applied sciences

MDPI

Article

Empirical Study of Test Case and Test Framework Presence in Public Projects on GitHub

Matej Madeja *, Jaroslav Porubän, Sergej Chodarev, Matúš Sulír and Filip Gurbáľ

Department of Computers and Informatics, Technical University of Košice, Letná 9, 042 00 Košice, Slovakia; jaroslav.poruban@tuke.sk (J.P.); sergej.chodarev@tuke.sk (S.C.); matus.sulir@tuke.sk (M.S.); filip.gurbal@tuke.sk (F.G.)

* Correspondence: info@madeja.sk

Abstract: Automated tests are often considered an indicator of project quality. In this paper, we performed a large analysis of 6.3 M public GitHub projects using Java as the primary programming language. We created an overview of tests occurrence in publicly available GitHub projects and the use of test frameworks in them. The results showed that 52% of the projects contain at least one test case. However, there is a large number of example tests that do not represent relevant production code testing. It was also found that there is only a poor correlation between the number of the word "test" in different parts of the project (e.g., file paths, file name, file content, etc.) and the number of test cases, creation date, date of the last commit, number of commits, or number of watchers. Testing framework analysis confirmed that JUnit is the most used testing framework with a 48% share. TestNG, considered the second most popular Java unit testing framework, occurred in only 3% of the projects.

Keywords: testing culture; code quality; testing framework; GitHub; test case presence

check for updates

Citation: Madeja, M.; Porubän, J.; Chodarev, S.; Sulír, M.; Gurbáľ, F. Empirical Study of Test Case and Test Framework Presence in Public Projects at GitHub. *Appl. Sci.* **2021**, *11*, 7250. https://doi.org/10.3390/app11167250

Academic Editor: Vito Conforti

Received: 9 July 2021
Accepted: 4 August 2021
Published: 6 August 2021

1. Introduction

Automated testing is considered an inevitable part of high-quality software projects [1]. Tests allow developers to automatically detect bugs in changed code, they can help them analyze and resolve the defects. The presence of test cases, however, does not ensure that project is bug-free. Gren and Antinyan [2] even found that there is only a weak correlation between testing and code quality.

To further analyze the relationships between code quality and the occurrence of tests, we need to better understand how developers write tests in industrial projects. There are recommendations of how tests should be written, but we do not know how they are written in practice. Many researchers have tried to clarify the motivation of writing tests [3–5], the impact of Test-driven development (TDD) on code quality [6–9], the effectiveness of tests on defective code [10–12], or the popularity of testing frameworks [13]. However, a manual analysis of test cases in projects could also be helpful to reveal and understand how tests are written. In order to perform such an analysis in the future, it is necessary to find out how many projects use automated tests, using which testing frameworks they are written, and how the frameworks and libraries are used simultaneously. At the same time, it is interesting to see the ratio of the word "test" in different parts of the project to the number of test cases and other project's metadata, such as number of watchers, number of commits, etc. This information can be useful in terms of further development of testing tools, using information from tests during program comprehension as well as mining repositories in other studies.

It is also interesting to know how many open-source projects contain automated tests at all. According to Cruz et al. [14], who analyzed 1000 repositories, 39% of projects include test cases. Kochhar et al. [1] executed a similar study on 20,817 and later on 50,000 projects with test case presence in 61.65% and 42.66% of repositories, respectively. The first study, however, was not conducted on a random set of projects, and the results of other ones published by Kochhar et al. varied too much, so it is needed to evaluate the test case presence on a larger sample of independent projects.

In our previous work [15], we had already analyzed the correlation between the occurrence of the word "test" in names and contents of files and the number of actual test cases in selected GitHub projects. We were using manual identification of test cases, so we needed to limit the number of projects. For this reason, we have analyzed only projects with the highest occurrence of the word "test". We have found that the Pearson correlation coefficient $r = 0.655$ and therefore there is only a weakly significant correlation. Despite the weak correlation with the number of test cases, we have found that the word "test" is very closely related to the occurrence of at least one executable test case. This means that the occurrence of this word in the project may indicate the presence of test cases.

In this work, we extend our analysis of the occurrence of the word "test". Our long-term goal is to streamline the program comprehension using the information from tests, but first, we need to find projects with tests, from which it will be possible to find out what information is placed in the tests and where. Searching for the string "test" to find test cases in a project can return code artifacts that are not tests, and we need to eliminate such cases. At the same time, if there exists a typical number of tests in projects, it might be helpful to find out why this number is common and whether the results do not include true negatives. Therefore, we need to manually analyze random and independent projects to find out in which code artifacts this string is located (e.g., documentation, production code, testing code, etc.) and how often it indicates a test case. Mentioned issue of detecting test cases in a code could also be related to the correlation of the number of the word "test" and other project attributes, such as activity or popularity. This paper answers the following questions:

RQ 1. *Is there a typical number of the word "test" occurrences across a project that is observed in a large number of projects? If so, what is the reason behind such specific number?*

RQ 2. *Is occurrence of the word "test" related to the creation date, date of last commit, number of commits, or number of watchers of a project?*

As a part of our previous work, we also developed an automatic identification approach for test cases with 97% accuracy. This allows us to extend the analysis to all GitHub projects from May 2019 with Java as a primary language. To find out how many projects use tests that could be used as possible sources of information about the production code, we will use the mentioned automatic identification approach in this paper. We need to investigate how many tests and testing frameworks are used and where developers place important information useful for program comprehension. Therefore, the aim of this study is not to analyze how tests change during the project life cycle, but whether they are present in the projects, how to find the projects that contain tests, and find how tests describe the production code given the framework used. In this paper, we also answer the following questions:

RQ 3. *What is the ratio of open-source projects that contain at least one test case?*

RQ 4. *What is the general correlation between the number of occurrences of the word "test" in all files' content of a project and the number of actual test cases in the project?*

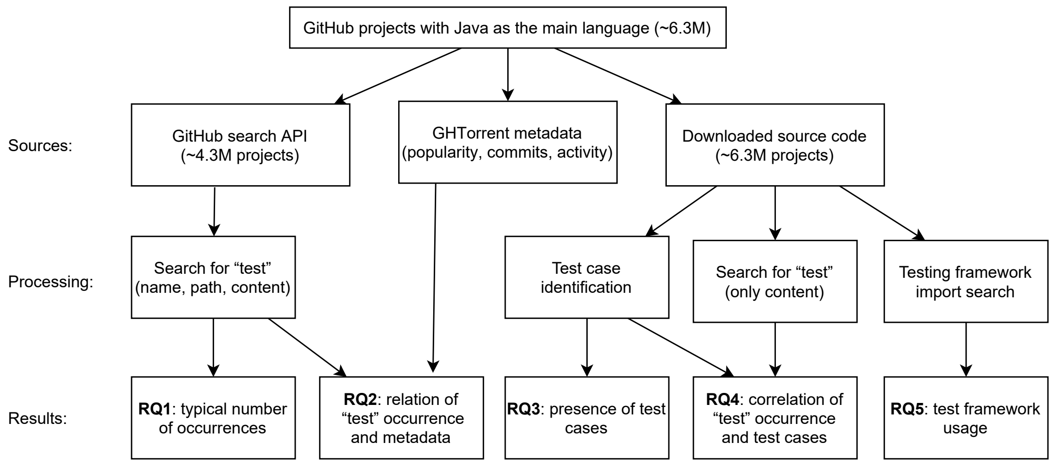
RQ 5. *Which test frameworks and libraries are used and how are they combined in projects?*

In the following sections, we describe the data gathering process, results, and threats to validity.

2. Method

The study was focused on public projects from GitHub (https://github.com/; accessed on 20 June 2021) that had Java as their most used (primary) programming language. To reduce the number of requests for the GitHub API, GHTorrent [16] was used, which mirrors project metadata from GitHub. We used a collection of 6.3 M projects from the May 2019 dump (https://ghtorrent.org/downloads.html; accessed on 20 June 2021); `mysql-2019-05-01` was downloaded, which is described in detail in our previous paper [15].

For a high-level overview of the method, highlighting the connections between data sources, processing techniques, and individual research questions, see Figure 1. To answer the first two research questions, we have used the raw data obtained in our previous study—results of searching for the word "test" using GitHub search API. Because of the limits of the API, only 4.3 M repositories were considered. For other research questions, the search for the word "test" was no longer performed using the GitHub Search API, but it was a part of the script following the rules of GitHub search to keep the data consistency (explained in detail in [15]). Therefore, we were not limited by GitHub API code indexing for search purposes anymore and we were able to analyze more projects, 6.3 M in total.



GitHub projects with Java as the main language (~6.3M)
Sources:
GitHub search API (~4.3M projects)
GHTorrent metadata (popularity, commits, activity)
Downloaded source code (~6.3M projects)
Processing:
Search for "test" (name, path, content)
Test case identification
Search for "test" (only content)
Testing framework import search
Results:
RQ1: typical number of occurrences
RQ2: relation of "test" occurrence and metadata
RQ3: presence of test cases
RQ4: correlation of "test" occurrence and test cases
RQ5: test framework usage

Figure 1. Study overview.

2.1. *Typical Number of "test" Occurrences*

We used 6 different datasets with the results of the search for the word "test" using GitHub API in each project. The datasets are formed based on different search parameters used. First, we used two types of file filtering and searched: (1) only in Java and Kotlin files and (2) in all project files. For each approach, the "test" string was searched in: (1) file content, (2) filename, and (3) file path. As a result, in each dataset, we had the number of the word "test" in each file of a project.

We used these data to find out if there exists some specific number of the word "test" occurrences that is very common among the projects (**RQ 1**). To understand the reasons for such common numbers, we have performed a manual investigation of 100 randomly selected projects from each search sample that has this typical number of occurrences.

Random selection was ensured by MySQL `rand()` (https://dev.mysql.com/doc/refman/8.0/en/mathematical-functions.html#function_rand; accessed on 20 July 2021) function. Each repository was downloaded, opened in an Integrated Development Environment (IDE), and all files with occurrences were analyzed to find out for what purposes the word "test" is used. The analyzed files were classified into the following groups:

- Testing code—occurrences in test cases of application.
- Testing helpers—help classes used for testing.
- Example tests—test not written by a developer but automatically generated by an IDE or a framework as an example.

- Configuration—project's configuration files, e.g., build tool ones (`pom.xml`, `build.gradle`, etc.).
- Production code —occurrences in production code, e.g., name of an identifier, production package name, etc.
- Documentation—occurrences in readme files, comments, or JavaDoc.
- Other files—e.g., data sources, SQL dumps, images, etc.

This way we tried to identify sources of frequently occurring numbers of the word "test" in different projects and datasets.

2.2. Correlation between Occurrence of "test" and Project Properties

To answer **RQ 2**, we created scatter graphs with the number of occurrences of the word "test" on the Y-axis and the creation date, date of the last commit, number of commits, or number of watchers of the project on the X-axis. These data were taken from the GHTorrent dataset. We have also calculated standard Pearson's correlation coefficient [17] for each search sample and project parameter. The correlation coefficient was calculated as follows:

$$r = \frac{\sum(x - m_x)(y - m_y)}{\sqrt{\sum(x - m_x)^2 \sum(y - m_y)^2}} \tag{1}$$

where m_x is the mean of the vector x (number of occurrences of the word "test") and m_y is the mean of the vector y (creation date, date of the last commit, number of commits, or number of watchers).

2.3. Number of Test Cases and Test Framework Usage

To answer the next three research questions, we have used the automatic test case detection script described in our previous paper [15]. The analysis was performed on the downloaded source code archives of all 6.3 M GitHub project with Java as the primary language. The analysis process had the following steps:

1. Get project name from GHTorrent.
2. Download ZIP with the source code of the project from GitHub.
3. Run analysis using the proposed script (https://github.com/madeja/test-overview-in-java/blob/master/ProjectAnalyzer.php; accessed on 20 July 2021).

The process of source code download and analysis takes substantial time, so to process all the projects we have used 34 parallel processes and we ran them from 22 January to 24 February 2021.

As a result, the script provided us the number of actual test cases present in each file of a project, and also the presence of import statements for testing frameworks and libraries. The presence of test cases and their number were used to answer **RQ 3**. To answer **RQ 4**, we compared the data to the number of occurrences of the word "test" in all files content of a project and used the Pearson's correlation coefficient.

To answer **RQ 5**, we searched for import statements of the 29 testing frameworks and libraries. The list of frameworks and corresponding import paths were based on our previous work [15], where we analyzed 50 Java testing frameworks and their properties. For this study, we excluded frameworks without identified import (mostly archived ones or test generators) or without the obligation to use the word "test", due to searching only in files with "test" presence in the content.

3. Results

A total of 340 M classes containing the word "test" in their content were analyzed by the automated script, in which 1124 M test cases were found. The whole dataset of the analysis is available at Zenodo (https://doi.org/10.5281/zenodo.4566740; accessed on 20 July 2021) (some data are shared with the dataset (https://doi.org/10.5281/zenodo.4566198; accessed on 20 July 2021) from our previous paper [15]).

3.1. Typical Number of "test" Occurrences

Figure 2 shows the number of occurrences of the word "test" in all projects analyzed via the Github API. Zero occurrences were skipped to make the graph more readable. It is possible to see that in *all files content* the typical number of occurrences of the word "test" was 4, in other search types it was 2.

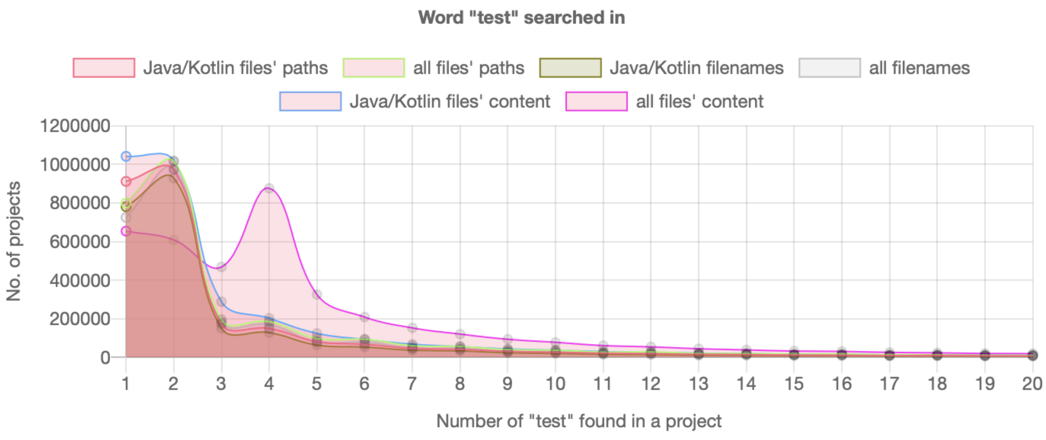


Figure 2. The number of occurrences of the word "test" for all Java primary language projects at GitHub.

Manual investigation of results of 600 randomly selected projects (100 for each search type) with typical numbers of "test" occurrences can be seen in Figure 3. In all search types a high incidence of *example tests* (58% occurrences of 600 manually analyzed projects), that are mostly generated by IDE or framework, was observed. They do not test the application functionality; therefore, they are not considered as real executable tests because they are not actually executed, e.g., in CI/CD. Example tests mostly occurred in Android projects with classes `ExampleUnitTest`, `ExampleInstrumentedTest`, or `ApplicationTest`. For non-Android projects, there were mostly test cases implemented as skeleton methods.

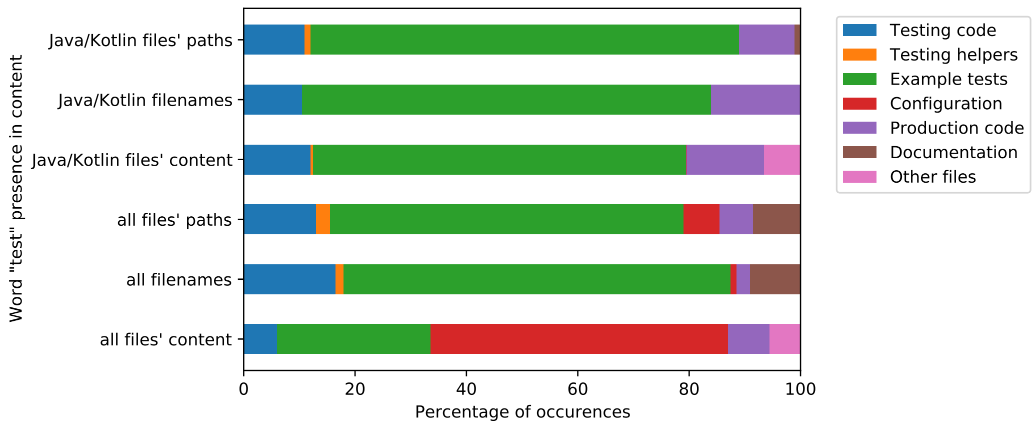


Figure 3. Usage proportion of the word "test" by file type in projects with 4 occurrences for search in all files content and 2 occurrences in other searches.

The second most common place of finding the word "test" was the *testing* code, which is the main objective we search for. As can be seen, only a small portion of occurrences represent such tests. This means that the peak values of "test" occurrence are mostly caused by automatically generated tests and not by actual tests of the production code.

The "test" string was also used in the *production code* for

- project's package name,
- identifiers, where "test" denotes "examine" or "validate" something,
- debugging or logging.

Although some occurrences in the production code were used to verify application functionality, they were intended for manual testing or debugging of the code. As these were not automated tests, such occurrences were included in the production code group. Presence in *configuration* files was high for *all files content* search due to files such as `pom.xml` or `gradle.build`, where the testing process was mostly configured, e.g., testing directories, framework, etc. Other places of finding were *documentation* (e.g., readme files, JavaDoc, comments), *test helpers* (e.g., abstract testing class), or *other files* (data, SQL dumps, images, dynamically generated content, etc.).

RQ 1 *Is there a typical number of the word "test" occurrences across a project that is observed in a large number of projects? If so, what is the reason behind such specific number?*

Yes, the most frequent occurrence of the word "test" is 4 for *all files content* search and 2 for *filename*, *path*, and *java files content* search. A manual investigation of randomly selected repositories showed that 58% of such search results were *example test cases*, which are mostly responsible for such typical occurrences in projects. For *all files content* search, there was the most occurred combination of two *example tests* and two *configuration files*. Test cases were found in 11% of all results and 9% occurred in production code.

3.2. *Occurrence of "test" and Project Properties*

To answer **RQ 2**, we created scatter graphs according to creation date, date of the last commit, number of commits, or number of watchers of the project (see Figure 4). Presented graphs are focused on the occurrences of the word "test" in all java files' content, because this method is able to find the most test cases in an unknown project code, regardless of the framework used. Results of other search types can be found in Appendix A. The graphs do not show projects over 600 occurrences of the searched term to make them more readable (loss of 0.1% of all projects). Regression lines included in the graphs can be used to estimate the number of "test" occurrences based on different project's properties. Pearson's correlation coefficient is calculated in each graph as *pearsonr* with a p-value to express the statistical significance of the correlation coefficient.

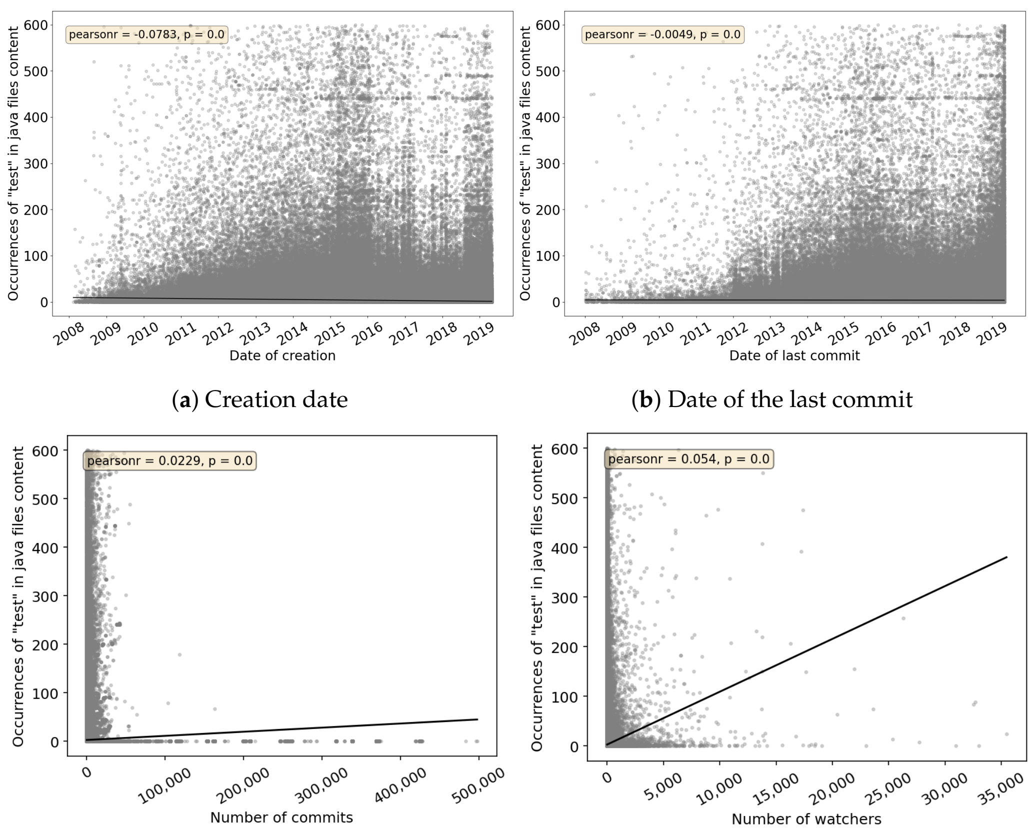
It can be seen from the graphs that there is no strong correlation between the observed properties of the project and the number of occurrences of the word "test" in all java files' content. Low correlation is mostly caused by a huge portion of projects with zero occurrences of the searched string. Considering the novelty of projects via creation date, we see increased incidence in 2015 and mid-2018 (Figure 4a). During these years we see an increased number of projects with a higher occurrence, i.e., more than 100. At the same time, considering the date of the last commit, the number of projects with a high number of "test" occurrences grows with the date of the last commit. This situation is probably related to the maintenance of the project because from a long-time perspective it is difficult to keep the project reliable and stable without automated tests. Thus, many newly created projects do not include tests at all (lack of the word "test" in a project), but the tests are implemented in the following years of the project life (presence of the word "test" in a project).

Although for the popularity (number of watchers), the highest Pearson's coefficient of all 4 observed project properties was reached, the correlation is still very weak. From this point of view, developers do not generally follow projects that are potentially more reliable (more tested). On the other hand, this fact can be related to our previous findings [15] that many projects with a high presence of the "test" string in the java file content are automatically committed from other Version Control Systems (e.g., Apache Subversion), where copied directories are used for branching, or they are just copies of other projects without the "fork" relation. Therefore, such projects are probably not followed by developers.

Observing the number of projects' commits revealed the fact that projects with >80,000 commits have a prevalence of zero "test" occurrences. The presence of the searched term varied in projects with common (lower) commit activity, therefore, it is not possible to conclude a common behavior of developers.

RQ 2 *Is occurrence of the word "test" related to the creation date, date of last commit, number of commits, or number of watchers of a project?*

No strong relation was found between the number of occurrences of the word "test" in all java files' content and the mentioned project properties. On the other hand, we observed that the number of projects with a higher "test" presence grows with the project's last commit date. Occurrence related to creation date was higher in 2015 and mid-2018, but not pointing to any trend or rule.



(**a**) Creation date (**b**) Date of the last commit

(**c**) The number of commits (**d**) The number of watchers

Figure 4. Correlation between the number of occurrences of the word "test" in all java files' content and the project's life cycle parameters.

3.3. Test Case Presence and Correlation with "test" in File Content

In Figure 5, it is possible to see the number of projects containing test cases in a specific quantity. Comparing this to Figure 2 we see a very similar course as in the occurrence of the word "test" in *java file content*. At the same time, we can notice that there are peaks in the same places. Example tests were not excluded and were considered relevant test cases, as it is still impossible to automatically detect example test cases. In order to detect example tests, we would need to manually decide if the particular test case verifies some functionality of production code or not. This process could be automated if we could identify Unit Under Test (UUT) from a test case, because then we would be able to distinguish whether a particular UUT is in the production code or not.

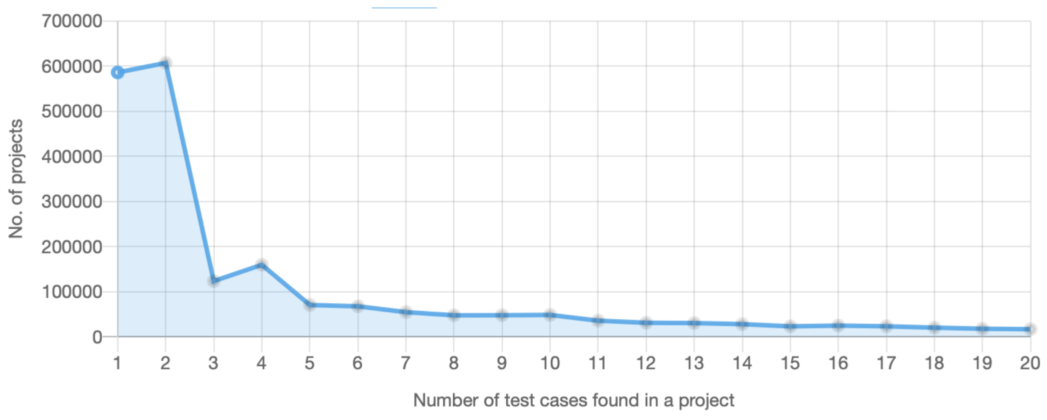


Figure 5. The real number of test cases in all GitHub projects with Java as a primary language.

In total 51.57% of all projects included at least one test case. On the other hand, considering the results of Section 3.1 where example tests consisted of 67% for *java files content*, we can estimate that in projects with two or less test cases, only 33% of them are non-example ones. Therefore, we can calculate the percentage of projects containing at least one test case without example ones (we only considered example tests of projects with two or less test cases):

$$P = \frac{N_{real} - N_{unreal}}{N_{all}} = \frac{3{,}239{,}308 - 1193{,}659 * 0.67}{6{,}280{,}818} = 38.84\% \tag{2}$$

where:

N_{real}—number of projects containing an actual test case;
N_{unreal}—number of projects with two or less test cases of which all are example ones;
N_{all}—all analyzed projects.

Of course, this is only an estimate, as even projects with a higher number of test cases can include also example ones.

RQ 3 *What is the ratio of open-source projects that contain at least one test case?*

51.57% of 6.3 M analyzed projects included at least one test case. Considering that probably only 33% of test cases detected in projects with two detected test cases are not example ones, we estimate that only 38.84% of projects include at least one non-example test case.

Figure 6 presents the correlation between all occurrences of the word "test" in *java files' content* of a project and the detected number of actual test cases in the particular project by the proposed script. Compared to the correlation in projects with a high incidence of the word "test" ($r = 0.655$, see [15]), the presence of test cases is lower, reaching Pearson's correlation coefficient of $r = 0.0253$, meaning a poor correlation. Excluding forked projects, the correlation was $r = 0.0393$, so the influence of forks is minimal. The result is probably caused by the occurrence of irrelevant projects, such as testing ones, homework repositories, clones, etc.

There is a demand to identify the relevance of projects. Some tools for this task already exist, e.g., *reaper* [18], but they are very computationally intensive. Our testing [15] shows that the mentioned tool is slow to use on a large number of projects; therefore, this tool was not usable for this study.

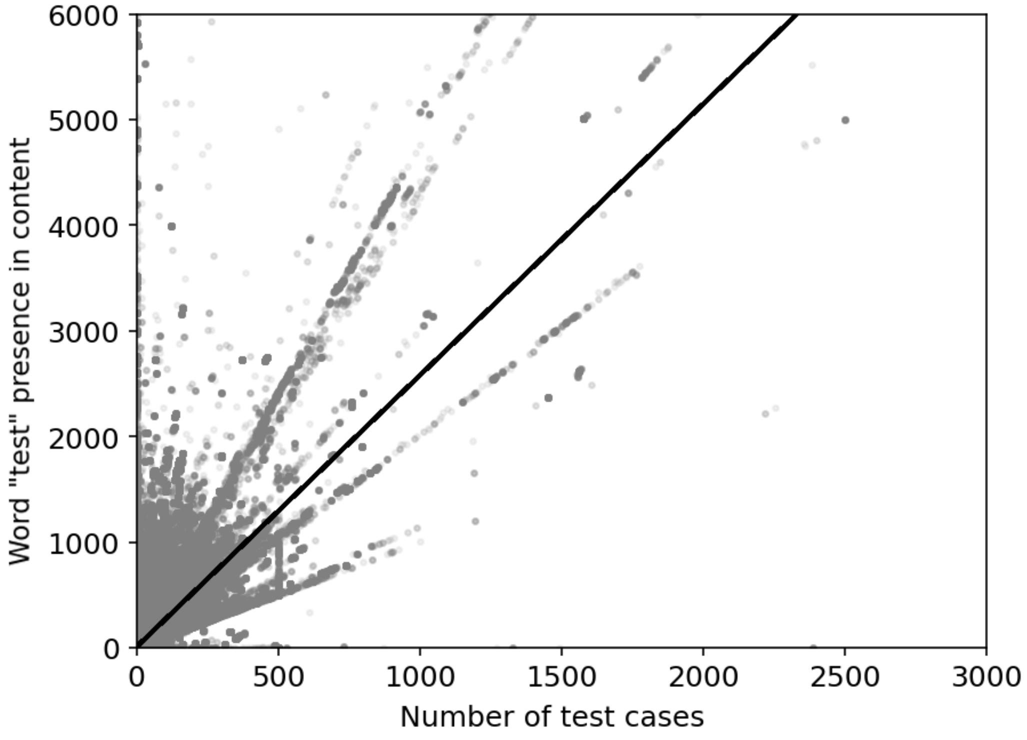


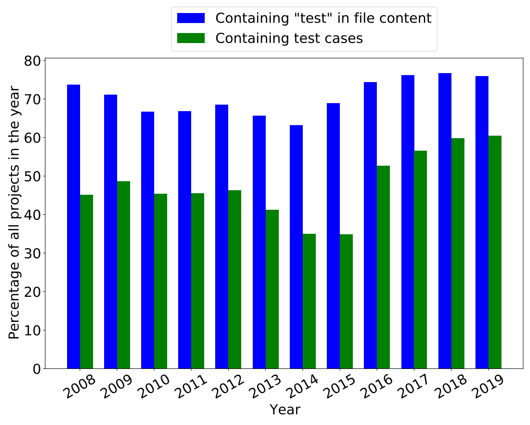
Figure 6. Correlation between the number of the "test" word occurrences in the all java files' content of a project and the number of test cases in the particular project in all GitHub projects with Java as a primary language.

RQ 4 *What is the general correlation between the number of occurrences of the word "test" in all files' content of a project and the number of actual test cases in the project?*

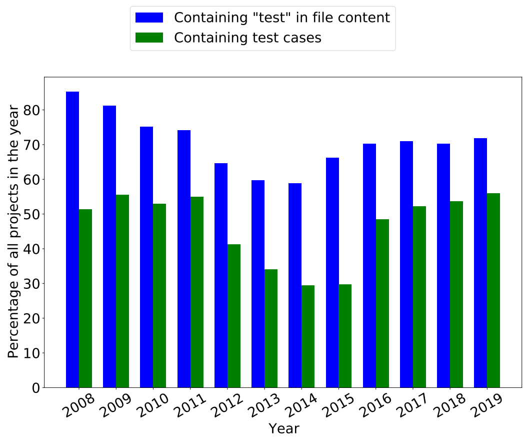
There is a poor correlation due to reached Pearson's correlation coefficient of $r = 0.0253$. The negative impact of forked projects was not confirmed, so the huge decrease in correlation was probably caused by irrelevant projects.

3.4. Number of Projects Potentially Containing Tests

Discussion in this section is an extension of **RQ 4**. Despite we found no correlation between the number of occurrences of the word "test" and the number of test cases, we decided to analyze the relation of the presence of this word and the presence of test cases. We know that most Java testing frameworks require the word "test" to implement a test case [15]. When considering this rule, the presence of this word in the project will in most cases also indicate the presence of tests. Figure 7 shows a percentage comparison of projects with non-zero "test" occurrences in files content and test case presence. Remember that data were collected in 2019 and the year 2019 does not include all projects created/committed, but only until 30 April 2019.



(**a**) By the year of creation.



(**b**) By the year of the last commit.

Figure 7. Year percentage comparison of projects with non-zero "test" occurrences and test cases.

It can be seen in Figure 7a,b, that annually approximately 60–80% projects include "test" in the files' content with a stabilized ratio of approximately 75% for the last four years. When comparing the proportion of projects with the occurrence of the word "test" and projects involving test cases, the annual difference median is 21.15% by year of creation and 23.54% by year of the last commit. These numbers mean that annually approximately 22% of projects contain the word "test" in the absence of an executable test case. We can consider it as a *false positive* indicator of test case identification using "test" occurrence in the project. Compared to results presented in Section 3.2, there is no increase of the searched term occurrences due to the last commit.

3.5. Testing Frameworks Usage

The use of testing frameworks can have a significant impact on the writing of tests. The number of particular framework imports found across all projects' java files is shown in Figure 8. Remember that imports were searched only in files that contained the word "test", i.e., with possible usage in a test class. For this reason, imports of some frameworks were not found, e.g., *etlUnit*, *GrandTestAuto*, and *BeanTest*. The number of imports expresses how many files (most often java classes) use the given framework. We can notice a huge prevalence of the *JUnit4* + *JUnit5* frameworks. (We divided JUnit into two groups due to different test case notation. JUnit3 uses "test" in the method name, and JUnit4+ uses the annotation @Test via its in-place binding [19] to the test method (currently only JUnit4 + JUnit5)). It is followed by *Mockito*, *Spring Testing Framework*, *Hamcrest*, and *JUnit3*. *TestNG* is often considered the second most popular framework, the occurrence of its imports is the 6th most common. On the other hand, it is necessary to take into account that, e.g., *Spring Testing Framework* is purpose-dependent, *Mockito* is a part of unit tests, but is not an equivalent to a full-fledged framework such as *JUnit* or *TestNG*.

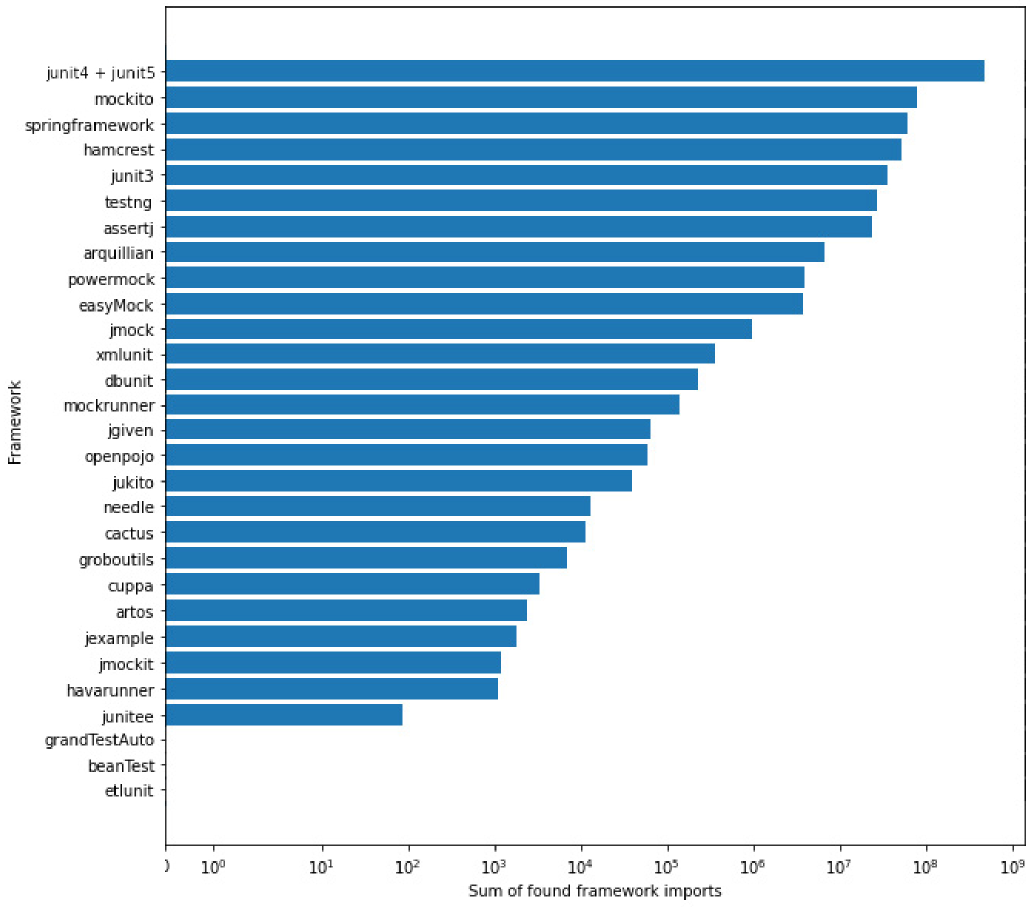


Figure 8. The number of found framework imports in all java files (logarithmic scale).

Figure 9 shows how many projects use a particular testing framework. Regardless of version, *JUnit* is used in 48% of projects. *TestNG* as the second most popular Java unit testing framework has a share of only 3%. In Table 1, percentages of all simultaneously used frameworks can be seen. Similar amounts of occurrences are grouped by colors for faster identification of simultaneously used frameworks with similar presence in the projects. We can notice that a large number of projects are used simultaneously with *JUnit3* and *JUnit4*/*JUnit5*, probably because the projects were created with *JUnit3*, and after the release of *JUnit4*, the tests were written in a new style. At the same time, *Mockito* is very often used with both *JUnit* versions. It is also interesting to pay attention to frameworks in which frequent zero percentages occurred. For example, *Artos* and *HavaRunner* occur only with *Hamcrest* and *JUnit*. It is clear from this that these two frameworks depend on *Hamcrest* which depends on *JUnit*. Therefore, from Table 1 it is possible to observe how particular frameworks are interdependent.

A detailed look at all simultaneous usages of the 5 most used frameworks is presented in Figure 10. *JUnit* versions were joined under one group because they are part of the same family. It is important to note that the percentages only take into account the frameworks shown in the graph, i.e., not with all searched frameworks. It can be seen that *JUnit* is mostly used alone, without any other frameworks. Its competitor *TestNG*, on the other hand, appeared never alone, but most often together with *JUnit*. This is probably because *JUnit* is used by default and *TestNG* is added manually by the developer to the project, without having to completely remove the competing framework. *Hamcrest* is often used solely with *JUnit* or with *JUnit* + *TestNG*. The presence of all five frameworks in the project is more than 1%, which is quite a high proportion compared to other combinations.

Table 1. Testing frameworks used simultaneously.

	junit3	junit4	testng	artos	arquillian	havarunner	jexample	assertj	hamcrest	xmlunit	cactus	cuppa	dbunit	easyMock	groboutils	jgiven	jmock	jmockit
junit4 + junit5	9.59016																	
testng	1.01206	1.39472																
artos	0	0.00008	0															
arquillian	0.40252	0.63740	0.23039	0														
havarunner	0.00002	0.00008	0	0	0													
jexample	0.00014	0.00053	0	0	0	0												
assertj	0.94495	3.29287	0.54514	0	0.24464	0	0											
hamcrest	4.18828	9.67182	1.01032	0.00003	0.40580	0.00005	0.00008	1.91533										
xmlunit	0.27999	0.29350	0.23651	0	0.15430	0	0	0.18731	0.27944									
cactus	0.01785	0.01302	0.00158	0	0.00698	0	0	0.00002	0.00695	0								
cuppa	0	0.00032	0.00018	0	0	0	0	0.00032	0.00013	0.00013	0							
dbunit	0.12417	0.17079	0.01697	0	0.00676	0	0	0.01633	0.11108	0.00347	0.00097	0						
easyMock	1.06113	1.32477	0.30343	0	0.17063	0	0	0.20254	0.81690	0.13797	0.00117	0	0.01403					
groboutils	0.01184	0.01224	0	0	0	0	0	0.00133	0.00403	0	0	0	0.00006	0.00665				
jgiven	0.00174	0.00436	0.00101	0	0.00018	0	0	0.00276	0.00240	0	0	0	0	0	0			
jmock	0.30995	0.44509	0.10624	0	0.00900	0	0.00014	0.17348	0.34444	0.00383	0.00088	0	0.01443	0.05858	0	0.00032		
jmockit	0.00075	0.00089	0.00045	0	0	0	0	0.00037	0.00080	0	0	0	0	0.00034	0	0	0.00089	
jukito	0.14024	0.14900	0.13586	0	0.13534	0	0	0.13955	0.14248	0.13502	0	0	0.00083	0.13610	0	0.00016	0.00054	0
junitee	0.00008	0.00008	0.00005	0	0	0	0	0	0.00002	0	0.00003	0	0	0.00002	0.00002	0	0	0
mockito	4.49015	9.67778	1.18571	0	0.35171	0	0.00006	2.33805	5.59345	0.27960	0.00332	0.00030	0.12530	0.64240	0.00634	0.00211	0.28067	0.00080
mockrunner	0.06369	0.08071	0.01024	0	0.00014	0	0.00002	0.03400	0.05126	0.00005	0.00005	0	0.00102	0.01782	0.00623	0	0.00703	0
needle	0.01547	0.01574	0.01448	0	0.00094	0	0	0.00264	0.01460	0.00016	0	0	0	0.00102	0	0	0	0
openpojo	0.00978	0.02729	0.00900	0	0.00166	0	0	0.00321	0.01622	0.00008	0	0	0.00003	0.00010	0	0	0.00002	0
powermock	1.09964	1.68869	0.30558	0	0.17527	0	0	0.37899	1.12151	0.14797	0.00163	0	0.05199	0.40081	0.00185	0.00059	0.02696	0.00042
springframework	1.81781	9.47724	0.67249	0	0.26236	0	0	1.81246	3.30872	0.24702	0.00692	0	0.12215	0.41259	0.00307	0.00241	0.20013	0.00038

	jukito	junitee	mockito	mockrunner	needle	openpojo	powermock
junitee	0						
mockito	0.14561	0.00002					
mockrunner	0	0	0.06650				
needle	0	0	0.01462	0			
openpojo	0.00002	0	0.02453	0.00035	0		
powermock	0.13564	0.00002	1.55510	0.01794	0.00847	0.01392	
springframework	0.13804	0.00002	3.22824	0.03247	0.01441	0.01037	0.52661

Legend of colors:
Occurrence in range (0, 0.0001)
Occurrence in range <0.0001, 0.001)
Occurrence in range <0.001, 0.01)
Occurrence in range <0.01, 0.1)
Occurrence in range <0.1, 1)
Occurrence in range <1, 100>
Note: All numbers in the table are in %.

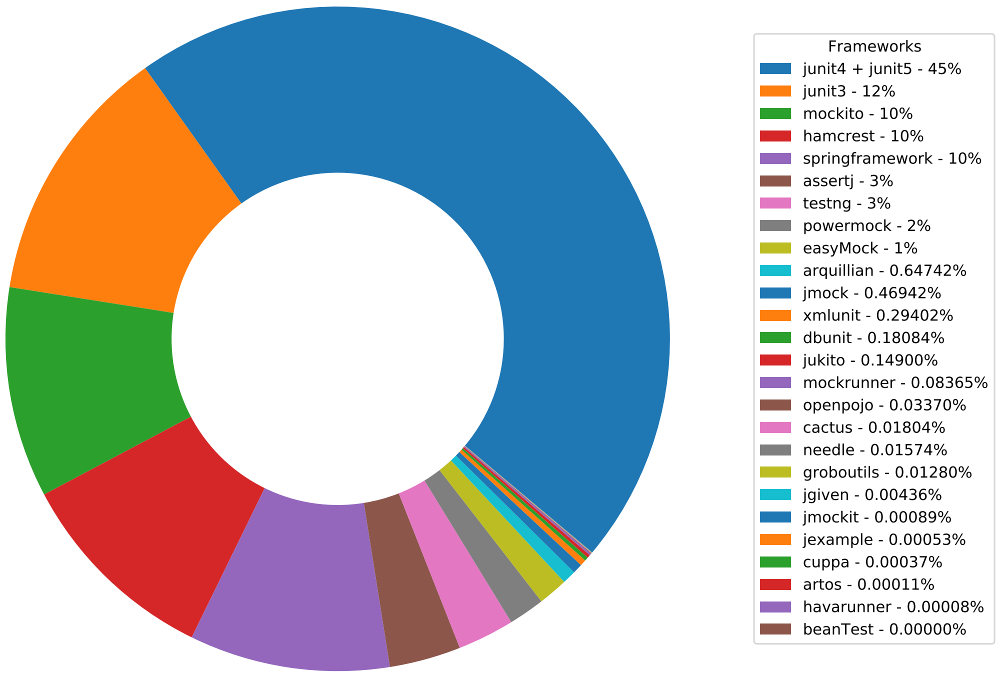


Figure 9. The most used Java testing frameworks in projects.

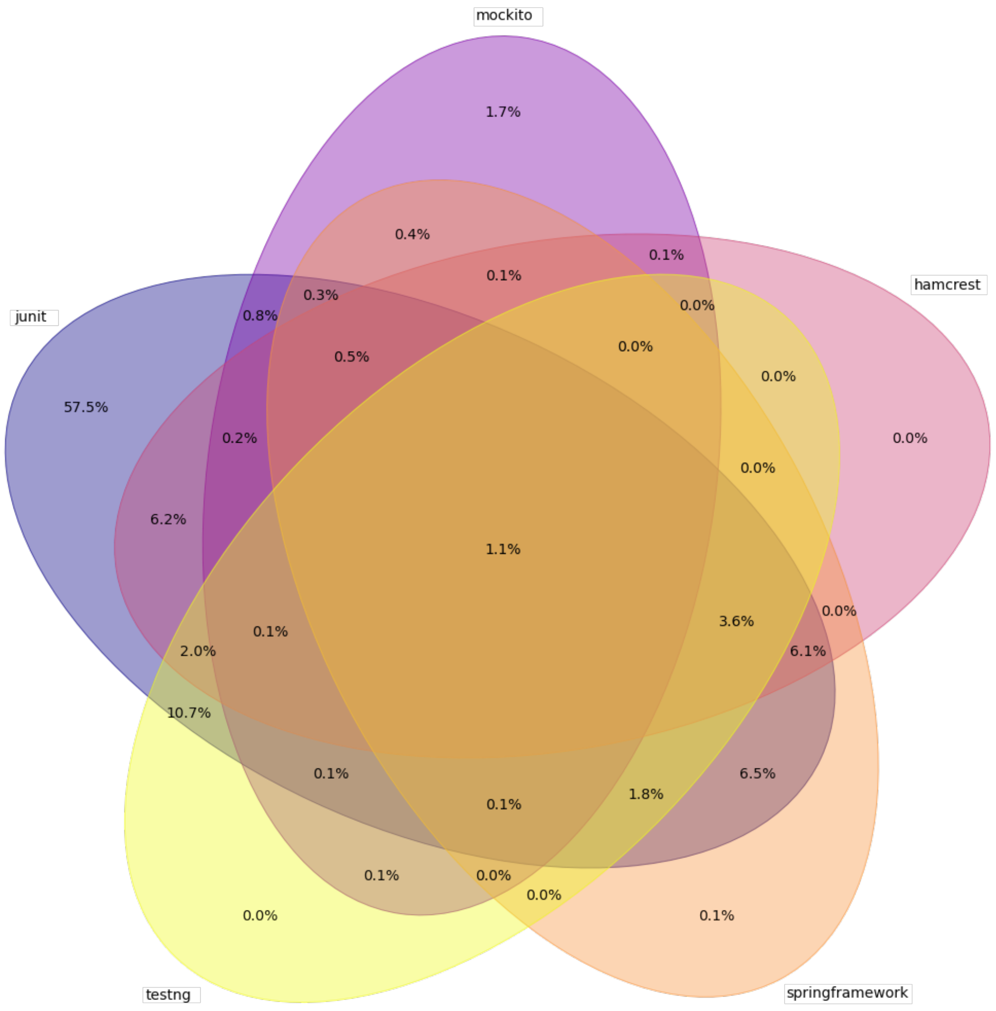


Figure 10. The first 5 simultaneously used testing frameworks.

RQ 5 *Which test frameworks and libraries are used and how are they combined in projects?*

The most widely used framework is *JUnit*, imported in 48% of projects (2.84% solely JUnit3, 35.53% solely JUnit4 with JUnit5, 9.59% JUnit3 together with JUnit4 and JUnit5), which is most often combined with *Mockito* (more than 10% of projects). *Spring Framework* tests and *Hamcrest*, which are also very popular, have a relatively large usage share too. *TestNG*, considered the second most popular Java testing framework, has only 3% share and is almost never used without *JUnit*, which is a very surprising result. A detailed overview of simultaneous usage of frameworks is given in Table 1.

4. Threats to Validity

In this section, we will discuss threats to the validity of this study, as suggested by Wohlin et al. [20].

4.1. Internal Validity

Answering **RQ 1** and **RQ 2** relied on the GHTorrent databank and GitHub API search algorithm. For all RQs, only projects with Java as a primary language were selected; therefore, testing practices out of this scope (e.g., Java used as a minority language) could have been lost. Our study was limited to projects containing the word "test" in different places and was depended on third-party frameworks. It is necessary to further investigate customized testing solutions, that are mostly represented by small Java programs which test the production code and do not use any official testing framework. The implementation of such programs is often significantly different and it is difficult to identify test cases. It must be taken into account that in this study the relevance of the projects was not taken into account, but all projects were analyzed, i.e., copies, forks, example projects, etc.

4.2. External Validity

To provide generalizable results, we analyzed a huge number of Java projects due to the popularity of Java, in order to achieve high variability of tests of different developers. The results can be used as a basis for further development of tools for test development and code quality improvement. Despite the presented observations, our findings, as is usual in empirical software engineering, may not be directly generalized to other systems, particularly to commercial or to the ones implemented in other programming languages.

5. Related Work

An overview of similar studies and their comparison with our study is presented in Table 2. As can be seen from the table, the main contributions of this paper compared to related work are the following:

- analysis on the largest sample of 4.3 M projects;
- considering the largest number of 26 unit testing frameworks;
- the finest granularity of the identification of a test case (at the level of the test method);
- platform-independent focus on Java programming language.

In the following paragraphs, we describe related work in more detail.

GitHub is often used for many studies to identify practices in independent projects. The quality and reliability of a project can be evaluated from several perspectives. In 2013, Kochhar et al. executed very similar empirical studies [1,21] of the adoption of testing in open source projects at GitHub, independently of programming language. They found out that 61% of the analyzed repositories include at least one test case, that projects with a larger number of developers have more test cases, and that number of test cases has a weak correlation with the number of bug reporters. We extend their study from 50,000 to millions of analyzed repositories, but solely focused on projects with Java as a primary language. At the same time, we search for a big set of testing frameworks and compare their usage in projects.

Table 2. The novelty of the study compared to related work.

Paper	Data Collection	Project Type	Source	Project Filtering	Number of Analyzed Projects	Analysis Level	Project Contains Tests When	Considered Unit Testing Frameworks	Projects Containing Unit Tests
Kochhar et al. [1]	semi-automated	independent	GitHub	random projects with manual review	20.8k	testing class	a file contains the word "test" in filename	not considered	61.65%
Kochhar et al. [21]					50k				42.66%
Cruz et al. [14]	automated	Android	F-Droid + Github	projects from F-Droid with code placed at GitHub	1k	project	a file contains import of a testing framework or it is configured in a config file	4	39.00%
Pecorelli et al. [11]	semi-automated	Android	F-Droid + Github	projects from F-Droid with code plased at GitHub	1.8k	testing class	a file contains "test" as prefix or suffix and observer labels the class as test	all identified by manual investigation	41.00%
Lin et al. [22]	automated	Android	GitHub	1. repository contains one AndroidManifest.xml 2. build.gradle contains "com.android.application" 3. 2 < declared components in the manifest 4. package name in an app market	12k	method	a file in "test" or "androidTest" directories containing @Test annotation	3	6.10%
this paper	automated	Java	GitHub	all projects with Java as primary language	4.3M	method	a file contains the word "test" in the content with a test method identified by static analysis [15]	26	51.57%

There was a similar study by Pecorelli et al. focused on the effectiveness of test cases on 1780 open-source Android applications [11]. They found that 59% of applications do not contain any test at all. The rest of the applications had tests with poor quality and very low effectiveness based on all computed metrics, and they are very likely to miss faults in production code. Authors claim that there may be a need to define novel indicators to better quantify the quality of test cases. Spadini et al. found that 71% of production code is more likely to contain defects when tested by smelly tests (poorly designed tests) and also it negatively impacts program comprehension [12].

Another study oriented towards Android applications was executed by Lin et al., where more than 3.5 million GitHub repositories were analyzed, and they identified more than 12,000 non-trivial and real-world Android apps [22]. They observed that only 8% of applications have any automated tests, only 6% unit tests and UI testing is less adopted than unit testing (4%). They conducted surveys to support their observations and found that developers tend to follow the same test automation practices across applications and popular projects are more likely to adopt test automation practices. However, they searched only projects with JUnit-based testing frameworks such as JUnit, Robolectric, Mockito, and Espresso.

Beller et al. [23] created a dataset of Travis CI usage in GitHub open-source projects, where they analyzed more than 1000 projects. They performed a general-purpose analysis of all build logs, and they also searched for outputs of common testing frameworks, as they are a common part of the build process. The framework detection is limited by the used build tools (Maven, Gradle, Ant) and frameworks (JUnit, PowerMock, Mockito, TestNG). They also gathered test results, i.e., how many tests were passed, failed, errored or skipped, execution time, etc.

Lemay [24] observed 1746 publicly available Java GitHub projects to infer usage and usability issues of developers with the Java language. He tried to measure the quality of programming languages. The study analyzed control flow statements, literal, operator usage, and null checks. Studying such repository properties can be beneficial for further language development. In our study, we focus on tests and the possibility of their identification.

The effect of programming languages on software quality was investigated by Ray et al. [25]. In 729 repositories they used a mixed-methods approach, combining multiple regression modeling with visualization and text analytics, to study the effect of language features such as static v.s. dynamic typing, strong v.s. weak typing on software quality. Considering effects such as team size, project size, and project history, they report that language design does have a significant, but modest effect on software quality. Strong typing is modestly better than weak typing, and among functional languages, static typing is also somewhat better than dynamic typing. They also found that functional languages are better than procedural ones.

Zerouali and Mens [13] reviewed the use of testing frameworks in 4532 open-source projects. They studied how frequently specific libraries are used over time and how they are used simultaneously. They analyzed only 13 testing frameworks, while we analyze 26 ones on a much larger sample of projects.

6. Discussion

The long-term goal of our research is to use information about the semantics and structure of tests to enrich production code with such information to improve program comprehension. In this way, software development will be cheaper and more reliable because a developer who comprehends the code can implement the specified functionality faster and with fewer errors. To use this idea, it is necessary to discuss how many software projects use tests. Because testing frameworks affect the placement of usable information for such enrichment, we also investigated what frameworks are used and how they are combined.

Because in the existing research the percentage of projects containing tests varied too much (39–62% of the analyzed projects, see [1,14]), in this paper, we executed a study with the largest dataset of 4.3 M projects for such analysis. To process so many projects, we used a script for static analysis [15]. Despite the achieved result that 52% of projects contain tests, we found that many of the test cases are example ones and do not contain actual information about the production code. These tests are generated, they do not contain relevant information to enrich the production code (e.g., a test containing `assert(true)` has no added value for code comprehension). Therefore, we need to look for a method for the detection that a test is beneficial for comprehension.

Because the word "test" is often used in testing frameworks, it is assumed that we will also find tests when searching for this word (see results in [15] with reached correlation $r = 0.655$). This method is also often used to search for tests, e.g., searching classes in the "test" directory or searching for classes containing "test" in the filename. This paper shows that globally there is a very low correlation between the number of the word "test" occurrences and the number of test cases in a project. We searched also for correlations between the number of the word "test" occurrences and creation date, date of the last commit, number of commits, or number of watchers, which could help search for projects containing test cases, but we found a very low correlation again, therefore, these approaches are not usable to reliably estimate the number of tests in a project.

We know from the study what frameworks are used and how they are combined. This will allow better targeting of tools that help create tests, generate documentation, or understand code to better target these tools for specific frameworks. For our goal of using the information from the tests, it will be possible to focus on the most used frameworks and other frameworks mostly combined with them. This will allow us to use the right information to help the developer comprehend the testing or production code.

7. Conclusions

In this paper, a study of the presence of test cases and the use of different frameworks in a huge set of GitHub projects using Java as the primary language was performed. In total, 6.258 M projects and 340 M of classes containing the word "test" in its content were analyzed, in which 1124 M test cases were found. Manually, 600 randomly selected projects containing a typical number of the word "test" occurrences were analyzed in different parts of the code, in a total of six groups. We calculated a general correlation between the word "test" in file content and the number of test cases in it. At the same time, we created an overview of testing frameworks usage, how they are combined and how they are dependent on each other. We summarize the most interesting findings from our study:

- Manual investigation of randomly selected repositories with a huge number of the same occurrences of "test" showed that 58% of such search results are *example test cases*.
- There is no strong relation between the occurrence of the word "test" and creation date, date of the last commit, number of commits, or number of watchers.
- A total of 51.57% of projects include at least one test case.
- There is a poor correlation between the number of occurrences of the word "test" in the file content and the number of test cases in such file ($r = 0.0253$).
- *JUnit* is used in 48% of projects and is mostly used with *Mockito*, *Spring Testing Framework*, and *Hamcrest*.
- *TestNG*, considered the second most popular Java unit testing framework, occurred only in 3% of projects and was never used without *JUnit*.

Based on the findings, the following additional contributions are concluded:

- Searching for "test" in file content cannot be directly used to estimate the number of test cases.
- Test cases are present in more than half of the projects, so mining information from tests about the production code is promising.
- An overview of testing frameworks usage and ways how they are combined in projects will allow practitioners and researchers to focus on the most used ones.

In future research, we will focus on the analysis of example tests and their automatic detection, as these tests do not really test the production code and, therefore, do not improve code quality. There is also a need to streamline tools detecting industrial projects with the possibility of excluding irrelevant ones, such as reaper, which was time-consuming to use on a large sample of projects (see our experience with this tool in paper [15]). It is also necessary to evaluate the impact of the presence of automated tests on code quality in proprietary projects, as the results may be diametrically different compared to open-source solutions.

Author Contributions: Conceptualization, M.M.; Data curation, M.M.; Formal analysis, M.M.; Investigation, M.M.; Methodology, M.M., J.P., S.C., M.S., and F.G.; Software, M.M.; Visualization, M.M.; Writing—original draft, M.M.; Writing—review and editing, M.M., J.P., S.C., M.S., and F.G. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Project VEGA No. 1/0762/19 Interactive pattern-driven language development.

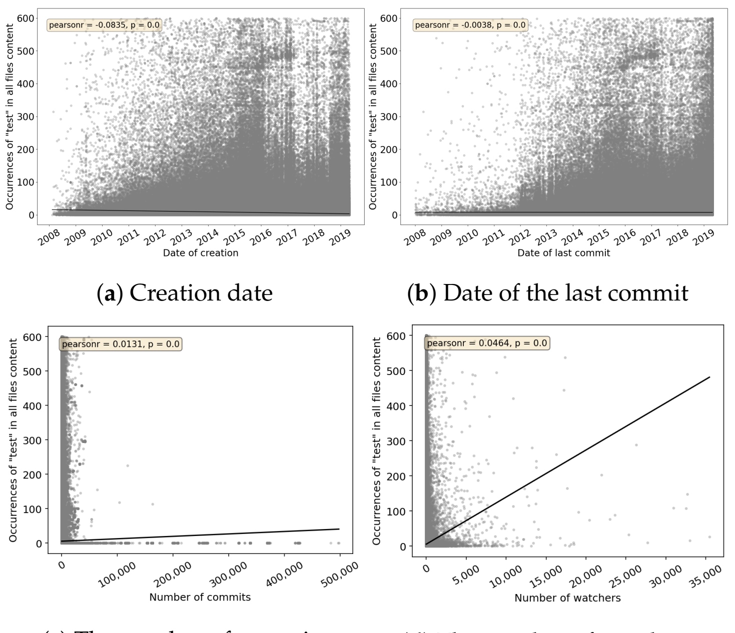
Data Availability Statement: Data supporting reported results can be found at https://doi.org/10.5281/zenodo.4566740 (accessed on 20 July 2021) and partially at https://doi.org/10.5281/zenodo.4566198 (accessed on 20 July 2021).

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Appendix A. Additional Graphs of "test" Presence and Project's Properties

This appendix is an extension of Section 3.2 as a result of deep mining Java projects of GitHub. We present correlation graphs of creation date, date of the last commit, number of commits, or number of watchers and:

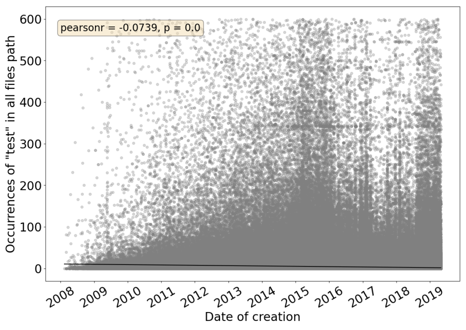
- "test" string occurrences in file content (all files, Figure A1),
- "test" string occurrences in file path (all files, Figure A2 + solely java files, Figure A3),
- "test" string occurrences in the filename (all files, Figure A4 + solely java files, Figure A5).



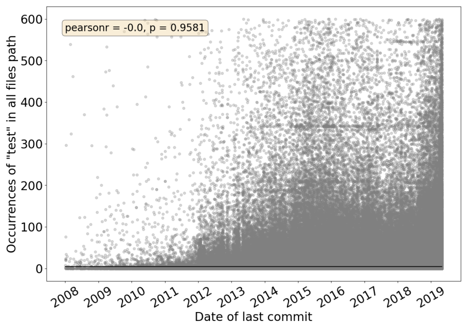
(**a**) Creation date (**b**) Date of the last commit

(**c**) The number of commits (**d**) The number of watchers

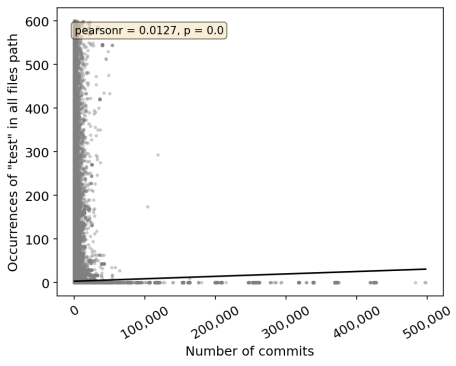
Figure A1. Correlation between the number of occurrences of the word "test" in all files content and project's life cycle parameters.



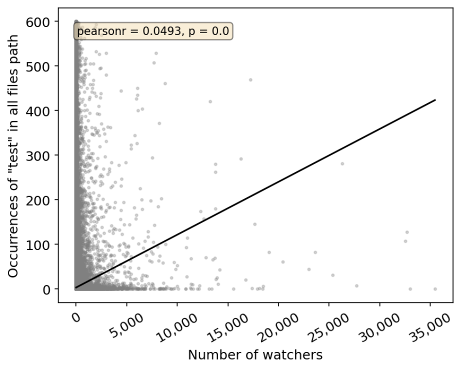
(**a**) Creation date



(**b**) Date of the last commit

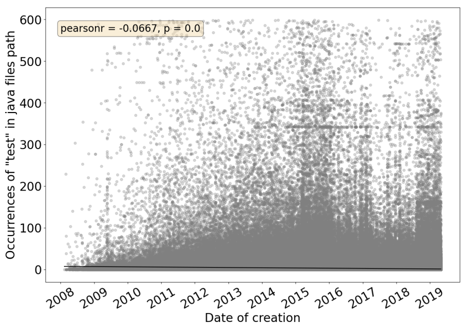


(**c**) The number of commits

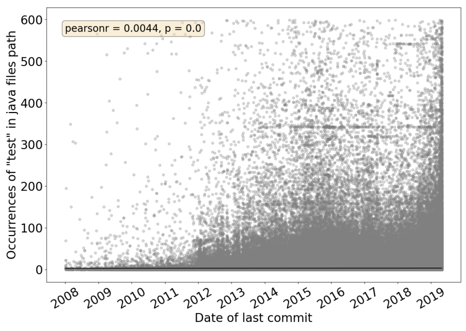


(**d**) The number of watchers

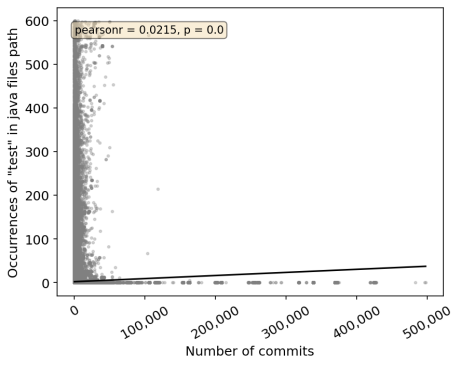
Figure A2. Correlation between the number of occurrences of the word "test" in all files path and project's life cycle parameters.



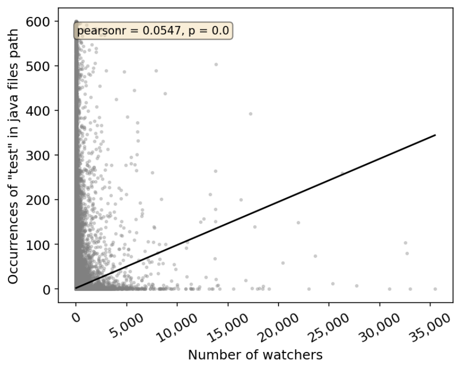
(**a**) Creation date



(**b**) Date of the last commit

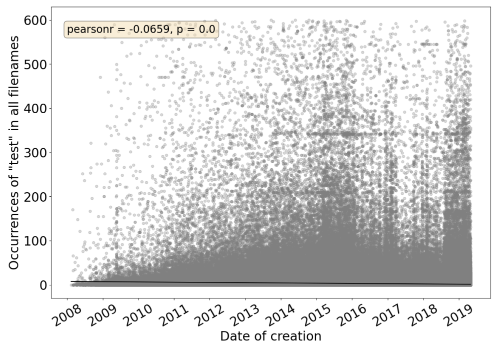


(**c**) The number of commits

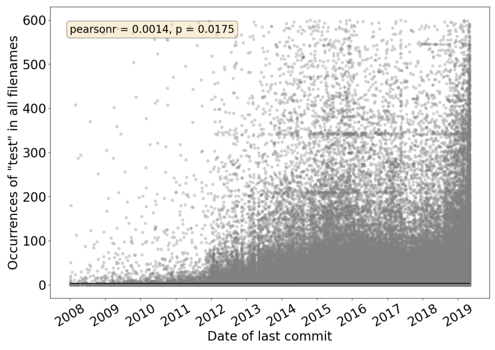


(**d**) The number of watchers

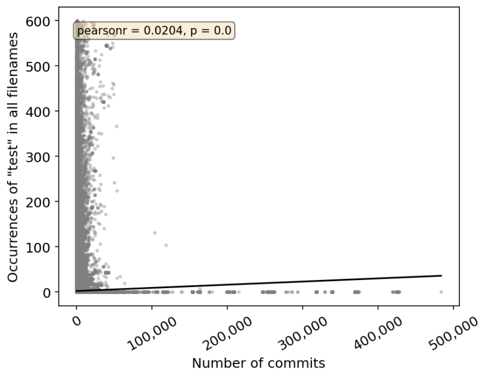
Figure A3. Correlation between the number of occurrences of the word "test" in java files path and project's life cycle parameters.



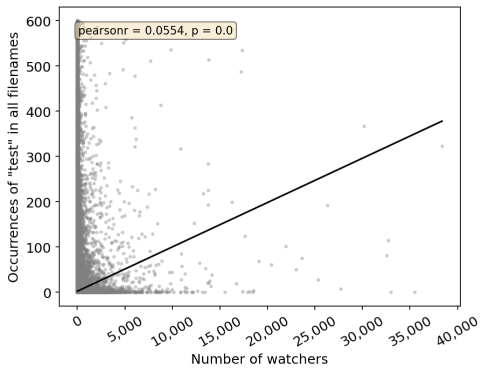
(**a**) Creation date



(**b**) Date of the last commit

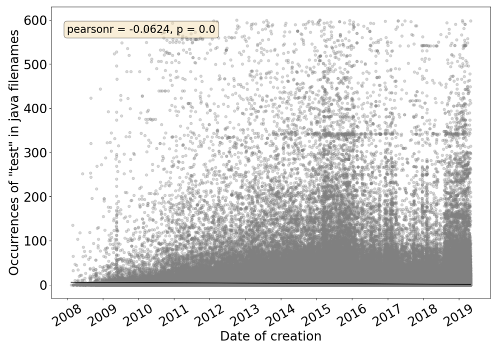


(**c**) The number of commits

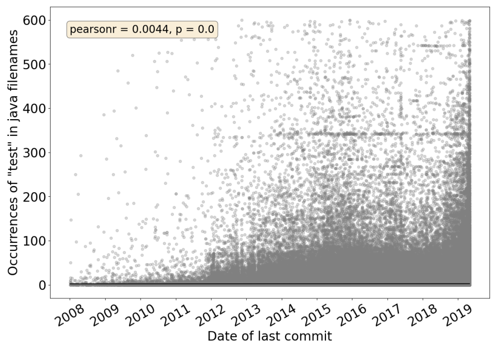


(**d**) The number of watchers

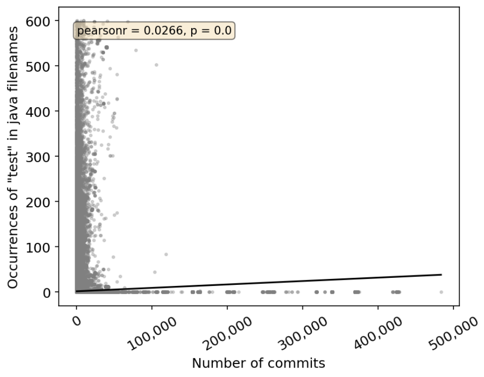
Figure A4. Correlation between the number of occurrences of the word "test" in all files filename and project's life cycle parameters.



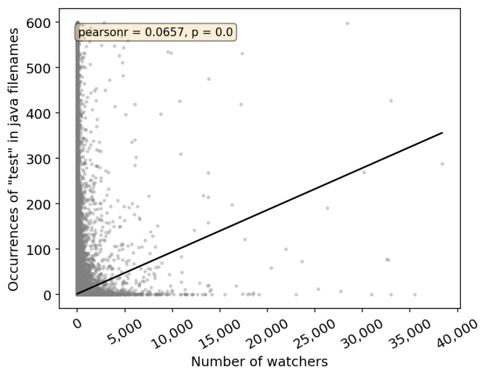
(**a**) Creation date



(**b**) Date of last commit



(**c**) The number of commits



(**d**) The number of watchers

Figure A5. Correlation between the number of occurrences of the word "test" in java files filename and project's life cycle parameters.

References

1. Kochhar, P.S.; Bissyandé, T.F.; Lo, D.; Jiang, L. An Empirical Study of Adoption of Software Testing in Open Source Projects. In Proceedings of the 2013 13th International Conference on Quality Software, Nanjing, China, 29–30 July 2013; pp. 103–112. [CrossRef]
2. Gren, L.; Antinyan, V. On the relation between unit testing and code quality. In Proceedings of the 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Vienna, Austria, 30 August–1 September 2017; pp. 52–56.
3. Linares-Vásquez, M.; Bernal-Cardenas, C.; Moran, K.; Poshyvanyk, D. How do Developers Test Android Applications? In Proceedings of the 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), Shanghai, China, 17–22 September 2017; pp. 613–622.
4. Beller, M.; Gousios, G.; Panichella, A.; Zaidman, A. When, How, and Why Developers (Do Not) Test in Their IDEs. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*; Association for Computing Machinery: New York, NY, USA, 2015; pp. 179–190. [CrossRef]
5. Kochhar, P.S.; Thung, F.; Nagappan, N.; Zimmermann, T.; Lo, D. Understanding the Test Automation Culture of App Developers. In Proceedings of the 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), Graz, Austria, 13–17 April 2015; pp. 1–10.
6. Fucci, D.; Erdogmus, H.; Turhan, B.; Oivo, M.; Juristo, N. A Dissection of the Test-Driven Development Process: Does It Really Matter to Test-First or to Test-Last? *IEEE Trans. Softw. Eng.* **2017**, *43*, 597–614. [CrossRef]
7. Bissi, W.; Serra Seca Neto, A.G.; Emer, M.C.F.P. The effects of test driven development on internal quality, external quality and productivity: A systematic review. *Inf. Softw. Technol.* **2016**, *74*, 45–54. [CrossRef]
8. Karac, I.; Turhan, B. What Do We (Really) Know about Test-Driven Development? *IEEE Softw.* **2018**, *35*, 81–85. [CrossRef]
9. Rafique, Y.; Mišić, V.B. The Effects of Test-Driven Development on External Quality and Productivity: A Meta-Analysis. *IEEE Trans. Softw. Eng.* **2013**, *39*, 835–856. [CrossRef]
10. Petrić, J.; Hall, T.; Bowes, D. How Effectively Is Defective Code Actually Tested? An Analysis of JUnit Tests in Seven Open Source Systems. In *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering*; Association for Computing Machinery: New York, NY, USA, 2018; pp. 42–51. [CrossRef]
11. Pecorelli, F.; Catolino, G.; Ferrucci, F.; De Lucia, A.; Palomba, F. Testing of Mobile Applications in the Wild: A Large-Scale Empirical Study on Android Apps. In *Proceedings of the 28th International Conference on Program Comprehension*; Association for Computing Machinery: New York, NY, USA, 2020; pp. 296–307. [CrossRef]
12. Spadini, D.; Palomba, F.; Zaidman, A.; Bruntink, M.; Bacchelli, A. On the Relation of Test Smells to Software Code Quality. In Proceedings of the 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), Madrid, Spain, 23–29 September 2018; pp. 1–12. [CrossRef]
13. Zerouali, A.; Mens, T. Analyzing the evolution of testing library usage in open source Java projects. In Proceedings of the 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), Klagenfurt, Austria, 20–24 February 2017; pp. 417–421.
14. Cruz, L.; Abreu, R.; Lo, D. To the attention of mobile software developers: guess what, test your app! *Empir. Softw. Eng.* **2019**, *24*, 2438–2468. [CrossRef]
15. Madeja, M.; Porubän, J.; Bačíková, M.; Sulír, M.; Juhár, J.; Chodarev, S.; Gurbáľ, F. Automating Test Case Identification in Java Open Source Projects on GitHub. *Comput. Inform.* **2021**, accepted. Available online: https://arxiv.org/abs/2102.11678 (accessed on 20 July 2021).
16. Gousios, G. The GHTorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*; IEEE Press: Piscataway, NJ, USA, 2013; pp. 233–236.
17. Kirch, W. (Ed.) Pearson's Correlation Coefficient. In *Encyclopedia of Public Health*; Springer: Dordrecht, The Netherlands, 2008; pp. 1090–1091. [CrossRef]
18. Munaiah, N.; Kroh, S.; Cabrey, C.; Nagappan, M. Curating GitHub for engineered software projects. *Empir. Softw. Eng.* **2017**, *22*, 3219–3253. [CrossRef]
19. Nosáľ, M.; Sulír, M.; Juhár, J. Source Code Annotations as Formal Languages. In Proceedings of the 2015 Federated Conference on Computer Science and Information Systems (FedCSIS), Lodz, Poland, 13–16 September 2015; pp. 953–964. [CrossRef]
20. Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M.C.; Regnell, B.; Wesslén, A. *Experimentation in Software Engineering*; Springer: Berlin/Heidelberg, Germany, 2012.
21. Kochhar, P.S.; Bissyandé, T.F.; Lo, D.; Jiang, L. Adoption of Software Testing in Open Source Projects—A Preliminary Study on 50,000 Projects. In Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering, Genova, Italy, 5–8 May 2013; pp. 353–356. [CrossRef]
22. Lin, J.W.; Salehnamadi, N.; Malek, S. Test Automation in Open-Source Android Apps: A Large-Scale Empirical Study. In Proceedings of the 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, VIC, Australia, 21–25 September 2020; pp. 1078–1089.
23. Beller, M.; Gousios, G.; Zaidman, A. TravisTorrent: Synthesizing Travis CI and GitHub for Full-Stack Research on Continuous Integration. In Proceedings of the 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), Buenos Aires, Argentina, 20–21 May 2017; pp. 447–450. [CrossRef]

24. Lemay, M.J. Understanding Java Usability by Mining GitHub Repositories. In *9th Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU 2018)*; OpenAccess Series in Informatics (OASIcs); Barik, T., Sunshine, J., Chasins, S., Eds.; Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik: Dagstuhl, Germany, 2019; Volume 67, pp. 2:1–2:9. [CrossRef]
25. Ray, B.; Posnett, D.; Filkov, V.; Devanbu, P. A Large Scale Study of Programming Languages and Code Quality in Github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*; Association for Computing Machinery: New York, NY, USA, 2014; pp. 155–165. [CrossRef]