A New Taxonomy of Bugs Based on the Location of the Correction: An Empirical Study

Mathieu Nayrolles*, Abdelwahab Hamou-Lhadj*, Abdou Maiga*, Alf Larsson†, Sigrid Eldh†

*Software Behaviour Analysis (SBA) Research Lab
ECE, Concordia University, Montréal, Canada
{m_nayrol, a_maiga, abdelw}@ece.concordia.ca

†Ericsson
Stockholm, Sweden
{alf.larsson, sigrid.eldh}@ericsson.com

***Abstract*—The abstract goes here.**

***Index Terms*—Bug taxonomy, Bug tracking systems, Empirical studies Software maintenance**

I. Introduction

In order to classify the research on the different fields related to software maintenance, we can reason about types of bugs at different levels. For example, we can group bugs based on the developers that fix them or using information about the bugs such as crash traces.

There have been several studies (e.g., [1], [2]) that study of the factors that influence the bug fixing time. These studies empirically investigate the relationship between bug report attributes (description, severity, etc.) and the fixing time. Other studies take bug analysis to another level by investigating techniques and tools for bug prediction and reproduction (e.g., [3], [4], [5]). These studies, however, treat all bugs as the same. For example, a bug that requires only one fix is analyzed the same way as a bug that necessitates multiple fixes. Similarly, if multiple bugs are fixed by modifying the exact same locations in the code, then we should investigate how these bugs are related in order to predict them in the future. Note here that we do not refer to duplicate bugs. Duplicate bugs are marked as duplicate (and not fixed) and only the master bug is fixed. As a motivating example, consider Bugs #AMQ-5066 and #AMQ-5092 from the Apache Software Foundation bug report management system (used to build one of the datasets in this paper). Bug #AMQ-5066 was reported on February 19, 2014 and solved with a patch provided by the reporter. The solution involves a relatively complex patch that modifies `MQTTProtocolConverter.java`, `MQTTSubscription.java` and `MQTTTest.java` files. The description of the bug is as follows:

When a client sends a SUBSCRIBE message with the same Topic/Filter as a previous SUBSCRIBE message but a different QoS, the Server MUST discard the older subscription, and resend all retained messages limited to the new Subscription QoS.

A few months later, another bug, Bug #AMQ-5092 was reported:

MQTT protocol converters does not correctly generate unique packet ids for retained and non-retained publish messages sent to clients. [...] Although retained messages published on creation of client subscriptions are copies of retained messages, they must carry a unique packet id when dispatched to clients. ActiveMQ re-uses the retained message's packet id, which makes it difficult to acknowledge these messages when wildcard topics are used. ActiveMQ also sends the same non-retained message multiple times for every matching subscription for overlapping subscriptions. These messages also re-use the publisher's message id as the packet id, which breaks client acknowledgment.

This bug was assigned and fixed by a different person than the one who fixed bug #AMQ-5066. The fix consists of modifying slightly the same lines of the code in the exact files used to fix Bug #AMQ-5066. In fact, Bug #5092 could have been avoided altogether if the first developer provided a more comprehensive fix to #AMQ-5066 (a task that is easier said than done). These two bugs are not duplicates since, according to their description, they deal with different types of problems and yet they can be fixed by providing a similar patch. In other words, the failures are different while the root causes (faults in the code) are more or less the same. From the bug handling perspective, if we can develop

a way to detect such related bug reports during triaging then we can achieve considerable time saving in the way bug reports are processed, for example, by assigning them to the same developers. We also conjecture that detecting such related bugs can help with other tasks such as bug reproduction. We can reuse the reproduction of an already fixed bug to reproduce an incoming and related bug.

Our aim is not to improve testing as it is the case in the work of Eldh [6] and Hamill et al.[7]. Our objective is to propose a classification that can allow researchers in the filed of mining bug 9 repositiories to use the taxonomy as a new criterion in triaging, prediction, and reproduction of bugs. By analogy, we can look at the proposed bug taxonomy in a similar way as the clone taxonomy presented by Kapser and Godfrey [8]. The authors proposed seven types of source code clones and then conducted a case study, using their classification, on the file system module of the Linux operating system. This clone taxonomy continues to be used by researchers to build better approaches for detecting a given clone type and being able to effectively compare approaches with each other.

We are interested in bugs that share similar fixes. By a fix, we mean a modification (adding or deleting lines of code) to an exiting file that is used to solve the bug. With this in mind, the relationship between bugs and fixes can be modeled using the UML diagram in Figure 1. The diagram only includes bugs that are fixed. From this figure, we can think of four instances of this diagram, which we refer to as bug taxonomy or simply bug types (see Figure 2).

Figure 1. Class diagram showing the relationship between bugs and fixed

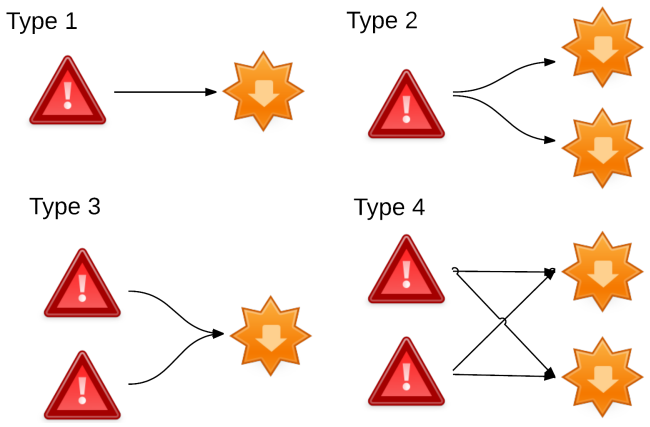

Figure 2. Proposed Taxonomy of Bugs

The first and second types are the ones we intuitively know about. Type 1 refers to a bug being fixed in one single location (i.e., one file), while Type 2 refers to bugs being fixed in more than one location. In Figure 2, only two locations are shown for the sake of clarity, but many more locations could be involved in the fix of a bug. Type 3 refers to multiple bugs that are fixed in the exact same location. Type 4 is an extension of Type 3, where multiple bugs are resolved by modifying the same set of locations. Note that Type 3 and Type 4 bugs are not duplicates, they may occur when different features of the system fail due to the same root causes (faults). We conjecture that knowing the proportions of each type of bugs in a system may provide insight into the quality of the system. Knowing, for example, that in a given system the proportion of Type 2 and 4 bugs is high may be an indication of poor system quality since many fixes are needed to address these bugs. In addition, the existence of a high number of Types 3 and 4 bugs calls for techniques that can effectively find bug reports related to an incoming bug during triaging. This is similar to the many studies that exist on detection of duplicates (e.g., [9], [10], [11]), except that we are not looking for duplicates but for related bugs (bugs that are due to failures of different features of the system, caused by the same faults). To our knowledge, there is no study that exmpirically examines bug data with these types in mind, which is the main objective of this section. More particularly, we are interested in the following research questions:

- RQ1: What are the proportions of different types of bugs?
- RQ2: How complex is each type of bugs?
- RQ3: How fast are these types of bugs fixed?

II. Preliminaries

Software maintenance, comprehension, evolution, specifications and testing are research areas overlapping each other in terms of terminology.

In this paper, we will use a precise set of definitions. We do not claim ownership of these definitions, they have been established using various resources [12], [13], [14], [15], [16].

We limit software maintenance to the following three artifacts:

- Bug report: A bug report describes a behavior observed in the field and considered abnormal by the reporter. Bug reports are submitted manually to bug report systems (bugzilla/jira). There is no mandatory format to report a bug, nevertheless, it should have: Version of the software / OS / Platform used, steps to reproduce, screen shots, stack trace and anything that could help a developer to assess the internal state of the software system.

- Crash report: A crash report is the last action a software system does before crashing. Crash reports are automatic (they have to be implemented into the software system by developer) and contain data (that can be proprietary) to help developers understand the crash (e.g. memory dump,...).
- Tasks: A task is a new feature, or the improvement of an existing feature, to be implemented in a future release of the software.

These artifacts are produced in response to the following phenomena:

- Software Bug: A software bug is an error, flaw, failure, defect or fault in a computer program or system that causes it to violate at least one of its functional or nonfunctional requirements.
- Error: An error is a mistake, misconception, or misunderstanding on the part of a software developer.
- Fault/defect: A fault (defect) is defined as an abnormal condition or defect at the component, equipment, or subsystem level which may lead to a failure. A fault (defect) is not final (the system still works) and does not prevent a given feature to be accomplished. A fault (defect) is a deviation (anomaly) of the healthy system that can be caused by an error or external factors (hardware, third parties, ...).
- Failure: The inability of a software system or component to perform its required functions within specified requirements.
- Crash: The software system encountered a fault (defect) that triggered a fatal failure from which the system could not recover from/overcome. As a result, the system stopped.

In the remaining of this section, we introduce the two types of software repositories: version control and project tracking system.

A. Version control systems

Version control consists of maintaining the versions of files — such as source code and other software artifacts [17]. This activity is a complex task and cannot be performed manually on real world project. Consequently, numerous tools have been created to help practitioners manage the version of their software artifacts. Each evolution of a software is a version (or revision) and each version (revision) is linked to the one before through modifications of software artifacts. These modifications consist of updating, adding or deleting software artifacts. They can be referred as `diff`, `patch` or `commit`[1]. Each `diff`, `patch` or `commit` have the following characteristics:

[1]These names are not to be used interchangeably as difference exists.

- Number of Files: The number of software files that have been modified, added or deleted.
- Number of Hunks: The number of consecutive code blocks of modified, added or deleted lines in textual files. Hunks are used to determine, in each file, how many different places the developer has modified.
- Number of Churns: The number of lines modified. However, the churn value for a line change should be at least two as the line has to be deleted first and then added back with the modifications.

In modern versioning systems, when maintainers make modifications to the source code want to version it, they have to do commit. The commit operation will version the modifications applied to one or many files.

Figure 3 presents the data structure used to store a commit. Each commit is represented as a tree. The root leaf (green) contains the commit, tree and parent hashes as same as the author and the description associated with the commit. The second leaf (blue) contains the leaf hash and the hashes of the files of the project.

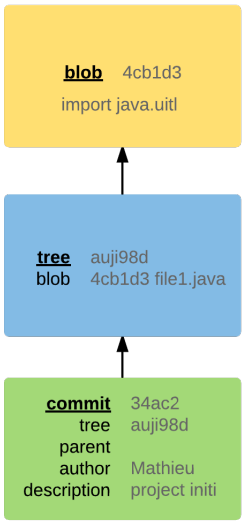

Figure 3. Data structure of a commit.

In this example, we can see that author “Mathieu” has created the file $file1.java$ with the message “project init”.

B. Project Tracking Systems

Project tracking systems allow end users to create bug reports (BRs) to report unexpected system behavior, manager can create tasks to drive the evolution forward and crash report (CRs) can be automatically created. These systems are also used by development teams to keep track of the modification induced by bug and to crash reports, and keep track of the fixes.

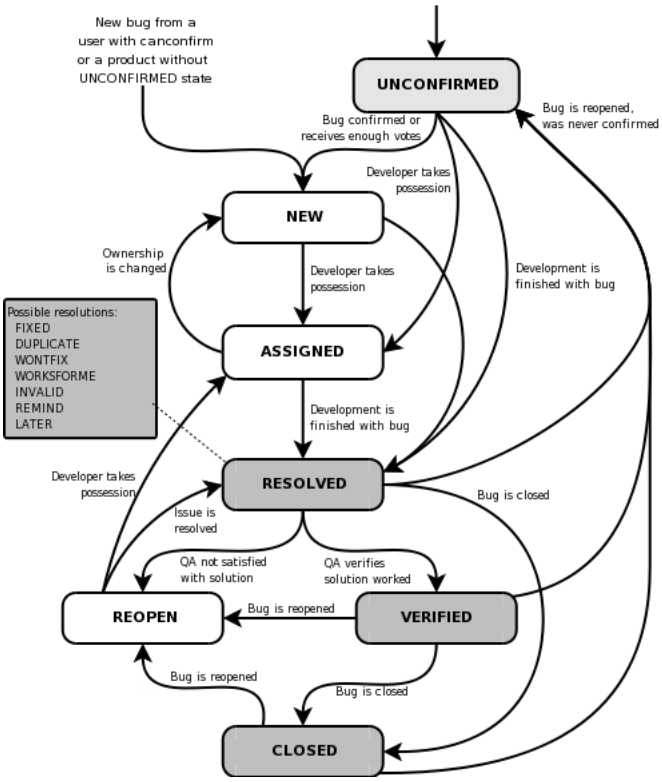

Figure 4. Lifecyle of a report [18]

Figure 4 presents the life cycle of a report. When a report is submitted by an end-user, it is set to the `UNCONFIRMED` state until it receives enough votes or that a user with the proper permissions modifies its status to `NEW`. The report is then assigned to a developer to be fixed. When the report is in the `ASSIGNED` state, the assigned developer(s) starts working on the report. A fixed report moves to the `RESOLVED` state. Developers have five different possibilities to resolve a report: `FIXED`, `DUPLICATE`, `WONTFIX`, `WORKSFORME` and `INVALID` [19].

- `RESOLVED/FIXED`: A modification to the source code has been pushed, i.e., a changeset (also called a patch) has been committed to the source code management system and fixes the root problem described in the report.
- `RESOLVED/DUPLICATE`: A previously submitted report is being processed. The report is marked as duplicate of the original report.
- `RESOLVED/WONTFIX`: This is applied in the case where developers decide that a given report will not be fixed.
- `RESOLVED/WORKSFORME`: If the root problem described in the report cannot be reproduced on the reported OS / hardware.
- `RESOLVED/INVALID`: If the report is not related to the software itself.

Finally, the report is `CLOSED` after it is resolved. A report can be reopened (sent to the `REOPENED` state) and then assigned again if the initial fix was not adequate (the fix did not resolve the problem). The elapsed time between the report marked as the new one and the resolved status are known as the *fixing time*, usually in days. In case of task branching, the branch associated with the report is marked as ready to be merged. Then, the person in charge (quality assurance team, manager, ect...) will be able to merge the branch with the mainline. If the report is reopened: the days between the time the report is reopened and the time it is marked again as `RESOLVED/FIXED` are cumulated. Reports can be reopened many times.

Tasks follow a similar life cycle with the exception of the `UNCONFIRMED` and `RESOLVED` states. Tasks are created by management and do not need to be confirmed in order to be `OPEN` and `ASSIGNED` to developers. When a task is complete, it will not go to the `RESOLVED` state, but to the `IMPLEMENTED` state. Bug and crash reports are considered as problems to eradicate in the program. Tasks are considered as new features or amelioration to include in the program.

Reports and tasks can have a severity[20]. The severity is a classification to indicate the degree of impact on the software. The possible severities are:

- blocker: blocks development and/or testing work.
- critical: crashes, loss of data, severe memory leak.
- major: major loss of function.
- normal: regular report, some loss of functionality under specific circumstances.
- minor: minor loss of function, or other problem where easy workaround is present.
- trivial: cosmetic problems like misspelled words or misaligned text.

The relationship between an report or a task and the actual modification can be hard to establish and it has been a subject of various research studies (e.g., [21], [22], [23]). This reason is that they are in two different systems: the version control system and the project tracking system. While it is considered a good practice to link each report with the versioning system by indicating the report $\#id$ on the modification message, more than half of the reports are not linked to a modification[23].

III. Study Design

The goal of this study is to analyze the location of bug fixes, with the purpose of classifying bug fixes into types. More specifically, this study aims to answer the folliwng two research questions:

- **RQ$_1$:** *What are the proportions of different types of bugs?* This research question aims to what extent bug can be classified according to their fix-locations and the proportion of each types. Specifically, we investigate if different types of bugs exists at all and if the proportion of different types in non-negligible. As discussed ealier, knowing, for example, that bugs of Type 2 and 4 are the most predominant ones suggests that we need to investigate techniques to help detect whether an incoming bug is of Types 2 and 4 by examining historical data. Similarly, if we can automatically identify a bug that is related to another one that has been fixed then we can reuse the results of reproducing the first bug in reproducing the second one.
- **RQ$_2$:** *How complex is each type of bugs?* This second research question aims to investigate the complexity of the different types of bug. More specifically, we analyze and discuss the complexity of different types of bugs using code and process metrics both. For the code aspect of the complexity, we compute the number of different files impacted by the fix and the number of hunks and churns. We do not compute any statical complexity metrics such as cyclomatic complexity [24]. For the process aspect of complexity, we analyze the severity of the bug, the amount of duplicate bug report submitted, the number of times a bug report gets re-opened, the number of comments and the time required to fix the bug.

A. Context Selection

The context of this study consists of the change history of 388 projects belonging to two software ecosystem, namely, Apache and Netbeans. Tbale I reports, for each of them, (i) the number of projects analyzed, (ii) size ranges in terms of the number of classes and KLOC, (iii) the overall number of commits and issues analyzed, and (iv) the average, minimum, and maximum length of the projects' story (in years).

Dataset	R/F BR	CS	Files	Projects
Netbeans	53,258	122,632	30,595	39
Apache	49,449	106,366	38,111	349
Total	102,707	229,153	68,809	388

Table I
DATASETS

All the analysed projects are hosted in *Git* or *Mercurial* repositories and have either a *Jira* or a *Bugzilla* issue tracker associated with it. The Apache ecosystem consists in 349 projects written in various programming languages (C, C++, Java, Python, ...) and uses *Git* and *Jira*. These projects represent the Apache ecosystem in its entirety; no system has been excluded from our study. The complete list can be found online[2]. The Netbeans ecosystem consists in 39 projects mostly written in Java. Similarly to the Apache ecosystem, we did not select some of the projects belonging to the Netbeans ecosystem but all of them. The Netbeans community uses *Bugzilla* and *Mercurial*.

The choice of the ecosystems to analyze is not random, but rather driven by the motivation to consider projects having (i) different sizes, (ii) different architectures, and (iii) different development bases and processes. Indeed, Apache project are extremely various in terms of size of the development team, purpose and technical choices [25]. On the other side, Netbeans have a relatively stable list of core developer and a common vision shared through the 39 related projects [26].

Cumulatively, these datasets span from 2001 to 2014. In summary, our consolidated dataset contains 102,707 bugs, 229,153 changesets, 68,809 files that have been modified to fix the bugs, 462,848 comments, and 388 distinct systems. We also collected 221 million lines of code modified to fix the bugs, identified 3,284 sub-projects, and 17,984 unique contributors to these bug report and source code version management systems. The cumulated opening time for all the bugs reaches 10,661 working years (3,891,618 working days) which translates into more than one billion dollars[27].

B. Data Extraction and Analysis

This subsection describes the data extraction and analysis process that we followed to answer our research questions.

1) What are the proportions of different types of bugs?: To answer **RQ$_1$**, we cloned the 349 *git* repositories belonging to the Apache ecosystem and the 39 *mercurial* repositories belonging to the Netbeans ecosystem. The raw size of the cloned source code alone, exluding binaries, images and other non-text file, is 163 GB. Then, we extracted all the 102,707 closed issue that have been resolved using the *RESOLVED/FIXED* tags. Indeed, this study aims to classify bugs according to their fix locations. If an issue is fixed by other means than *fixing* the source code, then, it falls outside the scope our study. In order to assign commits to issues we used is the regular expression-based approach by Fischer et al. [28] matching the issue ID in the commit note. Using this technic, we were able to link almost 40% (40,493 out of 102,707) of our resolved/fixed issues to 229,153 commits. An issue can be fixed with several commits.

We choose not to use more complex technics like ReLink, an approach proposed by Wu et al.[23], which considers the following constraints: (i) matching the committer/authors with issue tracking contributor name/email; (ii) the time interval between the commit and the last comment posted by the same author/contributor on the issue tracker must be less

[2]https://projects.apache.org/projects.html?name

than seven days; and (iii) Vector Space Model (VSM) cosine similarity between the commit note and the last comment referred above or greater than 0.7 because we belive that mining more than forty thousands issues is enough to be significant.

Using our generated consolidated dataset, we extracted the files f_i impacted by each commit c_i for each one of our 388 projects. Then, we classify the bugs according to the following:

- **Type 1:** A bug is tagged type 1 if it is fixed by modifying a file f_i and f_i is not involved in any other bug fix.
- **Type 2:** A bug is tagged type 2 if it is fixed by modifying several files $f_{i..n}$ and the files $f_{i..n}$ are not involved in any other bug fix.
- **Type 3:** A bug is tagged type 3 if it is fixed by modifying a file f_i and the file f_i is involved in fixing other bugs.
- **Type 4:** A bug is tagged type 4 if it is fixed by modifying several files $f_{i..n}$ and the files $f_{i..n}$ are involved in any other bug fix.

To answer this question, we analyze whether any type is predominant in the studied ecosystem, by testing the null hypothesis:

- H_{01A} : The proportion of Types does not change significantly across the studied ecosystems

We test this hypothesis by observing both a "global" (across ecosystem) and a "local" predominance (per ecosystem) of the different types of bugs. We must observe these two aspects to ensure that the predominance of a particular type of bug is not circumstantial (in few given systems only) but is also not due to some other, unknown factors (in all systems but not in a particular ecosystem).

We answer $\mathbf{RQ}_1$ in two steps. The first step is to use descriptive statistics; we compute the ratio of each types to the total number of bugs in the dataset.

In the second step, we compare the proportions of the different types of bugs with respect to the ecosystem where the bugs were found. We build the contingency table with these two qualitative variables (the type and studied ecosystem) and test the null hypothesis $\mathbf{H}_{01A}$ to assess whether the proportion of a particular type of bugs is related to a specific ecosystem or not.

We use the Pearson's chi-squared test to reject the null hypothesis H_{01A}. Pearson's chi-squared independence test is used to analyze the relationship between two qualitative data, in our study the type bugs and the studied ecosystem. The results of Pearson's chi-squared independence test are considered statistically significant at α = 0.05. If p-value $\leq$ 0.05, we reject the null hypothesis H_{01A} and conclude that the proportion of each types is different for each ecosystem.

Overall, the data extraction and manipulation for $\mathbf{RQ}_1$ (i.e., cloning repositories, linking commits to issues and tagging issues by type) took thirteen weeks on two Linux servers having 1 quadcore 3.10 GHz CPU and 12 Gb of RAM each.

2) How complex is each type of bugs?: To answer $\mathbf{RQ}_2$ we went through our 40,493 of our resolved/fixed issues and the linked 229,153 commits in order to compute code and process metrics for each of them. These metrics will then be used to assess the complexity of a bug. The computed process metrics are:

- The time t it took to resolve issue i.
- The number of issues dup tagged as duplicate of issue i.
- The number of time issue i got reopen $reop$.
- The number of comments $comment$ on issue i.
- The severity sev of the issue i.

The computed code metrics are:

- The number of files f impacted by issue i.
- The number of commit c required to fix the issue i.
- The number of hunks h required to fix the issue i.
- The number of churns ch required to fix the issue i.

We address the relation between types and the complexity of the bugs in using our metrics. We analyze whether Types 2 and 4 bugs are more complex to handle than Types 1 and 3 bugs, by testing the null hypotheses:

- H_{02S} : The severity of different types is not significantly different
- H_{02D} : Different types are not significantly more likely to get duplicated.
- H_{02R} : Different types are not significantly more likely to get reopened.
- H_{02T} : There is no statistically-significant difference between the duration of fixing of different types.

For each hypothesis, we build a contingency table with the qualitative variables type of bugs and the dependent variable.

We use the Pearson's chi-squared test to reject the null hypothesis H_{02D} (respectively H_{02R}) and H_{02S}. The results of Pearson's chi-squared independence test are considered statistically significant at α = 0.05. If a p-value $\leq$ 0.05, we reject the null hypothesis H_{02D} (respectively H_{02R}) andconclude the fact that the bug is more likely to be duplicated (respectively reopened) is related to the type of the bug and we reject H_{02S} and conclude that the severity level of the bug is related to the bug type.

IV. Analysis of the Results

This section reports the analysis of the results achieved aiming at answering our two research questions.

Table II
CONTINGENCY TABLE AND PEARSON'S CHI-SQUARED TESTS

Ecosystem	T1	T2	T3	T4	Pearson's chi-squared p-Value
Apache	1968 (14.3 %)	1248 (9.1 %)	3101 (22.6 %)	7422 (54 %)	
Netbeans	776 (2.9 %)	240 (0.9 %)	8372 (31.3 %)	17366 (64.9 %)	<0.01
Overall	2744 (6.8 %)	1488 (3.7 %)	11473 (28.3 %)	24788 (61.2 %)	
	Types 1 and 2		Types 3 and 4		
Apache	3216 (23.4 %)		10523 (76.6 %)		
Netbeans	1016 (3.8 %)		25738 (96.2 %)		<0.01
All	4232 (10.5 %)		36261 (89.5 %)		
	Types 1 and 3		Types 2 and 4		
Apache	5069 (36.9 %)		8670 (63.1 %)		
Netbeans	9148 (34.2 %)		17606 (65.8 %)		<0.01
All	14217 (35.1 %)		26276 (64.9 %)		

A. What are the proportions of different types of bugs?

Table II presents a contingency table and the results of the Pearson's chi-squared tests we performed on each types of bug. In addition to presenting bug types 1 to 4, Table II also presents regroupement of bug types: (a) Types 1 and 2 versus Types 3 and 4 and (b) Types 1 and 3 versus Types 2 and 4.

Types 3 (22.6% and 54%) and 4 (31.3% and 64.9%) are predominants compared to types 1 (14.3% and 9.1%) and 2 (6.8% and 3.7%) for the Apache and the Netbeans ecosystems, respectively. Obviously, this observation also holds for when the two ecosystems are combined. Overall, the proportion of different types of bug is as follows: 6.8%, 3.7%, 28.3%, 61.2% for types 1, 2, 3 and 4, respectively. The result of the Pearson's tests are bellow 0.01. As a reminder, we consider results of Pearson's tests statistically significant at α<0.05. Consequently, we can conclude that there is a predominance of Types 3 and 4 in all different ecosystems and this observation is not related to a specific ecosystem. When combined into our first group, Types 1 & 2 versus. Types 3 & 4, there are significantly more Types 3 and 4 (89.5 %) than Types 1 and 2 (10.5 %). In the second group, Types 1 & 3 versus. Types 2 & 4, there are significantly more Types 2 & and 4 (64.9%) than Types 1 & 3 (35.1%).

B. How complex is each type of bugs?

To answer $\mathbf{RQ}_2$, we analyse the complexity of each bug in terms of duplication, fixing time, comments, reopenning, files impacted, severity, changesets, hunks and chunks.

Figure 5 presents nine boxplots describing our complexity metric for each types of each ecosystem. In each sub-figure, the boxplots are organised as follows: (a) Types 1 to 4 of the Apache ecosystem, (b) Types 1 to 4 of the Netbeans ecosystem and (c) Types 1 to 4 of both ecosystems combined. For all the metrics, except the severity, the median is close to zero and we can observe many outliers. Tables III, IV and V present descriptive statistics about each metric for each type for the Apache ecosystem, the Netbeans ecosystem and both ecosystems combined, respectively. The descriptive Statistics used are μ:mean, $\sum$:sum, $\hat{x}$:median, σ:standard deviation and $\%$:percentage. In addition, to the descriptive statistics, these tables present matrixes of Mann-Whitney test for each metric and type. We added the ✓ symbol to the Mann-Whitney test results columns when the value is statistically significant (e.g. α<0.05) and ✗ otherwise. Tables VI and VII are built similarly to Tables III, IV and V at the exception that they present the data of types groups: Types 1 & 2 vs. Types 3 & 4 and Types 1 & 3 vs. Types 2 & 4. Finally, Table VIII presents the Pearson's chi-squared tests results for each complexity metrics for Types 1 to 4 and our two types combination. In what follows, we present our findings for each complexity metric. omplexity metrics are divided in two groups: (a) process and (b) code metrics. Process metrics refer to metrics that have been extracted from the project tracking system (i.e. fixing time, comments, reopening and severity). Code metrics are directly computed using the source code used to fix a given bug (i.e. files impacted, changesets required, hunks and chunks). We acknowledge that these complexity metrics only represent an abstraction of the actual complexity of a given bug as they cannot account for the actual thought processes and expertise required to craft a fix. However, we believe that they are an accurate abstraction. Moreover, they are used in several studies in the field also rely on these metrics to accurately approximate the complexity of bug [1], [29], [30], [31], [32].

1) Duplicate: The duplicate metric represents the number of time a bug gets resolved using the *duplicate* label while referencing one of the *resolved/fixed* bug of our dataset. The process metric is useful to approximate the impact of a given bug on the community. Indeed, for a bug to be resolved using the *duplicate*, it means that the bug has been reported before. The more a bug gets reported by the community, the more people are impacted enough to report it. Note that, for a bug_a

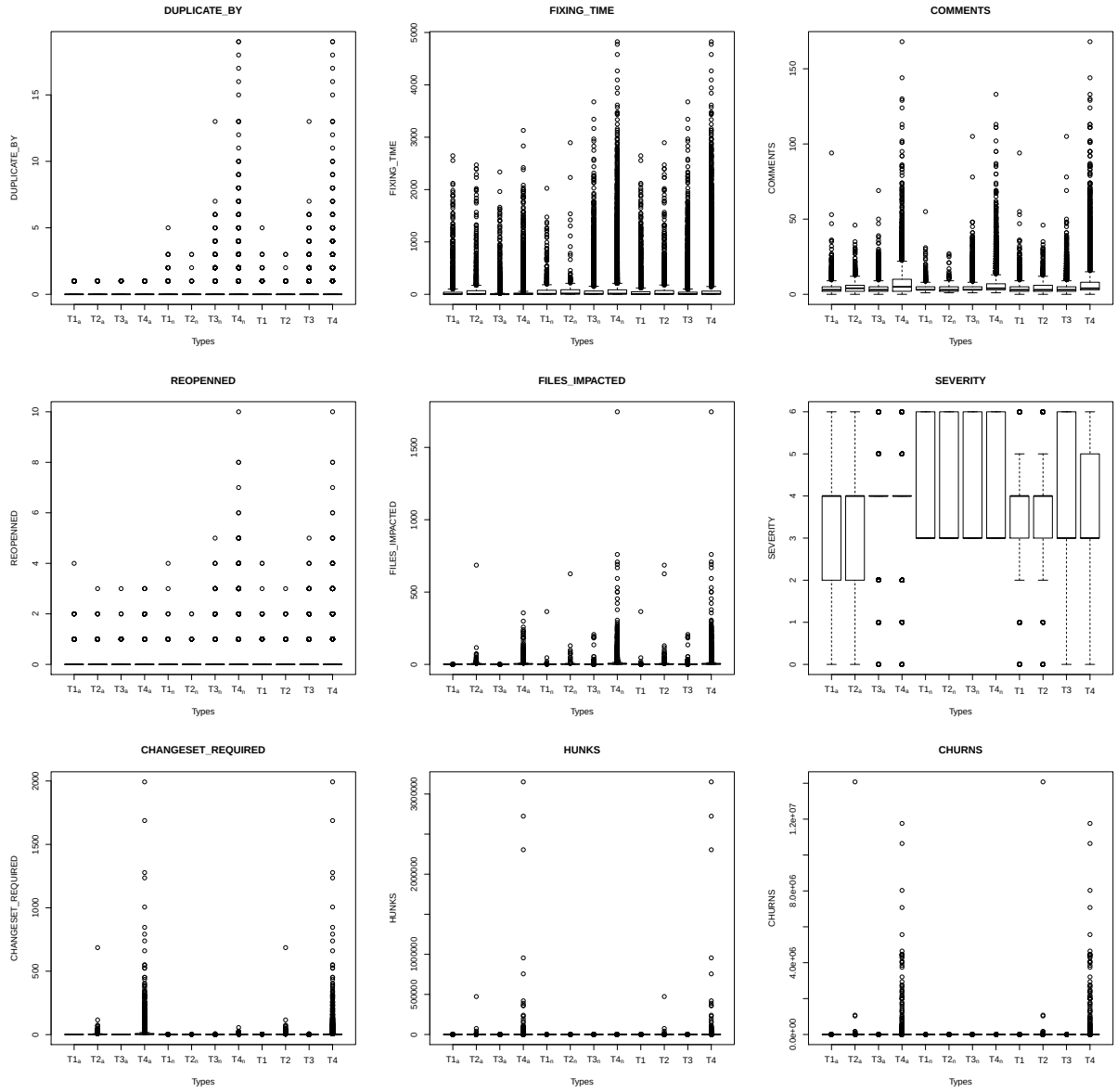

Figure 5. Complexity metrics boxplots. From left to right and top to bottom: Duplicate, Fixing time, Comments, Reopening, Files impacted, Severity, Changesets, Hunks and Chunks.

to be resolved using the *duplicate* label and referencing bug_b, bug_b does not have to resolved itself. Indeed, bug_b could be under investigation (i.e. *unconfirmed*) or being fixed (i.e. *new* or *assigned*).

In the Apache ecosystem, the types that are the most likely to get duplicated, ordered by ascending mean duplication rate, are T3 (0.016) $<$ T2 (0.022) $<$ T1 (0.026) $<$ T4 (0.029) and they represent 14.8%, 8.1%, 14.5% and 62.6% of the total duplications, respectively. The differences between duplication means by types, however, are only significant in 33.33% (4/12) of the case. Indeed, the mean duplication are only significant on the following cases: T1 vs. T3, T3 vs. T4. For the Apache ecosystem, we can conclude that $T4^1_{dup} \gg T1^2_{dup} \gg T3^4_{dup}$. We use the notation $x^r_m \gg y^r_m$ ($x^r_m \ll y^r_m$) to represent that x, along the metric m, is significantly greater (lower) than y, along the same metric, according to the mann-whitney tests (α<0.05). r represents the rank of x (y) according to m from 1 (higher percentage) to 4 (lower percentage).

Netbeans T1 0.086 (2.5%) T2 0.067 (0.6%) T3 0.074

Table VIII
PEARSON'S CHI-SQUARED TESTS FOR COMPLEXITY METRICS

Eco.	Metric	All	T1T2 v. T3T4	T1T3 v. T2T4
Apache	Dup.	<0.01	<0.01	<0.01
	Tim.	<0.01	<0.01	<0.01
	Com.	<0.01	<0.01	<0.01
	Reo.	<0.01	<0.01	<0.01
	Fil.	<0.01	<0.01	<0.01
	Sev.	<0.01	<0.01	<0.01
	Cha.	<0.01	<0.01	<0.01
	Hun.	<0.01	<0.01	<0.01
	Chur.	<0.01	<0.01	<0.01
Netbeans	Dup.	<0.01	<0.01	<0.01
	Tim.	<0.01	<0.01	<0.01
	Com.	<0.01	<0.01	<0.01
	Reo.	<0.01	<0.01	<0.01
	Fil.	<0.01	<0.01	<0.01
	Sev.	<0.01	<0.01	<0.01
	Cha.	<0.01	<0.01	<0.01
	Hun.	<0.01	<0.01	<0.01
	Chur.	<0.01	<0.01	<0.01
Overall	Dup.	<0.01	<0.01	<0.01
	Tim.	<0.01	<0.01	<0.01
	Com.	<0.01	<0.01	<0.01
	Reo.	<0.01	<0.01	<0.01
	Fil.	<0.01	<0.01	<0.01
	Sev.	<0.01	<0.01	<0.01
	Cha.	<0.01	<0.01	<0.01
	Hun.	<0.01	<0.01	<0.01
	Chur.	<0.01	<0.01	<0.01

(23.3%) T4 0.113 (73.6%)

Overall T1 0.043 (3.9%) T2 0.03 (1.5%) T3 0.058 (22.3%) T4 0.088 (72.3%)

2) Fixing time:

3) Comments:

4) Reopening:

5) Severity:

6) Files impacted:

7) Changesets:

8) Hunks:

9) Chunks:

V. CONCLUSION

The conclusion goes here.

VI. REPRODUCTION PACKAGE

We provide a repoduction package that is publicly available at https://research.mathieu-nayrolles.com/taxonomy/reproduction.zip. All the instructions needed to reproduce our results are self contained in the provided archive.

Table III
APACHE ECOSYSTEM COMPLEXITY METRICS COMPARISON AND MANN-WHITNEY TEST RESULTS.
μ:MEAN, $\sum$:SUM, $\hat{x}$:MEDIAN, σ:STANDARD DEVIATION, $\%$:PERCENTAGE

Types	Metric	μ	$\sum$	$\hat{x}$	σ	$\%$	T1	T2	T3	T4
	Dup.	0.026	51	0	0.2	14.8	n.a	✗(0.53)	✓(<0.05)	✗(0.45)
	Tim.	91.574	180217	4	262	21.8	n.a	✓(<0.05)	✓(<0.05)	✓(<0.05)
	Com.	4.355	8571	3	4.7	9.5	n.a	✓(<0.05)	✗(0.17)	✓(<0.05)
	Reo.	0.062	122	0	0.3	13.8	n.a	✗(0.29)	✓(<0.05)	✓(<0.05)
T1	Fil.	0.991	1950	1	0.1	3.7	n.a	✓(<0.05)	✗(0.28)	✓(<0.05)
	Sev.	3.423	6737	4	1.3	13.2	n.a	✗(0.18)	✓(<0.05)	✓(<0.05)
	Cha.	1	1968	1	0	1.9	n.a	✓(<0.05)	✓(<0.05)	✓(<0.05)
	Hun.	3.814	7506	3	2.4	0	n.a	✓(<0.05)	✓(<0.05)	✓(<0.05)
	Chur.	18.761	36921	7	48.6	0	n.a	✓(<0.05)	✗(0.09)	✓(<0.05)
	Dup.	0.022	28	0	0.1	8.1	✗(0.53)	n.a	✗(0.16)	✗(0.19)
	Tim.	115.158	143717	8	294.1	17.4	✓(<0.05)	n.a	✓(<0.05)	✓(<0.05)
	Com.	5.041	6291	4	4.7	7	✓(<0.05)	n.a	✓(<0.05)	✓(<0.05)
	Reo.	0.071	89	0	0.3	10.1	✗(0.29)	n.a	✓(<0.05)	✗(0.59)
T2	Fil.	4.381	5468	2	20.4	10.5	✓(<0.05)	n.a	✓(<0.05)	✓(<0.05)
	Sev.	3.498	4365	4	1.2	8.6	✗(0.18)	n.a	✓(<0.05)	✓(<0.05)
	Cha.	4.681	5842	2	20.4	5.5	✓(<0.05)	n.a	✓(<0.05)	✓(<0.05)
	Hun.	561.995	701370	14	13628.2	3.9	✓(<0.05)	n.a	✓(<0.05)	✓(<0.05)
	Chur.	14184.869	17702716	88	400710.2	8	✓(<0.05)	n.a	✓(<0.05)	✓(<0.05)
	Dup.	0.016	50	0	0.1	14.5	✓(<0.05)	✗(0.16)	n.a	✓(<0.05)
	Tim.	35.892	111300	1	151.8	13.5	✓(<0.05)	✓(<0.05)	n.a	✓(<0.05)
	Com.	4.422	13712	3	4.4	15.2	✗(0.17)	✓(<0.05)	n.a	✓(<0.05)
	Reo.	0.033	101	0	0.2	11.5	✓(<0.05)	✓(<0.05)	n.a	✓(<0.05)
T3	Fil.	0.994	3081	1	0.1	5.9	✗(0.28)	✓(<0.05)	n.a	✓(<0.05)
	Sev.	3.644	11300	4	1.1	22.2	✓(<0.05)	✓(<0.05)	n.a	✓(<0.05)
	Cha.	1	3101	1	0	2.9	✓(<0.05)	✓(<0.05)	n.a	✓(<0.05)
	Hun.	4.022	12472	3	3.4	0.1	✓(<0.05)	✓(<0.05)	n.a	✓(<0.05)
	Chur.	16.954	52573	6	49.8	0	✗(0.09)	✓(<0.05)	n.a	✓(<0.05)
	Dup.	0.029	216	0	0.2	62.6	✗(0.45)	✗(0.19)	✓(<0.05)	n.a
	Tim.	52.76	391586	4	182.2	47.4	✓(<0.05)	✓(<0.05)	✓(<0.05)	n.a
	Com.	8.313	61701	5	10.2	68.3	✓(<0.05)	✓(<0.05)	✓(<0.05)	n.a
	Reo.	0.077	570	0	0.3	64.6	✓(<0.05)	✗(0.59)	✓(<0.05)	n.a
T4	Fil.	5.633	41805	3	14	79.9	✓(<0.05)	✓(<0.05)	✓(<0.05)	n.a
	Sev.	3.835	28466	4	1	56	✓(<0.05)	✓(<0.05)	✓(<0.05)	n.a
	Cha.	12.861	95455	4	52.2	89.7	✓(<0.05)	✓(<0.05)	✓(<0.05)	n.a
	Hun.	2305.868	17114149	30	58094.7	96	✓(<0.05)	✓(<0.05)	✓(<0.05)	n.a
	Chur.	27249.773	202247816	204	320023.5	91.9	✓(<0.05)	✓(<0.05)	✓(<0.05)	n.a

Table IV
NETBEANS ECOSYSTEM COMPLEXITY METRICS COMPARISON AND MANN-WHITNEY TEST RESULTS.
μ:MEAN, $\sum$:SUM, $\hat{x}$:MEDIAN, σ:STANDARD DEVIATION, $\%$:PERCENTAGE

Types	Metric	μ	$\sum$	$\hat{x}$	σ	$\%$	T1	T2	T3	T4
	Dup.	0.086	67	0	0.4	2.5	n.a	✗(0.39)	✗(0.24)	✗(0.86)
	Tim.	92.759	71981	10	219.1	2.3	n.a	✓(<0.05)	✗(0.15)	✓(<0.05)
	Com.	4.687	3637	3	4.1	2.4	n.a	✓(<0.05)	✗(0.83)	✓(<0.05)
	Reo.	0.054	42	0	0.3	1.9	n.a	✗(0.1)	✗(0.58)	✓(<0.05)
T1	Fil.	1.735	1346	1	13.2	0.8	n.a	✓(<0.05)	✓(<0.05)	✓(<0.05)
	Sev.	4.314	3348	3	1.5	3.1	n.a	✗(0.66)	✓(<0.05)	✓(<0.05)
	Cha.	1.085	842	1	0.4	2	n.a	✗(0.99)	✗(0.26)	✓(<0.05)
	Hun.	4.405	3418	3	7	0.5	n.a	✓(<0.05)	✗(0.13)	✓(<0.05)
	Chur.	5.089	3949	2	12.5	0.3	n.a	✓(<0.05)	✓(<0.05)	✓(<0.05)
	Dup.	0.067	16	0	0.3	0.6	✗(0.39)	n.a	✗(0.73)	✗(0.39)
	Tim.	111.9	26856	16	308.6	0.9	✓(<0.05)	n.a	✓(<0.05)	✗(0.41)
	Com.	4.433	1064	3	4	0.7	✓(<0.05)	n.a	✓(<0.05)	✓(<0.05)
	Reo.	0.079	19	0	0.3	0.9	✗(0.1)	n.a	✗(0.11)	✗(0.97)
T2	Fil.	8.804	2113	2	42.7	1.3	✓(<0.05)	n.a	✓(<0.05)	✓(<0.05)
	Sev.	4.362	1047	3	1.5	1	✗(0.66)	n.a	✓(<0.05)	✓(<0.05)
	Cha.	1.075	258	1	0.3	0.6	✗(0.99)	n.a	✗(0.5)	✓(<0.05)
	Hun.	21.887	5253	8	62.7	0.7	✓(<0.05)	n.a	✓(<0.05)	✓(<0.05)
	Chur.	32.263	7743	8	125.8	0.7	✓(<0.05)	n.a	✓(<0.05)	✓(<0.05)
	Dup.	0.074	620	0	0.4	23.3	✗(0.24)	✗(0.73)	n.a	✓(<0.05)
	Tim.	87.033	728642	9	233.6	23.8	✗(0.15)	✓(<0.05)	n.a	✓(<0.05)
	Com.	4.73	39599	3	4.3	26.5	✗(0.83)	✓(<0.05)	n.a	✓(<0.05)
	Reo.	0.06	499	0	0.3	22.7	✗(0.58)	✗(0.11)	n.a	✓(<0.05)
T3	Fil.	1.306	10932	1	5.1	6.8	✓(<0.05)	✓(<0.05)	n.a	✓(<0.05)
	Sev.	4.021	33666	3	1.4	31.4	✓(<0.05)	✓(<0.05)	n.a	✓(<0.05)
	Cha.	1.065	8917	1	0.3	21	✗(0.26)	✗(0.5)	n.a	✓(<0.05)
	Hun.	5.15	43115	3	12.4	5.8	✗(0.13)	✓(<0.05)	n.a	✓(<0.05)
	Chur.	6.727	56317	2	22	4.9	✓(<0.05)	✓(<0.05)	n.a	✓(<0.05)
	Dup.	0.113	1959	0	0.7	73.6	✗(0.86)	✗(0.39)	✓(<0.05)	n.a
	Tim.	128.833	2237319	13	332.8	73	✓(<0.05)	✗(0.41)	✓(<0.05)	n.a
	Com.	6.058	105202	4	6.7	70.4	✓(<0.05)	✓(<0.05)	✓(<0.05)	n.a
	Reo.	0.094	1639	0	0.4	74.5	✓(<0.05)	✗(0.97)	✓(<0.05)	n.a
T4	Fil.	8.408	146019	4	25.1	91	✓(<0.05)	✓(<0.05)	✓(<0.05)	n.a
	Sev.	3.982	69159	3	1.4	64.5	✓(<0.05)	✓(<0.05)	✓(<0.05)	n.a
	Cha.	1.871	32494	2	1.2	76.4	✓(<0.05)	✓(<0.05)	✓(<0.05)	n.a
	Hun.	40.195	698022	13	98.3	93.1	✓(<0.05)	✓(<0.05)	✓(<0.05)	n.a
	Chur.	61.893	1074830	15	178.6	94	✓(<0.05)	✓(<0.05)	✓(<0.05)	n.a

Table V
APACHE AND NETBEANS ECOSYSTEMS COMPLEXITY METRICS COMPARISON AND MANN-WHITNEY TEST RESULTS.
μ:MEAN, $\sum$:SUM, $\hat{x}$:MEDIAN, σ:STANDARD DEVIATION, $\%$:PERCENTAGE

Types	Metric	μ	$\sum$	$\hat{x}$	σ	$\%$	T1	T2	T3	T4
	Dup.	0.043	118	0	0.3	3.9	n.a	✗(0.09)	✗(0.16)	✓(<0.05)
	Tim.	91.909	252198	6	250.6	6.5	n.a	✓(<0.05)	✓(<0.05)	✓(<0.05)
	Com.	4.449	12208	3	4.5	5.1	n.a	✓(<0.05)	✓(<0.05)	✓(<0.05)
	Reo.	0.06	164	0	0.3	5.3	n.a	✗(0.07)	✓(<0.05)	✓(<0.05)
T1	Fil.	1.201	3296	1	7	1.5	n.a	✓(<0.05)	✓(<0.05)	✓(<0.05)
	Sev.	3.675	10085	4	1.4	6.4	n.a	✗(0.97)	✗(0.17)	✓(<0.05)
	Cha.	1.024	2810	1	0.2	1.9	n.a	✓(<0.05)	✓(<0.05)	✓(<0.05)
	Hun.	3.981	10924	3	4.3	0.1	n.a	✓(<0.05)	✓(<0.05)	✓(<0.05)
	Chur.	14.894	40870	5	42.2	0	n.a	✓(<0.05)	✓(<0.05)	✓(<0.05)
	Dup.	0.03	44	0	0.2	1.5	✗(0.09)	n.a	✓(<0.05)	✓(<0.05)
	Tim.	114.632	170573	9	296.4	4.4	✓(<0.05)	n.a	✓(<0.05)	✗(0.15)
	Com.	4.943	7355	3	4.6	3.1	✓(<0.05)	n.a	✗(0.72)	✓(<0.05)
	Reo.	0.073	108	0	0.3	3.5	✗(0.07)	n.a	✓(<0.05)	✗(0.47)
T2	Fil.	5.095	7581	2	25.4	3.6	✓(<0.05)	n.a	✓(<0.05)	✓(<0.05)
	Sev.	3.637	5412	4	1.3	3.4	✗(0.97)	n.a	✗(0.44)	✗(0.1)
	Cha.	4.099	6100	2	18.7	4.1	✓(<0.05)	n.a	✓(<0.05)	✓(<0.05)
	Hun.	474.881	706623	12	12481.7	3.8	✓(<0.05)	n.a	✓(<0.05)	✓(<0.05)
	Chur.	11902.19	17710459	62	366988	8	✓(<0.05)	n.a	✓(<0.05)	✓(<0.05)
	Dup.	0.058	670	0	0.4	22.3	✗(0.16)	✓(<0.05)	n.a	✓(<0.05)
	Tim.	73.21	839942	6	215.8	21.6	✓(<0.05)	✓(<0.05)	n.a	✓(<0.05)
	Com.	4.647	53311	3	4.3	22.2	✓(<0.05)	✗(0.72)	n.a	✓(<0.05)
	Reo.	0.052	600	0	0.3	19.5	✓(<0.05)	✓(<0.05)	n.a	✓(<0.05)
T3	Fil.	1.221	14013	1	4.4	6.6	✓(<0.05)	✓(<0.05)	n.a	✓(<0.05)
	Sev.	3.919	44966	3	1.4	28.4	✗(0.17)	✗(0.44)	n.a	✓(<0.05)
	Cha.	1.048	12018	1	0.3	8.1	✓(<0.05)	✓(<0.05)	n.a	✓(<0.05)
	Hun.	4.845	55587	3	10.7	0.3	✓(<0.05)	✓(<0.05)	n.a	✓(<0.05)
	Chur.	9.491	108890	3	32.3	0	✓(<0.05)	✓(<0.05)	n.a	✓(<0.05)
	Dup.	0.088	2175	0	0.6	72.3	✓(<0.05)	✓(<0.05)	✓(<0.05)	n.a
	Tim.	106.056	2628905	9	297.9	67.6	✓(<0.05)	✗(0.15)	✓(<0.05)	n.a
	Com.	6.733	166903	4	8	69.6	✓(<0.05)	✓(<0.05)	✓(<0.05)	n.a
	Reo.	0.089	2209	0	0.4	71.7	✓(<0.05)	✗(0.47)	✓(<0.05)	n.a
T4	Fil.	7.577	187824	3	22.4	88.3	✓(<0.05)	✓(<0.05)	✓(<0.05)	n.a
	Sev.	3.938	97625	3	1.3	61.8	✓(<0.05)	✗(0.1)	✓(<0.05)	n.a
	Cha.	5.162	127949	2	29	85.9	✓(<0.05)	✓(<0.05)	✓(<0.05)	n.a
	Hun.	718.58	17812171	16	31804.5	95.8	✓(<0.05)	✓(<0.05)	✓(<0.05)	n.a
	Chur.	8202.463	203322646	28	175548.3	91.9	✓(<0.05)	✓(<0.05)	✓(<0.05)	n.a

Table VI
TYPES 1& AND 2 VERSUS TYPES 3 & 4 COMPLEXITY METRICS COMPARISON AND MANN-WHITNEY TEST RESULTS.
μ:MEAN, $\sum$:SUM, $\hat{x}$:MEDIAN, σ:STANDARD DEVIATION, %:PERCENTAGE

Ecosystem	Metric	Types 1 and 2					Types 3 and 4					Mann-Whitney
		μ	$\sum$	$\hat{x}$	σ	%	μ	$\sum$	$\hat{x}$	σ	%	p-value
Apache	Dup	0.025	79	0	0.2	22.9	0.025	266	0	0.2	77.1	✗(0.82)
	Time	100.726	323934	6	275.1	39.2	47.789	502886	3	17 3.9	60.8	✓(<0.05)
	Com	4.621	14862	3	4.7	16.5	7.166	75413	4	9.1	83.5	✓(<0.05)
	Reop	0.066	211	0	0.3	23.9	0.064	671	0	0.3	76.1	✗(0.74)
	Files	2.307	7418	1	12.8	14.2	4.266	44886	2	11.9	85.8	✓(<0.05)
	Severity	3.452	11102	4	1.2	21.8	3.779	39766	4	1	78.2	✓(<0.05)
	Change	2.428	7810	1	12.8	7.3	9.366	98556	3	44.2	92.7	✓(<0.05)
	Hunks	220.422	708876	4	8491.9	4	1627.542	17126621	15	48799.9	96	✓(<0.05)
	Churns	5516.056	17739637	15	249654.4	8.1	19224.593	202300389	72	269046.2	91.9	✓(<0.05)
Netbeans	Dup	0.082	83	0	0.4	3.1	0.1	2579	0	0.6	96.9	✗(0.92)
	Time	97.281	98837	11	243.2	3.2	115.237	2965961	12	304.8	96.8	✗(0.76)
	Com	4.627	4701	3	4	3.1	5.626	144801	4	6.1	96.9	✓(<0.05)
	Reop	0.06	61	0	0.3	2.8	0.083	2138	0	0.4	97.2	✗(0.08)
	Files	3.405	3459	1	23.9	2.2	6.098	156951	2	21.1	97.8	✓(<0.05)
	Severity	4.326	4395	3	1.5	4.1	3.995	102825	3	1.4	95.9	✓(<0.05)
	Change	1.083	1100	1	0.4	2.6	1.609	41411	1	1.1	97.4	✓(<0.05)
	Hunks	8.534	8671	3	31.9	1.2	28.795	741137	8	82.7	98.8	✓(<0.05)
	Churns	11.508	11692	3	63.1	1	43.949	1131147	8	149.5	99	✓(<0.05)
Overall	Dup	0.038	162	0	0.2	5.4	0.078	2845	0	0.5	94.6	✓(<0.05)
	Time	99.899	422771	7	267.8	10.9	95.663	3468847	8	275	89.1	✓(<0.05)
	Com	4.623	19563	3	4.6	8.2	6.073	220214	4	7.1	91.8	✓(<0.05)
	Reop	0.064	272	0	0.3	8.8	0.077	2809	0	0.3	91.2	✗(0.21)
	Files	2.57	10877	1	16.2	5.1	5.566	201837	2	18.9	94.9	✓(<0.05)
	Severity	3.662	15497	4	1.4	9.8	3.932	142591	3	1.3	90.2	✓(<0.05)
	Change	2.105	8910	1	11.2	6	3.86	139967	2	24.1	94	✓(<0.05)
	Hunks	169.553	717547	4	7403	3.9	492.754	17867758	9	26297.9	96.1	✓(<0.05)
	Churns	4194.548	17751329	10	217637.4	8	5610.202	203431536	13	145192.5	92	✓(<0.05)

Table VII
TYPES 1& AND 3 VERSUS TYPES 2 & 4 COMPLEXITY METRICS COMPARISON AND MANN-WHITNEY TEST RESULTS.
μ:MEAN, $\sum$:SUM, $\hat{x}$:MEDIAN, σ:STANDARD DEVIATION, $\%$:PERCENTAGE

Ecosystem	Metric	Types 1 and 3					Types 2 and 4					Mann-Whitney
		μ	$\sum$	$\hat{x}$	σ	$\%$	μ	$\sum$	$\hat{x}$	σ	$\%$	p-value
Apache	Dup.	0.02	101	0	0.1	29.3	0.028	244	0	0.2	70.7	✓(<0.05)
	Tim.	57.51	291517	2	203.7	35.3	61.742	535303	4	203.3	64.7	✓(<0.05)
	Com.	4.396	22283	3	4.5	24.7	7.842	67992	5	9.7	75.3	✓(<0.05)
	Reo.	0.044	223	0	0.2	25.3	0.076	659	0	0.3	74.7	✓(<0.05)
	Fil.	0.993	5031	1	0.1	9.6	5.452	47273	3	15.1	90.4	✓(<0.05)
	Sev.	3.558	18037	4	1.2	35.5	3.787	32831	4	1	64.5	✓(<0.05)
	Cha.	1	5069	1	0	4.8	11.684	101297	4	49	95.2	✓(<0.05)
	Hun.	3.941	19978	3	3.1	0.1	2054.846	17815519	26.5	54002	99.9	✓(<0.05)
	Chur.	17.655	89494	7	49.3	0	25369.15	219950532	180	332850.4	100	✓(<0.05)
Netbeans	Dup.	0.075	687	0	0.4	25.8	0.112	1975	0	0.7	74.2	✓(<0.05)
	Tim.	87.519	800623	9	232.4	26.1	128.602	2264175	13	332.5	73.9	✓(<0.05)
	Com.	4.726	43236	3	4.2	28.9	6.036	106266	4	6.7	71.1	✓(<0.05)
	Reo.	0.059	541	0	0.3	24.6	0.094	1658	0	0.4	75.4	✓(<0.05)
	Fil.	1.342	12278	1	6.2	7.7	8.414	148132	4	25.4	92.3	✓(<0.05)
	Sev.	4.046	37014	3	1.4	34.5	3.988	70206	3	1.4	65.5	✓(<0.05)
	Cha.	1.067	9759	1	0.3	23	1.86	32752	2	1.2	77	✓(<0.05)
	Hun.	5.087	46533	3	12	6.2	39.945	703275	13	98	93.8	✓(<0.05)
	Chur.	6.588	60266	2	21.4	5.3	61.489	1082573	15	178.1	94.7	✓(<0.05)
Overall	Dup.	0.055	788	0	0.3	26.2	0.084	2219	0	0.6	73.8	✓(<0.05)
	Tim.	76.819	1092140	6	223.1	28.1	106.541	2799478	9	297.8	71.9	✓(<0.05)
	Com.	4.608	65519	3	4.3	27.3	6.632	174258	4	7.9	72.7	✓(<0.05)
	Reo.	0.054	764	0	0.3	24.8	0.088	2317	0	0.4	75.2	✓(<0.05)
	Fil.	1.217	17309	1	5	8.1	7.437	195405	3	22.6	91.9	✓(<0.05)
	Sev.	3.872	55051	3	1.4	34.8	3.921	103037	3	1.3	65.2	✓(<0.05)
	Cha.	1.043	14828	1	0.2	10	5.102	134049	2	28.5	90	✓(<0.05)
	Hun.	4.678	66511	3	9.8	0.4	704.78	18518794	16	31033.2	99.6	✓(<0.05)
	Chur.	10.534	149760	4	34.5	0.1	8411.977	221033105	30	191558.8	99.9	✓(<0.05)

REFERENCES

[1] C. Weiß, T. Zimmermann, and A. Zeller, "How Long will it Take to Fix This Bug ?" in *Fourth International Workshop on Mining Software Repositories (MSR'07)*, no. 2, 2007, p. 1.

[2] H. Zhang, L. Gong, and S. Versteeg, "Predicting bug-fixing time: an empirical study of commercial software projects," in *International Conference on Software Engineering*. IEEE Press, may 2013, pp. 1042–1051. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486931

[3] N. Chen, "Star: stack trace based automatic crash reproduction," Ph.D. dissertation, The Hong Kong University of Science and Technology, 2013.

[4] S. Kim, T. Zimmermann, E. J. Whitehead Jr., and A. Zeller, "Predicting Faults from Cached History," in *29th International Conference on Software Engineering (ICSE'07)*. IEEE, may 2007, pp. 489–498. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=4222610

[5] M. Nayrolles, A. Hamou-Lhadj, T. Sofiene, and A. Larsson, "JCHARMING : A Bug Reproduction Approach Using Crash Traces and Directed Model Checking," in *Proceedings of the 22nd International Conference on Software Analysis, Evolution, and Reengineering, IEEE, 2015*, 2015, pp. 101–110.

[6] S. Eldh, "On Test Design," Ph.D. dissertation, Mälardalen, 2001.

[7] M. Hamill and K. Goseva-Popstojanova, "Exploring fault types, detection activities, and failure severity in an evolving safety-critical software system," *Software Quality Journal*, vol. 23, no. 2, pp. 229–265, apr 2014. [Online]. Available: http://link.springer.com/10.1007/s11219-014-9235-5

[8] M. W. G. Cory Kapser, "Toward a Taxonomy of Clones in Source Code: A Case Study." [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.5.6056

[9] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of Duplicate Defect Reports Using Natural Language Processing," in *29th International Conference on Software Engineering*. IEEE, may 2007, pp. 499–510. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=4222611

[10] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10*, vol. 1. New York, New York, USA: ACM Press, may 2010, p. 45. [Online]. Available: http://dl.acm.org/citation.cfm?id=1806799.1806811

[11] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering - ASE 2012*. New York, New York, USA: ACM Press, 2012, p. 70. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6494907

[12] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, jan 2004. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=1335465

[13] M. J. Pratt, "Introduction to ISO 10303the STEP Standard for Product Data Exchange," *Journal of Computing and Information Science in Engineering*, vol. 1, no. 1, p. 102, mar 2001. [Online]. Available: http://computingengineering.asmedigitalcollection.asme.org/article.aspx?articleid=1399190

[14] I. Burnstein, *Practical Software Testing: A Process-Oriented Approach*. Springer Science & Business Media, 2006. [Online]. Available: https://books.google.com/books?hl=en{&}lr={&}id=IM7iBwAAQBAJ{&}pgis=1

[15] J. Radatz, A. Geraci, and F. Katki, "IEEE standard glossary of software engineering terminology," *IEEE Std*, vol. 610121990, no. 121990, p. 3, 1990.

[16] J. A. Whittaker, J. Arbon, and J. Carollo, *How Google Tests Software*. Addison-Wesley, 2012. [Online]. Available: https://books.google.com/books?hl=en{&}lr={&}id=VrAx1ATf-RoC{&}pgis=1

[17] A. Zeller, *Configuration management with version sets: A unified software versioning model and its applications*, 1997.

[18] Bugzilla, "Life Cycle of a Bug," 2008.

[19] T. Koponen, "Life cycle of defects in open source software projects," in *Open Source Systems*. Springer, 2006, pp. 195–200.

[20] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. New York, New York, USA: ACM Press, 2008, p. 308. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1453101.1453146

[21] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, oct 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=630830.631291

[22] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein, "The missing links," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering - FSE '10*. New York, New York, USA: ACM Press, nov 2010,

p. 97. [Online]. Available: http://dl.acm.org/citation.cfm?id=1882291.1882308

[23] R. Wu, H. Zhang, S. Kim, and S. Cheung, "Relink: recovering links between bugs and changes," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering.*, 2011, pp. 15–25. [Online]. Available: http://dl.acm.org/citation.cfm?id=2025120

[24] T. J. McCabe and C. W. Butler, "Design complexity measurement and testing," *Communications of the ACM*, vol. 32, no. 12, pp. 1415–1425, dec 1989. [Online]. Available: http://dl.acm.org/citation.cfm?id=76380.76382

[25] G. Bavota, G. Canfora, M. D. Penta, R. Oliveto, and S. Panichella, "The Evolution of Project Inter-dependencies in a Software Ecosystem: The Case of Apache," in *2013 IEEE International Conference on Software Maintenance*. IEEE, sep 2013, pp. 280–289. [Online]. Available: http://dl.acm.org/citation.cfm?id=2550526.2550583

[26] X. O. Wang, E. Baik, and P. T. Devanbu, "System compatibility analysis of Eclipse and Netbeans based on bug data," in *Proceeding of the 8th working conference on Mining software repositories - MSR '11*. New York, New York, USA: ACM Press, may 2011, p. 230. [Online]. Available: http://dl.acm.org/citation.cfm?id=1985441.1985479

[27] Us News, "Software Developer Salary Information," 2014. [Online]. Available: http://money.usnews.com/careers/best-jobs/software-developer/salary

[28] M. Fischer, M. Pinzger, and H. Gall, "Populating a Release History Database from version control and bug tracking systems," in *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.* IEEE Comput. Soc, pp. 23–32. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=1235403

[29] R. K. Saha, S. Khurshid, and D. E. Perry, "An empirical study of long lived bugs," in *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. IEEE, feb 2014, pp. 144–153. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6747164

[30] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *2013 35th International Conference on Software Engineering (ICSE)*. Ieee, may 2013, pp. 382–391. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6606584

[31] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceeding of the 28th international conference on Software engineering - ICSE '06*. New York, New York, USA: ACM Press, may 2006, p. 361. [Online]. Available: http://dl.acm.org/citation.cfm?id=1134285.1134336

[32] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proceedings. 27th International Conference on Software Engineering, 2005.* IEEe, 2005, pp. 284–292. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=1553571