Introducing Automated Unit Testing into Open Source Projects

Christopher Oezbek

Introducing Automated Unit Testing into Open Source Projects

Christopher Oezbek

Freie Universität Berlin
Institut für Informatik
Takustr. 9, 14195 Berlin, Germany
christopher.oezbek@fu-berlin.de

Summary. To learn how to introduce automated unit testing into existing medium scale Open Source projects, a long-term field experiment was performed with the Open Source project FreeCol. Results indicate that (1) introducing testing is both beneficial to the project and feasible for an outside innovator, (2) testing can enhance communication between developers, (3) an active stance is important for engaging the project participants to fill a newly vacant position left by a withdrawal of the innovator.

1 Introduction

The Open Source development paradigm based on copyleft licenses, global distributed development and volunteer participation has become an alternative development model for software, competing on par with proprietary solutions in many areas. Open Source software especially has established a good track record related to quality measures such as number of post-release defects or time to resolution for bug reports [8, 10] based on its open access to source code, openness to participation and use of peer review [13].

The present study originated in the question how to further improve a project's ability to produce high-quality software. From a software engineering perspective the answer proposed in previous work was to introduce innovative processes and tools into Open Source projects [9]. But is such introduction feasible? How must an innovator act to achieve adoption of the introduced innovation? The present study is a first exploration on these questions.

Quality assurance was chosen as the area for improvement and automated unit testing [14] as the innovation, because it represents a well-known and established quality assurance practice from industry, which should easily provide benefit to Open Source projects. Methodologically, an introduction conducted by a researcher is in-between action research [1] and a field experiment [5], because the researcher is interacting in the field but using his own agenda.

The study proceeded in four steps: First, a theoretical model was built of how to introduce automated testing to make the process reproducible by others. This model prescribes activities and goals for lurking [11], joining and acting [2, 12],

collaborating and phase-out of the innovator and is shown in Figure 1. Second, the project FreeCol was selected from the project hoster SourceForge.net based on several criteria such as being medium-sized and open for outside participation to ensure interesting interaction and relevant results. FreeCol was started in March 2002, trying to recreate the turn-based strategy game Colonization.[1] FreeCol is a client-server application written in Java and regularly ranked in the top 50 of Open Source projects at the SourceForge.net with on average 16,500 copies downloaded per month. The project has 60 members enlisted on the project page[2] of which 46 are designated as developers and 13 of which are deemed active.[3] The project already had one test case using JUnit at the beginning of this study. Third, testing was introduced into this project, which took place in April and May 2007 following the phase model shown in Figure 1 and resulting in 57 test-cases. In September 2007 the test-suite was broken by a large scale refactoring and the project maintainers asked for a repair, which was performed as a last activity in the project. Fourth and last, the outcome of the introduction was analyzed post-hoc in September 2009 by means of (1) data mining the source code repository [6] of the project for test coverage and test failures [15] and (2) qualitatively analyzing the mailing-list communication on testing.[4]

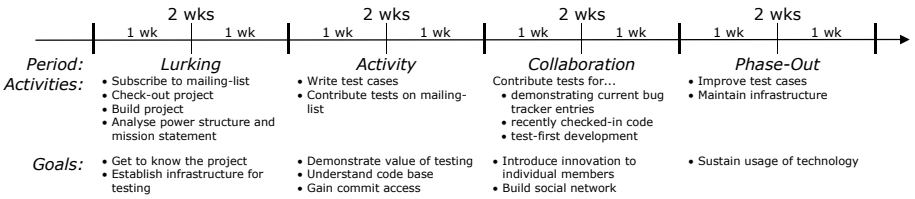

Fig. 1. Phases in the introduction process.

2 Results

Looking back from April 2007 to August 2009 we find the introduction a success based on four quantitative indicators: (1) On average 9.9 test cases were being added per month, raising their number from 73 at the departure of the innovator to 277 in August 2009 (Figure 1) covering 23% of the source code (see Figure 1); a respectable figure for a UI-oriented application such as FreeCol. (2) The percentage of commits affecting test cases is stable between 10.0% to

[1] `http://www.freecol.org/history.html`

[2] `http://sourceforge.net/project/memberlist.php?group_id=43225`

[3] `http://www.freecol.org/team-and-credits.html`

[4] All scripts used for producing the results in this study as well as intermediate data to reproduce the statistical analysis are available at `http://www.inf.fu-berlin.de/inst/ag-se/pubs/test-intro2009data.zip`.

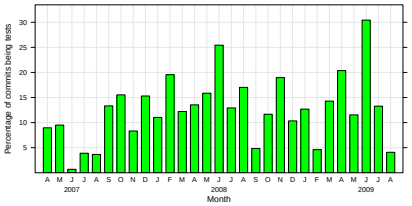

Fig. 2. Percentage of test commits

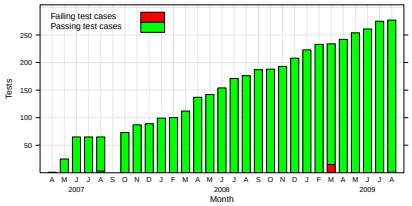

Fig. 3. Number of test cases

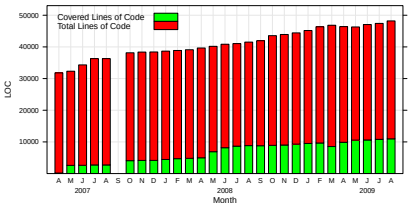

Fig. 4. Lines covered

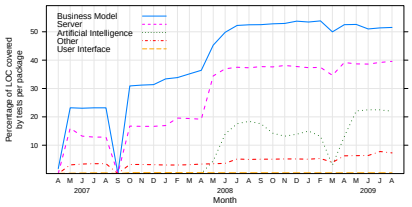

Fig. 5. Coverage per modules

15.1% per month with 95% confidence. (3) The source code passed the tests in most months (Figure 1). (4) Of the 32 developers who have ever committed to the FreeCol source code, 16 participated in testing. On the mailing-list several developers voiced their positive attitude towards testing, e.g. [fc:2518][5] and [fc:3351].

2.1 Insights into automated testing

The most interesting insight regarding the *use* of testing is that test-cases have been used in FreeCol to enhance communication in two ways: (1) If facing a defect without knowledge to repair or understand it, we have seen developers write failing test cases which reproduce the failure and use the test-case as a more concise alternative for communicating the failure (for instance [fc:2606] [fc:2610] [fc:2640] [fc:2696] [fc:3983]). (2) When facing ambiguity about how FreeCol should behave, we have seen developers codify their opinion as test cases [fc:3276] [fc:3056] or existing tests being the starting-point for discussions about how FreeCol should behave [fc:1935]. This is a second major advantage beside the regression detecting abilities of having a test suite (see for instance [fc:3961] or [fc:4431]).

As a second insight we found that testing varied largely by module. While the business logic including the game objects attained more than 50% coverage, other areas such as the server module at 40% and the artificial intelligence module at 22% are less tested and UI testing is completely absent from FreeCol (see

[5] Citations such as [fc:2518] are hyperlinks to e-mails from the Freecol Developer Mailing-list and are numbered in the order they were posted.

Figure 1). How to expand the coverage of underrepresented modules remains an open question.

2.2 Insights into innovation introduction

On introducing innovations two main results were found: (1) FreeCol excelled at incrementally expanding innovation usage over a long time and maintaining the existing code base. Yet, it required assistance by an innovator or particularly skilled individual to achieve radical expansion of coverage. (2) When detaching from FreeCol Open Source project, the innovator needed to signal this to release ownership of responsibilities and code.

The first insight was deduced from the two notable expansions in coverage over the last two years. The first was the expansion of coverage from 0.5% to 10% by the innovator when introducing automated testing in 2007, and the second in April and May of 2008 when one developer expanded coverage from 13% to 20%. Otherwise coverage remained stable over the two years, in contrast to the number of test cases which increased constantly (see Figure 1). On the mailing-list a hint can be found that this is due to the difficulty of constructing scaffolding for new testing scenarios [fc:4147]. This leads to a question regarding our understanding of Open Source projects: If—as studies consistently show—learning ranks highly among Open Source developers' priorities for participation [3], then why is it that coverage expansion was conducted by just two project participants? It seems that the innovator and the one developer both brought existing knowledge about testing into the project and that project participants' affinity for testing and their knowledge about it expanded only very slowly. A similar result was reported by Hahsler who studied adoption of design patterns. He found for most projects that only one developer used patterns [4, p.121]. This should strike us as strange, if sharing of best practices and knowledge did occur frequently.

The second insight for innovation introduction resulted from phasing-out the innovator's involvement in May and September 2007. The first attempt in May failed and the test suite was unmaintained during a large-scale refactoring and soon "spectacularly broken", as one maintainer put it. Comparing this with the second more successful departure in September, which resulted in the tests being maintained by one of the maintainers, we find that the primary difference in behavior is one of signaling and ownership. When the innovator first detached, ownership was neither considered nor was the withdrawal communicated to the project. Yet, as Mockus et al. found in their case study of Apache and Mozilla, code ownership is achieved implicitly for code the developer is "known to have created or to have maintained consistently" [8, p.318]. And while such code ownership "doesn't give them [the owners] any special rights over change control", it stipulates a barrier for other developers to engage with the code.[6]

[6] See for instance [7] for a discussion on code ownership as an important part of the mental model of developers.

Only when the test suite broke completely after the refactoring, did it become apparent that it was unmaintained. Thus, when phasing-out the innovator's engagement again after fixing the test suite in September 2007, one discussion (see [fc:2182]) was sufficient to create an understanding of shared code ownership in testing. When the innovator disengaged, one of the maintainers picked up the role of maintaining the test cases successfully, keeping the percentage of test affecting commits at around 10% of the total commits (see Figure 1), until another developer assumed a more active role in testing.

When analyzing the contributions of developers to the testing effort, we find that besides the innovator and the maintainer there were two individuals who contributed extensively to testing. Interestingly, as their contribution increased and waned over time, the maintainer who had already picked up the testing effort initially seemed to adjust his own contribution accordingly. As contributions of the other developers never exceeded five testing commits per month, it seems that the project adopted a flexible code ownership strategy. In this approach, the role of a "test master" exists who contributes heavily to testing and is pivotal to the expansion of test coverage and development of knowledge regarding testing. This role is not formally but rather implicitly assigned and acknowledged explicitly in the project only for instance when a core developer — stumped by a difficulty regarding testing — asked: "Any suggestions, particularly from the resident test expert [name of developer]?" [fc:4446].

3 Limitations and conclusion

This study presents a first exploration into the research area of actively improving an Open Source project and, as a single case using unit testing as the innovation, can not generalize far. Other projects might have different attitudes towards testing, the domain of the software might make testing more difficult, or the researcher as the innovator could have introduced a noticeable bias. For future work, more projects, other innovations and more data source per project should thus be studied, though an active approach like in this study can not be scaled very far due to the effort associated with each case.

To conclude, this study has shown that the introduction of a code-centric process innovation such as automated testing is feasible for an outside innovator using a four-stage model. Regarding automated testing this study has found (1) a number of episodes in which test cases were used for communicating bug reports, and (2) a lack of the state of the practice regarding automated testing. The results for the innovator are that (1) external participants are important for the radical expansion of innovation use, and (2) signaling the departure of the innovator is important even for an innovation which has an explicit signaling mechanisms such as test cases failures. Open questions were raised about the extent to which participants are able to learn about new innovations.[7]

[7] Acknowledgments: Dan Delorey provided the author with a list of all java projects on Sourceforge.net that had more than 5 active developers over the course of 2006.

References

1. D. E. Avison, F. Lau, M. D. Myers, and P. A. Nielsen. Action research. *Commun. ACM*, 42(1):94–97, 1999.
2. N. Ducheneaut. Socialization in an Open Source Software community: A socio-technical analysis. *Computer Supported Cooperative Work (CSCW)*, V14(4):323–368, Aug. 2005.
3. R. A. Ghosh, R. Glott, B. Krieger, and G. Robles. Free/Libre and Open Source Software: Survey and study – FLOSS – Part 4: Survey of developers. Final Report, International Institute of Infonomics University of Maastricht, The Netherlands; Berlecon Research GmbH Berlin, Germany, June 2002.
4. M. Hahsler. A quantitative study of the adoption of design patterns by Open Source software developers. In S. Koch, editor, *Free/Open Source Software Development*, chapter 5, pages 103–123. Idea Group Publishing, 2005.
5. G. W. Harrison and J. A. List. Field experiments. *Journal of Economic Literature*, 42(4):1009–1055, Dec. 2004.
6. H. Kagdi, M. L. Collard, and J. I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance and Evolution: Research and Practice*, 19(2):77–131, 2007.
7. T. D. LaToza, G. Venolia, and R. DeLine. Maintaining mental models: a study of developer work habits. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 492–501, New York, NY, USA, 2006. ACM.
8. A. Mockus, R. T. Fielding, and J. Herbsleb. Two case studies of Open Source Software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, 2002.
9. C. Oezbek and L. Prechelt. On understanding how to introduce an innovation to an Open Source project. In *Proceedings of the 29th International Conference on Software Engineering Workshops (ICSEW '07)*, Washington, DC, USA, 2007. IEEE Computer Society. reprinted in UPGRADE, The European Journal for the Informatics Professional 8(6):40-44, December 2007.
10. J. W. Paulson, G. Succi, and A. Eberlein. An empirical study of open-source and closed-source software products. *IEEE Transactions on Software Engineering*, 30(4):246–256, 2004.
11. J. Preece, B. Nonnecke, and D. Andrews. The top five reasons for lurking: Improving community experiences for everyone. *Computers in Human Behavior*, 20(2):201–223, 2004. The Compass of Human-Computer Interaction.
12. L. Quintela García. Die Kontaktaufnahme mit Open Source Software-Projekten. Eine Fallstudie. Bachelor thesis, Freie Universität Berlin, 2006.
13. E. S. Raymond. The cathedral and the bazaar. *First Monday*, 3(3), 1998.
14. J. A. Whittaker. What is software testing? And why is it so hard? *IEEE Software*, 17(1):70–79, 2000.
15. H. Zhu, P. A. V. Hall, and J. H. R. May. Software unit test coverage and adequacy. *ACM Comput. Surv.*, 29(4):366–427, Dec. 1997.

Many thanks also to Gesine Milde, Florian Thiel, Lutz Prechelt, the FreeCol maintainers and test masters, and two anonymous reviewers who read early versions of this paper.