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

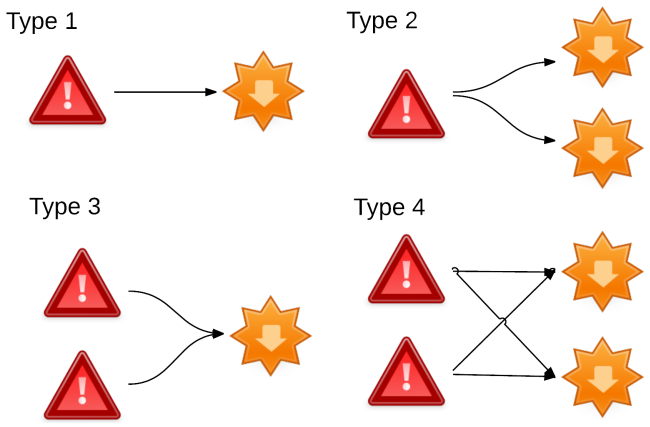

Figure 2. Proposed Taxonomy of Bugs

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.

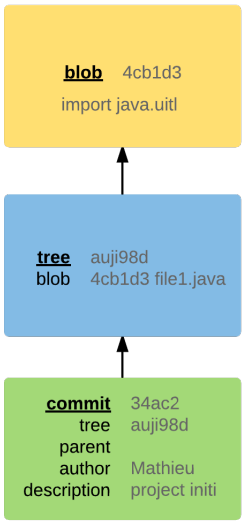

Figure 3. Data structure of a commit.

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.

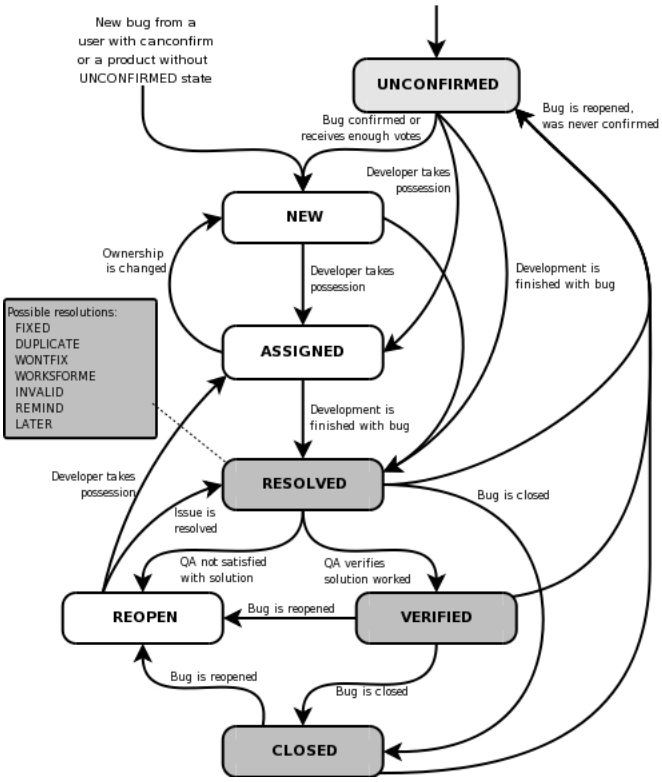

Figure 4. Lifecyle of a report [18]

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 [**?**]. 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).

Table I
DATASETS

Ecosystem	#Projects	#Classes	#Commits	R/F BR
Netbeans	39	277,741	53,258	30,595
122,632				
Apache	349	3,784,423	49,449	38,111
106,366				
Total	388	4,062,164	102,707	68,809
229,153				

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 [**?**]. On the other side, Netbeans have a relatively stable list of core developer and a common vision shared through the 39 related projects [**?**].

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[**?**].

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. [**?**] 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

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

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 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. Conclusion

The conclusion goes here.

V. 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.

Appendix A

Table II lists all the top-level projects we analysed for this study.

Table II: List of softwares

Parsers	
Mime4j	Apache James Mime4J provides a parser, MimeStreamParser, for e-mail message streams in plain rfc822 and MIME format
Xerces	XML parsers for c++, java and perl
Xalan	XSLT processor for transforming XML documents into HTML, text, or other XML document types.
FOP	Print formatter driven by XSL formatting objects (XSL-FO) and an output independent formatter.
Droids	intelligent standalone robot framework that allows to create and extend existing droids (robots).
Betwit	XML introspection mechanism for mapping beans to XML
Databases	
Drill	Schema-free SQL Query Engine for Hadoop, NoSQL and Cloud Storage
Tez	Frameword for complex directed-acyclic-graph of tasks for processing data.built atop Apache Hadoop YARN.
HBase	Apache HBase is the Hadoop database, a distributed, scalable, big data store.
Falcon	Falcon is a feed processing and feed management system aimed at making it easier for end consumers to onboard their feed processing and feed management on hadoop clusters.
Cassandra	Database with high scalability and high availability without compromising performance
Hive	Data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage using SQL
Sqoop	Tool designed for efficiently transferring bulk data between Apache Hadoop and structured datastores such as relational databases.
Accumulo	Sorted, distributed key/value store is a robust, scalable, high performance data storage and retrieval system.
Lucene	Full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.
CouchDB	Store your data with JSON documents. Access your documents and query your indexes with your web browser, via HTTP.
Phoenix	OLTP and operational analytics in Hadoop for low latency applications
OpenJPA	Java persistence project that can be used as a stand-alone POJO persistence layer or integrated into any Java EE
Gora	Provides an in-memory data model and persistence for big data
Optiq	framework that allows efficient translation of queries involving heterogeneous and federated data.
HCatalog	Table and storage management layer for Hadoop that enables users with different data processing tools
DdlUtils	Component for working with Database Definition (DDL) files
Derby	Relational database implemented entirely in Java
DBCP	Supports interaction with a relational database
JDO	Object persistence technology
Web & Services	
Wicket	Server-side Java web framework
Service Mix	The components project holds a set of JBI (ava Business Integration) components that can be installed in both the ServiceMix 3 and ServiceMix 4 containers.
Shindig	Apache Shindig is an OpenSocial container and helps you to start hosting OpenSocial apps quickly by providing the code to render gadgets, proxy requests, and handle REST and RPC requests.
Felix	Implement the OSGi Framework and Service platform and other interesting OSGi-related technologies under the Apache license.
Trinidad	JSF framework including a large, enterprise quality component library.
Axis	Web Services / SOAP / WSDL engine.
Synapse	Lightweight and high-performance Enterprise Service Bus
Giraph	Iterative graph processing system built for high scalability.
Tapestry	A component-oriented framework for creating highly scalable web applications in Java.
JSPWiki	WikiWiki engine, feature-rich and built around standard JEE components (Java, servlets, JSP).
TomEE	Java EE 6 Web Profile certified application server extends Apache Tomcat.
Knox	REST API Gateway for interacting with Apache Hadoop clusters.
Flex	Framework for building expressive web and mobile applications
Lucy	Search engine library provides full-text search for dynamic programming languages
Camel	Define routing and mediation rules in a variety of domain-specific languages, including a Java-based Fluent API, Spring or Blueprint XML Configuration files, and a Scala DSL.
Pivot	Builds installable Internet applications (IIAs)
Celix	Implementation of the OSGi specification adapted to C
Traffic Server	Fast, scalable and extensible HTTP/1.1 compliant caching proxy server.
Apache Net	Implements the client side of many basic Internet protocols. The purpose of the library is to provide fundamental protocol access, not higher-level abstractions.
Sling	Innovative web framework
Axis	Implementation of the SOAP (”Simple Object Access Protocol”) submission to W3C.
Shale	Web application framework, fundamentally based on JavaServer Faces.
Rave	web and social mashup engine that aggregates and serves web widgets.
Tuscany	Simplifies the task of developing SOA solutions by providing a comprehensive infrastructure for SOA development and management that is based on Service Component Architecture (SCA) standard.
Pluto	Implementation of the Java Portlet Specification.
ODE	Executes business processes written following the WS-BPEL standard
Muse	Java-based implementation of the WS-ResourceFramework (WSRF), WS-BaseNotification (WSN), and WS-DistributedManagement (WSDM) specifications.
WS-Commons	Web Services Commons Projects

Geronimo	Server runtime that integrates the best open source projects to create Java/OSGi
River	Network architecture for the construction of distributed systems in the form of modular co-operating services
Commons FileUpload	Makes it easy to add robust, high-performance, file upload capability to your servlets and web applications.
Beehive	Java Application Framework that was designed to simplify the development of Java EE based applications.
Aries	Java components enabling an enterprise OSGi application programming model.
Empire Db	Relational database abstraction layer and data persistence component
Commons Daemon	Java based daemons or services
Click	JEE web application framework
Stanbol	Provides a set of reusable components for semantic content management.
CXF	Open-Source Services Framework
Sandesha2	Axis2 module that implements the WS-ReliableMessaging specification published by IBM, Microsoft, BEA and TIBCO
Neethi	Framework for the programmers to use WS Policy
Rampart	Provides implementations of the WS-Sec* specifications for Apache Axis2.
AWF	web server
Nutch	Web crawler
HttpAsyncClient	Designed for extension while providing robust support for the base HTTP protocol
Portals Bridges	Portlet development using common web frameworks like Struts, JSF, PHP, Perl, Velocity and Scripts such as Groovy, JRuby, Jython, BeanShell or Rhino JavaScript.
Stonehenge	set of example applications for Service Oriented Architecture that spans languages and platforms and demonstrates best practices and interoperability.
Cloud & Big data	
Whirr	Set of libraries for running cloud services
Ambari	Aimed at making Hadoop management simpler by developing software for provisioning, managing, and monitoring Apache Hadoop clusters.
Karaf	Karaf provides dual polymorphic container and application bootstrapping paradigms to the Enterprise.
Hadoop	Software for reliable, scalable, distributed computing.
Hama	framework for Big Data analytics which uses the Bulk Synchronous Parallel (BSP) computing model.
Twill	Abstraction over Apache Hadoop YARN that reduces the complexity of developing distributed applications
Hadoop MapReduce	Framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.
Tajo	Big data relational and distributed data warehouse system for Apache Hadoop
Sentry	System for enforcing fine grained role based authorization to data and metadata stored on a Hadoop cluster.
Oozie	Workflow scheduler system to manage Apache Hadoop jobs.
Solr	Provides distributed indexing, replication and load-balanced querying, automated failover and recovery, centralized configuration
Airavata	Software framework that enables you to compose, manage, execute, and monitor large scale applications
JClouds	Multi-cloud toolkit for the Java platform that gives you the freedom to create applications that are portable across clouds while giving you full control to use cloud-specific features.
Impala	Native analytic database for Apache Hadoop.
Libcloud	Python library for interacting with many of the popular cloud service providers using a unified API.
Slider	deploy existing distributed applications on an Apache Hadoop YARN cluster
MRUNIT	Java library that helps developers unit test Apache Hadoop map reduce jobs.
Stratos	Framework that helps run Apache Tomcat, PHP, and MySQL applications and can be extended to support many more environments on all major cloud infrastructures
Mesos	Abstracts CPU, memory, storage, and other compute resources away from machines
Helix	A cluster management framework for partitioned and replicated distributed resources
Argus	Centralized approach to security policy definition and coordinated enforcement
DeltaCloud	API that abstracts differences between clouds
MRQL	Query processing and optimization system for large-scale, distributed data analysis, built on top of Apache Hadoop, Hama, Spark, and Flink.
Provisionr	create and manage pools of virtual machines on multiple clouds
Curator	A ZooKeeper Keeper.
ZooKeeper	Open-source server which enables highly reliable distributed coordination
Bigtop	Infrastructure Engineers and Data Scientists looking for comprehensive packaging, testing, and configuration of the leading open source big data components.
Yarn	split up the functionalities of resource management and job scheduling/monitoring into separate daemons.
Messaging and Logging	
Activemq	Messaging queue
Qpid	Messaging queue
log4cxx	Logging framework for C++
log4j	Logging framework for Java
log4net	Logging framework for .Net
Flume	Distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data.
Samza	The project aims to provide a near-realtime, asynchronous computational framework for stream processing.
Pig	Analyzing large data sets that consists of a high-level language for expressing data analysis programs.
Chukwa	Data collection system for monitoring large distributed systems

BookKeeper	Replicated log service which can be used to build replicated state machines.
Apollo	Faster, more reliable, easier to maintain messaging broker built from the foundations of the original ActiveMQ.
S4	Processes continuous unbounded streams of data.
Graphics	
Commons Imaging	Pure-Java Image Library
PDFBox	Java tool for working with PDF documents.
Batik	Java-based toolkit for applications or applets that want to use images in the Scalable Vector Graphics (SVG)
XML Graphics Commons	consists of several reusable components used by Apache Batik and Apache FOP
UIMA	UIMA frameworks, tools, and annotators, facilitating the analysis of unstructured content such as text, audio and video.
Dependency Management and build systems	
Tentacles	Downloads all the archives from a staging repo, unpack them and create a little report of what is there.
Ivy	Transitive dependency manager
Rat	Release audit tool, focused on licenses.
Ant	drive processes described in build files as targets and extension points dependent upon each other
EasyAnt	Improved integration in existing build systems
IvyIDE	Eclipse plugin which integrates Apache Ivy's dependency management into Eclipse
NPanday	Maven for .NET
Maven	software project management and comprehension tool
Networking	
Mina	100% pure java library to support the SSH protocols on both the client and server side.
James	Delivers a rich set of open source modules and libraries, written in Java, related to Internet mail communication which build into an advanced enterprise mail server.
Hupa	Rich IMAP-based Webmail application written in GWT (Google Web Toolkit).
Etch	cross-platform, language and transport-independent framework for building and consuming network services
Commons IO	Library of utilities to assist with developing IO functionality.
File systems and repository	
Tika	detects and extracts metadata and text from over a thousand different file types
OODT	Apache Object Oriented Data Technology (OODT) is a smart way to integrate and archive your processes, your data, and its metadata.
Commons Virtual File System	Provides a single API for accessing various different file systems.
Jackrabbit Oak	Scalable and performant hierarchical content repository
Directory	Provides directory solutions entirely written in Java.
SANDBOX	Subversion repository for Commons committers to function as an open workspace for sharing and collaboration.
Misc	
Harmony	Modular Java runtime with class libraries and associated tools.
Mahout	Machine learning applications.
OpenCMIS	Apache Chemistry OpenCMIS is a collection of Java libraries, frameworks and tools around the CMIS specification.
Apache Commons	Apache project focused on all aspects of reusable Java components
Shiro	Java security framework
Cordova	Mobile apps with HTML, CSS & JS
XMLBeans	Technology for accessing XML by binding it to Java types
State Chart XML	Provides a generic state-machine based execution environment based on Harel State Tables
excalibur	lightweight, embeddable Inversion of Control
Commons Transaction	Transactional Java programming
Velocity	collection of POJO
BCEL	analyze, create, and manipulate binary Java class files
Abdera	Functionally-complete, high-performance implementation of the IETF Atom Syndication Format
Commons Collections	Data structures that accelerate development of most significant Java applications.
Java Caching System	Distributed caching system written in Java
OGNL	Object-Graph Navigation Language; it is an expression language for getting and setting properties of Java objects, plus other extras such as list projection and selection and lambda expressions.
Anything To Triples	library that extracts structured data in RDF format from a variety of Web documents.
Axiom	provides an XML Infoset compliant object model implementation which supports on-demand building of the object tree
Graft	debugging and testing tool for programs written for Apache Giraph
Hivemind	Services and configuration microkernel
JXPath	defines a simple interpreter of an expression language called XPath

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