# Revisiting Assert Use in GitHub Projects

Pavneet Singh Kochhar
School of Information Systems
Singapore Management University, Singapore
kochharps.2012@smu.edu.sg

David Lo
School of Information Systems
Singapore Management University, Singapore
davidlo@smu.edu.sg

## ABSTRACT

Assertions are often used to test the assumptions that developers have about a program. An assertion contains a boolean expression which developers believe to be true at a particular program point. It throws an error if the expression is not satisfied, which helps developers to detect and correct bugs. Since assertions make developer assumptions explicit, assertions are also believed to improve understandability of code. Recently, Casalnuovo et al. analyse C and C++ programs to understand the relationship between assertion usage and defect occurrence. Their results show that asserts have a small effect on reducing the density of bugs and developers often add asserts to methods they have prior knowledge of and larger ownership. In this study, we perform a partial replication of the above study on a large dataset of Java projects from GitHub (185 projects, 20 million LOC, 4 million commits, 0.2 million files and 1 million methods). We collect metrics such as number of asserts, number of defects, number of developers and number of lines changed to a method, and examine the relationship between asserts and defect occurrence. We also analyse relationship between developer experience and ownership and the number of asserts. Furthermore, we perform a study of what are different types of asserts added and why they are added by developers. We find that asserts have a small yet significant relationship with defect occurrence and developers who have added asserts to methods often have higher ownership of and experience with the methods than developers who did not add asserts.

## CCS CONCEPTS

**•Theory of computation →Assertions;**

## KEYWORDS

Assertions, Replication Study, GitHub

## 1 INTRODUCTION

An assertion is a statement that helps developers test an assumption that they have about a piece of code. An assertion contains a boolean expression which needs to be satisfied for the execution of the subsequent statements; otherwise, an `AssertionError` would be raised. Assertions support design by contract (DbC) style of programming, where a developer defines what a method is supposed to do and verifies it during actual execution [37]. Asserts can be used to enforce preconditions, postconditions and invariants to check the change in the state caused by a piece of code. Often, assertions provide an effective way to detect and correct bugs early on in the development life cycle as an `AssertionError` can help developers in quickly finding the source and reason of an error. An assert statement can be viewed as a unit test which is directly embedded in the code and tests the program on real data. Thus, assertions complement unit and integration tests. Assertions also serve as documentation to help developers understand the working of a program, thus, improving readability and maintainability. Many popular languages such as C, C++, and Java provide support for assertions.

Recently, Casalnuovo et al. study 69 most popular C and C++ projects on GitHub [12]. They investigate several questions such as relationship between asserts and defect occurrence, how assertion usage relates to developer characteristics such as code ownership and experience, etc. They use hurdle regression model and other statistical methods such as the Mann-Whitney-Wilcoxon test to understand the relationship between assertions and defect count. They find that assertions have a negative small but significant relationship with defect occurrence and adding asserts to methods with more developers can reduce the chance of new bugs. They also find that asserts are made to a method by developers who have larger ownership of and more experience with the method.

Casalnuovo et al.'s study provides interesting results; however, the study investigates only projects written in C and C++ and results may not generalize to other popular languages like Java. According to TIOBE programming community index [1], which takes into consideration factors such as number of skilled engineers world-wide, courses and third party vendors, in September 2016, Java is the most popular language. Additionally, Bissyande et al. find that on GitHub there are many more projects written in Java than any other languages [9]. Moreover, another ranking done by IEEE Spectrum which uses 12 metrics from 10 data sources such as GitHub, Stack Overflow, Twitter, Reddit, Hacker News etc. also finds that Java is the most popular language [13]. This motivates us to analyse how developers in open-source Java projects use asserts and their relationships with defect occurrence.

To understand the usage of asserts in Java projects, we perform a partial replication of study by Casalnuovo et al. [12]. We replicate the first two research questions from the previous study and examine a new research question. We collect a larger dataset of 185 Apache Java projects hosted on GitHub. We use Apache projects as they have been extensively studied in the past owing to their high quality [5, 42, 45]. We write a parser to collect all the methods changed over the history of the project by analyzing *git* logs and gather statistics such as number of asserts added, lines changed and number of developers of each method. Similar to Casalnuovo

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*EASE'17, Karlskrona, Sweden*
© 2017 ACM. 978-1-4503-4804-1/17/06…$15.00
DOI: http://dx.doi.org/10.1145/3084226.3084259

et al. [46], we analyze the commit logs to find occurrences of bug fixing commits. We build regression models and employ statistical tests to verify several hypotheses, e.g., asserts impact defect occurrence. Moreover, we examine the relationships between number of asserts added to methods and developer ownership and experience. Furthermore, we perform an in-depth analysis on the types of asserts added by developers.

We examine the following research questions:

*RQ1: How does assertion use relate to defect occurrence?*

*RQ2: How does assertion use relate to developer characteristics such as code ownership and experience?*

*RQ3: How are asserts used by developers?*

The contributions of our work are as follows:

(1) We perform a large-scale replication study to analyze the impact of usage of asserts on defect occurrence in Java projects.
(2) We examine the impact of ownership and experience of developers on assertion usage.
(3) We perform an in-depth qualitative study on 575 distinct methods, each containing at least one assert statement, which are randomly extracted from 185 Apache projects, to understand assert usage patterns.
(4) We release our dataset publicly to allow other researchers to replicate our study and perform complementary studies to find interesting characteristics of assertion usage.

The structure of the remainder of the paper is as follows. In Section 2, we briefly describe the original study and its results. In Section 3 and Section 4, we explain methodology for data analysis and findings of three research questions, respectively. We discuss implications of our study and threats to validity in Section 5, related work in Section 6, and conclusion and future work in Section 7.

## 2 SUMMARY OF THE ORIGINAL STUDY

In this section, we provide a brief overview of the original study design and results [12]. We encourage interested readers to read the original study for more details. The original study analyzed 69 C and C++ projects from GitHub sorted by popularity, to understand the impact of asserts on defect occurrence. The authors use `git log` to collect several metrics at the method level such as number of added lines of code, number of developers who have changed a method, number of bug fix commits associated with a method, and number of asserts added over the lifetime of a method.

To identify asserts added, the authors use the keyword `assert` and to find bug fix commits, they use several keywords such as 'error', 'defect', 'flaw', 'bug', 'fix', 'issue', etc. The authors analyze the relationship between adding asserts and ownership and experience of developers with respect to the method. The authors also build call graphs of 18 C applications to understand the role of methods in the overall system and what network positions are associated with assertions in a call-graph network. Furthermore, the authors categorize projects into different domains such as applications, database, framework, library, code analyzer and middleware. The authors use hurdle regression models [11] to understand the impact of presence/absence of number of asserts, added lines of code and, number of developers on the outcome variable, i.e., number of defects. To assess the significance of relationships, the previous study uses Wilcoxon test and computes Cohen's d for effect size.

We briefly describe the questions investigated by Casalnuovo et al. and their results:

*RQ1: How does assertion use relate to defect occurrence?*

The results show that the asserts have a negative and significant relationship with defect occurrence. Adding assert has a significant influence on bugs and the effect of the first assert is more than that of the subsequent ones.

*RQ2: How does assertion use relate to the collaborative/human aspects of software engineering, such as ownership and experience?*

Developers who have added asserts have higher ownership and experience of methods.

*RQ3: What aspects of network position of a method in a call-graph are associated with assertion placement?*

The results show that methods with asserts have higher hub score than those without. However, no conclusive results were found for other network measures such as authority, in-degree, out-degree and betweeness.

*RQ4: Does the domain of application of a project relate to assertion use?*

The authors find that the application domain has no impact on the number of assertions added.

**Our Study:** We re-investigate the first two research questions (RQ1 and RQ2) on a large dataset of Java projects. We do not examine RQ3 and RQ4 as the previous study shows that the results either show no correlation or are inconclusive. Instead, we perform an in-depth qualitative study on the usage of asserts by Java developers. In the following sections, we explain the methodology and findings of our study.

## 3 METHODOLOGY

In this section, we first present our study subjects in Section 3.1. We then present our data collection procedure in Section 3.2, and statistical methods we use to analyze the data in Section 3.3.

### 3.1 Study Subjects

To understand the relationship between assert usage and bug occurrence, we first collect all Apache Java projects hosted on GitHub. Since GitHub contains many toy projects (c.f. [29]), we focus on a subset of projects hosted in GitHub which are owned by Apache Foundation. Apache projects are often of high quality and have been used in many past studies [5, 42, 45]. This gives us an initial set 342 projects. Following Casalnuovo et al. [12], we filter out projects which have added less than ten asserts over the lifetime of the project. In the end, we get 185 projects which include several popular projects such as Apache Hadoop, HttpClient, Maven, etc. These projects span various categories such as data processing, web utility, build management, database, etc. and have thousands of developers spread around the world contributing to them. Table 1 shows the statistics of subjects we analyze.

### 3.2 Data Processing

***Retrieving and pre-processing project evolution history:*** For all the projects in our dataset, we collect the full history of all non-merge commits along with the commit logs, author data, commit

**Table 1: Study Subjects.**

| | | |
|---|---|---|
| Project Details | #Projects | 185 |
| | #Authors | 2,791 |
| | KLOC | 20,033 |
| | #Files | 201,600 |
| | #Methods | 1,993,828 |
| | #Assert Methods | 30,253 |
| | Period | 12/98 - 04/16 |
| #All Commits | Total | 4,852,069 |
| | With Asserts | 7,540 |
| #Bugfix Commits | Total | 29,867 |
| | With Asserts | 741 |

dates and patches, which show all the changes made to files within that commit. We use the command `git log -U1 -w`, where `-U1` specifies `git` to download the commit patches for the commit and `-w` gives the method names for which code has been added. We use the keyword 'test' to filter out test files, as assertions used in test cases are different from the ones used in source code. Similar to Casalnuovo et al. [12], we use error related keywords: 'error', 'defect', 'flaw', 'bug', 'fix', 'issue', 'mistake', 'fault', and 'incorrect' to identify bug fix commits. We check the commit messages for the presence of above keywords to filter out bug fix commits from non bug fix ones.

***Computing relevant statistics:*** We run CLOC[1] tool on the latest version of the project to count the total lines of code. CLOC gives information about the total number of lines of source code, blank lines and comment lines of files in a given repository. Furthermore, it also counts the number of files in a project for each programming language used by that project. To identify usages of asserts, following Casalnuovo et al. [12], we develop a parser which searches for "assert" keyword in the commit patches. Our parser parses the added and deleted lines (excluding comment lines) to count the number of asserts added and deleted for each commit. We sum all the asserts added to each method on a per commit basis. Similarly, we count the total number of lines added to a method over its lifetime. Furthermore, we collect the total number of commits and developers for each method. We collect these statistics for each method, and each developer who has contributed to the methods.

***Open card sort:*** For each project, we collect methods that have added an assert statement and are changed in a bug fix commit. We select these methods as we want to analyze what developers do to prevent future bugs. Then, we randomly select a maximum of 10 methods from each project. One thing to note is that some projects have less than 10 methods which are both edited in a bug fix commit as well as contain an added assert statement. In total, we have 575 distinct methods from 185 projects in our dataset.

To analyze why and how asserts are added, we perform an open card sort to form categories. Open card sort is used to structure information and form categories [36]. We generate a card for each method, which contains the commit message, project name, commit id, class name and the actual method. The cards were then manually grouped into several categories similar to what was performed by LaToza et al. [34]. To categorize a method, we read its commit message and patch to understand the functionality it is trying to implement and the context in which assert statements have been added. Based on the usage of asserts in a method, we put the method into an existing category or create a new one. These categories are not pre-defined but rather chosen during the card sort. At the end of the card sort, if there were are only few methods in a category, we combine several related categories to a bigger category. Similarly, if the number of methods in a category is large, we break it into several categories. To broaden our perspective, we also involve non-authors to assist us in the card sort.

[1]http://cloc.sourceforge.net/

### 3.3 Statistical Methods

We build several regression models and run several statistical tests to answer our research questions.

We use regression analysis to understand the relationship between a dependent variable and a set of independent variables considering a set of control variables. We use the same dependent, independent and control variables as used by Casalnuovo et al. [12]. The variables in our data are counts, e.g., number of asserts, number of developers, and number of bug fixing commits. The values of some of these variables are zero for many methods since many methods have no asserts or no defects. Thus, our data consists of methods with zero-assert/zero-defect and methods with non-zero asserts/non-zero defects. Following Casalnuovo et al. [12], we use hurdle regression model in R, which consists of two components: a hurdle component and a count component. The hurdle component models overcoming a hurdle, i.e., the effect of going from zero defect to one defect, and the second part (count component) models the effect of going from a non-zero value to another non-zero value. To check for excessive multi-collinearity, we measure variance inflation factors (VIFs), which shows how much the variances of the coefficients are inflated due to linear dependencies between independent and control variables. We use the commonly accepted threshold value of 5 to filter out correlated variables [15]. All models that we create in this study satisfy this threshold.

We use Wilcoxon test, which is a non-parametric test, to compare the difference between two distributions. We also compute Cohen's d, to measure the effect sizes and use boxplots to visualize the distributions.

## 4 FINDINGS

In this section, we describe findings to answer our research questions.

### 4.1 RQ1: How does assertion use relate to defect occurrence?

**Motivation:** Assertions are used by developers to check the state of a program with the purpose of finding bugs early and preventing future bugs by making program assumptions explicit in code. Casalnuovo et al. have shown that a weak negative correlation between assertion use and number of defects exists among C and C++ program [12]. Unfortunately, it is unclear if the same weak correlation exists for Java programs. This motivates us to revisit the same research question that Casalnuovo et al. investigate but with different subject programs.

**Table 2: Hurdle model comparing number of bug fixing commits in a method (as dependent variable) with number of asserts in the method (as independent variable) considering number of lines of code modified (LOC) and number of developers as control variables.**

| | Hurdle Component total_bug (as binary) *logistic* | | | | | Count Component total_bug *poisson* | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Coeff. | Std. Error | z-val | Pr(> \|z\|) | | Coeff. | Std. Error | z-val | Pr(> \|z\|) | |
| log(LOC) | 0.051 | 0.002 | 27.105 | $< 2e^{-16}$ | *** | 0.295 | 0.002 | 130.247 | $< 2e^{-16}$ | *** |
| developers | 0.425 | 0.002 | 212.346 | $< 2e^{-16}$ | *** | 0.156 | 0.001 | 149.061 | $< 2e^{-16}$ | *** |
| asserts | -0.032 | 0.004 | -8.806 | $< 2e^{-16}$ | *** | -0.013 | 0.003 | -4.085 | $4.4e^{-05}$ | *** |
| constant | -2.462 | 0.005 | -523.708 | $< 2e^{-16}$ | *** | -1.609 | 0.007 | -245.402 | $< 2e^{-16}$ | *** |
| observations | 1,903,098 | | | | | 4,201 | | | | |
| log likelihood | -1,051,000 | | | | | | | | | |
| AIC | 2,101,677 | | | | | | | | | |

$^{***}p < 0.001$, $^{**}p < 0.01$, $^{*}p < 0.05$

**Table 3: Regression models for methods changed by many and fewer developers.**

| | Many Developers | | | | | Fewer Developers | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Coeff. | Std. Error | z-val | Pr(> \|z\|) | | Coeff. | Std. Error | z-val | Pr(> \|z\|) | |
| log(LOC) | 0.227 | 0.026 | 8.785 | $< 2e^{-16}$ | *** | 0.385 | 0.028 | 13.603 | $< 2e^{-16}$ | *** |
| developers | 0.154 | 0.007 | 22.314 | $< 2e^{-16}$ | *** | 0.498 | 0.073 | 6.837 | $8.11e^{-12}$ | *** |
| asserts | -0.012 | 0.005 | -2.179 | 0.029 | * | -0.007 | 0.005 | -1.297 | 0.195 | |
| constant | -1.248 | 0.099 | -12.509 | $< 2e^{-16}$ | *** | -2.647 | 0.146 | -18.113 | $< 2e^{-16}$ | *** |
| observations | 1354 | | | | | 2847 | | | | |
| AIC | 3865.529 | | | | | 3651.301 | | | | |
| log likelihood | -1929 | | | | | -1822 | | | | |

$^{***}p < 0.001$, $^{**}p < 0.01$, $^{*}p < 0.05$

**Methodology:** We use the hurdle model described in Section 3.3 with the number of asserts in a method and number of times a method is changed in bug fix commits as independent and dependent variable respectively. The control variables are number of lines changed in and number of developers contributed to a method.

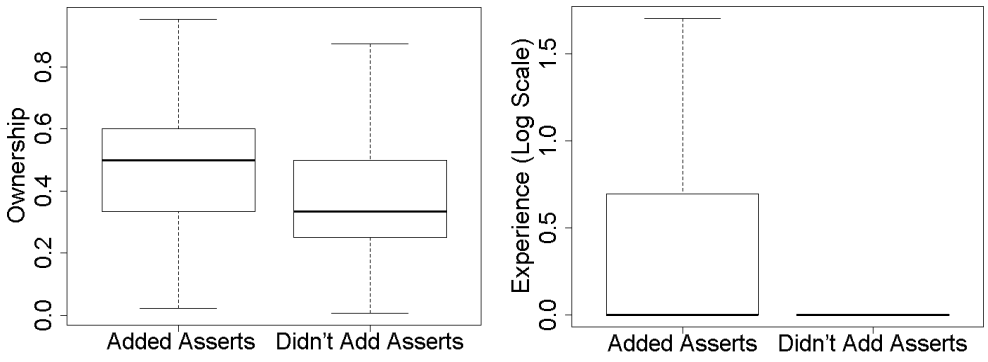
**Findings:** Table 2 shows the coefficients of the hurdle model. The first column shows the coefficients of the hurdle component and the second one for the count component. Hurdle component models dependent variable changing from 0 to 1, whereas count component models values changing from a non-zero value to another non-zero value.

For the hurdle component, we can note that the coefficient corresponding to the assert variable is negative. Thus adding the first assert is negatively correlated to the number of bugs. To estimate the impact of asserts, we can take the exponential of the assert coefficient. Adding the first assert to a method leads to an expected reduction to the number of bugs by a factor of exp(-0.032) = 0.968 or 3.2%. The p-value is less than 0.001 showing that the result is statistically significant. This shows that adding the first assert to a method has a significant impact on the defect occurrence of the method.

For the count component, the coefficient corresponding to the assert variable is also negative, and the p-value is also less than 0.001. By taking the exponential of the assert coefficient, we can find that a unit increase in the number of asserts relates to a decrease in defect occurrence by a factor of exp(-0.013) = 0.987 or 1.3%. Comparing the results of the count and hurdle components, we can observe that adding the first assert has almost double the impact on the number of defects compared to adding subsequent asserts. Our results corroborate the findings of Casalnuovo et al. [12]. They also find that adding the first assert has a prominent effect; however, our results show a stronger negative correlation for asserts added after the first one.

To further examine the impact of non-zero asserts on non-zero defects, we use 4,201 methods which have at least one assert added and one defect found over the history of the methods. We divide the dataset into two parts based upon the median of number of developers. One group contains methods with the number of developers less than or equal to the median, while the other group has methods with more than the median number of developers. Similar to Casalnuovo et al. we use Poisson regression to build a regression model for each group. Note that we do not need to use the Hurdle model for this setting since all methods have non-zero asserts and non-zero defects. Table 3 shows the coefficients of the regression models built.

From the assert coefficients of the models and the p-values, we can observe that adding asserts has a significant effect only when many developers are involved. For methods with many developers contributing to them, adding an assert leads to decrease in the bug fixing commits by a factor of exp(-0.012) = 0.988 or 1.2%. For methods with fewer developers, the coefficient of number of asserts is negative but insignificant. Thus, adding asserts to methods with many developers matters more than adding asserts to methods with fewer developers. Our results are similar to the findings by




**(a) Ownership** **(b) Experience**

**Figure 1: Distribution of ownership and experience for developers who added and did not add asserts.**

Casalnuovo et al. on C and C++ programs; they also find that asserts have a significant effect only when many developers are involved.

*Adding asserts to methods can lead to lower defect occurrence, with the first assert being more important than the subsequent ones. Asserts added to methods with many developers has a larger effect on the number of defects compared to those added to methods with fewer developers.*

## 4.2 RQ2: How does assertion use relate to developer characteristics such as code ownership and experience?

**Motivation:** In this research question, we investigate the characteristics of developers who introduce asserts to methods. Asserts are not trivial to write as often they require conceptual understanding of the code. Thus, we hypothesize that developers adding asserts have high degree of understanding of the logic as compared to developers who just change code. This research question investigates if this hypothesis is supported based on our data. A similar question was investigated by Casalnuovo et al. for C and C++ programs, while we consider Java programs.

**Methodology:** To answer RQ2, we first calculate the ownership and experience of developers for each method similar to Casalnuovo et al. Ownership is calculated as the percentage of changes made to a particular method by a developer. For example, if there are 500 commits made to a method $m$ over its lifetime and of those 500 commits, developer $d$ made 100 commits, then $d$'s ownership of $m$ is 0.2. We calculate the experience of a developer to a method by counting the total number of commits made by the developer to the method. Next, for each method $m$, we separate developers into two groups: those who have added asserts and those who did not. If any of the groups is empty, we omit the method from the rest of the analysis. We then calculate the median ownership and experience of each group of developers for each method and use box plot to compare these two groups.

**Findings:** Figure 1a shows the distribution of median ownership of developers who have added asserts to methods and the ones who did not. From the boxplot, we can observe that developers who have added asserts have larger ownership. The median value for ownership of developers for methods with added asserts is 0.50, whereas the corresponding value for methods without asserts is 0.33. The lower and upper quartile values for ownership of developers for methods with added asserts are 0.33 and 0.60, whereas the corresponding values for methods without asserts are 0.24 and 0.50, respectively. We perform Wilcoxon test and find that the difference between the two groups is significant with a p-value of less than $2.2e^{-16}$. We have also computed Cohen's d and find that the effect size is medium. This suggests that users with higher ownership to a method have higher likelihood of adding asserts.

Figure 1b shows a box plot which compares the distribution of median experience of developers of the two groups for each method. The lower and upper quartile values for experience of developers for methods with added asserts are 0.00 and 0.69, whereas the corresponding values for methods without asserts are 0.00 each. The median values for the two groups are zeroes as large number of methods are changed by only one developer. We find that 79.17% of the methods for which no assert was added were changed by only one developer, thus, the lower quartile, upper quartile and median values are the same. As such, we observe that the box for experience values of developers for methods without asserts reduces to a line. Wilcoxon test confirms that there is a difference between the mean of the median experiences of developers of these two groups (p-value is less than $2.2e^{-16}$). We have also computed Cohen's d and find that the effect size is small. This suggests that users with higher experience to a method have higher likelihood of adding asserts.

*Developers who have added asserts to methods often have higher ownership of and experience with the methods than developers who did not add asserts.*

## 4.3 RQ3: How are asserts used by developers?

**Motivation:** From our findings to the previous research questions, we find that adding asserts has a significant effect on defect occurrence and developers who add asserts have high ownership and experience. In this research question (which was not investigated by Casalnuovo et al.), we want to investigate the types of asserts added by developers. A better understanding of asserts will help future developers to add asserts which are more likely to find bugs.

**Methodology:** To answer RQ3, we collect methods in which developers added asserts. To get a representative sample from all projects in our dataset while limit the manual effort involved in the card sorting process, we randomly select a maximum of ten distinct methods, which have at least one assert and are related to at least one buggy commit, from each project in our dataset. However, some projects have less than ten methods which satisfy the above criteria. In those cases, we select all the methods in that project. Next, we manually analyze 575 distinct methods and perform open card sort, which is described in Section 3, to find categories.

**Findings:** We highlight eight categories that emerge from our card sort below.

*A. Null Condition Check:*

During the card sort, we find that many of the methods check for null deferences using assertion in two main ways: by using of the inequality operand, e.g., assert ⟨variable⟩ != null, and by invoking the `assertNotNull` method from the `Assert` class. We analyze the data flow from all the sampled methods within this category and find that developers make use of the variable under test at the later part of the method after checking for null dereferences. An example that belongs to this category is the following method from Apache Geronimo GShell[2], a framework for building rich command-line applications, which encrypts a given string and outputs its corresponding bytes. The method first checks if the buffer and the encrypting key are not null. If both these values are not null, only then the method proceeds further.

**Title - Assert checks for null condition.**
**Project - Apache Geronimo GShell**
**Class - org.apache.geronimo.gshell.remote.message.**
**CryptoAwareMessageSupport.java**
**Commit - b216e35b4843b78f3834c60a7e08abbd5dedba30**

```
+ protected void encryptString(final ByteBuffer out, final
     Key key, final String str) throws Exception {
+   assert out != null;
+   assert key != null;
+   ...
+ }
```

*B. Process State Check:*

Asserts are often used to enforce that a process is in a particular state (e.g., whether a process is still running or it has terminated successfully) before certain operations are performed. In the following example from Apache Reef[3], the assert checks the state of a job driver, which can be *INIT, WAIT_EVALUATOR, READY, RUNNING* etc. The method *onNext* proceeds only if the job driver is in the *WAIT_EVALUATOR* state.

**Title - Assert checks for state of job driver.**
**Project - Apache Reef**
**Class - org.apache.reef.examples.scheduler.SchedulerDriver.java**
**Commit - 10f96514379179b8b2cf11fcf350418787302078**

```
+ public void onNext(ActiveContext context) {
+   synchronized (lock) {
+   assert (state == State.WAIT_EVALUATOR);
+   ...
+   }
+ }
```

*C. Initialization Check:*

We find that developers also often use asserts to check if a resource has been initialized. The following code shows an example from Apache ActiveMQ Apollo[4], a messaging broker which translates a message from one messaging protocol to another. In this example, the method *add* adds element E to the queue controller. The method uses `synchronized` to ensure that only one thread can execute the code enclosed and then checks if the queue is initialized using an assert statement.

**Title - Assert checks for Initialization of queue**
**Project - Apache ActiveMQ Apollo**
**Class - org.apache.activemq.queue.ExclusivePersistentQueue.java**
**Commit - 3131481436673062ee49c69a4ab120f373af671e**

```
+ public void add(E elem, ISourceController<?> source) {
+    synchronized (this) {
+    assert initialized;
+    ...
+   }
+ }
```

*D. Resource Check:*

Developers often perform operations on resources such as files and directories. Prior to such operations it is important to check if the file or directory the code is trying to use exists or not. Our analysis shows that developers use assert statements to verify presence of resources. In the following example from Apache Tentacles[5], which automates interactions with the repository releases containing large number of artifacts, the assert statement added to the method checks if the file is a directory.

**Title - Assert checks if a file is a directory.**
**Project - Apache Tentacles**
**Class - org.apache.creadur.tentacles.Files.java**
**Commit - e96309356c2cb4e5c91dc1c8f04f0aed1d3e7170**

```
  public static void mkdirs(File file) {
+  final boolean isDirectory = file.isDirectory();
+  assert isDirectory : "Not a directory: " + file;
  }
```

*E. Resource Lock Check:*

Developers often check if a resource is locked by another process before performing any operation. In the following code from Apache ManifoldCF solr-3.x Integration, the method *close* removes all references to *SegmentReader* and commits any pending changes but before performing any operation, it checks if the thread has locked the current instance of class *IndexWriter*. Only if there is a lock on the object, the subsequent statements will be executed otherwise the program will throw an `AssertionError`.

[2]http://geronimo.apache.org/gshell/
[3]https://reef.apache.org/
[4]https://activemq.apache.org/apollo/
[5]http://creadur.apache.org/tentacles/

**Title - Assert checks for lock on IndexWriter object**
**Project - Apache ManifoldCF solr-3.x Integration**
**Class - org.apache.lucene.index.IndexWriter.java**
**Commit - a27b2c434adbece0e04c1a91b1c6163f0097ad98**

```
  synchronized void close() throws IOException {
+   assert Thread.holdsLock(IndexWriter.this);
     ...
  }
```

*F. Minimum and Maximum Value Constraint Check:*

We find several examples of asserts which are used to check if a given value is above certain minimum limit or less than a certain maximum value. Such criteria checks are done to ensure that the program does not behave unexpectedly when the given value is used. In the following example from Apache AsterixDB Hyracks[6], the method *validate* checks if the value of *lowRangeTuple* and *highRangeTuple* are less than and more than the *frameTuple* value, respectively.

**Title - Assert checks for minimum and maximum value.**
**Project - Apache AsterixDB Hyracks**
**Class - org.apache.hyracks.storage.am.btree.frames.**
**BTreeNSMInteriorFrame.java**
**Commit - 3b89343c7d02388c836128f65401d133629d761e**

```
+ public void validate(PageValidationInfo pvi) throws
      HyracksDataException {
+     if (!pvi.isLowRangeNull) {
+    assert cmp.compare(pvi.lowRangeTuple, frameTuple) <
      0;
+     }
+     if (!pvi.isHighRangeNull) {
+    assert cmp.compare(pvi.highRangeTuple, frameTuple) >=
      0;
+     }
+  }
```

*G. Collection Data and Length Check:*

Collections such as arrays, lists and maps are commonly used in programming to store data. Before accessing the values of a collection using a data structure operation, asserts are at times used to check if the operation is valid. For example, for an array, assertion can be used to check if the index used to access one of its elements is still within the bound of the array. In the following example from Apache Mahout[7], which provides a programming environment for creating scalable machine learning applications, the method *mergeR* accepts two two-dimensional array and merges their values. The assert method ensures that the length of the input arrays are equal before iterating over these collections to perform further operations.

**Title - Assert checks for array length.**
**Project - Apache Mahout**
**Class - org.apache.mahout.math.hadoop.stochasticsvd.**
**GivensThinSolver.java**
**Commit - 151de0d737501af5dcfee8a21bc7d18ff6edddc8**

```
+ public static void mergeR(double[][] r1, double[][] r2) {
+   int kp = r1[0].length;
+   assert kp == r2[0].length;
+   double[] cs = new double[2];
+   for (int v = 0; v < kp; v++) {
+       for (int u = v; u < kp; u++) {
+   ...
+ }
```

*H. Implausible Condition Check:*

Asserts are also used in switch-case and if-else statements to check for implausible condition. In the following example from Apache Felix[8], an open-source implementation of OSGi Framework and Service platform[9], the method matches a character variable in the switch statement. If none the conditions satisfy, the default condition will be selected and assert false statement will be executed. This statement informs the developer that a supposedly implausible condition has happened. Such information will help developer to check the value of the variable being matched in the switch statement and prevent any unwanted program behaviors.

**Title - Assert informs of implausible situation.**
**Project - Apache Felix**
**Class - org.apache.felix.gogo.runtime.shell.Tokenizer.java**
**Commit - c578f444bb3c2bc192ee74294398fdfac8a237a9**

```
+ private CharSequence group(){
+   switch (ch)
+   {
+     case '{':
+         ...
+     default:
+     assert false;
+   }
+ }
```

*Asserts are used for several purposes such as null check, resource lock check, initialization check, implausible condition check among many others.*

# 5 DISCUSSION

## 5.1 Implications

**For Researchers:**

In this study, we highlight the impact of asserts on defect occurrence. Our results show that asserts have a significant effect on the number of bugs and strengthen the findings of Casalnuovo et al [12] by investigating a different programming language and larger dataset. From our qualitative analysis of assert usage, we find that developers make use of asserts in different ways to help them in debugging and preventing future bugs. We find that a large number of asserts are used to check for null condition. This is intuitive since null check is the most basic condition check before performing

[6] http://incubator.apache.org/projects/asterixdb.html
[7] http://mahout.apache.org/
[8] http://felix.apache.org/
[9] https://www.osgi.org/developer/specifications/

operations on a variable. Our results corroborate previous studies which find wide usages of null check [19, 47]. Many static analysis tools have been developed to check for null dereferences [20, 26, 27]. However, static analysis tools have several drawbacks such as scalability issues, limited interprocedural analysis, etc. [40]. Further research is required to assist developers in finding locations where asserts need to be added to prevent null dereferences. Furthermore, developers use assertions for many other purposes. Our study sheds light on these different usages. Schiller et al. argue that there is a need to curate best industry practices for usages of contracts as design patterns [47]. Our study takes one step in that direction by identifying 8 categories of how assertions are used in practice. Further studies are needed though to fully understand assert usage patterns in different projects and programming languages, which will help in establishing design patterns for contracts in general. Just like object-oriented design patterns have been helpful by providing best solutions to common problems faced by developers during software development [2, 48, 51], contract design pattern can provide similar benefit and achieve similar impact.

Moreover, further studies are required to understand the perception of developers on the usefulness of asserts. As writing asserts requires knowledge of the surrounding code, it would be worthwhile to examine how much effort goes into understanding and adding asserts to programs. Often, books and online articles contend that asserts improve the readability of programs, however, research studies are required to verify these claims. Current studies on asserts have mainly looked into the relationship between assert and code quality [4, 10, 12, 33, 39].

**For Practitioners:**

Our study shows that adding asserts can lead to lower defect occurrence for Java projects. This result can serve as a motivation for Java developers to add asserts to their code as it can potentially lead to an improvement in software quality. Our results shed light on the different usage patterns of asserts. Apart from commonly used *null check*, we find that developers employ asserts for a wide range of purposes such as resource lock check, initialization check, minimum and maximum value constraint check, implausible condition check, among many others. These usage patterns can assist developers to better use asserts. New software developers are often unsure on how to become better engineers [6]. A past study shows that developers must have certain skills like passionate, open-minded, creative, adaptable, hardworking, knowledgeable, productive etc. [35]. Apart from having these non-technical skills, developers have to continuously update their technical know-how. Our results, in complement with a previous study [12], can motivate practitioners to make use of concepts like asserts as they are shown to lead to reduction in defects in three different languages, i.e., C, C++ and Java, and results are possibly generalizable to other languages that support asserts.

**For Educators:**

Assertions are often taught as part of programming courses in undergraduate curriculum. However, students mention that learning assertions is difficult [49] and they often face problems in writing pre- and post-conditions due to insufficient background in mathematics [23]. Our findings inform that assertions are helpful in debugging and reducing the number of bugs. Thus, more effort is needed to improve learning of these concepts. Past studies show that there is need to restructure computer science curriculum to help students understand complex topics like discrete mathematics and usage of tools that assist students in writing pre- and post-conditions, thus, helping them ensure the correctness of methods [23]. Many students prefer learning programming by referring to examples and by repetitive practice than by simply listening to lectures [50]. We present several examples of usages of asserts which can guide educators and students in understanding different usage patterns and logic on how to write better asserts.

## 5.2 Threats to Validity

Our study uses a similar methodology that was used by Casalnuovo et al. [12]. Thus, our study shares many threats to validity as this prior work.

**Construct Validity:** First, we use bug fixing commits as a proxy for defect occurrences. We choose not to use number of issue reports as a proxy because not all the projects we investigate use an issue tracking system and not all bugs are logged in the issue tracking system [8]. Still, we acknowledge that a bug may be represented by more than one bug fixing commits. Furthermore, some bug fixing commits may not be described as such in commit logs and thus may be missed in our study. It is also possible that a commit is incorrectly associated with a bug. Unfortunately, manually identifying the exact number of bugs for a large dataset like ours is very difficult.

**External Validity:** We consider projects hosted on GitHub. As such, our results may not be valid for projects on other platforms. However, we reduce this threat by analysing a large number of projects. Also, we consider only projects from Apache Software Foundation. As such, our results might not generalize to other open-source projects. However, we consider a large dataset and Apache projects are spread out over various domains, which can mitigate this threat to some extent. Many past studies have also only focused on Apache projects [42, 44, 45].

# 6 RELATED WORK

In this section, we explain the related work on assertions and their impact on quality. We also highlight related works that perform replication studies.

## 6.1 Studies on Assertion Usage and Impact on Quality

Hoare seminal work introduces a method of reasoning about correctness of a program, also known as Hoare logic [25]. Hoare's technique allows programmers to express program properties that can be automatically checked. This work influences research in automated reasoning about software correctness. Chalin et al. conduct a survey and investigate assertion usage in various projects [14]. The participants of the survey consist of over 200 industry developers and questions related to the developers' use of assertion in their projects as well as the reporting format of discovered errors are asked. The survey results highlight that many of the participants have used assertions in their work. It is also reported that a majority of the assertions are pre- and post-condition checks. Jones at al. conduct a separate study on 21 Java, Eiffel and C# projects [19].

They find a correlation between the number of assertions used and the size of a project. Similar to Chalin et al., they find that a majority of the assertions are pre- and post-condition checks.

Polikarpova et al. perform a study to compare automatically-generated assertions and human-written assertions. More specifically, they compare automatically-generated assertions that are produced by Daikon (a contract inference tool) with human-written assertions [41]. They report that automatically-generated assertions are valuable since they can supplement human-written ones. Schiller et al. has also conducted a comparative study between Daikon's automatically-generated assertions and human-written assertions [47]. They report that the majority of the human-written assertions are on checking null dereferences and Daikon's automatically-generated assertions are more varied expressing invariants on state updates and conditional properties among others.

Briand et al. conduct studies to locate the source of errors with and without the use of assertions [10]. It is reported that by using assertions, developers are able to detect between 75% - 80% of the errors and developers appreciate the use of assertions as it aids in their debugging. If more fine-grained assertions are added, developers can detect an extra 10% of the errors. However, they report that this could result in extra effort from the developers and may not seem justifiable. In a separate study, Baudry et al. has reported a similar finding [4]. Developers report that the use of assertions generally eases their debugging. However, it is also reported that the quality of assertions matter more to developers than their quantity. Muller et al. conduct a study to investigate the relationship between the use of the assertions and developer's quality and productivity [39]. They observe that the use of assertion increases developer's productivity and the written code is more robust. Kudrjavets et al. study two commercial project components from Microsoft [33].

Our study builds upon the above mentioned studies by analyzing the relationship between assertion usage and defect occurrence further. We replicate the study by Casalnuovo et al. [12] by analyzing a total of 185 Java projects. This is much more than the projects analyzed by the above-mentioned studies. Additionally, we also investigate additional research questions which investigate the relationship between developer ownership and experience with the introduction of assertions, and various usage patterns of assertions.

## 6.2 Replication Studies

Bird et al. examine the relationship between ownership and software failure for Windows Vista and Windows 7 and find that high levels of ownership are associated with less defects [7]. Foucault et al. perform a replication of the above study on 7 open-source projects and find contradicting results, i.e., fault-proneness is affected more by module size than ownership metrics [21]. Herzig et al. replicate the above two studies and define several new ownership metrics [24]. Their findings confirm the results observed in the original study by Bird et al [7]. Mockus examines a project from Avaya to study the impact of organizational volatility on customer reported defects [38]. The results show that departures from the organization leads to higher defects while adding new members has no impact on software quality. Donadelli et al. replicate the above study by Mockus on Google Chrome and find differing results, i.e., after normalizing by the highly correlated number of co-owners, the number of people who leave or join the project both lead to lower post-release defects [17].

Zimmermann et al. use network analysis to analyze dependencies between different pieces of code to help managers find units that are more defect-prone [53]. They find that network measures can predict 60% of the binaries considered as critical by developers and models using network measures have 10% higher recall than the ones using complexity metrics. Premraj and Herig perform a replication of [53] on 3 open-source Java projects and find that results are the same using similar experimental setup [43]. However, using setups more commonly used in industry such as forward-release and cross-project prediction, network measures do not offer any advantage. Erdogmus et al. performed a study on students to analyze two groups: one which wrote test-driven development (TDD) while other followed conventional technique [18]. They found that group using TDD wrote more tests and was more productive. Fucci et al. performed a replication of Erdogmus et al. [18] and further analyze the relationship of process conformance and software quality [22]. They find that in TDD there is a correlation between process conformance and quality, but not with productivity. Apart from these replications, there are studies that look into various aspects of replication such as concepts, classifications, guidelines, and other themes [3, 16].

## 6.3 Large Scale Studies on GitHub

Kochhar et al. investigate thousands of projects to investigate the correlations between number of test cases and various project development characteristics, including the lines of code and the size of development teams [30, 31]. In another study, Kochhar et al. analyse 628 projects from GitHub to investigate the impact of using multiple languages on software quality [32]. Jiang et al. collect thousands of forks from GitHub to understand the forking phenomenon in GitHub [28]. Zhang et al. propose an approach to detect similar repositories and evaluate their approach on 1000 popular Java repositories on GitHub [52].

# 7 CONCLUSION AND FUTURE WORK

Assertions help developers test assumptions about the code. Assertions are supported by many programming languages such as C, C++, Java, Python, etc. Not only assertions help in debugging and fixing errors, they also serve as documentation to inform developers about what the code does. In this replication study, we collect 185 Apache Java projects, which contains over 20M LOC, 0.2M files, 1M methods and 4M commits, to analyze the usage of asserts. We perform several regression analyses to observe the impact of assertions on defect occurrence as well how assertion use relates to developer ownership and experience. Furthermore, we perform an open card sort on 575 distinct methods from projects in our dataset to understand how asserts are used by developers.

Our empirical study leads to the following findings:

(1) Adding asserts to a method have a small yet significant relationship with defect occurrence.
(2) Developers who have added asserts to a method often have high experience with and larger ownership of the method than developers who did not.

(3) Developers often use asserts to check for null condition, initialization, process state, resource lock, implausible condition, etc.

As a future work, we want to perform a larger study by investigating many more projects written in many more programming languages. We also plan to investigate the relationships between each of the different types of assert and defect occurrence. We also want to augment our study with a developer survey to get a comprehensive insight on the usefulness and limitations of asserts, as well as common usage patterns that are perceived to be the most useful.

## DATASET

Our dataset is made publicly available and it can be downloaded from: *https://github.com/smusis/assert-usage*

## REFERENCES


[1] Last accessed - Jan 31, 2016. TIOBE Programming Community index Jan 2016. http://www.tiobe.com/tiobe-index/.

[2] Giuliano Antoniol, Gerardo Casazza, Massimiliano Di Penta, and Roberto Fiutem. 2001. Object-oriented design patterns recovery. *Journal of Systems and Software* 59, 2 (2001), 181–196.

[3] Maria Teresa Baldassarre, Jeffrey Carver, Oscar Dieste, and Natalia Juristo. 2014. Replication Types: Towards a Shared Taxonomy. In *EASE*. 18:1–18:4.

[4] Benoit Baudry, Yves Le Traon, and Jean-Marc Jézéquel. 2001. Robustness and Diagnosability of OO Systems Designed by Contracts. In *METRICS*. 272.

[5] Gabriele Bavota, Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. 2015. How the Apache Community Upgrades Dependencies: An Evolutionary Study. *Empirical Software Engineering* 20, 5 (2015), 1275–1317.

[6] Andrew Begel and Beth Simon. 2008. Novice Software Developers, All over Again. In *ICER*. 3–14.

[7] Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, and Premkumar Devanbu. 2011. Don'T Touch My Code!: Examining the Effects of Ownership on Software Quality. In *ESEC/FSE*. 4–14.

[8] Tegawendé F. Bissyandé, David Lo, Lingxiao Jiang, Laurent Réveillère, Jacques Klein, and Yves Le Traon. 2013. Got issues? Who cares about it? A large scale investigation of issue trackers from GitHub. In *ISSRE*. 188–197.

[9] Tegawendé F. Bissyandé, Ferdian Thung, David Lo, Lingxiao Jiang, and Laurent Réveillère. 2013. Popularity, Interoperability, and Impact of Programming Languages in 100,000 Open Source Projects. In *COMPSAC*. 303–312.

[10] L. C. Briand, Y. Labiche, and H. Sun. 2002. Investigating the Use of Analysis Contracts to Support Fault Isolation in Object Oriented Code. In *ISSTA*. 70–80.

[11] A Colin Cameron and Pravin K Trivedi. 2013. *Regression Analysis of Count Data*. Number 53. Cambridge Univrsity Press.

[12] Casey Casalnuovo, Prem Devanbu, Abilio Oliveira, Vladimir Filkov, and Baishakhi Ray. 2015. Assert Use in GitHub Projects. In *ICSE*. 755–766.

[13] Stephen Cass. Last accessed - September 30, 2016. The 2015 Top Ten Programming Languages. http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages.

[14] P. Chalin. 2005. Logical foundations of program assertions: what do practitioners want?. In *SEFM*. 383–392.

[15] Jacob Cohen, Patricia Cohen, Stephen G. West, and Leona S. Aiken. 2003. Applied Multiple Regression/correlation Analysis for the Behavioral Sciences. *Lawrence Erlbaum* (2003).

[16] Cleyton V. C. de Magalhães, Fabio Q. B. da Silva, and Ronnie E. S. Santos. 2014. Investigations About Replication of Empirical Studies in Software Engineering: Preliminary Findings from a Mapping Study. In *EASE*. 37:1–37:10.

[17] Samuel M. Donadelli, Yue Cai Zhu, and Peter C. Rigby. 2015. Organizational Volatility and Post-release Defects: A Replication Case Study Using Data from Google Chrome. In *MSR*. 391–395.

[18] Hakan Erdogmus, Maurizio Morisio, and Marco Torchiano. 2005. On the Effectiveness of the Test-First Approach to Programming. *TSE* 31, 3 (2005), 226–237.

[19] H. Christian Estler, Carlo A. Furia, Martin Nordio, Marco Piccioni, and Bertrand Meyer. 2014. *FM 2014: Formal Methods: 19th International Symposium*. Springer International Publishing, Chapter Contracts in Practice, 230–246.

[20] Cormac Flanagan, K. Rustan M. Leino, Mark Lillibridge, Greg Nelson, James B. Saxe, and Raymie Stata. 2002. Extended Static Checking for Java. In *PLDI*. 234–245.

[21] Matthieu Foucault, Jean-Rémy Falleri, and Xavier Blanc. 2014. Code Ownership in Open-source Software. In *EASE*. 39:1–39:9.

[22] Davide Fucci, Burak Turhan, and Markku Oivo. 2014. Conformance Factor in Test-driven Development: Initial Results from an Enhanced Replication. In *EASE*. 22:1–22:4.

[23] Timothy S. Gegg-Harrison, Gary R. Bunce, Rebecca D. Ganetzky, Christina M. Olsson, and Joshua D. Wilson. 2003. Studying Program Correctness by Constructing Contracts. In *ITiCSE*. 129–133.

[24] Michaela Greiler, Kim Herzig, and Jacek Czerwonka. 2015. Code Ownership and Software Quality: A Replication Study. In *MSR*. 2–12.

[25] C. A. R. Hoare. 1969. An Axiomatic Basis for Computer Programming. *Commun. ACM* 12, 10 (1969), 576–580.

[26] David Hovemeyer and William Pugh. 2004. Finding Bugs is Easy. *SIGPLAN Notices* 39, 12 (2004), 92–106.

[27] David Hovemeyer and William Pugh. 2007. Finding More Null Pointer Bugs, but Not Too Many. In *PASTE*. 9–14.

[28] Jing Jiang, David Lo, Jia-Huan He, Xin Xia, Pavneet Singh Kochhar, and Li Zhang. 2017. Why and how developers fork what from whom in GitHub. *Empirical Software Engineering* 22, 1 (2017), 547–578.

[29] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. 2014. The Promises and Perils of Mining GitHub. In *MSR*. 92–101.

[30] Pavneet Singh Kochhar, Tegawendé F. Bissyandé, David Lo, and Lingxiao Jiang. 2013. Adoption of Software Testing in Open Source Projects-A Preliminary Study on 50, 000 Projects. In *CSMR*. 353–356.

[31] Pavneet Singh Kochhar, Tegawendé F. Bissyandé, David Lo, and Lingxiao Jiang. 2013. An Empirical Study of Adoption of Software Testing in Open Source Projects. In *QSIC*. 103–112.

[32] Pavneet Singh Kochhar, Dinusha Wijedasa, and David Lo. 2016. A Large Scale Study of Multiple Programming Languages and Code Quality. In *SANER*. 563–573.

[33] Gunnar Kudrjavets, Nachiappan Nagappan, and Thomas Ball. 2006. Assessing the Relationship Between Software Assertions and Faults: An Empirical Investigation. In *ISSRE*. 204–212.

[34] Thomas D. LaToza, Gina Venolia, and Robert DeLine. 2006. Maintaining Mental Models: A Study of Developer Work Habits. In *ICSE*. 492–501.

[35] Paul Luo Li, Andrew J. Ko, and Jiamin Zhu. 2015. What Makes a Great Software Engineer?. In *ICSE*. 700–710.

[36] Bella Martin, Bruce Hanington, and Bruce M. Hanington. 2012. *Universal Methods of Design: 100 Ways to Research Complex Problems, Develop Innovative Ideas, and Design Effective Solutions*. Rockport,.

[37] Bertrand Meyer. 1992. Applying "Design by Contract". *Computer* 25, 10 (1992), 40–51.

[38] Audris Mockus. 2010. Organizational Volatility and Its Effects on Software Defects. In *FSE*. 117–126.

[39] M. M. Muller, R. Typke, and O. Hagner. 2002. Two controlled experiments concerning the usefulness of assertions as a means for programming. In *ICSM*. 84–92.

[40] Mangala Gowri Nanda and Saurabh Sinha. 2009. Accurate Interprocedural Null-Dereference Analysis for Java. In *ICSE*. 133–143.

[41] Nadia Polikarpova, Ilinca Ciupa, and Bertrand Meyer. 2009. A Comparative Study of Programmer-written and Automatically Inferred Contracts. In *ISSTA*. 93–104.

[42] Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. 2011. Ecological Inference in Empirical Software Engineering. In *ASE*. 362–371.

[43] R. Premraj and K. Herzig. 2011. Network Versus Code Metrics to Predict Defects: A Replication Study. In *ESEM*. 215–224.

[44] Foyzur Rahman, Daryl Posnett, and Premkumar Devanbu. 2012. Recalling the "Imprecision" of Cross-project Defect Prediction. In *FSE*. 61:1–61:11.

[45] Foyzur Rahman, Daryl Posnett, Israel Herraiz, and Premkumar Devanbu. 2013. Sample Size vs. Bias in Defect Prediction. In *ESEC/FSE*. 147–157.

[46] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. 2014. A Large Scale Study of Programming Languages and Code Quality in Github. In *FSE*. 155–165.

[47] Todd W. Schiller, Kellen Donohue, Forrest Coward, and Michael D. Ernst. 2014. Case Studies and Tools for Contract Specifications. In *ICSE*. 596–607.

[48] Alan Shalloway and James R Trott. 2004. *Design patterns explained: a new perspective on object-oriented design*. Pearson Education.

[49] M. Sivasakthi and R. Rajendran. 2011. Learning difficulties of 'object-oriented programming paradigm using Java': students' perspective. *Indian Journal of Science and Technology* 4, 8 (2011), 983–985.

[50] Phit-Huan Tan, Choo-Yee Ting, and Siew-Woei Ling. 2009. Learning Difficulties in Programming Courses: Undergraduates' Perspective and Perception. In *ICCTD*. 42–46.

[51] John Vlissides, Richard Helm, Ralph Johnson, and Erich Gamma. 1995. Design patterns: Elements of reusable object-oriented software. *Reading: Addison-Wesley* 49, 120 (1995), 11.

[52] Yun Zhang, David Lo, Pavneet Singh Kochhar, Xin Xia, Quanlai Li, and Jianling Sun. 2017. Detecting similar repositories on GitHub. In *SANER*. 13–23.

[53] Thomas Zimmermann and Nachiappan Nagappan. 2008. Predicting Defects Using Network Analysis on Dependency Graphs. In *ICSE*. 531–540.