Lessons Learned from an XP Experiment with Students: Test-First Needs More Teachings

Thomas Flohr[1] and Thorsten Schneider[2]

[1] Software Engineering Group, University of Hannover,
Welfengarten 1, 30167 Hannover, Germany
Thomas.Flohr@Inf.Uni-Hannover.de
[2] S²e (Secure Software Engineering),
Lange Strasse 33, 32051 Herford, Germany
Schneider@secure-software-engineering.com
http://www.secure-software-engineering.com

Abstract. For most XP techniques only a few experimental results on their effects are available. In October 2004 we started a medium-term experiment to investigate the impact of test-first compared to a classical-testing approach. We carefully designed a controlled experiment and conducted it with 18 graduated students randomly assigned to 9 pairs. Hypotheses dealt with development speed, number of test-cases and the test-coverage when applying the testing approaches. Results show differences however not significant ones. This paper also addresses other observations we made during the experimental run. Two major problems strongly affect the results of the experiment: the low number of data points and the non-trivial question, whether students really applied test-first all the time. Although we cannot provide any new results on testing to the research community, this paper contains valuable information about further experimental studies on this topic.

1 Introduction

The test-first approach is an XP technique [1] answering the question, when test-cases should be written. When applying the test-first approach one normally traverses a cycle similar to:

1. Write one single test-case.
2. Run this test-case. If it fails continue with step 3. If the test-case succeeds, continue with step 1.
3. Implement the minimal code to run the test-case successfully.
4. Run the test-case again. If it fails again, continue with step 3. If the test-case succeeds, continue with step 5.
5. Refactor the implementation to achieve the simplest design possible.
6. Run the test-case again, to verify that the refactored implementation still succeeds the test-case. If it fails, continue with step 5. If the test-case succeeds, continue with step 1, if there are still requirements left in the specification.

One claimed advantage of test-first is that one never writes more code than absolutely necessary. It is also said, that test-first leads to greater confidence in code correctness and when performing refactorings, etc.

J. Münch and M. Vierimaa (Eds.): PROFES 2006, LNCS 4034, pp. 305 – 318, 2006.

Of course these are hypotheses, which must be proven through experimental runs and depends on how good developers follow the process of test-first.

But what about the normal approach of testing? We called this approach classical-testing. It can be defined by the following process:

1. Read the specification.
2. Design the program
3. Write a few lines of code, some method(s), class(es), package(s) or the whole application.
4. Write some (new) or no test-suites, tests-sets, tests or test-cases gained from the specification.
5. Run the tests. If they succeed, continue with step 1, 2, 3 or 4. If the tests fail, continue with step 6. If there are no more further requirements and the code has been tested enough (perspective of the developer), exit the classical-testing cycle.
6. Remove the errors in the implementation and continue with step 5.

The classical-testing approach is less specified and does not give a good answer when to write a test-case. Our main research question concentrated on several aspects:

1. The difference between both testing approaches regarding the amount of time needed to complete the same number of story-cards (development speed).
2. The difference between both testing approaches regarding the test-coverage.
3. The difference between both testing approaches regarding the number of test-cases.

To verify (or falsify) our hypotheses, we conducted an experiment with students in October 2004. We opted for a medium-term experiment of several weeks (altogether 40 hours) because short-term studies observing development processes are limited to use small and artificial tasks. We had two groups: one applied test-first and the other one was our control group applying the classical-testing approach. Each group contained several pairs, so in fact the pairs were our subjects. The design of the experiment is outlined in section 3 and a much more detailed description can be found in [2].

Overall, this paper is structured as follows: section 2 contains the hypotheses we had using parts of the GQM approach as proposed by Basili [3, 4]. Section 4 contains the threats to validity. Section 5 deals with the execution of the experiment. In section 6 we give an overview of the results we received from the experimental run. Section 7 contains the discussion of the observations we made and how we judge their significance. A final conclusion follows in chapter 8.

1.1 Related Work

There is only small number of publications dealing with the observation of test-first in controlled experiments. Most empirical studies about XP techniques report on pair programming [5], XP or agile methods in general [6, 7].

Müller and Hagner observed 19 graduate students with some experience in XP [8]. In their experiment test-first was compared to traditional development separately. The students were divided in two groups to apply one of the approaches each. Each group had to implement the main class of a graph library which only contains declarations of methods. The experiment included unit and acceptance tests. Objects of