An Empirical Study On Leveraging Logs During Bug Fixes

Suhas Kabinna
School of Computing
Queen's University
Kingston, Ontario, Canada K7L 2N8
Email: kabinna@queensu.ca

Weiyi Shang
School of Computing
Queen's University
Kingston, Ontario, Canada K7L 2N8
Email: swy@queensu.ca

Ahmed E. Hassan
School of Computing
Queen's University
Kingston, Ontario, Canada K7L 2N8
Email: ahmed@queensu.ca

***Abstract*—Logging is a practice used by software developers to record and convey important information during the execution of a system. Logs can be used to output the behavior of the system when running, to monitor the choke points of a system, and to help in debugging the system.**

These logs are valuable sources of information for developers in debugging large scale software systems. Prior research has shown that over 41% of log changes are done during bug fixes. However there exists little knowledge about how logs leveraged during bug fixes.

In this paper we sought to study the leverage of logs during bug fixes through case studies on 3 open source systems like Hadoop, HDFS and Qpid from the Apace Foundation. We found that logs are statistically significantly more changed in bug fixes than other code changes. We find four different types of log modifications, that developers make to logging statements during bug fixes namely - changes to logging level, changes to the textual content of a log, changes to log variables (parameters) and relocation of logs. We found that developers add more variables to log during bug fixes . Finally we found that issue reports with log changes have larger code churn, but require less people, less time and less discussion to get fixed. Our study demonstrates the importance logs play in bug fixes and motivates developers to log more.

I. Introduction

Logging is a software practice leveraged by developers to record useful information during the execution of a system. Logging can be done through simple *printf* statements or through the usage of logging libraries such as 'Log4j', 'Slf4j', 'JCL' etc. Each log contains a textual part which gives information about the context, a variable part which contains information about the events, and logging levels which provide the verbosity of the logs. An example of a logging statement is shown below

```
LOG.info("Connected to " + host);
```

Logs are widely used in practice. Prior research has shown that logs are leveraged for detecting anomalies [1], [2], [3], monitoring performance of systems[4], maintenance of large scale systems [3] and debugging [25]. The valuable information in logs leads to a new market for log maintenance platforms like Splunk[4], XpoLog [5], Logstash [6]which assist developers in analyzing logs.

Logs are used extensively to help developers in fixing bugs in large software systems . For example, in the JIRA issue HBASE-3403 (commit 1056484), a bug is reported when a module does not exit upon system failure. In order to fix this bug developers record more information in the existing logs. After the bug is fixed, the added information in the logs are used by developers to prevent similar bugs in the system.

Research has shown that logs are used by developers extensively during the development of software systems [7]. Prior research performs a manual study on bug fixing changes. The results show that 41% of the log changes are used to field debugging[25]. However there exists no large scale empirical study to investigate how logs are leveraged during bug fixing.

In this paper we perform an empirical study on the leverage of logs during bug fixes on three open source systems namely Hadoop, HBase and Qpid. In particular we sought to answer following research questions.

RQ1: Are logs leveraged more during bug fixes?

We found that logs are leveraged more frequently during bug fixes than non-bug fixing changes . In particular, adding and modifying logs is statistically significant during bug fixing commits than non-bug fixing commits, with non-trivial effect size. This suggests that developers leverage logs during bug fixes.

RQ2: What types of modifications to logs are more frequent during bug fix?

We manually identified four types of log modifications from our case studies. The 4 types of changes are *'Logging level change', 'Text Modification', 'Variable Modification' and* 'Relocating logs'. We found that *'Text Modification', 'Variable Modification' and* 'Relocating' are statistically significant, with medium to high effect sizes in bug fixing commits. This shows that developers believe that adding more information to logs can help fix bugs.

RQ3: Are logs useful in bug fixes?

We found that logs help in a faster resolution of bugs with less developer involvement. We found that bug fixing commits with log changes, have higher total churn. This implies that logs are leveraged more to fix complex bugs. After controlling the code churn, we found that the issues with log changes takes less time to get resolved, need less developer involvement and have less discussions.

The rest of this paper is organized as follows. Section 2 is the related work in field of logging, Section 3 contains details

of our methodology in gathering and extracting the data for our study. We also present an overview of the statistical techniques we will be using in this paper. Section 4 contains the details case studies and the results we obtained. Section 5 will contain the limitations of our study and any threats to validity. Section 6 will contain further work we intend to do and will conclude the paper.

II. Related Work

In this section, we present the related work of this paper. In particular, we present the prior research that performs log analysis to for large software systems and empirical studies on logs.

A. Log Analysis

Prior work leverage log analysis for testing and detecting anomalies in large scale systems. *Shang et al.* [8] propose an approach to leverage logs in verifying the deployment of Big Data Analytic applications. Their approach analyzes logs in order to find differences between running in a small testing environment and a large field environment. *Lou et al.* [9] propose an approach to leverage variable values printed in logs to detect anomalies in large systems. Based on the variable values in logs, their approach creates invariants (e.g., equations). Any new logs that violates the invariant are considered to be a a sign of anomalies. *Fu et al* [10] built a Finite State Automaton (FSA) using unstructured logs and to detect performance bugs in distributed systems. *Xu et al* [2] link logs to logging statements in source code to recover the text and and the variable parts of log messages. They applied Principal Component Analysis (PCA) to detect system anomalies. Tan et al. [11] propose a tool named SALSA, which constructs state-machines from logs. The state-machines are further used to detect anomalies in distributed computing platforms. Jiang et al. [12] study the leverage of logs in troubleshooting issues from storage systems. They find that logs assist in a faster resolution of issues in storage systems. Beschastnikh et al. [13], [14] designed automated tools that infers execution models from logs. These models can be used by developers to understand the behaviours of concurrent systems. Moreover, the models also assist in verifying the correctness of the system and fixing bugs. To assist in fixing bugs using logs, Yuan *et al*. [15] propose an approach to automatically infer the failure scenarios when a log is printed during a failed run of a system.

Jiang et al. [16], [17], [18], [19] proposed log analysis approaches to assist in automatically verifying results from load tests. Their log analysis approaches first automatically abstract logs into system events [16]. Based on the such events, they identified both functional anomalies [17] and performance degradations [18] in load test results. In addition, they proposed an approach that leverage logs to reduce the load test that are performed in user environment [19].

The extensive prior research of log analysis motivate our paper to study how logs are leveraged during bug fixes. Our findings confirms that logs are widely leveraged during bug fixes and the use of logs assists developers in a faster resolution of logs with fewer people and less discussion involved.

B. Empirical studies on logs

Prior research has performed empirical studied on logs and logging characteristics. Yuan et al. [7] studies the logging characteristics in four open source systems. They find that over 33% of all log changes are after thoughts and logs are changed 1.8 times more than entire code. Fu *et al.* [20] performed an empirical study on where developer put logging statements. They find that logging statements are used for assertion checks, return value checks, exceptions, logic-branching and observing key points. The results of the analysis was evaluated by professionals from the industry and F-score of over 95% was achieved.

Shang et al. [21] signify the fact that there is gap between operators and developers of software systems, especially in the leverage of logs. They performed an empirical study on the evolution both static logging statements and log lines outputted during run time [25], [22]. They find that logs are co-evolving with the software systems. However, logs are often modified by developers without considering the needs of operators. Furthermore, *Shang et al* [23] find that understanding logs is challenging. They examine user mailing lists from three large open-source projects and find that users of these systems have various issues in understanding logs. *Shang et al.* propose to leverage different types of development knowledge, such as issue reports, to assist in understanding logs.

Prior research by *Yuan et al.* [24] shows that logs need to be improved by providing additional information. Their tool named *Log Enhancer* can automatically provides additional control and data flow parameters into logs.

The most related prior research by Shang *et al*. [25] empirically study the relationship of logging practice and code quality. Their manual analysis sheds some lights on the fact that some logs are changed due to field debugging. They also show that there is a strong relationship between logging practice and code quality. Our paper focused on understanding how logs are leveraged during bug fixes. Our results show that logs are leveraged extensively during bug fixes and assist in a quick resolution of bugs.

III. Methodology

In this section, we describe our method for preparing data to answer our research questions.

The aim of this paper is to understand what roles logging statements play in bug fixes. To answer this question, we conduct a case study on three open source projects. Hadoop, Hbase and Qpid have extensive logging and suit our case study and table 1 highlights the overview of the three subject systems.

Hadoop[1]: Hadoop is an open source software framework for distributed storage and distributed processing of big data on clusters. It uses the MapReduce data-processing paradigm

[1]http://hadoop.apache.org/

TABLE I
OVERVIEW OF THE DATA

Projects	Hadoop		Hbase		Qpid	
	Buggy	Non-Buggy	Buggy	Non-Buggy	Buggy	Non-Buggy
# of Rev	7,366	12,300	5,149	7,784	1,824	5,684
Code Churn	4,09K	3.2M	1.4M	2.18M	175k	2.3M
Log Churn	4,311	23,838	4,566	12,005	597	10,238

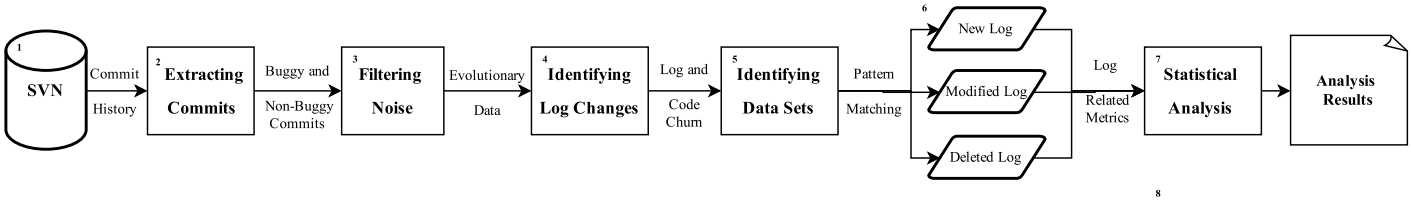

Fig. 1. Overview of our cast study approach

and the debugging process of Hadoop has been studied in prior research [25], [9], [2]. We used Hadoop releases 0.16.0 to 2.0.

HBase[2]: Apache HBase is a distributed, scalable, big data store. We used Hbase releases 0.10 till 0.98.2.RC0. Those release have more than 4 years of development in Hbase from 2010 till 2014.

Qpid[3]: Qpid is an open source messaging system that implements a Advanced Message Queuing Protocol (AMQP). We studied from release 0.10 till release 0.30 of QPID. Those releases are from 2011 to 2014.

We used SVN to collect the commit history and JIRA to collect the issue reports from all the projects in our case study.

1) Data Collection and filtering: We used SVN to study the evolution of Java source code in the three projects. We extract the changes made in each commit and using this data calculate the metrics to answer our research questions. The entire process is broken down into 3 steps and is illustrated in Figure 1.

2) Extracting Commits : The first step in our approach is to extract buggy and non-buggy commits. We extracted a list of all commits from SVN and the commit message from each commit. We extracted a list of all JIRA issues related to bug fixes. As developers mention the JIRA issue numbers in the commit messages, we matched the commit messages against the JIRA issues to identify all the defect fixing changes. If a commit message does not contain a JIRA issue we search search for bug-fixing key words like 'fix' or 'bug'. Prior research has shown that such heuristics can identify bug-fixing commits with a high accuracy.

[2]http://hbase.apache.org/

[3]https://qpid.apache.org

3) Filtering Noise: After separating our data into buggy and non-buggy commits, we calculated the churn for each commit. As commits contain changes to non-Java files, we filtered out the changes to non-Java files from both the datasets.

We found that some commits have a high code churn because of branch and merge operations. To filter out such commits with high code churn(over 50,000), we consider only those commits which have both code addition and deletion. For example - in the commit 952,410 the total churn is over 100,000 and it has no deletion of code. To filter branching commits with churn less than 50,000 we use the change-list file where branching commits fall under 'BRANCH_SYNC' category.

4) Identifying Log changes : To identify the log changes in the datasets, we manually looked at some of the commits to find common patterns in the logging statements. Some of the patterns were specific to a particular project. For example a logging statement from Qpid uses 'QPID_LOG'-
QPID_LOG(error, "Rdma: Cannot accept new connection (Rdma exception): " + e.what());

Some patterns are uniform across projects due to the use of same logging libraries. For example -
LOG.debug("public AsymptoticTestCase(String"+ name +") called")

Using regular expressions to match these patterns, we automated the process of finding all the logging statements in our data sets. For example, *Log4j* is used widely in Hadoop and HBase. In both projects, logging statements have method invocation like "LOG", followed by log-level and other information. We count every such invocation as a logging statement.

A. Identifying log modifications.

After identifying the logging statements in each commit, we found two types of log changes.

Added Log: This type includes all log lines added in a commit.

Deleted Log: This type includes all log lines deleted in a commit.

Since SVN *diff* does not provide a built in feature to track modification to a file line by line, modifications to logging statements are shown as added and deleted logging statements. To track these modifications we used levenshtein measures [1] . We consider a pair of added and deleted logging statement to be a log modification, if the levenshtein distance of lesser than 5 or ratio of higher than 0.5. For example, the logging statements show below have levenshtein distance of 16 and ration of 0.86 when we compare both the logging statements entirely. Hence this is categorized as a log modification.

+ LOG.debug("Call: " + method.getName() + " took " + callTime + "ms");
- LOG.debug("Call: " + method.getName() + " " + callTime);

After identifying log modifications we obtained three new data sets namely:

1) Modified Logs: This includes all the modified logging statements in a commit.
2) New Logs: This includes all those logs which were newly added in a commit. To obtain this we removed all the added logs from the modified logs.
3) Removed Logs: This includes all those logs which were deleted in a commit. Similar to new logs we removed all the deleted logs from the modified logs.

We use this data to answer the three RQ's in the next section.

IV. Study Results

In this section we present our study results by answering our research questions. For each question, we discuss the motivation behind it, the approach to answering it and finally the results obtained.

RQ1: Are logs leveraged more during bug fixes?

Motivation: Prior research has shown that logs are used in the by fixing process. During debugging, developers update log statements also, so future occurrences of a similar bug can be resolved easily with the updated information in the log statements. However, there has been no large scale empirical study to show how extensively logs are leveraged in debugging.

Approach: We try to find if there is a difference between bug fixing and non-bug fixing commits with respect to log churn. To do this, we used the data sets obtained in previous section i.e Modified, New and Removed logs, and we calculated code churn for each data set. We used 'Total code churn' of a revision to control the other metrics. The 3 new metrics are:

$$Modified\,Churn = Modified\,Log/Total\,Churn$$
$$New\,Churn = New\,Log/Total\,Churn$$
$$Removed\,Churn = Removed\,Log/Total\,Churn$$

To determine if this is a statistically significant difference for these metrics, in buggy and non-buggy commits we use the *MannWhitney U test* (Wilcoxon rank-sum test)[2]. *Wilcoxon test* gives p-value as the result. A p-value of ≤ 0.05 means that the test difference between the two data sets is statistically significant and we may reject the null hypothesis (i.e., the two populations are the same). By rejecting the null hypothesis, we can accept the alternative hypothesis, which tells us if one population is statistically significantly larger than the other. We use *Wilcoxon test* because we our metrics are highly skewed and as it is a non-parametric test, the distribution of population does not factor.

We also use *effect sizes* to measure how big is the difference in log changes (new logs, modified logs, removed logs) between the bug fixing and non-bug fixing commits. Unlike *Wilcoxon-test*, which only tells us if the differences of the mean between two distributions are statistically significant, effect sizes quantify the difference between two distributions. Researchers have shown that reporting only the statistical significance may lead to erroneous results (i.e., if the sample size is very large, p-value can be small even if the difference is trivial). We use *Cohen's d* to quantify the effects [**?**]. *Cohen's d* measures the effect size statistically, and has been used in prior engineering studies. *Cohen's d* is defined as:

$$Cohen's\ d = \frac{\bar{x}_1 - \bar{x}_2}{s}, \tag{1}$$

where $\bar{x}_1$ and $\bar{x}_2$ are the mean of two populations, and s is the pooled standard deviation [**?**]. As software re-engineering has different thresholds for *Cohen's d* [26], the new scale is shown below.

$$Effect\,Size = \begin{cases} 0.16 < & Trivial \\ 0.16 - 0.6 & Small \\ 0.6 - 1.4 & Medium \\ 1.4 > & Large \end{cases}$$

Results: **Logs are modified more in bug fixing commits than non-bug fixing commits**. Table 2 shows that modified log churn is statistically significant for all subject systems and has non-trivial effect sizes in Qpid and Hadoop. This is because modifying existing log to provide additional information is more efficient than writing new logs. Research has shown that 36% of log messages are modified at-least once as after-thoughts [7]. This means that in most cases the existing logging messages do not convey all the information necessary for a bug fix. An example of this type of change is shown below:

+ log.trace("setConnectionURL(" + Util.maskUrlForLog (connectionURL) ")");
- log.trace("setConnectionURL(" + connectionURL + ")");

Developers add new logs more during bug fixes. From table 2, new log churn is statistically significant in all subject

[1]http://xlinux.nist.gov/dads/HTML/Levenshtein.html

[2]http://www.ime.unicamp.br/ dias/Ch10.wilcoxon.pdf

TABLE II
P VALUES AND EFFECT SIZE OF FOR COMPARISON . A POSITIVE EFFECT SIZE MEANS BUG FIXING COMMITS ARE LARGER. P-VALUES ARE BOLD IF IT IS LESS THAN THRESHOLD LIMIT OF 0.05

Projects	Hadoop		Hbase		Qpid	
Metrics	P-Values	Effect Size	P-Values	Effect Size	P-Values	Effect Size
Modified Log Churn	**2.88e-4**	0.167(small)	**0.0353**	0.0886	**0.0281**	0.329(small)
New Log Churn	**0.00202**	0.0078	**0.00353**	0.134	**0.0032**	0.234(small)
Deleted Log Churn	0.087	-0.0455	**0.00489**	0.120	**0.00952**	0.042

systems but only Qpid has non-trivial effect size. This implies in some cases developers need to add more logging statements in some places in the source code. For new projects like Qpid some important source code is not well logged. Therefore developers need to add logging statements. For mature projects such as Hadoop and Hbase, source code is well logged so, developers focus more effort on improving existing logging statements rather than adding new logging statements.

Developers do not delete logs during bug fixes. We observed that deleted log churn is statistically significant in only Hbase and Qpid and all subject systems have trivial effect sizes. This implies that in both bug fixing and non-bug fixing commits, developers do not remove logging statements. Prior research has shown that developers only delete log lines only when confident about their source code [**?**]. In Hadoop we observe that the effect size is negative which implies logging statements are removed more from non-bug fixing commits than bug fixing commits.

> *Developers log more during bug fixes than other types of changes. In particular logs are modified more often during the debugging process than other activities. This implies that, logs are leveraged during bug fixes*

RQ2: What types of modifications to logs are more frequent during bug fix ?

Motivation*:* From RQ 1 we found that logs are modified more during bug fixes. In this RQ, we want to know how logs are leveraged during bug fixes, in particular the different types of modifications to logs.

Approach*:* We performed a manual analysis on the modified logging statements to identify the different types of log modifications. We first collected all the commits which had logging statement changes in our projects. We selected a 5% random sample from all the commits with logging statement changes. We followed a iterative process [**?**] to identify the different types of logging modifications, that developers make in the source code till we cannot find any new types of modifications. We identified 4 different types of log modifications and their distribution is shown in table 3. The four types of changes are described below:

Relocating: In this only white spaces are changed or the logging statement is kept intact but moved to a different place in the file.

TABLE III
DISTRIBUTION OF LOG MODIFICATIONS

Projects	Hadoop (%)	Hbase (%)	Qpid (%)
Relocating	82.6	61.4	55.8
Text Modification	7.85	12.1	18
Variable Modification	7.9	8.4	12.5
Logging Level Change	3.85	5.4	13.6

Text Modification: In this type, the text printed from the log is modified.

> *- Logger.warn(" Sample Text Goes Here " + printAVariable);*
> *+ Logger.warn(" New Sample Text Goes Here " + printAVariable);*

Variable Modification : In this type, variable part of log is modified , keeping the text constant(if both variable and text changes in logs, we consider them as new logs).

> *- Logger.warn(" Sample Text Goes Here " + printAVariable);*
> *+Logger.warn(" Sample Text Goes Here " + printAVariable + printBVariale);*

Logging Level Change : In this type only change is the log level change.

> *- Logger.warn(" Sample Text Goes Here " + printAVariable);*
> *+ Logger.debug(" Sample Text Goes Here " + printAVariable);*

We created an automated tool to label log modifications into the four categories. For every commit we calculated the number of log modifications in each commit and used *total churn* as the controlling measure. The four metrics

After categorization, we found the churn for these categories and used *total churn* as the controlling measure. The 4 metrics obtained are- (1) Relocating log Churn, (2) Text Modification Churn, (3) Variable Modification Churn and (4) Logging Level Churn.

After finding the churn in each of these categories we tried to find how significant are these compared to the non bug fixing commits. We consider only those commits which have log churn to do this comparisons. The total churn is used as

TABLE IV
P-VALUES AND EFFECT SIZE FOR LOG MODIFICATIONS

Projects	Hadoop		Hbase		Qpid	
Metrics	P-values	Effect Size	P-values	Effect Size	P-values	Effect Size
Relocating	**1.69e-11**	0.260(small)	**6.33e-03**	0.2092(small)	**9.14e-08**	0.987(med)
Text Modificaton	**7.75e-04**	0.153(small)	**2.94e-05**	0.308(small)	**4.68e-08**	0.531(small)
Variable Modification	**1.94e-06**	0.447(small)	**3.51e-04**	0.614(med)	**5.19e-05**	1.209(med)
Log LevelChange	**0.0057**	0.412	0.153	-0.05	0.341	0.396

a controlling measure. We used *Wilcoxon* test and *Cohens.d*, to measure the significance of bug fixing commits.

Results: **We found that developers modify variables more in bug fixes.** From table 3, we observe that variable modification is statistically significant in all the subject systems and has effect size in medium range. This implies that developers modify parameters in their logs, to provide has the highest effect size among all the types of log changes and is statistically significant in all 3 projects. This implies that developers modify the parameters in their logs,to provide more information about the system. This is useful because if any other bugs originate in that module, the additional data can help in easy resolution. It also means that developers do not provide all the details the first time logs are written and update them as after-thoughts.

We found that Text Modification is more in bug fixes. From table 3, we see that textual modification is significant in all the projects in our case study. This implies that developers value more in providing contextual data to an existing log rather than writing new logging statements.

We found log relocation is usually done when the same log statements are moved/copied to multiple places as part of code change. We looked at over 100 commits across all three projects to understand this behavior. We observed that in majority of the cases, developers use logging statements before try, catch blocks or do not check all the conditions. When a bug is raised the logs help in fixing the issue, but along with the fix they place the original logging statement in a try catch block and handle further exceptions properly.For example in 792,522 revision of Hadoop, we see that the log messages are not placed in the appropriate 'try', 'catch' blocks. We also see that in some parts of the code the exceptions are not even present and later added by developers. In this commit the same logging message is copied in multiple places and hence its falls under relocation category.

From table 2, we see that this form of changes a major bulk of logging changes that occur in both data sets. This shows us that developers when writing code omit safe coding practices and do not use try, catch and finally blocks in their code.

Another example of this 1,042,282 revision in which a log statement is moved from the beginning of code block to the end of block. From our manual study (section 5) we found that this is mostly done to improve the readability or increase the performance. For example in, HBASE-4288 the logs are moved into new blocks, so they are printed only when 'Trace/Debug' option is enabled.

We observe that log level changes has P-values less than 0.05 only for Hadoop project. We also see that in table 2, logging level changes are the least among all the types of logging changes. This implies developers do not leverage log levels during bug fixes. This can be because -
1. Developers do not know what logging level is correct
2. During bug fixes they enable all levels i.e Debug and Trace, so they do not know which levels are incorrect.

> *Variable modification is more significant when compared to other types of log modifications*
> ***Implication**: Developers believe adding more parameters/variables to logging statements, helps in the debugging process. This shows that logging statements evolve continually and can change multiple times across revisions. It also means developers can spend more time or make use of the logging tools present to log more in first commit so subsequent changes are not necessary and debugging is faster.*

RQ3: Are logs useful in bug fixes?

Motivation: In our previous research questions we found that logs are modified more frequently in bug fixes. However, we cannot come to any inference about the usefulness of logs in bug fixes. In this RQ we try to understand the usefulness of leveraging logs during bug fixes.

Approach: To find the usefulness of logs in bug fixes we identified all JIRA issues of type 'bug' from the three subject systems. We obtained the code commits for each of these JIRA issues and identified log churn for each commit. We then split the JIRA issues into (1) bugs fixed with log changes (2) bugs fixed without log changes. We calculate the total code churn for each of the JIRA issues. We use the total code churn to measure the complexity of the issue. We then compare the total code churn of bug fixes with log changes, against bug fixes without log changes. We use *Wilcoxon* test to find the statistical significance and *Cohens.d* to measure the size of the difference.

We then parsed the JIRA issue files for each commit and extracted three metrics namely -

TABLE V
P-VALUES AND EFFECT SIZE FOR BUG FIXES

Projects	Hadoop		Hbase		Qpid	
Metrics	P -values	Effect Size	P -values	Effect Size	P -values	Effect Size
Total Churn	**2.2e-16**	0.563(small)	**2.2e-16**	0.093	**3.15e-08**	0.270(small)
Resolution Time	**4.26e-03**	-0.145(small)	**7.44e-14**	-0.167(small)	0.0865	-0.119
# of Comments	**2.2e-16**	-0.507(small)	**5.16e-11**	-0.289(small)	**2.34e-03**	-0.227(small)
# of Developers	**2.2e-16**	-0.577(small)	**2.2e-16**	-0.538(small)	**4.73e-02**	-0.375(small)

1) **Resolution Time:** This is the time taken from the time a bug is opened till its resolved. For example in JIRA if bug was reported in 1^{st} Feb 2015 and closed in 5^{th} Feb 2015 it means the time taken to fix the bug was 4 days. We used to measure how long it takes for a bug to be fixed.
2) **Number of Comments:** This metric gives the total number of comments are present in a JIRA issue. We obtained this by finding the number of comment id's present in the JIRA XML file. We used this measure how much effort is needed to discuss on fixing a bug.
3) **Number of Developers:** Every task in JIRA is generally assigned to particular developer and there number of viewers who are interested in the issue and help in resolving it. This metric gives the number of such unique developers who commented on the JIRA issue posts and were involved in resolving it. We obtained this by finding all the unique author names in the XML file. We used this as a metric as it shows human effort needed. More number of developers involved and commenting on a issue list, signifies more human effort spent resolving the bug.

We controlled the metrics by total code churn. We used *Wilcoxon* test to measure the statistical significance of each metric in bugs fixed with log changes with bugs fixed without log changes. *Cohens.D* is used to measure the size of the difference of each metric.

Results: **We found that the logs are used to fix more complex bugs**. From table 4 we see that average code churn per commit is significantly higher for commits with log changes and has non-trivial effect sizes. This implies that when dealing with more complex bugs, developers may leverage logs more.

We found that bugs are easier to fix with logs. After controlling the churn, we found that given two bugs of same complexity the one with log changes takes lesser time to get resolved and needs lesser number of developers involved in the fix with less discussion. This implies that logs provide useful information for developers to discuss, diagnose and fix bugs easily. For example, logs are used in fixing issue HBASE-3074 (commit 1,005,714). In this JIRA issue we see the very first comment is to provide additional details in the logging message about where the connection manager fails. When we looked at the commit, we see that the developers add the name of the existing server which has gone stale in the logging statements. This additional data helps trace the cause of the failure and helps in the debugging process.

TABLE VI
OVERVIEW OF THE DATA SETS WITH CODE CHURN

Projects	Metrics	With Log Change	Without Log Change
Hadoop	Average Churn	371	41
	Average Time	62 days	48 days
	Average comments	20	16
	Average users	7	6
HBase	Average Churn	692	295
	Average Time	36 days	26 days
	Average comments	27	19
	Average users	6	6
QPid	Average Churn	1,193	145
	Average Time	96 days	64 days
	Average comments	5	4
	Average users	3	2

Logs are leveraged during complex bugs and help in quicker resolution. Developers leverage logging statements to fix complex bugs. Bug fixes with log changes are resolved quicker with fewer people and less discussions. This implies provide useful information to fix bugs.

V. MANUAL STUDY ON LEVERAGING LOGS BUG FIXES

From our research question we found that logs help in reducing the time and human effort necessary during the debugging process. We performed a manual analysis to find out where and in what scenarios logs are used. We first collected all the bug fixing commits which had logging statement changes in our projects. We then selected a random sample from all the issue reports (180 samples) with confidence level 95% and interval of 5%.

Q1. What often are logs leveraged in the debugging process ?

We found that 70.7% of the sample used logs directly for fixing the buggy issues. We looked into the JIRA discussion posts and we observer that over 128 of the reports, made use of log dumps, master log records or the developers in their comments mentioned logs to trace the problem. We observed that developers use logging statements to trace the bugs and even use the exceptions generated (example -QPID-2979). z‘

In the remaining 30% of the issue reports developers make use of exceptions (example HBASE-3654) or reproduce the issue (example QPID-4312) and fix them.

Q2. What type of information do developers look in logs ?

61% of the reports (78) made use of both variable (dynamic objects) and textual part present in the logging statements to resolve the bugs. We observed that developers output the system state, the server/connection name, time, logging level, and even object names in the log lines. We found that these details are generally sought after by developers more during debugging process. For example - In HBASE-4797, developers provide a system generated log dump on JIRA. We observe that developers draw many conclusions using the logs present and this helps in resolving the bug. In this particular example, developers used the time-stamp, region-servers and the sequence-id's present in the logs. We also observed that these commits were associated with higher code churn implying, developers need more information to fix more complex problems.

39 % of the reports used only the log message themselves to diagnose the problem. We observed that majority of these bugs (38 commits), were caused by the logs themselves. These issues were either trivial from typos (example - commit 1333321), haphazard logs (example - commit 1333321) to complex issues degrading performance (example - commit 999046).

***Findings**: 70% of the issues leverage logs during bug fixes, out of which 61% of the issues leverage both textual and variable parameters in a logging statement.*

VI. Limitations and Threats to Validity

External Validity: In this study we found that logs are leveraged during bug fixes more than any other type of changes. We looked at three projects from the Apache Software Foundation namely Hadoop, HBase and Qpid. This is also limiting factor as these projects are all Java based and we have not looked into other programming languages.

Internal Validity: In our initial research questions, we tried to find if there is relation which proves logs getting changed implies a high probability of bug fix or not. This was shown to be true and we deduced that logs are in-fact changed more often during bug fixes than other types of changes. But there can be instances where logs are used in bug fix, but they are not changed as they provide sufficient information. In such cases its impossible to determine if logs were useful or not.

Although our study shows presence of logs reduces time of resolution, number of developers and number of discussions we do not claim any causal relationship. There are many other contributing factors and further analysis is necessary.

Construct Validity: When correcting the data, we found that some of the revisions were only related to branching or merging . To eliminate this we looked at all SVN commits and removed branching commits from our data set. We also used keywords to find branching and merging commits. Such keyword based filtering may not be entirely correct. Although prior research uses similar approach, better ways to filter branching and merging commits are needed.

The other major limitation is user defined logs in the Java Code. When searching for log lines we looked at specific patters in the files. These patters were 'Log', 'Logger', 'LOGGER', 'LOG', 'log' and few other variants. But the users can define their own names for these statements and it would make it impossible to search for such user defined functions in the change commits.

VII. Conclusion and Future Work

In this work we looked at Apache Hadoop, Hbase and Qpid to find out if logs are more useful in bug fixes or not.We first try to establish how we are going to measure this by checking for log modifications that are part of a bug fixing commit. In our initial research questions we find that logs are modified and added during bug fixes more than other types of developmental tasks

We identified 4 different types of logging changes namly 'Text Modification', 'Variable Modification', 'Restructuring' and 'Logging Level Changes'. We saw that 'Variable Modification had the highest effect size in all projects, followed by 'Restructuring' and 'Text Modification'. This showed that developers tend to add or delete or change variables more often during bug fixes. We also saw that developers tend to change the position of logging statements or change the text in logs more frequently during bug fixes.

Next, we studied the bug fixes with log changes and found that 1) Bug fixes with log changes take more time to get resolved, 2) Bug fixes with log changes have more comments and users in the discussion. We found that given 2 bugs of same complexity, the one without log changes takes longer time to be fixed, needs more number of developers to be involved in the fix and has long discussions post also.

Finally we did a manual analysis were we found developers use logging statements to understand the rationale behind logs. We found over 70.7 % of the cases directly used logs in the debugging process. We found that over 61 % of the time logs are used to understand the rationale behind bugs and find which specific conditions triggers bugs.

This proves that logs are in-fact helpful in bug fixes and are useful for quick resolution of bugs with less involvement from developers. It shows that if logs added during bug fixes the resolution time and development can reduce during subsequent bug fixes and help other developers.

References

[1] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Online system problem detection by mining patterns of console logs," in *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, ser. ICDM '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 588–597. [Online]. Available: http://dx.doi.org/10.1109/ICDM.2009.19

[2] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, ser. SOSP '09. New York, NY, USA: ACM, 2009, pp. 117–132. [Online]. Available: http://doi.acm.org/10.1145/1629575.1629587

[3] M. Syer, Z. M. Jiang, M. Nagappan, A. Hassan, M. Nasser, and P. Flora, "Leveraging performance counters and execution logs to diagnose memory-related performance issues," in *Proceedings of ICSM 2013: the 29th IEEE International Conference on Software Maintenance , Year = 2013, Month = Sept, Pages = 110-119, Doi = 10.1109/ICSM.2013.22, ISSN = 1063-6773, Keywords = program diagnostics;program testing;public domain software;software performance evaluation;execution logs;large-scale enterprise system;load tests;memory-related performance issues;open-source system;performance counters;software diagnosis;software systems;Couplings;Lifting equipment;Memory management;Radiation detectors;Standards;Transient analysis;Visualization;Execution Logs;Load Testing;Performance Counters;Performance Engineering*.

[4] L. Bitincka, A. Ganapathi, S. Sorkin, and S. Zhang, "Optimizing data analysis with a semi-structured time series database," in *Proceedings of the 2010 Workshop on Managing Systems via Log Analysis and Machine Learning Techniques*, ser. SLAML'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 7–7. [Online]. Available: http://dl.acm.org/citation.cfm?id=1928991.1929002

[5] Xpolog, "http://www.xpolog.com/."

[6] logstash, "http://logstash.net."

[7] D. Yuan, S. Park, and Y. Zhou, "Characterizing logging practices in open-source software," in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 102–112. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337223.2337236

[8] W. Shang, Z. M. Jiang, H. Hemmati, B. Adams, A. E. Hassan, and P. Martin, "Assisting developers of big data analytics applications when deploying on hadoop clouds," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 402–411. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486842

[9] J.-G.-L. Q.-F.-S. Y. Y-Xu and J-Li., "Mining invariants from console logs for system problem detection." in *In Proc. of 2010 USENIX Annual Technical Conference(ATC 10)*., 2010.

[10] Q. F. J. L. Y. Wang and J. Li., "Execution anomaly detection in distributed systems through unstructured log analysis." in *In Proc. of 9th IEEE International Conference on Data Mining (ICDM 09)*, 2009.

[11] J. Tan, X. Pan, S. Kavulya, R. Gandhi, and P. Narasimhan, "Salsa: Analyzing logs as state machines," in *Proceedings of the First USENIX Conference on Analysis of System Logs*, ser. WASL'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 6–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855886.1855892

[12] W. Jiang, C. Hu, S. Pasupathy, A. Kanevsky, Z. Li, and Y. Zhou, "Understanding customer problem troubleshooting from storage system logs," in *Proccedings of the 7th Conference on File and Storage Technologies*, ser. FAST '09. Berkeley, CA, USA: USENIX Association, 2009, pp. 43–56. [Online]. Available: http://dl.acm.org/citation.cfm?id=1525908.1525912

[13] I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M. D. Ernst, "Leveraging existing instrumentation to automatically infer invariant-constrained models," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. New York, NY, USA: ACM, 2011, pp. 267–277. [Online]. Available: http://doi.acm.org/10.1145/2025113.2025151

[14] I. Beschastnikh, Y. Brun, M. D. Ernst, and A. Krishnamurthy, "Inferring models of concurrent systems from logs of their behavior with csight," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 468–479. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568246

[15] D. Yuan, H. Mai, W. Xiong, L. Tan, Y. Zhou, and S. Pasupathy, "Sherlog: Error diagnosis by connecting clues from run-time logs," in *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XV. New York, NY, USA: ACM, 2010, pp. 143–154. [Online]. Available: http://doi.acm.org/10.1145/1736020.1736038

[16] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "An automated approach for abstracting execution logs to execution events," *J. Softw. Maint. Evol.*, vol. 20, no. 4, pp. 249–267, Jul. 2008. [Online]. Available: http://dx.doi.org/10.1002/smr.v20:4

[17] Z. M. Jiang, A. Hassan, G. Hamann, and P. Flora, "Automatic identification of load testing problems," in *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, Sept 2008, pp. 307–316.

[18] ——, "Automated performance analysis of load tests," in *In Proceedings of ICSM 2009: the 2009 IEEE International Conference on Software Maintenance*, Sept 2009, pp. 125–134.

[19] Z. M. Jiang, A. Avritzer, E. Shihab, A. E. Hassan, and P. Flora, "An industrial case study on speeding up user acceptance testing by mining execution logs," in *Proceedings of the 2010 Fourth International Conference on Secure Software Integration and Reliability Improvement*, ser. SSIRI '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 131–140. [Online]. Available: http://dx.doi.org/10.1109/SSIRI.2010.15

[20] Q. Fu, J. Zhu, W. Hu, J.-G. L. R. Ding, Q. Lin, D. Zhang, and T. Xie, "Where do developers log? an empirical study on logging practices in industry," in *ICSE Companion 2014 Companion Proceedings of the 36th International Conference on Software Engineering*. ACM New York, NY, USA 2014 ISBN: 978-1-4503-2768-8 doi¿10.1145/2591062.2591175, 2014-05-31 2014, pp. Pages 24–33.

[21] W. Shang, "Bridging the divide between software developers and operators using logs," in *Software Engineering (ICSE), 2012 34th International Conference on*.

[22] W. Shang, Z. M. Jiang, B. Adams, A. E. Hassan, M. W. Godfrey, M. Nasser, and P. Flora, "An exploratory study of the evolution of communicated information about the execution of large software systems," *Journal of Software: Evolution and Process*, vol. 26, no. 1, pp. 3–26, 2014. [Online]. Available: http://dx.doi.org/10.1002/smr.1579

[23] U. L. L. U. D. Knowledge, "Weiyi shang, meiyappan nagappan, ahmed e. hassan, zhen ming jiang," in *The 30th International Conference on Software Maintenance and Evolution (ICSME 2014)*, 2014.

[24] D. Yuan, J. Zheng, S. Park, Y. Zhou, and S. Savage, "Improving software diagnosability via log enhancement," *ACM Trans. Comput. Syst.*, vol. 30, no. 1, pp. 4:1–4:28, Feb. 2012. [Online]. Available: http://doi.acm.org/10.1145/2110356.2110360

[25] W. Shang, M. Nagappan, and A. Hassan, "Studying the relationship between logging characteristics and the code quality of platform software," *Empirical Software Engineering*, vol. 20, no. 1, pp. 1–27, 2015. [Online]. Available: http://dx.doi.org/10.1007/s10664-013-9274-8

[26] V. B. Kampenes, T. Dybå, J. E. Hannay, and D. I. K. Sjøberg, "A systematic review of effect size in software engineering experiments," *Information and Software Technology*, vol. 49, no. 11-12, pp. 1073–1086, Nov 2007.