Usage of Tests in an Open-Source Community

Benoît Verhaeghe, Vincent Blondeau, Nicolas Anquetil, Stéphane Ducasse

Usage of Tests in an Open-Source Community:

A Case Study with Pharo Developers

Benoît Verhaeghe[1] Nicolas Anquetil[1]
Stéphane Ducasse[1]

[1] Univ. Lille, CNRS, Inria, Centrale Lille,
UMR 9189 - CRIStAL,
F-59000 Lille, France
{firstname.lastname}@inria.fr

Vincent Blondeau[1,2]

[2] Worldline,
Z.I A Rue de la Pointe
59113 Seclin, France
{firstname.lastname}@worldline.com

Abstract

During the development, it is known that tests ensure the good behavior of applications and improve their quality. We studied developers testing behavior inside the Pharo community in the purpose to improve it. In this paper, we take inspiration from a paper of the literature to enhance our comprehension of test habits in our open source community. We report results of a field study on how often the developers use tests in their daily practice, whether they make use of tests selection and why they do. Results are strengthened by interviews with developers involved in the study. The main findings are that developers run tests every modifications of their code they did; most of the time they practice test selection (instead of launching an entire test suite); however they are not accurate in their selection; they change their selection depending on the duration of the tests and; contrary to expectation, test selection is not influenced by the size of the test suite.

Keywords Regression Test Selection, Case study, Interviews, Pharo Community.

1. Introduction

With the increase in complexity of software applications, the need to test every piece of code becomes compulsory. For a long time, tests were done by running the application manually. This practice is not dead, but with the improvement of testing tools, frameworks, or processes, there is a push to automate testing and make it more systematic.

In Pharo, developers are encouraged to make and run automated tests for their applications. Here, contrary to some companies, the lake of tests is not present [Blondeau et al., 2017].

Moreover, we can run all the tests of Pharo if we want to be sure that we break nothing. But a long time is necessary to run them all. Developers are encouraged to run only the tests that cover methods impacted by their work.

Yoo and Harman [2012] expose several solutions to this problem: First, test suite minimization identifies tests covering the same piece of software and remove them. The test suite is reduced, but not drastically because, in practice, only a few tests are redundant in term of coverage. Once the suite is reduced, all tests are run. Second, test suite prioritization first runs the tests that could be impacted by the developer modifications and second all the other tests. Such a solution is valuable but, one still needs to wait for the whole test suite execution to know that there is no error. Third, test selection — or RTS, Regression Test Selection — Engström et al. [2010], Graves et al. [2001], Yoo and Harman [2012] selects only the tests that could be impacted by modifications. In this paper, we focus only on the latter solution.

Pharo does not offer integrated test selection mechanisms other than manually selecting and running one test, one or more classes, or one or more packages.

Based on our hypothesis that Pharo developers do not miss the right tests after a modification in the source code. we decided to (i) characterize the testing behavior (or habits) of Pharo community developers; and (ii) look for solutions on how to improve it.

We found three publications in the literature that matched these goals.

First, Blondeau et al. [2017] studies developer testing habits in an major IT Company. This research would cover well our need to characterize developer testing behavior. However, most of the projects in the study are closed source. Consequently, the conclusion may not apply for Pharo (closed source projects are known to behave differently from open-source ones Zimmermann et al. [2009]). This paper was inspired the two following. Second, Beller et al.

ISBN [to be supplied]. . . $15.00

[2015] studies the usage of the IDEs by the developers to understand "When, How, and Why Developers Test". This research would cover well our need to characterize the testing behavior of developers. Third, Gligoric et al. [2014] is a "Comparison of Manual and Automated Test Selection". As such, it seems to match well our second goal of verifying whether test selection can improve testing practice. However, most of the developers in the study were students and again, the conclusions may not apply to our context.

We wanted to make a similar case study in the Pharo environment. Therefore, we mainly based our study on Blondeau et al. [2017]. A secondary goal of the case study (out of the scope of this paper) is also to collect base data to be able to detect possible impact of future (automated test selection) actions on Pharo developers.

The main findings of our study are:

- As we expected it, the number of tests passing is more important in the Pharo community than it is in the other study.
- We found that the Pharo developers run often their tests.
- Contrary to expectation, Pharo developers practice the test selection depending of the duration of the tests. However, the number of tests in a test session is not important. Developers prefer to select the tests to run by their relevancy.

In Section 2, we give background information on our problem and describe the case study proposed by Blondeau et al. [2017] that inspired our study. Then, in Section 3, we define the research questions and describe our experiment. Section 4 analyses and compares the results of the case study. Section 5 presents the threats to validity of our case study. Finally, we present the related works and conclude in Section 6 and 7 respectively.

2. Problem Description

Pharo contains at least 41 790 tests, executing all tests after each change can turn into an operation requiring several minutes. In addition it is unclear that the tests can have side effect and could change the code. This can produce some dirty objects around. Pharo developers run automatically their tests through Pharo IDE and very frequently all the tests from Jenkins [1] server.

2.1 Regression Test Selection

Before presenting the case studies of the literature, we introduce here the concept of Regression Test Selection (RTS).

More formally, following Rothermel and Harrold [1993], the test selection problem is defined:

Problem: Given program *P*, its modified version *P'*, and test set *T* used previously on *P*. Find a way, making use of *T*, to gain sufficient confidence in the correctness of *P'*.

A solution to this problem is to only select the tests in *T* that exercise the modified code in *P'*. The outcome of the other tests should not have changed since they are not impacted Yoo and Harman [2012].

Some test case selection approaches are based on the notion of dependency graph. The general idea is that tests can be said to depend on the source code that they exercise. After a piece of code is changed, a test case selection technique just needs to navigate the dependency graph and go back from the changed piece of code to the tests that depend on it. A change can be any modification of the source code, even it has no impact on the application behavior. Figure 1 illustrates this principle for two methods and two tests: testMethod1 depends on method1 and method2 (testMethod1 calls method1 and method2), testMethod2 depends only on method2. If method1 is changed, only testMethod1 needs to be re-run as the outcome of testMethod2 cannot have changed. This is, of course, a simplified example. In real cases, establishing dependencies from tests to code is made difficult by many adverse factors Blondeau et al. [2016].

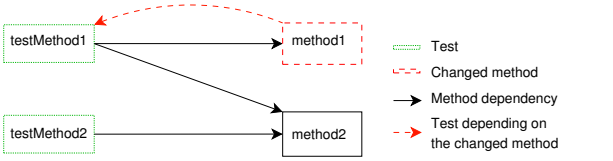


Figure 1. Test Selection Simple Case

We now proceed to describe the previously published case study that we take inspiration from.

2.2 Paper: What are the Testing Habits of Developers ?

The paper we take inspiration from is the one of Blondeau et al. [2017]. The authors study the testing habits of 32 participants. The study, led in an major IT Company, was composed by developers, technical leaders and architects of the company. They asked them to install a plugin in their IDE which records code changes and test executions. Consequently, this research might also not fit well our context.

Here are the main findings of interest:

- There is a low correlation ($\rho = 0.11$) between the amount of code changed immediately before a test session and the number of manually selected tests in that session;
- The developers perform test selection (58% of the test execution) and more than the half of the test session ran only one test;
- The amount of test executions does not depend on the size of the test suite nor on the duration of the tests. We could observe whether there is the same phenomenon in an open-source community.

[1] Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks such as building, testing, and deploying software. Jenkins can be installed through native system packages, or run standalone by any machine with the Java Runtime Environment installed.

Blondeau et al. [2017] also took inspiration from Gligoric et al. [2014] and Beller et al. [2015]. The results of these two other papers will be compared to ours in Section 4.

3. Experimental Setup

This section presents the research questions we want to explore and the methodology to answer them.

3.1 Research questions

We kept all the request questions that are of interesting for us and where data where available.

Like Blondeau et al. [2017], we set the following research questions for our case study:

RQ1: How and Why Developers Run Tests?

- **RQ1.1** Do developers test their code changes?
- **RQ1.2** How long does a test run take?
- **RQ1.3** Do quick tests lead to more test executions?
- **RQ1.4** Do developers practice test selection?
- **RQ1.5** What are common scenarios for manual RTS?

RQ2: How Do Developers React to Tests Run?

- **RQ2.1** How frequently tests pass and fail?
- **RQ2.2** How long does it take to fix a failing test?

RQ3: How and Why Developers Perform Test Selection?

- **RQ3.1** Does manual test selection depend on size of test suites?
- **RQ3.2** Does manual test selection depend on size of code changes?

3.2 Experimental protocol

Participation to the case study was voluntary. We send a message on the Pharo Discord[2] and on its mailing list to inform about our project. The volunteers had to install a plugin in one of them Pharo image. These plugin has been made to collect the data (see after). It is completely transparent for the participants.

For our experiment, we need data on the test executed and on the source code. The major difficulty was to collect data from an user who starts a test but decides to interrupt it (in Pharo, you can add a breakpoint in a test then abandon the test. The same behavior is present for the error or the warning.). Another problem is that users can modify their code during the test (directly inside the debugger).

Because we would like volunteers, it was inconceivable they waste time because of our recording. Thanks to Pharo, we had all the necessary tools to detect users code changes and their execution of tests. So, we collected precise information like Beller et al. [2015]. We collect test information through a plugin that were developed for Pharo. Theses pieces of informations are recorded:

Developer id: Unique id given to the developer;

Version code: List of all packages present on the developer image;

Test method: Id specifying how the developer executes its test session;

Test session start: Timestamp of the launch of the test runner;

Test session end: Timestamp at the end of the last test execution;

Tests executed: List of each test executed in the session with the following details:

- **Test name:** Name of the test method with its class.
- **Test duration:** Duration of the method execution;
- **Test status:** Result of the test: PASS, FAIL (wrong assert), ERROR (unexpected exception or unfinished test), or SKIPPED;

We also collect precise data for each code changes:

Modification time: Timestamp of the modification;

Method/Class name: Full name of the method, the class or the package which has been modified;

Snapshot: The lines of code corresponding to the modification.

The plugin records each action of the developer and centralize the data in a server.

3.3 Filtering and Massaging Data

As usual for *in vivo* case studies, filtering and massaging data to get meaningful answers, was a major task. We discuss here some of the hypotheses we had to make.

Test session. As Beller et al. [2015], we consider that a test session is at least an execution of one test by the Pharo test runner between two code modifications. A test session can be composed of one or several tests.

However, because we are considering test selection with tools that are not well suited for it, and, in the goal to compare the results to the ones of Blondeau et al. [2017], we introduces another concept (Agglomerated Session, see next). Consider a developer who want to run two specific tests from two different classes. With Pharo, the options are either to run all the tests of a class or to run independently the two tests methods. Developers often choose the second option. This means we will have two test sessions.

Moreover, in Pharo, the test execution can be stopped when an error is raised. Then, the Pharo debugger is automatically opened. So, the developer can change his code and run again the test inside it. In this case, we decided that the developer has run two sessions. The first one begins when the developer run it and has for result the error which has

[2] Discord is a free voice and text chat platform where the Pharo community discuss about any concern on Pharo, its usage, and its applications .

been raised (ERROR or FAIL). The second one begins when the developer run again the test. It has for result the final test execution result.

Agglomerated test session. An agglomerated test session is a set of successive tests sessions. Two successive test sessions on the same project from the same developer id will be agglomerated as long as they is no code modification between the two test session. It was been facilitated by Pharo because we can recording every code modification.

We collect code modification by author, an agglomerated test session begins at the run of a test and ends when the developer modifies its code.

Code tested. We also need to determine which code is being tested. Because we collect test data and code modification data, we are able to find exactly which part of the source code has been tested.

Code change. By collecting all the data about code modification, we can compute code change. We compare the original code with the code generated from all the modifications we collect from our plugin users.

Amount of code changes. Some research questions require to evaluate the amount of code changed. We consider any modification made by a developer as a line modified and so as a code modification. Pharo users are incited to create only small methods. When a developer saves one of its modification, most of the time only one line is changed.

Gligoric et al. [2014] used the number of AST node differences between two versions of the code.

3.4 Interviews with the Participants

To extract more insights from the participants to the case study, we conducted an interview at the begin of the gathering of the results [Wohlin et al., 2000]. The participation to the interviews is on voluntary basis. Its goal is not only to get more context of the environment of the developers but also to have some explanations on the qualitative results extracted from the recording of the tests executions.

For the major part of the volunteers, we conducted 20-30 minutes face to face discussion. Because of their availability, we gave them a survey to complete. After a brief description of themselves (their experience in the community) and a quick description of their project, we asked them to describe their behavior about testing in the context of the application they are working on: how they create tests, how they manually select them, launch them, what actions they take after a failing test. The result of the interviews are integrated in each research question to explain the quantitative results found by monitoring the developers.

4. Results and Discussion

We now present our results and compare them to the three previous case studies of Beller et al. [2015], Blondeau et al. [2017], Gligoric et al. [2014].

These results were obtained between June 9th, 2017 and July 3rd, 2017. The length of the study is short but the results we have are enough to draw some conclusions. Tables 1 and 2 present some descriptive statistics on the case studies.

Table 1. Descriptive statistics on the four case studies

	Blondeau et al. [2017]	Beller et al. [2015]	Gligoric et al. [2014]	us
# Developers	32	48	14	20
# Projects	64	73	17	-
# Test sessions	14 686	3 424	5 757	2 409
# Agglomerated sessions	13 611	-	-	1 468
# Test executions	153 763	10 840	264 562	269 417
Tests / Session	10.5	3.2	45.9	111.8
Sessions / Developer	458.9	71.3	411.2	120.45
Study Duration (months)	10	4	3	1

We have 20 participants. So, our study is closer to the third paper Gligoric et al. [2014] which have 14 developers as participants. The second paper Beller et al. [2015] have 48.

Among ours, 8 accepted to be interviewed. These participants are students (5) and experimented (7 years) Pharo developers (3).

We have less test sessions (866) than Gligoric et al. [2014] with a five times shorter study duration.

We also have less tests per session (2.25) and less sessions per developers (54.125) than the other study.

But we have 269 417 test executions. It is more than Beller et al. [2015] which seven times shorter duration of study. If our study lasted 10 months, we should have 2 694 170 tests executions. It is more than the double of tests made by the volunteers of the study of Blondeau et al. [2017] in the same duration.

This can already be seen as a good indicator for test practice in the Pharo community.

As an additional indicator, we give in Table 2 statistics for developers of our case study: number of days of collecting data, number of test sessions, and number of sessions per day where tests have been made (activity days), all developers combined.

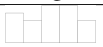
We can note that Pharo developers run more test session per activity day (Median: 17) than the developers in the first paper (Median 8). From this last number, it seems that testing is a major habit for the Pharo community.

4.1 RQ1: How and why developers run tests?

RQ1.1 Do developers test their code changes?

For this question, we evaluate whether there is a correlation between the number of tests run and the number of changes to source code. We used Spearman correlation as our data do not follow a normal distribution. The correlation is good $\rho = 0.61$ thus confirming that more code change tend to lead to more tests.

Table 2. Descriptive statistics per developer

	Min	Q_1	Median	Mean	Q_3	Max	Histogram
Calendar Days	0	6.75	13.5	12.6	18.5	24	
Activity Days	0	0	3	3.7	6.75	11	
Sessions	0	0	37.5	120.4	179.8	613.0	
Sess./Activity Day	1	6	17	32.12	35.5	264	

Beller et al. [2015] differentiates the number of changes to test code or number of changes to production code. They have a good correlation ($\rho = 0.66$) with test code changes, and a weak one ($\rho = 0.38$) for production code. Our correlation is more important than their results. So, the Pharo developers tends to test more the changes they produced than in the other studies.

From the interviews, it seems that developers would like to run tests after they made changes in their application. However, it was hard to find how to test changes related to virtual machine or complex systems. Moreover, developers confess that they often do not run tests after minor changes. The main reason given is lack of time.

RQ1.2: How long does a test run take?

Test sessions are almost instantaneous half of the time, and 25% of the tests lasts 11 seconds (results are similar for agglomerated sessions with respectively 3 and 22 sec.). Moreover, 7.7% of the test sessions take longer than one minute and 4.5% take longer than two minutes (respectively 13% and 8.3% for the agglomerated sessions). Detailed results can be found in Table 3.

We measured a maximum duration of the test sessions of 6 days 11 h 30. Because the data are anonymous, we can not ask further information about this test. It is possible that a developer start a test. The test fails and a debugger opens. The execution is stopped but the test duration is still increasing until the developer resume or abort the execution of the test.

Beller et al. report that 50% of their test sessions finish in less than 0.5 seconds and over 75% of the sessions finish within 5 seconds. For their test sessions, 7.4% take longer than one minute and 4.8% take more than two minutes. They conclude that most of the test sessions are short.

We have the same conclusion about test sessions. Most of them are very short.

RQ1.3: Do quick tests lead to more test executions?

We have to evaluate the correlation between test duration and the number of test executed in a session. We are expecting that short tests will be executed more often than the long ones. Consequently, the correlation should be negative.

However, our Spearman correlation was $\rho = -0.07$. Beller et al. [2015] and Blondeau et al. [2017] respectively obtain a positive correlation of $\rho = 0.26$ and $\rho = 0.20$. Both lead to the conclusion that there is no relevant correlation.

These results are contrary to expectation, faster tests are not executed more often (corollary: longer tests are not executed less often).

Interviewed people explained that they run the tests that cover the part of the application they changed without distinction of the duration of the test. Exceptions are made when the tests are too long (some minutes). In this case, the tests will be run after a code session or, at least, at the end of the development.

RQ1.4: Do developers practice test selection?

We report 53% of the agglomerated sessions with only one test, 22% with more than 5 tests, and 10% with more than 50 tests. We can reach the conclusion that Pharo developers practice test selection.

Beller et al. [2015] reports that 87% of test sessions include only one test case, 6.2% include more than 5 tests, and 2.9% more than 50 tests. From this, they concluded that their developers did practice test selection. Gligoric et al. [2014] reports 3 594 test sessions (62.4%) with only one test. The study of Blondeau et al. [2017] reports that 58% of the agglomerated session contains only one test.

It seems the Pharo developers and those in the papers of Gligoric et al. [2014] and Blondeau et al. [2017] select less "aggressively", *i.e.,* less test sessions consisting of only one test.

For us, in 50% of the test sessions, 12.5% of the available tests of the project are selected, and in 75%, 73.31% are selected (See Table 3).

Beller et al. [2015] further notes that in 50% of the test sessions, only 1% of the available tests of the project are selected, and in 75% of the cases, 12.5% are selected. For Beller et al. [2015] all tests are launched in 3.7% of the cases

We report that test selection occurs in 46% of the studied test sessions, less than Gligoric et al. [2014] (59.19%) and the other paper (about 96.3%[3]).

Finally we report an average selection ratio (number of executed tests divided by number of available tests) of 81.9%. For Gligoric et al. [2014], this ratio is almost the same with 9.0%. So, it seems Pharo developers practice the test selection but less than the participants of the studies of Gligoric et al. [2014] and Blondeau et al. [2017].

[3] Our statistics from their numbers

Thanks to the interviews, we identified two profiles of testers: Some developers run all the tests for a project each time they modify their code. The others select the tests to launch from their feeling and their experiences. They select the tests they think cover the last modifications.

RQ1.5: What are common scenarios for manual RTS?
The highlighted two scenarios for the test selection are:

- "Once a test fail, developers try to fix it and re-run the same test case until it passes"
- "The developers run group of tests and try to fix the failing one(s). For each correction they re-run the totality of the group of tests"

We found the first patterns in the studies of Gligoric et al. [2014] and Blondeau et al. [2017]. The first case appeared more than the second one during the interviews. Because we are in the Pharo context, developers also fix their tests inside the debugger by manually inspect how it behaves. They use this way of manual debugging instead of re-run the test because they can easily access to the data of the test during its execution. So it is easier to detect the error thanks to this Pharo feature.

4.2 RQ2: How do developers react to test runs?

RQ2.1: How frequently tests pass and fail?
In our case, on 269 417 tests executions, the ratio of failing tests is 1.16% (3 125), and the ratio of passing tests is 98.24% (262 675). We can also report 0.6% (1 616) of skipped tests. In Beller et al. [2015], on 10 840 tests executions, 65% (7 047) fail and 35% pass successfully. And Blondeau et al. [2017], on 153 763 tests executions, 13% (20 272) fail and 83% pass successfully.

We found a much lower ratio of failing tests in our case study. By interviewing developers, we can propose some explanations:

- The Pharo community run more tests than the developers of the others studies. So the number of tests passing is higher because they are executed many times.
- The tests do not pass because the corresponding behavior is not implemented yet. But the developers skip them to avoid having failing tests. This way, all the test suite is passes.

RQ2.2: How long does it take to fix a failing test?
In our case study, for the failing tests that get fixed, 50% are resolved in approximatively 1 hour and 53 minutes and 75% within approximately 2 days. The maximum duration that we observed to fix a test is 8 days, 14 hours and 10 minutes.

In Beller et al. [2015], for 70% of the tests (2 051), the authors observed at least one successful execution and 30% have no successful execution. Therefore a significant part of the tests are never fixed. For the 2 051 failing tests that are fixed at some point, 50% are executed with success within 10 minutes and 75% within 25 minutes.

In Blondeau et al. [2017], 50% of the failing tests are resolved in approximatively 20 minutes and 75% within approximately 17 hours 20 minutes.

Results for this question can also be found in Table 3.

We can notice that that the duration of a fix in Pharo is quicker than in the others studies.

4.3 RQ3: How and why developers perform test selection?

RQ3.1: Does manual test selection depend on size of test suites?
In our study, all developers performed test selection. We have an average of 1072 tests per developer, a minimum of 0, and a maximum of 14 458. We can conclude that developers performed manual test selection regardless of the size of their test suites.

In Gligoric et al. [2014], almost all developers performed manual test selection. They also had a wide range of test suite sizes. The authors further report an average of 174.3 tests per project; the minimum was 1 test, and the maximum was 2 216 tests. In Blondeau et al. [2017], all developers performed test selection. The authors found an average of 254.1 tests per project, a minimum of 1, and a maximum of 2 216.

The interviews reveal that Pharo developers are concerned by the duration of tests. However, if a test is too long to run (more than 2 minutes), they run it less times or they skip it and never run it again until the end of their developments.

RQ3.2: Does manual test selection depend on size of code changes?
We consider the relationship between the size of recent code changes and the number of tests that developers select in each test session. This correlation is $\rho = 0.33$ which is low. However, our correlation is superior to those of Gligoric et al. [2014] ($\rho = 0.28$) and Blondeau et al. [2017] ($\rho = 0.11$).

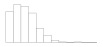
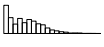
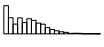
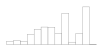
We conclude that the developers are not really influenced by the size of code changes for their test selection. But, it seems that in Pharo, developers may test more often than those who participate to the study of Gligoric et al. [2014] and Blondeau et al. [2017].

The interviewed explained they tried to select only the tests that only cover the code changes. An explanation is that the volunteers are not accurate in their selection.

5. Validity discussion

This section discusses the validity of our case study using validation scheme defined by Runeson and Höst [2009]. The construct validity, the internal validity, and the external validity are presented.

Table 3. Comparison of our results with those of Blondeau et al. [2017]. (When computing number of tests per session, we give results for test sessions and agglomerated sessions to match Blondeau et al.'s case study). Histograms are in log scale

			min	Q_1	median	Q_3	max	*unit*	Histogram
RQ1.2	Test session duration	us (test sess.)	0	0	1	11.0	559 800.0	*second*	
		us (agglom.)	0	0	3.0	22.0	559 800.0	*second*	
		Blondeau (test sess.)	0	1.0	3.0	16.0	15 820.0	*second*	
		Blondeau (agglom.)	0	1.0	3.0	18.0	15 820.0	*second*	
		Beller	0	0.03	0.5	3.4	73.8	*second*	
RQ1.4	Percentage of executed tests	us	0	0	0.1	73	73	%	
		Blondeau et al	0	1.2	4	17.8	100	%	
		Beller	0	1	1	12.5	100	%	
RQ2.2	Time to fix failing test	us	0	4.47	113	2 880	12 370	*minute*	
		Blondeau et al	0	3.1	4 981.0	1 042.0	359 600	*minute*	
		Beller	0	1.7	65.1	25.0	4 881	*minute*	

5.1 Construct Validity

Construct validity indicates whether the studied measures really represent what is investigated according to the research questions. The purpose of this study is to evaluate the behavior of developers of the company about testing.

All the tests are launched inside the Pharo IDE (Integrated Development Environment) contrary to the others studies of Beller et al. [2015], Blondeau et al. [2017], Gligoric et al. [2014]. The IDE also facilitate the usage of a type of tests selection. Moreover, the habits of developers on testing through their IDE or external tools are not known. So, the kind of the IDE and the external tools may create a bias in the data.

For research question 3.1, we supposed that the maximum number of tests per person is the number of unique tests executed by the developers. Because of the short duration of the study, it is possible they did not run all the tests of their projects.

We also change the way to measure the number of changes. For research question 1.1 and 3.2, we calculated the number of modifications. In fact, the Pharo developers are encouraged to create only short methods. So, when they modify their code, they often modify only one line. Modifications lead to two lines can insert some bias in the data. We could have used the number of AST node differences as Gligoric et al. [2014] to compute the number of modified line.

Some developers did not use our plugin because they are not using SUnit[4] for their tests. In fact, they are working on complex systems that they can not test with the default test framework. Consequently, our results can diverge from the real behavior of these developers.

The presence of continuous integration is another bias that can persist in the case study. Pharo developers use their own continuous integration platform to test the integrity of their code. Consequently, less tests can be made locally.

5.2 Internal Validity

Internal validity indicates whether no other variable except the studied one impacted the result.

The developers know that they are under study, Hawthrone effect[5] may have taken place, as the others experiments.

The sample may be biased towards developers who are actively interested in testing because participation was voluntary. In this sense, our results could be an overestimation of the real testing practices. However, people participating being in the core team of Pharo, where tests good practices are promoted. It may not represent the common behavior of the community.

5.3 External Validity

External validity indicates whether it is possible to generalize the findings of the study.

[4] The Pharo default test framework.

[5] Tendency of people to work harder and perform better when they participate in an experiment.

We make our case study close to the Blondeau et al. [2017] one, in different conditions.

All projects are studied in the Pharo environment. This can be an issue to generalize on other programming languages where testing is more difficult to do and done less intensively.

6. Related Works

The case study of Blondeau et al. [2017] is already described in Section 2.2 and those of Beller et al. [2015], Gligoric et al. [2014] are described in the paper of Blondeau et al. [2017].

Buffardi and Edwards [2014] led a study on seven students. They created a feedback tool through a website which show to the students if their code is enough concerned by their tests. The authors also surveyed to understand how students use the tests. They found that students are reluctant to adhere to test-first procedures. Moreover, they explained that students think testing is like debugging and not a part of the programming session. Like our, the paper of Buffardi and Edwards [2014] evaluates the testing habit and offer a way to improve it.

Orso and Rothermel [2014] summarized the different kinds of testing we can use during development. They described the testing strategies and how works the regression testing. Orso and Rothermel provides also information on frameworks for test execution. To develop plugins which collect data, we had to know how users can test their work. They particularly detailed the unit tests and the continuous integration. Both are used by the Pharo developers. They concluded, it is hard to describe all the testing possibilities. But their paper expose the main ones.

Panichella et al. [2016] explained it is difficult to understand the result of generated test. In fact, generated tests are hard to maintain and provide unusable data for "simple" developers. So, the authors developed a tool to generate test cases summaries. After the implementation of the tool, they study the impact of their work. The authors led an interview on 30 participants from both industry and academia. They concluded, their tool helps developers to find more bugs reducing testing effort.

7. Conclusion

To enhance the test usage in the Pharo community, we need to identify how developers behave. This paper is based on the study of Blondeau et al. [2017] and on two previous studies Beller et al. [2015], Gligoric et al. [2014]. The first one was led in a major IT company in a closed-source environment. The second was made on open-source project. In the last one, many participants are student but there are also industrials. Our study is just in the border of them. We are in an open-source context, with students and experimented developers.

Our main findings are:

- As expected, the number of tests which pass (94.45%) is more important in the Pharo community than it is in the others studies (less than 70%).
- We found that the Pharo developers run often their tests. For the same duration of study. They run two time more tests than the developers in the study of Blondeau et al. [2017].
- As in the study of Blondeau et al., developers do perform test selection, and to a high-level. For example, only 56% of the test sessions contain only one test. But, they are less aggressive in their selection because 71% of the tests are selected 50% of the time.
- The Pharo developers practice test selection depending of the duration of the tests. However, the number of tests in a test session is not important. Developers select the tests to run by their relevancy.

As future work, we plan to develop a tool to help the developers of the community for the selection of test. We would like to provide them the list of tests that can be impacted by their last modifications. The results collected in this study will be used to compare how they will behave after the implementation of the tool.

Acknowledgments

This work was supported by the Ministry of Higher Education and Research, and the Nord-Pas de Calais Regional Council, CPER Nord-Pas de Calais/FEDER DATA Advanced data science and technologies 2015-2020.

References

Moritz Beller, Georgios Gousios, Annibale Panichella, and Andy Zaidman. When, How, and Why Developers (Do Not) Test in Their IDEs. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 179–190, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3675-8. . URL http://doi.acm.org/10.1145/2786805.2786843.

Vincent Blondeau, Anne Etien, Nicolas Anquetil, Sylvain Cresson, Pascal Croisy, and Stéphane Ducasse. Test Case Selection in Industry: An Analysis of Issues Related to Static Approaches. *Software Quality Journal*, pages 1–35, 2016. ISSN 1573-1367. . URL http://dx.doi.org/10.1007/s11219-016-9328-4.

Vincent Blondeau, Anne Etien, Nicolas Anquetil, Sylvain Cresson, Pascal Croisy, and Stéphane Ducasse. What are the Testing Habits of Developers? A Case Study in a Large IT Company. In *Proceedings of the 21st IEEE International Conference on Software Maintenance and Evolution (ICSME'17), To appear*, Shanghai, China, August 2017.

Kevin Buffardi and Stephen H. Edwards. A formative study of influences on student testing behaviors. In *Proceedings*

of the 45th ACM Technical Symposium on Computer Science Education, SIGCSE '14, pages 597–602, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2605-6. . URL http://doi.acm.org/10.1145/2538862.2538982.

Emelie Engström, Per Runeson, and Mats Skoglund. A Systematic Review on Regression Test Selection Techniques. *Information and Software Technology*, 52(1):14–30, 2010.

Milos Gligoric, Stas Negara, Owolabi Legunsen, and Darko Marinov. An Empirical Evaluation and Comparison of Manual and Automated Test Selection. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pages 361–372. ACM, 2014.

Todd L. Graves, Mary Jean Harrold, Jung-Min Kim, Adam Porter, and Gregg Rothermel. An Empirical Study of Regression Test Selection Techniques. *ACM Trans. Softw. Eng. Methodol.*, 10(2):184–208, April 2001. ISSN 1049-331X. . URL http://doi.acm.org/10.1145/367008.367020.

Alessandro Orso and Gregg Rothermel. Software testing: A research travelogue (2000–2014). In *Proceedings of the on Future of Software Engineering*, FOSE 2014, pages 117–132, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2865-4. . URL http://doi.acm.org/10.1145/2593882.2593885.

Sebastiano Panichella, Annibale Panichella, Moritz Beller, Andy Zaidman, and Harald C. Gall. The impact of test case summaries on bug fixing performance: An empirical investigation. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 547–558, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3900-1. . URL http://doi.acm.org/10.1145/2884781.2884847.

Gregg Rothermel and Mary Jean Harrold. A Safe, Efficient Algorithm for Regression Test Selection. In *Proceedings of the International Conference on Software Maintenance (ICSM '93)*, pages 358–367. IEEE, September 1993.

Per Runeson and Martin Höst. Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *Empirical software engineering*, 14(2):131–164, 2009.

Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000. ISBN 0-7923-8682-5.

S. Yoo and M. Harman. Regression Testing Minimization, Selection and Prioritization: A Survey. *Software Testing, Verification and Reliability*, 22(2):67–120, 2012. ISSN 1099-1689. . URL http://dx.doi.org/10.1002/stvr.430.

Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 91–100. ACM, 2009.