Reference: In Proceedings of the Fourth Conference on Open Source Systems
(OSS 2008). Springer Verlag, 2008, to appear.

Continuous Integration in Open Source Software Development

Amit Deshpande

TechSolve, Inc.
6705 Steger Drive
45237 Cincinnati, OH, U.S.A.

amit@amit-deshpande.com

Dirk Riehle

SAP Research, SAP Labs LLC
3475 Deer Creek Rd
94304 Palo Alto, CA, U.S.A.

dirk@riehle.org

Abstract. Commercial software firms are increasingly using and contributing to open source software. Thus, they need to understand and work with open source software development processes. This paper investigates whether the practice of continuous integration of agile software development methods has had an impact on open source software projects. Using fine-granular data from more than 5000 active open source software projects we analyze the size of code contributions over a project's life-span. Code contribution size has stayed flat. We interpret this to mean that open source software development has not changed its code integration practices. In particular, within the limits of this study, we claim that the practice of continuous integration has not yet significantly influenced the behavior of open source software developers.

1 Introduction

Open source software is having a major impact on software and its production processes. Many software products today contain at least some open source software components. Some commercial products are completely open source software. For commercial software firms it is therefore important to understand open source software development processes and ensure that employed developers are capable of participating in them [13].

Open source software development processes are different from commercial software development processes like the spiral model [5] or agile methods [4]. Open source processes can vary from project to project. With no single well-defined open source process, it is unclear for hiring managers what skills to look for when hiring people for open source software projects.

One common hypothesis is that open source software processes are similar to agile development processes and hence that agile method skills could serve as a proxy for open source software development skills [12] [14]. However, this hypothesis has never been validated. Indeed, it is impossible to prove as long as open source processes remain as vaguely defined as they are today.

This paper analyses the use of continuous integration within open source software projects. Continuous integration is a practice where software developers work

in small increments, contributing code to the project frequently, and ensuring that the project compiles and passes its test suites at any time [11].

Continuous integration is a core agile methods practice. If it is also an important open source development practice, managers can hire more confidently developers for open source projects who have been educated in agile methods and who have appropriate experience with continuous integration.

To answer the question "Have open source projects increasingly been using continuous integration?" we looked at the individual actions of developers from 5122 active open source software projects. Specifically, we looked at the size (in source lines of code) of code contributions and their frequency.

The paper is organized as follows. Section 2 frames our hypothesis. Section 3 discusses our database and the details of the validation approach. Section 4 presents the results of the analysis. Section 5 discusses the shortcomings and limitations of the analysis. Section 6 discusses related work. Section 7 concludes the paper.

2 Is Open Source Using Continuous Integration?

Agile software development methods like Extreme Programming "embrace change": There is no insistence on following a master-plan, but the willingness to respond to (requirements) change quickly [3]. This is in stark contrast to traditional approaches to software development with their insistence on detailed planning ahead.

It is not obvious how open source projects relate to agile methods. According to Fugetta, "[Agile methods] can be equally applied to proprietary and open source software" [1]. Thus, with the advent of agile methods around 1997/1998, one might expect to see an adoption of agile methods practices in open source software development over the last ten years.

Thus, we are asking: *What is the impact of (the arrival of) agile methods on open source software development processes?* And then, more specifically: *Is continuous integration a widely used practice in open source software projects?* The answer to this question is important to software firms who engage with open source software projects, because they need to hire, cultivate, and maintain the appropriate skill sets of their employees to work with open source projects.

We chose continuous integration as a representative practice of agile methods like Extreme Programming [3] or SCRUM [6]. "Continuous integration is the practice of making small well-defined changes to a project's code base and getting immediate feedback to see whether the test suites still pass" [11].

If continuous integration had not been employed in open source projects before its first formulations around 1997 but has been increasingly employed since then, we would expect to see a statistically significant behavioral change in how open source software developers contribute code to open source projects. Specifically: *We would expect to see that the average size of code contributions, the individual check-in into a source code repository, would have gone down over the last ten years, and we would expect to see that the average frequency of such check-ins has gone up.*

To investigate this hypothesis, we analyzed more than 5000 active open source software projects, as discussed in the following sections.

3 Data Source and Approach

For the analysis, we use the database of the open source analytics firm Ohloh.net [10]. The database contains 5122 current and active open source projects from 1990 until May 2007.

We analyze the last 16 years from January 1990 to December 2006. The database contains the most popular open source projects as measured by the number of in-links to their website. The in-links are provided by the Yahoo! search engine.

Ohloh.net goes down to the level of the individual developer action. Specifically, Ohloh.net provides each individual commit action of all the projects over their entire history. A commit is the action with which a developer contributes a piece of code to the project's repository.

3.1 The Commit Action

A commit action is the atomic contribution of source code to a code repository. We measure the amount of work that went into a commit in Source Lines of Code (SLoC) added, changed, and removed, ignoring empty lines and comments. SLoC is based on applying the Unix diff command to two consecutive source file versions.

Unfortunately, Ohloh.net counts a changed line as an old line removed and a new line added. One line of code edited will be tracked as one line added and one line removed. Table 1 represents the resulting three cases. From this data alone we cannot determine the exact number of lines of code affected. Thus, in Section 3.3 we will take two approaches representing the upper and the lower bound respectively.

3.2 Commit Action Filtering

We want to study the size of a commit over time. In order to perform a valid analysis we impose four filters to exclude special and undesired events:

1. The initial commit (creation) of a file
2. The final commit (deletion) of a file
3. Commits of a size larger than three standard deviations above the average
4. Commits of a size smaller than three standard deviations below the average

Filter 3 eliminates the cases where a developer adds a large amount of external code to an existing file and Filter 4 eliminates commits where a large amount of code is removed, like in a major refactoring. None of these events helps us understand changes in commit size patterns.

3.3 Analysis of Commit Size

Approach 1 adds the number of lines of code added in one commit to the number of lines of code removed in the same commit. Following the discussion of Section 3.1, editing a single line will be counted as one line added plus one line removed. Thus, the size calculated by Approach 1 represents the upper bound for the commit size.

Table 1: Data semantics (SLoC = Line of Source Code)

Case	Commit Action	SLoC Added	SLoC Removed
1	One line of code added	1	0
2	One line of code removed	0	1
3	One line of code edited	1	1

Figure 1 shows the average size of a commit (as defined by Approach 1) per week from 1990 to 2006. The size is shown on the Y-axis and time is shown on the X-axis. To smooth out fluctuations and to highlight the longer-term trends or cycles we fit a moving average (period = 10) curve on the plot. The commit size is nearly constant and shows no significant trend or pattern over time.

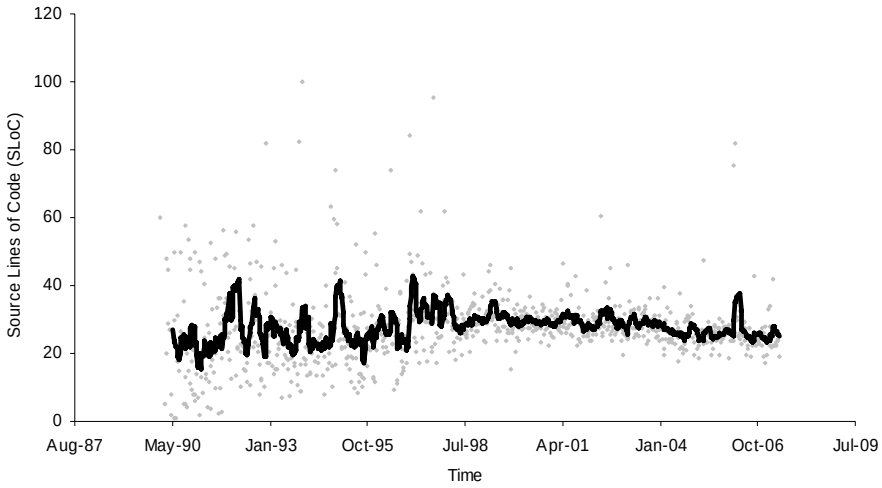

Figure 1: Upper bound of commit size in SLoC over time (Approach 1)

Approach 2 subtracts the number of lines of code removed from the number of lines of code added. Thus, an edited line will not be counted, and the value reflects the net change to the size of a file only. Thus, the commit size as calculated by Approach 2 represents the lower bound for the actual size of the commit.

Figure 2 shows the average size of a commit (as defined by Approach 2) per week using the same axes labeling and moving average curve. The commit size is nearly constant and shows no significant trend or pattern over time.

We tested the validity of our results using hypothesis testing. The null hypothesis is: The average commit size in any year is equal to the average size during the next year. The research hypothesis is: The average commit size in any year is greater than the average commit size during the next year.

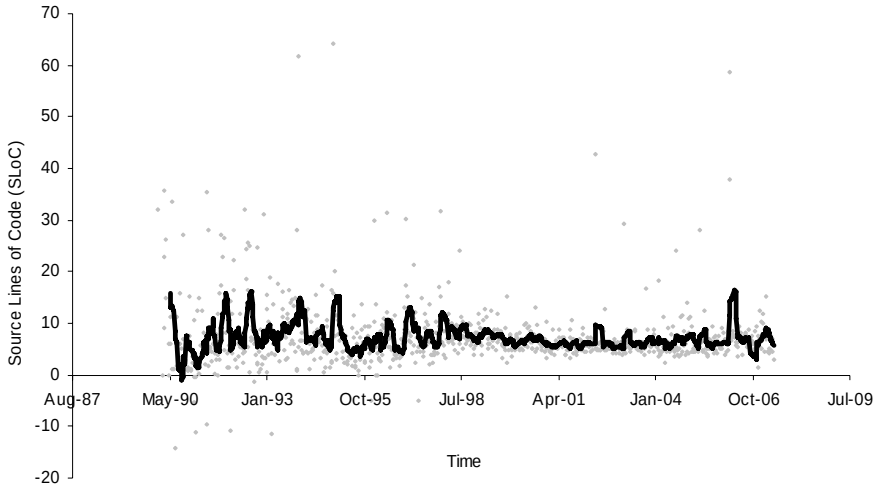

Figure 2: Lower bound of commit size in SLoC (Approach 2)

No clear picture resulted at the 95% confidence level. In a few cases, we could reject the null hypothesis, in most we could not. Within the limits of the study we conclude that the commit size remained constant with no significant trend or pattern over time.

4 Results of Analysis

Our analysis does not validate the hypothesis that open source software developers are practicing continuous integration. Our indicator, the average size of a commit, remained almost constant over the years with no significant trend or pattern. We applied a similar analysis (not given in this short paper) which showed that the commit frequency has remained stable over time as well.

One explanation is that open source projects have not adopted continuous integration on any significant scale. An alternative interpretation is that open source software development has always practiced continuous integration. In that case, the advent of agile methods did not lead to any changes, simply because it had long arrived in open source software development before.

5 Limitations of Analysis

The analysis and the conclusions we draw have at least the following limitations.

- *Sample size.* We considered 5122 open source projects. The total number of open source projects is much larger. However, our data source focuses on active projects of significant popularity. So we believe the sample we are using is relevant for analyzing agile methods practices in open source.

- *Data quality.* We couldn't calculate exact numbers of commit size due to the way Ohloh.net tracks changes to source code files. (See Section 3.1.) We believe that working with a lower/upper bound does not restrict the validity of our results as we care about the trend and not the actual values.
- *Data incompleteness.* Some amount of revision control information in open source projects has been lost forever, as projects have moved from one code repository to another. However, this is old data, and we believe the lack of some of the early histories do not affect the validity of our conclusions.
- *Improper summation across different programming languages.* One may argue that commit size depends on the programming language. We analyzed variation across languages and found that the programming language does not have a significant impact on the results presented in this paper.
- *Improper summation across different project sizes.* Some open source projects are small, some are large. Barry Boehm argues that agile methods don't scale well beyond a certain size limit [2]. Our data confirms within the given limitations that the practice of continuous integration has not found increasing adoption in open source projects over the last ten years.

We are working to remove these and other limitations. However, we believe that while the respective critiques can be made, the effects are limited, as argued above.

6 Related Work

Angioni et al. describe MADD, a "Methodology for Agile Distributed Development" and compare the similarity, differences and applicability of agile practices in open source development [9]. Less than 73% of the developers were knowledgeable about agile development practices. Less than 10% of the developers used core agile practices like pair programming or small iterations.

Turnu et al. studied the effect of applying agile methodology in the open source development environment [8]. Specifically, they studied the effects of applying Test Driven Development on open source development by using a simulation model developed from the Apache HTTP server project data. They concluded that use of agile methods in open source development can yield better results in terms of code quality.

7 Conclusions

Software firms seeking to work with open source software need to understand open source software development processes. Our analysis of the size of code contributions by open source software developers indicates that the core agile methods' practice of continuous integration has not had a significant impact on open source development practices. Hence, we conclude within the limits of our study that agile method skills are not a good proxy for open source software development skills.

Acknowledgements

We would like to thank Prem Devanbu and Gregorio Robles for their feedback on earlier versions of the paper as well as their encouragement for the work presented. We also would like to thank Oliver Arafat and Mario Fernandez for proofreading the paper. Finally, Amit Deshpande performed parts of this work while being employed at SAP Labs, LLC.

References

[1] Fugetta, A.: Open Source Software—An Evaluation. The Journal of Systems and Software (66) 77-90.
[2] Boehm, B.: Get Ready for Agile Methods, with Care. IEEE Computer (January 2002) 64-69.
[3] Beck, K.: Extreme Programming Explained: Embrace Change. Addison Wesley, 1999.
[4] Cockburn, A.: Agile Software Development. Addison-Wesley, 2001.
[5] Boehm, B.: A Spiral Model of Software Development and Enhancement. IEEE Computer (May 1988) 61-72.
[6] Schwaber, K.: Agile Software Development with SCRUM. Prentice-Hall, 2001.
[7] Lawton, M.: Open Source Business Models. IDC, 2007. Retrieved on Sept 13, 2007, from http://www.idc.com/getdoc.jsp?containerId=IDC_P13018
[8] Turnu, I., Melis, M., Cau, A., Setzu, A., Concas, G., and Mannaro, K. Modeling and Simulation of Open Source Development using an Agile Practice. J. Syst. Archit. 52, 11 (Nov. 2006) 610-618.
[9] Angioni, M., Sanna, R., Soro, A.: Defining a Distributed Agile Methodology for an Open Source Scenario. In Scotto, M., Succi, G. (eds.): Proceedings of the First International Conference on Open Source Systems. Springer Verlag (2005) 209-214.
[10] Ohloh Corporation, see http://www.ohloh.net.
[11] Duvall, P.: Continuous Integration: Improving Software Quality and Reducing Risk. Addison-Wesley, 2007.
[12] Koch, S.: Agile Principles and Open Source Software Development: A Theoretical and Empirical Discussion. In: Extreme Programming and Agile Processes in Software Engineering. Springer-Verlag (2004) 85–93.
[13] Morkel Theunissen, WH, Boake, A. Kourie, DG. In Search of the Sweet Spot: Agile Open Collaborative Corporate Software Development. In Proceedings of the 2005 SAISCIT: 268-277.
[14] Warsta, J. and Abrahamsson, P. Is Open Source Software Development Essentially an Agile Method? In: 3rd Workshop on Open Source Software Engineering, Portland, Oregon, USA.