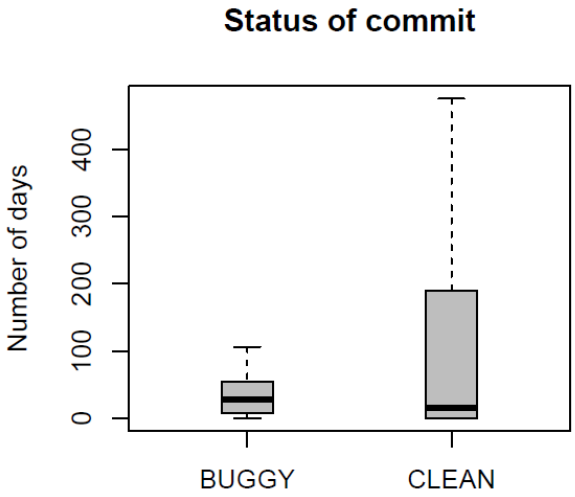

Figure 1. Apache ActiveMQ Apollo project difference in DAYS between commits.

A boxplot in a Figure 2 represents how the data of the COMMITS variable are distributed in clean and buggy commits. It shows that the number of commits within buggy data set is lower than in clean data set. Therefore, we can estimate that the lower value of COMMITS variable may cause a higher possibility of a bug introduction.

The data of the LOC factor results a box plot in a Figure 3, showing that the values of the number of LOC changed in a buggy data set are higher than in clean data set. So, we can assume that there is an influence of this factor for the induction of the bug.

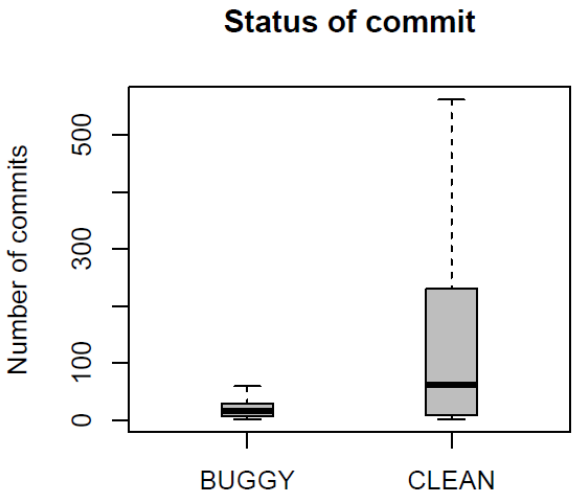

Figure 2. ApacheMQ Apollo project difference in COMMITS between commits.

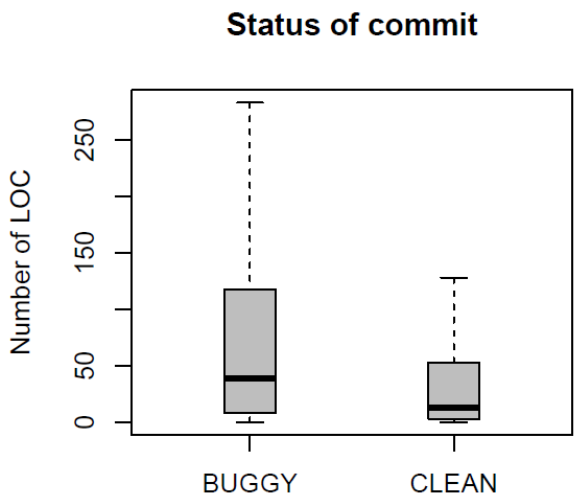

Figure 3. ApacheMQ Apollo project difference in LOC changed between commits.

Statistical tests were performed on these data and their results are put together in a Table 1.

Table 1. Results of statistical tests of the Apache ActiveMQ Apollo project.

Variable name	p-value	Effect size
DAYS	0.8007	0.00430 (negligible)
COMMITS	< 2.2e-16	0.37418 (medium)
LOC	< 2.2e-16	-0.27470 (small)
DEVELOPERS	< 2.2e-16	0.56287 (large)

Statistical tests were performed on these data. Using the data from the Figure 4 we can claim that:

1. As the p-value for the variable DAYS is >=0.05, the null hypothesis ***H1₀*** is accepted. The possibility of introducing a bug inside a software system is not related with the duration (in days) between a bug-introducing commit and a previous commit where the bug-introducing file was modified by the same developer. Also, the effect size of this factor is negligible.
2. The p-value of the COMMITS factor is less than 0.05, so we can reject the **H2₀.** Therefore we can claim that the possibility of introducing a bug inside a software system is related with the number of commits between the bug-introducing commit and the previous commit where the bug-introducing file was modified by the same developer. Buggy commits tend to have a lower number of commits between 2 specified commits than the clean commits and the effect size of this factor is medium.
3. The p-value of the LOC factor is less than 0.05, so the **H3₀** can be rejected. We can state that the possibility of introducing a bug inside a software system is related with a number of LOC modified in a file in a bug-introducing commit and a previous commit done by the same developer. The effect size of it for the induction of a bug in this project is small.
4. The p-value of the DEVELOPERS factor is less than 0.05, so we can reject the **H4₀**. The possibility of introducing a bug inside a software system is related with a number of developers who have modified a file between a bug-introducing commit and a previous commit done by the same developer. Also, this factor has a large effect size to influence a bug induction in this project. It is more likely that a file not modified by any other developer between a commit and a commit before it will induce a bug.

Apache Ant Ivy Project

Figure 4. The Apache Ant Ivy project difference of a buggy and clean commits in a number DEVELOPERS.

The Figure 4 represents a box plot of the DEVELOPERS factor. As the values in the buggy data set are higher, we can predict that there probably is some influence of this factor to the introduction of a bug.

The following data in a Table 2 contains the results of statistical tests performed for Apache Ant Ivy project.

Table 2. Results of statistical tests of the Apache Ant Ivy project.

Variable name	p-value	Effect size
DAYS	0.2103	-0.00926 (negligible)
COMMITS	< 2.2e-16	-0.07331 (negligible)
LOC	0.00099	0.02436 (negligible)
DEVELOPERS	< 2.2e-16	-0.21674 (small)

Regarding to the results of the statistical tests we can state that:

1. The p-value for the variable DAYS is more than 0.05, so the **H1₀** is accepted.
2. The p-value of the variable COMMITS is less than 0.05, so the **H2₀** is rejected. However, the effect size of this factor is negligible.

3. The p-value of the LOC factor is less than 0.05, so the **$H3_0$** is rejected. The lower number of LOC changed can increase a possibility of a bug induction in a commit. However, the effect size of this factor is negligible.
4. The p-value of the DEVELOPERS factor is less than 0.05, therefore the **$H4_0$** is rejected. Also, it has a small effect size. It is more likely that a file modified by larger number of other developer between a commit and a commit before it will induce a bug.

Apache Commons IO project

From the box plot of a DAYS factor (Figure 5) we can assume that there probably is an influence of this factor to introduce a bug.

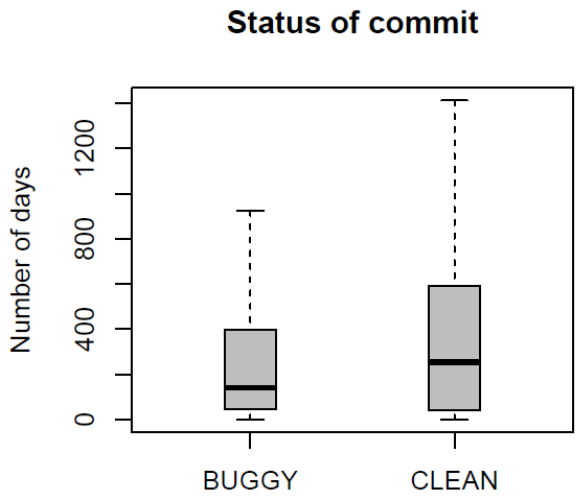

Figure 5. The Apache Commons IO project difference of a buggy and clean commits in a number of DAYS.

Figure 6. The Apache Commons IO project difference of a buggy and clean commits in a number of DEVELOPERS.

A box plot in a Figure 6 represents the distribution of a DEVELOPERS factor. We can assume that this factor probably makes some influence on a possibility of a bug introduction in a commit.

Table 3. Results of statistical tests of the Apache Commons IO project.

Variable name	p-value	Effect size
DAYS	6.239e-05	0.13716 (negligible)
COMMITS	0.1013	-0.05614 (negligible)
LOC	7.902e-06	-0.15303 (small)
DEVELOPERS	2.813e-10	0.21377 (small)

The statistical tests performed on this project result those conclusions in a Table 3:

1. The p-value for the variable DAYS is less than 0.05, so the **$H1_0$** is rejected. Also, the effect size of this factor is negligible.
2. The p-value of the variable COMMITS is more than 0.05, so the **$H2_0$** is accepted.
3. The p-value of the LOC factor is less than 0.05, so the **$H3_0$** is rejected. The effect size for this factor is small.
4. The p-value of the DEVELOPERS factor is less than 0.05, so the **$H4_0$** is rejected. So, the lower amount of the LOC changed may make a higher possibility for a commit to induce a bug in a file. The effect size of this factor is small.

Apache DB DdlUtils project

The Figure 7 represents a box plot of a difference in DAYS variable between clean and buggy data sets of this project. We can observe that this factor might influence the possibility of introduction of a bug in a software system.

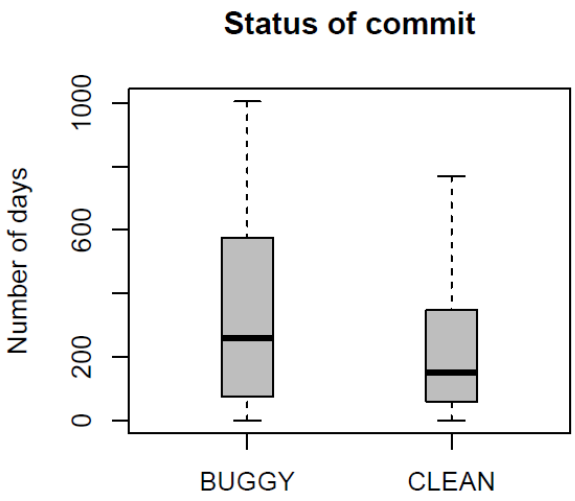

Figure 7. The Apache DB DdlUtils project difference in DAYS between clean and buggy data sets.

The results of statistical tests performed on the data of this project are represented on a Table 4.

Table 4. Results of statistical tests of the Apache DB DdlUtils project.

Variable name	p-value	Effect size
DAYS	< 2.2e-16	-0.20903 (small)
COMMITS	0.00049	-0.03776 (negligible)
LOC	1.436e-11	-0.07317 (negligible)
DEVELOPERS	4.529e-05	0.03269 (negligible)

From the data of the table we can state that:

1. The p-value of the DAYS factor is less than 0.05, so the **$H1_0$** is rejected. In this project, the bigger number of days between commits makes a higher possibility for a commit to induce a bug in the project and the effect size of this factor is small.
2. As the p-value of the COMMITS factor is less than 0.05, the **$H2_0$** is rejected. We state that this factor influences the possibility for a commit to induce a bug in this project. However, the effect size of this factor is negligible.
3. The p-value of the LOC factor is less than 0.05, so the **$H3_0$** is rejected. The possibility of introducing a bug in a software system is related with this factor. However, the effect size of this factor is negligible.
4. The p-value of the DEVELOPERS factors is less than 0.05, so the **$H4_0$** is rejected. However, the effect size is negligible for this factor.

Apache Drill project

The results of the statistical tests performed on the data extracted from this project are shown in the Table 5.

Table 5. Results of statistical tests of the Apache Drill project.

Variable name	p-value	Effect size
DAYS	2.293e-09	-0.06322 (negligible)
COMMITS	0.00385	-0.03057 (negligible)
LOC	0.00011	-0.04075 (negligible)
DEVELOPERS	0.5701	0.00600 (negligible)

From the data of the table we can state that:

1. The p-value of the DAYS variable is less than 0.05, so the **$H1_0$** is rejected. However, the effect size of this factor is negligible in this project.
2. The p-value of COMMITS variable is less than 0.05, so the **$H2_0$** is rejected. However, the effect size of this factor is negligible in this project.
3. The p-value of the LOC factor is less than 0.05, so the **$H3_0$** is rejected. However, the effect size of this factor is negligible on this project.
4. The p-value of the DEVELOPERS factor is more than 0.05, so the **$H4_0$** is accepted.

Apache Karaf project

Table 6. Results of statistical tests of the Apache Karaf project.

Variable name	p-value	Effect size
DAYS	4.065e-10	-0.12342 (negligible)
COMMITS	3.034e-10	-0.12436 (negligible)
LOC	0.04483	-0.03961 (negligible)
DEVELOPERS	9.541e-10	-0.12042 (negligible)

The results of performed statistical tests on this project are represented in a Table 6. From the data of the table we can state:

1. The p-value of the DAYS variable is less than 0.05, so the **$H1_0$** is rejected. However, the effect size of this factor is negligible in this project.
2. The p-value of COMMITS variable is less than 0.05, so the **$H2_0$** is rejected. However, the effect size of this factor is negligible in this project.
3. The p-value of the LOC factor is less than 0.05, so the **$H3_0$** is rejected. However, the effect size of this factor is negligible on this project.
4. The p-value of the DEVELOPERS factor is less than 0.05, so the **$H4_0$** is rejected. However, the effect size of this factor is negligible in this project.

Apache Pivot project

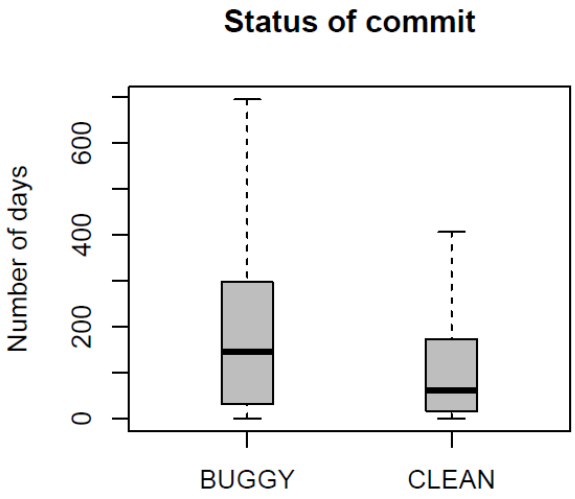

Figure 8. The Apache Pivot project difference in DAYS between clean and buggy data sets.

The box plot in a Figure 8 represents the values of the DAYS factor in this project. From these data we can assume that this factor is probably making an influence on the induction of a bug in this project.

The Figure 9 represents a box plot of the DEVELOPERS variable of this project. We can state that there probably is an influence of this factor on a likelihood to introduce a bug.

Figure 9. The Apache Pivot project difference in DEVELOPERS between clean and buggy data sets.

Table 7. Results of statistical tests of the Apache Pivot project.

Variable name	p-value	Effect size
DAYS	< 2.2e-16	-0.23623 (small)
COMMITS	< 2.2e-16	-0.13905 (negligible)
LOC	< 2.2e-16	0.04515 (negligible)
DEVELOPERS	< 2.2e-16	-0.36362 (medium)

The results of statistical tests performed on these data are in the Table 7. We can state that:

1. The p-value of the DAYS variable is less than 0.05, so the **$H1_0$** is rejected. Also, the effect size of this factor is small in this project.
2. The p-value of COMMITS variable is less than 0.05, so the **$H2_0$** is rejected. However, the effect size of this factor is negligible in this project.
3. The p-value of the LOC factor is less than 0.05, so the **$H3_0$** is rejected. However, the effect size of this factor is negligible on this project.
4. The p-value of the DEVELOPERS factor is more than 0.05, so the **$H4_0$** is accepted. Also, the effect size of this factor is medium in this project.

Overall results

After the analysis was done separately for every project, all the results of statistical tests performed were put into 4 tables, for each of the analyzed factor separately.

A Table 8 represents results of the DAYS variable from every analyzed project.

Table 8. The results of DAYS variable from every project analyzed.

Project name	p-value	Effect size
ActiveMQ Apollo	0.8007	0.00430 (negligible)
Ant Ivy	0.2103	-0.00926 (negligible)
Commons IO	6.239e-05	0.13716 (negligible)
DdlUtils	< 2.2e-16	-0.20903 (small)
Drill	2.293e-09	-0.06322 (negligible)
Karaf	4.065e-10	-0.12342 (negligible)
Pivot	< 2.2e-16	-0.23623 (small)

Regarding on these results we can state that in 5 of 7 projects the possibility of introducing a bug inside a software system is related with the duration (in days) between the bug-introducing commit and the previous commit where the bug-introducing file was modified by the same developer. However, 3 of them have a negligible effect size and only 2 of them have a small effect size. We can state that only in some projects the possibility of introducing a bug inside a software system is related with the duration (in days) between the bug-introducing commit and the previous commit where the bug-introducing file was modified by the same developer.

Table 9. The results of COMMITS variable from every project analyzed.

Project name	p-value	Effect size
ActiveMQ Apollo	< 2.2e-16	0.37418 (medium)
Ant Ivy	< 2.2e-16	-0.07331 (negligible)
Commons IO	0.1013	-0.05614 (negligible)
DdlUtils	0.00049	-0.03776 (negligible)
Drill	0.00385	-0.03057 (negligible)
Karaf	3.034e-10	-0.12436 (negligible)
Pivot	< 2.2e-16	-0.13905 (negligible)

Results of the analysis show that the possibility of introducing a bug inside 6 of 7 analyzed software systems is related with the number of commits between a bug-introducing commit and a previous commit where a bug-introducing file was modified by the same developer. However, the effect size of this variable is negligible. We can state that the COMMITS variable impacts the possibility of the bug induction in a software system, but the effect size of it is negligible in most of the projects and only in very few of projects it is larger than negligible.

Table 10. The results of LOC variable from every project analyzed.

Project name	p-value	Effect size
ActiveMQ Apollo	< 2.2e-16	-0.27470 (small)
Ant Ivy	0.00099	0.02436 (negligible)
Commons IO	7.902e-06	-0.15303 (small)
DdlUtils	1.436e-11	-0.07317 (negligible)
Drill	0.0001169	-0.04075 (negligible)
Karaf	0.04483	-0.03961 (negligible)
Pivot	< 2.2e-16	0.04515 (negligible)

Results show that in every of 7 analyzed projects the possibility of introducing a bug inside a software system is

related with a number of LOC modified in a file in a bug-introducing commit and a previous commit by the same developer. However, only 2 of them have effect size larger than negligible, so we can state that variable LOC influences an induction of a bug only in some software projects.

Table 11. The results of DEVELOPERS variable from every project analyzed.

Project name	p-value	Effect size
ActiveMQ Apollo	< 2.2e-16	0.56287 (large)
Ant Ivy	< 2.2e-16	-0.21674 (small)
Commons IO	2.813e-10	0.21377 (small)
DdlUtils	4.529e-05	0.03269 (negligible)
Drill	0.5701	0.0060 (negligible)
Karaf	9.541e-10	-0.12042 (negligible)
Pivot	< 2.2e-16	-0.36362 (medium)

Results of the analysis show that in 6 of 7 projects the possibility of introducing a bug inside a software system is related with the number of developers who have modified the file between a bug-introducing commit and a previous commit done by the same developer.

To sum up all the results of the analysis, we can conclude that the DAYS variable makes influence for a possibility to induce a bug only in some projects, but it also should be considered as a possible factor for higher or lower possibility of the induction of a bug.

The COMMITS variable has meaningful (medium) effect size only in 1 of 7 analyzed projects, but it should be considered as a possibly important factor to influence a possibility of an introduction of a bug.

The LOC variable influences a bug induction only in some projects, but it should be considered as a factor which influences the possibility of an introduction of a bug.

However, the DEVELOPERS factor had the most recognizable influence for a possibility of an induction of a bug in analyzed projects. Therefore it should be considered as the most important factor of all 4 factors which were analyzed on this empirical study.

V. Related work.

Śliwerski et al. in 2005 ("When do changes induce fixes?") performed analysis on the day of the week to influence the introduction of a bug in a project and identified the Friday as the day when the developer is the most likely to introduce a bug [8].

Thomas Zimmermann et al. in 2006 ("Automatic Identification of Bug-Introducing Changes") described a more accurate implementation of SZZ algorithm by removing possible false positive and negative changes on bug-fixing commits [9].

D. Izquierdo-Cortazar et al. in 2012 ("Are Developers Fixing Their Own Bugs? Tracing Bug-fixing and Bug-seeding Committers") investigated that a fixing of a bug usually is done on the code modified not more than by two developers. It was done by tracking all the lifecycle of the line of code by its authorship [10].

Kim et al. [11] used a bug prediction technique based on machine learning to classify commits into clean and buggy sets. The authors used different factors extracted from the commit information than the ones used in this empirical study.

VI. Threats to validity.

There are several threats which could affect the validity of this empirical study. They will be reviewed in this chapter. First threat that could affect the validity is an assumption that the bug-inducing commit is the most recent commit where the file was modified before the bug-fixing commit. It is impossible to ensure that the bug was created on that exact commit and not on any other commit in the past.

The results performed on selected projects cannot be generalized because for any group of projects and in different context the results may be different.

This empirical study can only be applied to Java Apache codebase projects, therefore it is also impossible to generalize the results of analysis to any other software projects developed using other programming languages.

The analyzed set of projects may be too small to reach more generalized conclusion.

The experience of the developer is also not taken into account on this empirical study, though it may highly influence an induction of a bug in a project.

VII. Conclusion and future work.

The relationship between the 4 investigated factors and the possibility of a bug induction in a software system should be taken into consideration when developing software projects as a way to increase the quality of a software and to decrease a number of bugs introduced. This empirical study analyzed the 4 factors: the number in days, the number in commits, the number of LOC changed and the number of developers who modified the same file during the given period of time between 2 commits done by the same developer where the same file was modified. The analysis of these factors exposed that none of them were influencing the possibility of an induction of a bug in all selected software projects. However, the analysis of the results shows that a number of commits factor should hardly be taken into consideration as it has a meaningful influence to induce a bug only in one project. Two other factors – a number of days and a number of LOC changed influence a possibility of the introduction of a bug in some projects. A factor of number of developers has meaningful influence to a possibility of a bug induction in more than half (4 of 7) projects analyzed.

To conclude the results, all 4 investigated factors should be considered as possible to influence an introduction of a

bug in a software system. However, the influence of every factor may not be present in every software project analyzed.

The future work on this study will include the analysis of more software projects to get more generalized results and conclusions. Also, the modifications of Git SZZ algorithm may be applied to find buggy commits more accurately.

REFERENCES

[1] H. Kagdi, M. L. Collard and J. I. Maletic, "A survey and taxonomy of approaches for mining software repositories in the context of software evolution", Journal of Software Maintenance and Evolution: Research and Practice, vol. 19, pp. 77-131, 2007.

[2] J. Loeliger, M. McCullough, "Version Control with Git: Powerful tools and techniques for collaborative software development", 2nd edition, O'Reilly Media, pp. 1-2, 2012.

[3] Giovanni Marco Dall'Olio, Jaume Bertranpetit, Hafid Laayouni, "The annotation and the usage of scientific databases could be improved with public issue tracker software", Database 2010: baq035 doi:10.1093/database/baq035, pp. 1-2, published online December 23, 2010.

[4] Hideaki Hata, Osamu Mizuno, Tohru Kikuno, "Bug Prediction Based on Fine-Grained Module Histories", ICSE International Conference on Software Engineering, sec. E, pp. 204-206, 2012-06-02

[5] https://developer.atlassian.com/jiradev/jira-apis/jira-rest-apis.

[6] http://www.physics.csbsju.edu/stats/box2.html

[7] Melinda R. Hess, Jeffrey D. Kromrey, "Robust Confidence Intervals for Effect Sizes: A Comparative Study of Cohen's d and Cliff's Delta Under Non-normality and Heterogeneous Variances", The annual meeting of the American Educational Research Association, pp. 2-3, April 12, 2004.

[8] J. Śliwerski, T. Zimmermann, and A. Zeller, "When Do Changes Induce Fixes?" in Proceedings of the 2005 International Workshop on Mining Software Repositories, New York, NY, USA, 2005, pp. 1–5.

[9] Sunghun Kim, Thomas Zimmermann, Kai Pan and E. James Whitehead Jr., "Automatic Identification of Bug-Introducing Changes", 21st IEEE International Conference on Automated Software Engineering (ASE'06), 2006, pp. 2-8.

[10] Daniel Izquierdo-Cortazar, Andrea Capiluppi, Jesus M. Gonzalez-Barahona, "Are Developers Fixing Their Own Bugs? Tracing Bug-fixing and Bug-seeding Committers", International Journal of Open Source Software and Processes (IJOSSP), 3(2): 23 - 42, 2011.

[11] Sunghun Kim, E. James Whitehead Jr., Member and Yi Zhang, "Classifying Software Changes: Clean or Buggy?", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 34, NO. 2, pp. MARCH/APRIL 2008.

Appendix 1.

Figure 1. The ApacheMQ Apollo project difference in a number of developers between the buggy and clean data sets.

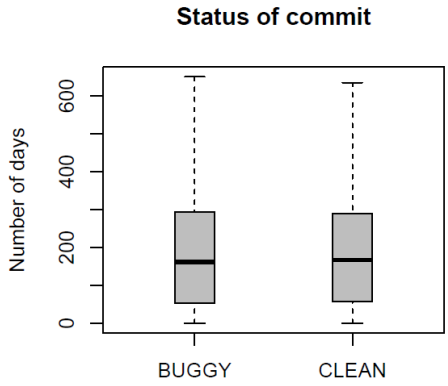

Figure 2. The Apache Ant Ivy project difference of a buggy and clean commits in a number of days.

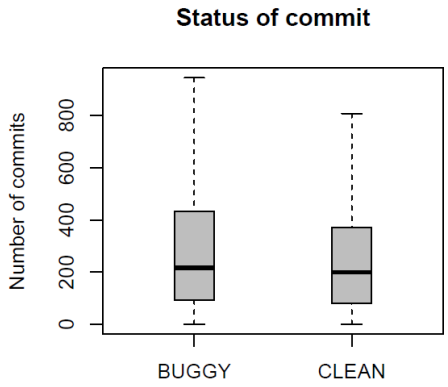

Figure 3. The Apache Ant Ivy project difference of a buggy and clean commits in a number of COMMITS.

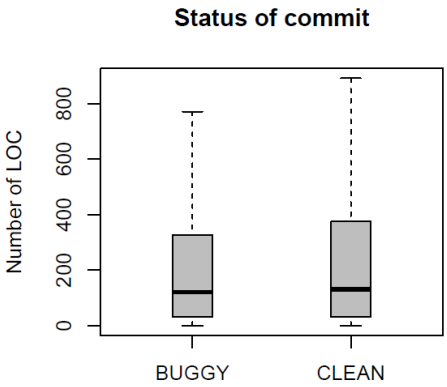

Figure 4. The Apache Ant Ivy project difference of a buggy and clean commits in a number of LOC changed.

Figure 5. The Apache Commons IO project difference of a buggy and clean commits in a number of COMMITS.

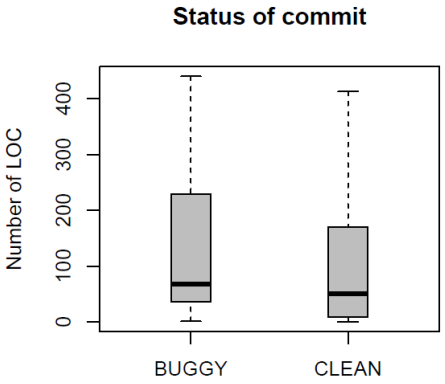

Figure 6. The Apache Commons IO project difference of a buggy and clean commits in a number of LOC changed.

Figure 7. The Apache DB DdlUtils project difference of buggy and clean commits in a number of COMMITS.

Figure 8. The Apache DB DdlUtils project difference of buggy and clean commits in a number of LOC changed.

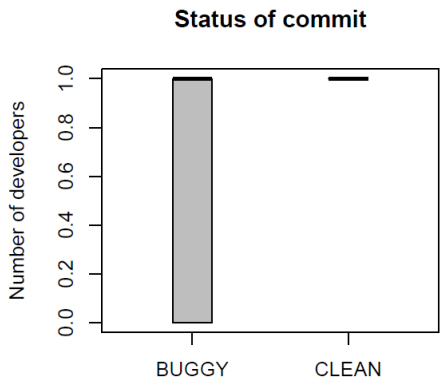

Figure 9. The Apache DB DdlUtils project difference of buggy and clean commits in a number of DEVELOPERS.

Figure 10. The Apache Drill project difference in DAYS between clean and buggy data sets.

Figure 11. The Apache Drill project difference between buggy and clean commits in a number of COMMITS.

Figure 12. The Apache Drill project difference between buggy and clean commits in a number of LOC changed.

Figure 13. The Apache Drill project difference between buggy and clean commits in a number of DEVELOPERS.

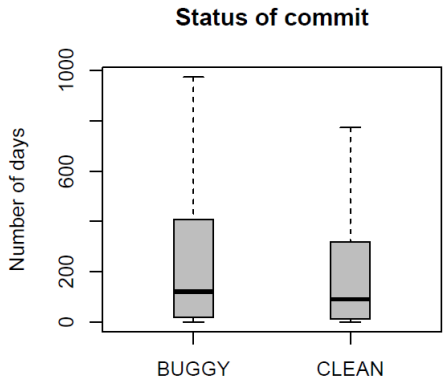

Figure 14. The Apache Karaf project difference between buggy and clean commits in a number of DAYS.

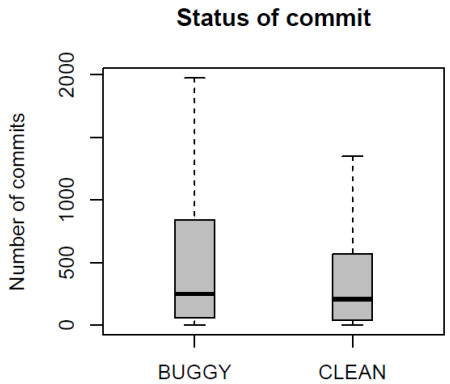

Figure 15. The Apache Karaf project difference between buggy and clean commits in a number of COMMITS.

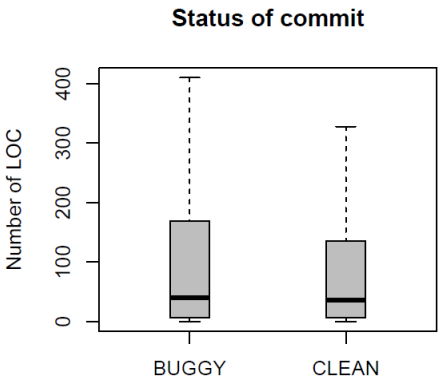

Figure 16. The Apache Karaf project difference between buggy and clean commits in a number of LOC changed.

Figure 17. The Apache Karaf project difference between buggy and clean commits in a number of DVELOPERS.

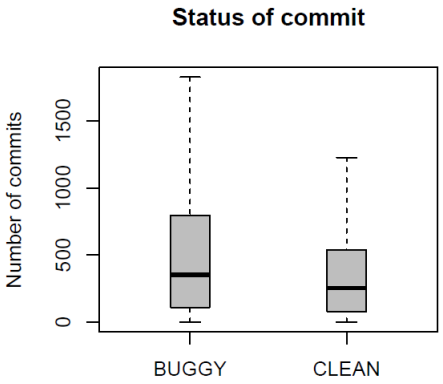

Figure 18. The Apache Pivot project difference between buggy and clean commits in a number of COMMITS.

Figure 19. The Apache Pivot project difference between buggy and clean commits in a number of LOC changed.