Analysis And Observations Of The Evolution Of Testing Library Usage

Work in Progress

Ahmed Zerouali
Software Engineering Lab, UMons
ahmed.zerouali@umons.ac.be

Abstract

Many software development projects frequently rely on testing-related libraries to test the functionality of the software product automatically and efficiently. To obtain insights in the nature of the evolution of testing library usage, we empirically analyzed the usage of eight testing-related libraries in 6,424 open source *Java* projects hosted on *GitHub*. We observed how frequently specific (pairs of) versions of libraries are used over time, for how much they are used within a project and we identified the delay to upgrade to a new version. We also identified over time the most used packages of libraries and we analyzed if groups of packages are usually used together. We studied the evolution of the number of test *Java* files and we also studied how often developers use testing libraries to test classes that provide a particular functionality. We found that some versions of certain libraries are quickly adopted than the others and some of them are quickly upgraded. We observed that most packages of some libraries tend to be used in a few numbers of *Java* files. These findings may pave the way for recommendation tools that allow project developers to choose the most appropriate library and library developers to better maintain their library.

Copyright © by the paper's authors. Copying permitted for private and academic purposes.

Proceedings of the Seminar Series on Advanced Techniques and Tools for Software Evolution SATToSE 2017 (sattose.org). 07-09 June 2017, Madrid, Spain.

1 Introduction

In object-oriented software systems, and open source software systems in particular, it is common practice to implement systems on top of frameworks and libraries and to rely on external libraries to improve code reuse, reduce development cost and promote programmers' productivity. Nearly every programming code that developers write will have a library call and most of today's software projects heavily depend on the use of these libraries [Mileva09]. Such libraries come with a, hopefully well-documented, API (application programming interfaces).

From the point of view of library users, in order to improve the way in which a library is used, it is useful to understand how other projects use the library, how and which library functions are often used together, which new releases are the first to be adopted and used.

From the point of view of library developers, it is useful to provide insights about the common practices and to assess the popularity of their API functionalities in a given ecosystem and to prevent them from risk factors such as breaking changes and eventually migrations to other competitor libraries.
In our research, motivated by the importance of unit testing, we decided to study testing related libraries evolution and usage to better understand how they are used and eventually analyze the real drivers and the actual requirements needed for a testing or mocking library to be adopted or abandoned. This is the focus of our empirical study, in which we analyze the evolution of the usage of eight aforementioned testing-related libraries(*JUnit*, *TestNG*, *Spring*, *Hamcrest*, *AssertJ*, *Mockito*, *EasyMock* and *PowerMock*) in 6,424 open source *Java* projects hosted on *GitHub*.
Our longitudinal study of testing related libraries and frameworks usage in *Java* projects addresses the following research questions:

RQ1: How long does it take before a *Java* project upgrades to a new released version of a testing-related library?
RQ2: How frequently are packages of testing libraries used?
RQ3: How does the number of test files in *Java* projects evolve over time?
RQ4: How often do *Java* projects use *JUnit* to test classes that provide a particular functionality?

2 Methodology

We focused only on open source *Java* projects extracted from *GitHub*, because *Java* is one of the most popular programming languages in *GitHub* and the most popular programming language according to *TIOBE*[1], and because we need to have full access to the projects historical data and source code.
Using our prepared corpus that we used in our earlier work [Zerouali17] and retained from *GitHub* Archive, and a list of the most popular testing and mocking libraries on *Maven* Repository (*JUnit*, *TestNG*, *Spring*, *Hamcrest*, *AssertJ*, *Mockito*, *EasyMock* and *PowerMock*), we extracted all import statements that exist in each *Java* file and all project dependencies existed in each Project Object Model file (pom.xml) for all the corpus. We found that among 20,688 *Java* projects that we have, only 6,424 projects can be considered, only this group of projects makes use of *Maven* and makes use of at least one of the declared testing libraries. For the source code analysis and exploration, and the extraction of the tested *Java* classes, we used the srcML toolkit [2], an infrastructure for the exploration, analysis, and manipulation of source code.

3 Research Questions and Preliminary Results

This section addresses our research questions by means of observations and visualizations.

3.1 How long does it take before a *Java* project upgrades to a new released version of a testing-related library?

To answer this question, for each project, for the first snapshot of each month of its lifetime, we extracted the metadata available on the *Maven* POM file and we identified the versions used and related to our considered testing libraries.
Figure 1 shows the delay to adopt a new released library version and the duration of use of this version. We observe that projects that make use of *TestNG*, *PowerMock* and *AssertJ* are quickly upgrading their library. More than 20% of these projects upgraded their library version in the first months, where the other ones take more time before upgrading to a new released version. Most of the versions of testing libraries tend to be used for less than two years before upgrading or switching to another library version.

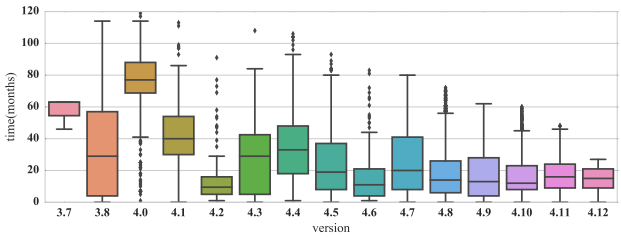

Figure 2: Latency to adopt a new released *JUnit* version in months.

We also observed that some versions of the same library are quickly adopted than the others (i.e., the first releases of the major version *JUnit* 4.0. See Figure 2) which might be explained by the fear of the incompatible API changes that are made on the new release.

3.2 How frequently are packages of testing libraries used?

To give an overview to the developers about the usage of their library packages, we analyzed the source code of all *Java* files of all projects that used one of the considered libraries.
As shown in Figure 3, in the analysed *Java* projects, we observed that most of testing library packages tend to be used in only a few numbers of *Java* files. We also found that more than 90% of the projects that use testing related libraries, use only a few numbers of the available packages in the used library.
Figure 4 shows that for unit testing libraries (*JUnit* and *TestNG*), the proportion of projects that use different packages of these libraries is higher than the proportion in the other categories. However, *AssertJ* (a recent asserting library) packages seem to be used in a different way. We also noticed that the usage of these packages is not related to the size of the used library.
We also analyzed how different packages and classes of the same library are used within a project, and we found that most packages are not necessary used together, but for some packages, we found patterns where they were usually used together.

[1] https://www.tiobe.com/tiobe-index/
[2] http://www.srcml.org/

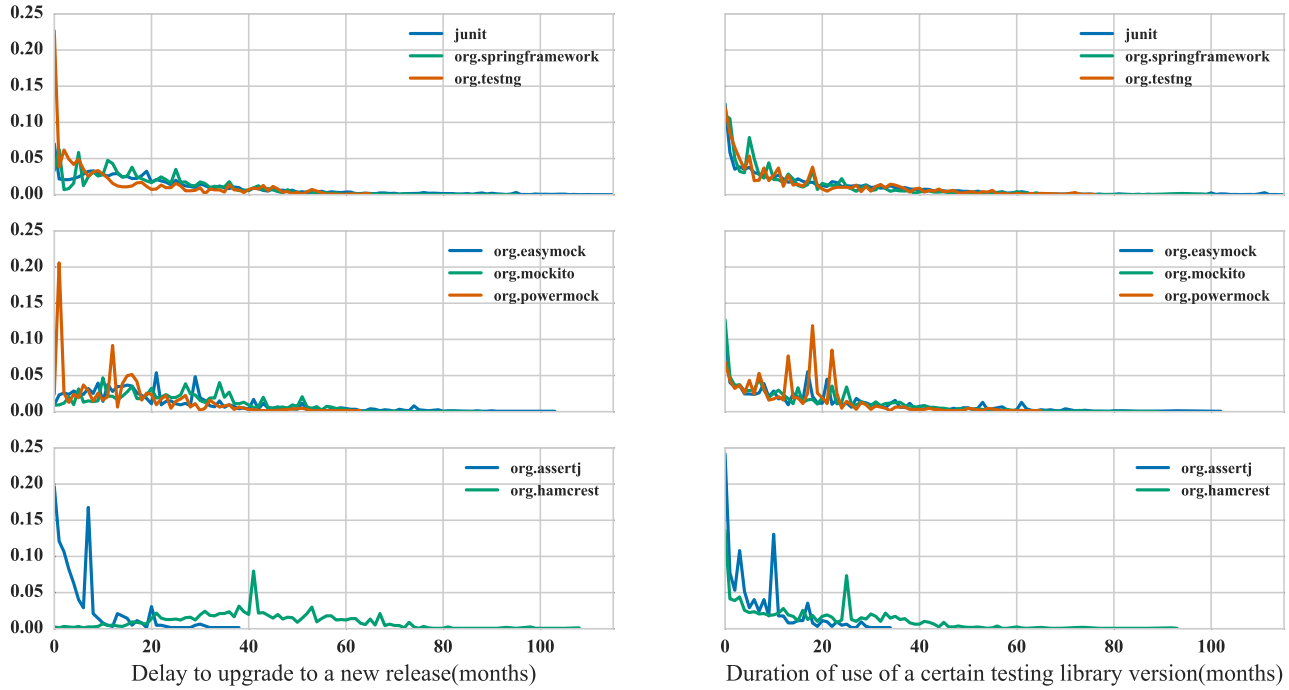

Figure 1: The proportion of the adopt latency and the duration of use of library versions across different projects.

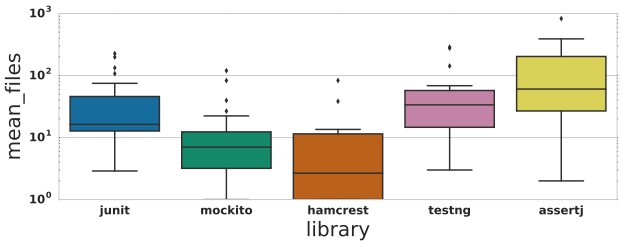

Figure 3: The distribution of the mean number of test *Java* files in a project using different packages of different testing related libraries.

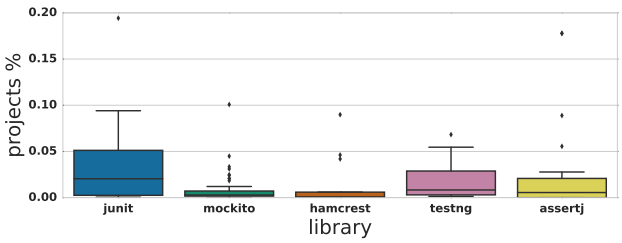

Figure 4: The distribution of the proportion of projects that use different packages of different testing related libraries.

3.3 How does the number of test files in *Java* projects evolve over time?

To answer this question, we calculated for each project, for the first snapshot of each month, the number of test *Java* files existing in this project. We considered only the unit testing libraries, *JUnit*, *TestNG* and *Spring*. Figure 5 shows for the three libraries, the evolution over time of the number of *Java* files that contain test cases in the *Java* projects. We found that in the last 8 years, the proportion of the numbers of test *Java* files existing within a *Java* project is always less than 15% of all *Java* files in this project. Before 2009, *TestNG* was used in a higher numbers of test *Java* files than *JUnit*, but after 2009 *JUnit* test files numbers increased to beat the decreasing numbers of *TestNG* test files, while *Spring*'s test files proportion was always under 5% of all *Java* files.

3.4 How often do *Java* projects use *JUnit* to test classes that provide a particular functionality?

To know how often we use testing libraries to test classes that provide a particular functionality, and because it is important for developers to avoid the myriad of problems that can occur when tests corrupt the database and cause subsequent tests to fail, we chose database classes as candidates. For 1,150 *Java* projects that used *JDBC* or *JPA* or *Hibernate* for their database access, and used *JUnit* as a testing library, we identified all *Java* classes that make use of these database libraries and all test *Java* classes, and then we analyzed if the database classes are being tested. We chose *JUnit* so we can have enough projects for this

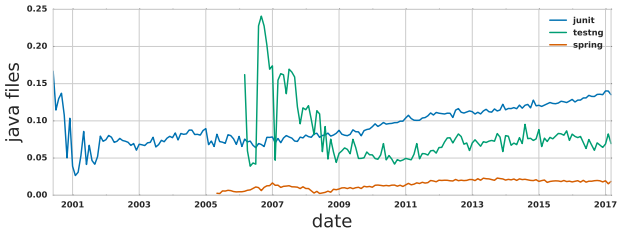

Figure 5: The evolution of the proportion of the test *Java* files over time.

empirical analysis. We found that half of these classes number is tested, and with a small difference, classes that used *JDBC* are more likely to be tested than the classes that used *JPA* or *Hibernate*. See Figure 6.

We also analyzed the co-evolution of the tested database classes and the other tested classes and we observed that the proportion of the database classes that are tested is slightly decreasing compared to the other tested classes(i.e., classes using all other types of libraries). See Figure 7.
Which is normal since software projects evolve and develop more functionalities using other libraries.

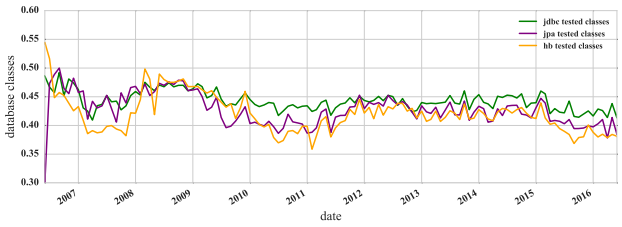

Figure 6: The evolution of the proportion of the tested database *Java* classes

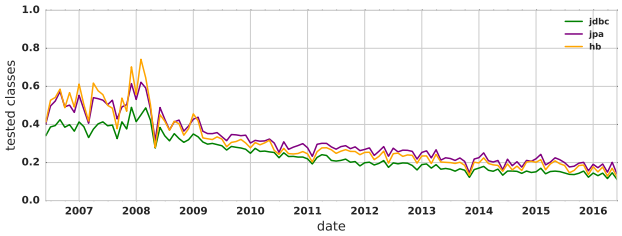

Figure 7: The co-evolution of the tested database *Java* classes and the other tested classes.

3.4.1 Acknowledgment

This research is part of FRFC research projects T.0022.13 and J.0023.16 financed by F.R.S.-FNRS, Belgium.

4 Threats to Validity

Our results may not be generalisable to non-*Java* projects or to projects that do not rely on the build automation tool *Maven*. While we studied the usage of eight *Java* libraries only, the proposed methodology is applicable to other combinations of libraries as well. Our results may be biased by the fact that we consider all type of open source *Java* projects without classifying these projects in different categories.

5 Conclusion

We analyzed the usage of eight popular testing, matching and mocking libraries in a large corpus of *GitHub*-hosted *Java* projects. We observed that some libraries are rapidly upgraded within a project after the release of a new version, moreover, certain versions of certain libraries tend to take more time before their adoption than the others.

We found that certain libraries are not fully used, only few numbers of packages are used by all projects that make use of these libraries, we also observed that some packages are usually used together within a *Java* project.

We also analyzed the test coverage in projects that used database related libraries and we found that half of the classes that make use of them are not tested.

Our findings can facilitate the way to library users for better usage by giving them sharp insights of how others make use of such libraries. Library developers also can benefit from this analysis to better understand how their major library versions are being used in practice, in order to provide incentives to increase their library's adoption rate and to avoid its users to migrate to competing libraries.

For future work, after we finish our current study, and based on a previous one [Zerouali17], we will provide recommendation tools on testing library usage to better assist library users as well as library developers.

References

[Qiu16] D Qiu, B Li, and H Leung. Understanding the API usage in *Java*. Information and Software Technology 73 (2016): 81-100.

[Sawant16] A A Sawant, and A Bacchelli. fine-GRAPE: fine-grained APi usage extractoran approach and dataset to investigate API usage. Empirical Software Engineering (2016): 1-24.

[Myers16] B A Myers, J Stylos. Improving API Usability. CACM 59(6): 62-69

[Teyton12] C Teyton, JR Falleri, X Blanc. Mining Library Migration Graphs. vol. 00, no. , pp. 289-298, 2012, doi:10.1109/WCRE.2012.38

[Kabinna16] S Kabinna, CP Bezemer, W Shang, A E. Hassan. Logging library migrations: a case study for the apache software foundation projects", MSR 2016

[Zerouali17] A Zerouali, T Mens. Analyzing the evolution of testing library usage in open source Java projects. Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on. IEEE, 2017.

[Mileva09] YM Mileva, V Dallmeier, M Burger, A Zeller Mining trends of library usage. Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops. ACM, 2009.