The Commenting Practice of Open Source

Oliver Arafat
Siemens AG, Corporate Technology
Otto-Hahn-Ring 6, 81739 München
oarafat@gmail.com

Dirk Riehle
SAP Research, SAP Labs LLC
3410 Hillview Ave, Palo Alto, CA 94304, USA
dirk@riehle.org

Abstract

The development processes of open source software are different from traditional closed source development processes. Still, open source software is frequently of high quality. This raises the question of how and why open source software creates high quality and whether it can maintain this quality for ever larger project sizes. In this paper, we look at one particular quality indicator, the density of comments in open source software code. We find that successful open source projects follow a consistent practice of documenting their source code, and we find that the comment density is independent of team and project size.

Categories and Subject Descriptors D.2.8 [**Metrics**]: Software Science

General Terms: Measurement, Documentation, Languages

1. Introduction

Open source software has become an important part of commercial software development and use. Its continued growth emphasizes this importance [1]. Projects like the Linux kernel and the Apache web server demonstrate that open source software can be of high quality. Most interestingly, open source projects have reached a size and complexity that rivals the size of some of the largest commercial projects [2], yet they are being developed in a manner quite different from traditional software engineering processes.

Our research goal is to understand the processes and practices of open source software development and to assess whether they can be applied in a corporate environment. This has become particularly important because most well-known processes find it hard to scale up to larger project sizes. Traditional life-cycle processes like the waterfall model are best used in contexts where the problem domain is well understood [15]. Agile software development methods can cope with changing requirements and poorly understood problem domains, but typically require co-location of developers and fail to scale to large project sizes [16].

A host of successful open source projects in both well and poorly understood problem domains and of small to large sizes suggests that open source can cope both with changing requirements and large project sizes. In this paper we focus on one particular code metric, the comment density, and assess it across 5,229 active open source projects, representing about 30% of all active open source projects. We show that commenting source code is an ongoing and integrated practice of open source software development that is consistently found across all these projects. This practice is independent of the chosen programming language, the age of project, the size of the project in lines of code, and their team sizes.
The contributions of this paper are the following:

- It assesses the metric of comment density for the first time for open source projects on a broad scale;
- It shows that commenting source code is a consistently exercised practice of open source software development;
- It reviews a variety of dependencies between properties of open source projects and their comment density.

The paper is organized as follows. Section 2 reviews our data source and the taken approach. Section 3 gives an aggregate overview of comment density in open source projects, discusses how it varies by programming language, and shows how commenting source code is a consistently followed practice in open source. Section 4 reviews the dependencies of comment density on multiple variables relevant to scaling up projects. Section 5 summarizes our conclusions and discusses threats to their validity. Section 6 reviews related work and Section 7 ends the paper with some final conclusions and an outlook on future work.

2. Data source, filters, and definitions

Our analyses use the database of the open source analytics firm Ohloh, Inc. [9]. The data is accessible through an API

[10]. We work with a database snapshot of March 2008, but have cut off all analysis data after December 31st, 2007. The database contains data from about 10,000 open source projects, including project name, description, committer information, and the code contribution history of a project.

In this paper we are interested in active well-working open source projects, not dead projects. We define and apply an active project filter to let a project pass only if it is at least two years old and if the code activity of the last year has been at least 60% of the activity of the previous year. This active project filter reduces the original 10,000 projects to 5,229 projects. Using a comparable approach, Daffara estimates that there were about 18,000 active open source projects in the world by August 2007 [7], so our sample size represents about 30% of the total population.

The code contribution history is a time series of commits (code contributions) to the source code repository. A ***commit*** represents a set of changes to the source code performed as one chunk of work. When analyzing commits we apply filters to improve data quality. For example, we filter out file rename and move operations where no real work has been done.

A commit consists of multiple diffs. A diff describes the differences between two consecutive versions of the same file as changed in the commit. It is split into three parts: The number of lines of source code that have been added to the file or removed from it, the number of comment lines that have been added or removed, and the number of empty lines that have been added or removed.

- A ***source line of code***, or SLoC, is a physical line in a source file that contains source code.
- A ***comment line***, or CL, is a physical line in a source file that represents a comment.
- A ***line of code***, or LoC, is either a source line of code or a comment line.
- An empty line is just that.

The Ohloh diff tool recognizes every comment character and characters respectively that are defined and valid within one particular programming language such as the triple quotes in Ruby.Furthermore, it also accounts for external mark up languages such as Plain Old Documentation (POD) which is widely used in Perl. Additionally, the Ohloh diff tool recognizes comments that span multiple lines [11]. It does not, however, recognize whether a code line was changed; rather, it counts a changed line as an old line removed and a new line added. While it is not possible to determine a posteriori whether a line was changed or removed and then added, heuristics exist to predict which variant was the case. Most variants of the Unix tool diff, for example, implement such a heuristic by solving the Longest Common Subsequence problem. We have developed a statistic that determines the probabilities of whether a line was changed or removed and added [12]. Our algorithms use this statistic to determine aggregate values like commit sizes and comment densities.

The ***commit size*** of a commit is the number of lines of code affected in a commit, whether added, removed, or changed. When calculating commit sizes we apply the statistic explained above.

The ***comment density*** of a file or a group of files or the whole source code base of a project is defined as the number of comment lines divided by the number of lines of code of the same code body [4].

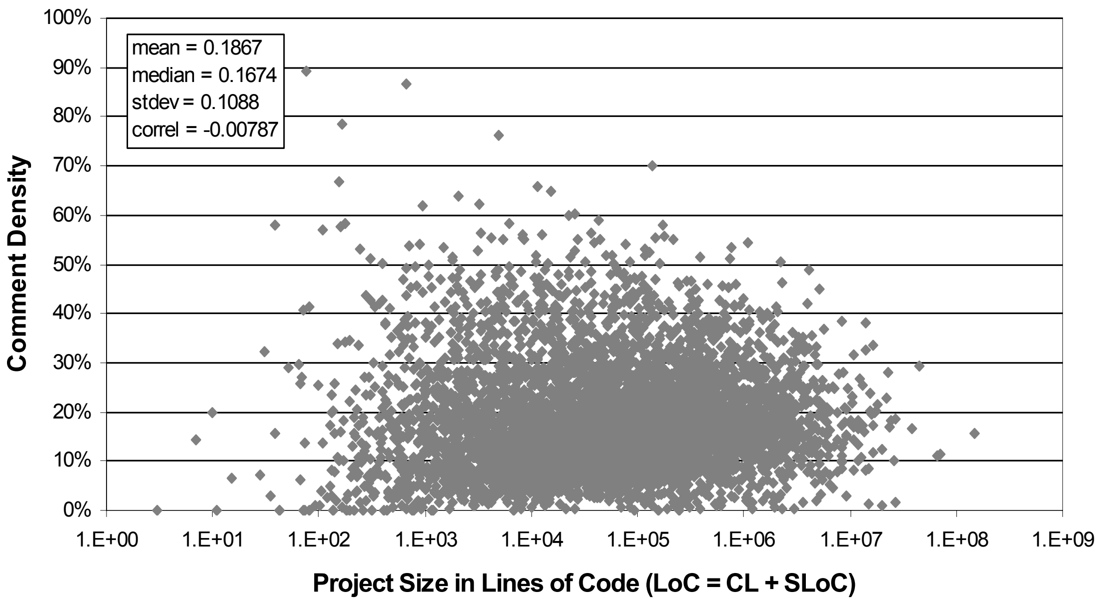


Figure 1: Comment density as a function of lines of code for a given project.

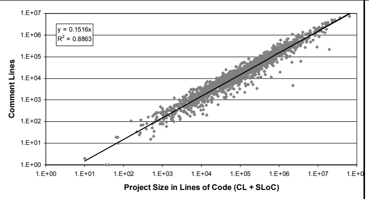
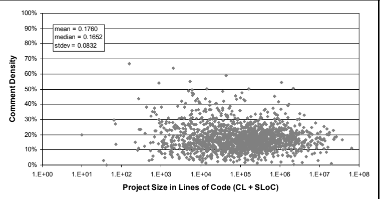
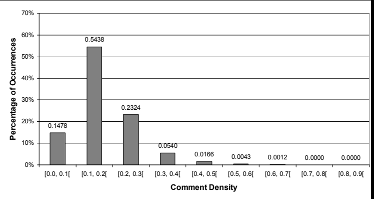
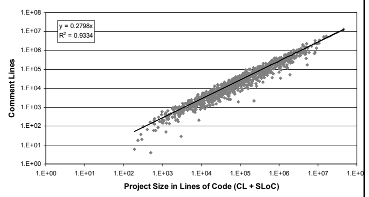
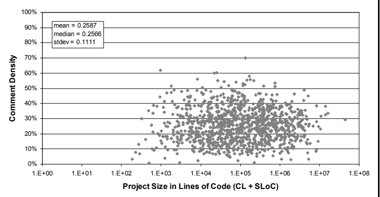
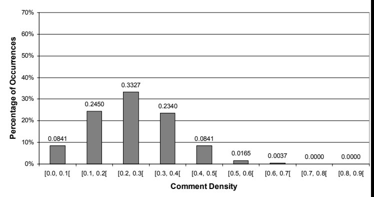
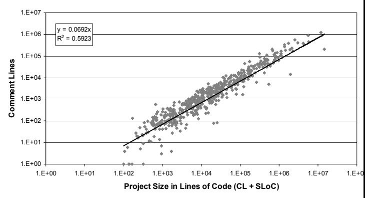
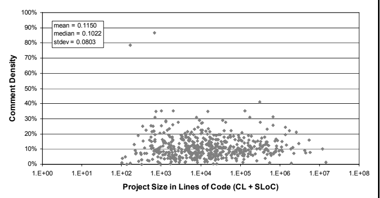
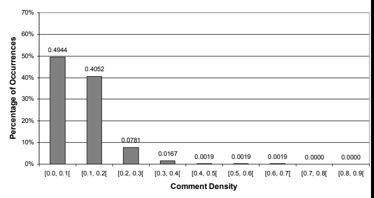
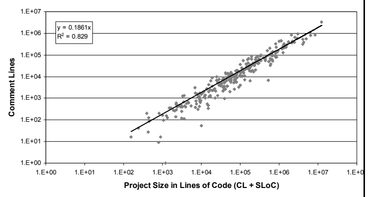
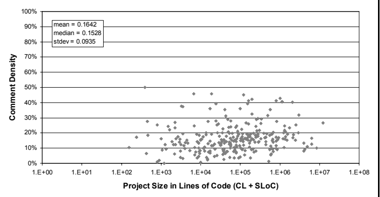
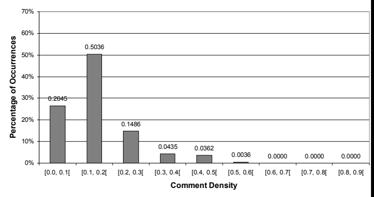
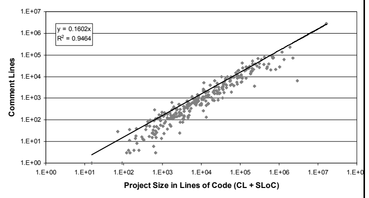
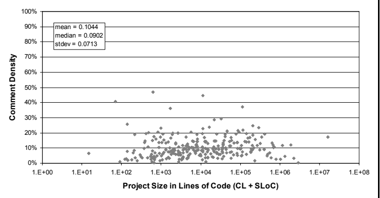
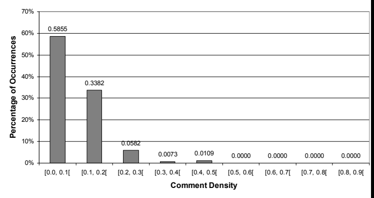
	Comment lines as a function of project size	Comment density as a function of project size	Binned distribution of projects as a function of comment density
C and C++			
	y = 0.1516x; R^2 = 0.8863; mean = 0.1760; median = 0.1652; stdev = 0. 0832; population size = 1621		
Java			
	y = 0.2798x; R^2 = 0.9334; mean = 0.2587; median = 0.2566; stdev = 0.1111; population size = 1085		
Python			
	y = 0.0692x; R^2 = 0.5923; mean = 0.1150; median = 0.1022; stdev = 0.0803; population size = 534		
JavaScript			
	y = 0.1861x; R^2 = 0.829; mean = 0.1642; median = 0.1528; stdev = 0.0935; population size = 276		
Perl			
	y = 0.1602x; R^2 = 0.9464; mean = 0.1044; median = 0.0902; stdev = 0.0713; population size = 273		

Figure 2: Comment density of projects with different dominant programming languages.

3. Comment density in open source

This Section provides an overview of comment density in open source projects, discusses how it varies by programming language, and investigates the practice of commenting source code. We find the commenting code is a common consistently exercised practice of open source projects.

3.1 Overview

As we have already shown in [18] and depicted in figure 1 the average comment density in our sample distribution is about 19% so about one line of code in five lines is a comment line varying widely per individual project with a standard deviation of 10.88%. The amount of comments in a given source code body can be interpreted as an indicator of its quality and maintainability [4] [5].

3.2 Influence of programming languages

The comment density varies significantly by programming language. Figure 2 shows the graphs, their models, and mean, median, and standard deviation for five popular languages. Of the five languages, Java has the highest mean of comment lines per source lines at 25.87% or one comment line for three source code lines. Perl has the lowest mean with 10.44% or about one comment line for nine source code lines.

It is interesting to discuss the differences between programming languages. It appears to be less a difference between any two particular programming languages, but rather it seems to be a difference between categories of languages.

One way of categorizing the five languages is by whether they are statically or dynamically typed. This puts C/C++ and Java in one category and Python, JavaScript, and PERL in the other. All three dynamically typed languages have a lower average comment density than the two statically typed languages.

Another way of categorizing the programming languages is by lineage. C/C++, Java, and JavaScript fall into the dominant C-paradigm of programming languages, while PERL and Python do not. This choice might be justified by the closeness of the average comment density between C/C++ (17.60%) and JavaScript (16.42%).

Example factors that might influence the amount of comment lines in source code:

- Expressiveness of language (and hence the need (or lack thereof) for more documentation);
- The use of IDEs and auto-generate comment features of such IDEs.

Without a detailed analysis of each programming language and the dominant practices around it we cannot predict what percentage of comment lines are real content lines and which are just empty stubs. Here, we drop further investigation into the reasons of such variation and postpone it to future work.

3.3 The commenting practice in open source

Figure 3 shows the comment density as a function of the amount of source lines of code in a given commit. This Figure is rich in information.

First, for those commits with zero SLoC, the comment density is naturally 100%. Not shown in the graph, the average number of comment lines for zero-SLoC commits is 47.55 comment lines with a standard deviation of 570.1. Moreover, of the 6,622,901 commits in our database after the filters, the zero-SLoC commits count is 164,054, representing 2.477% of all commits. In other words, about 2.5% of the code contributions in our sample population of open source projects, or about every 20th commit, exclusively serve documentation purposes.

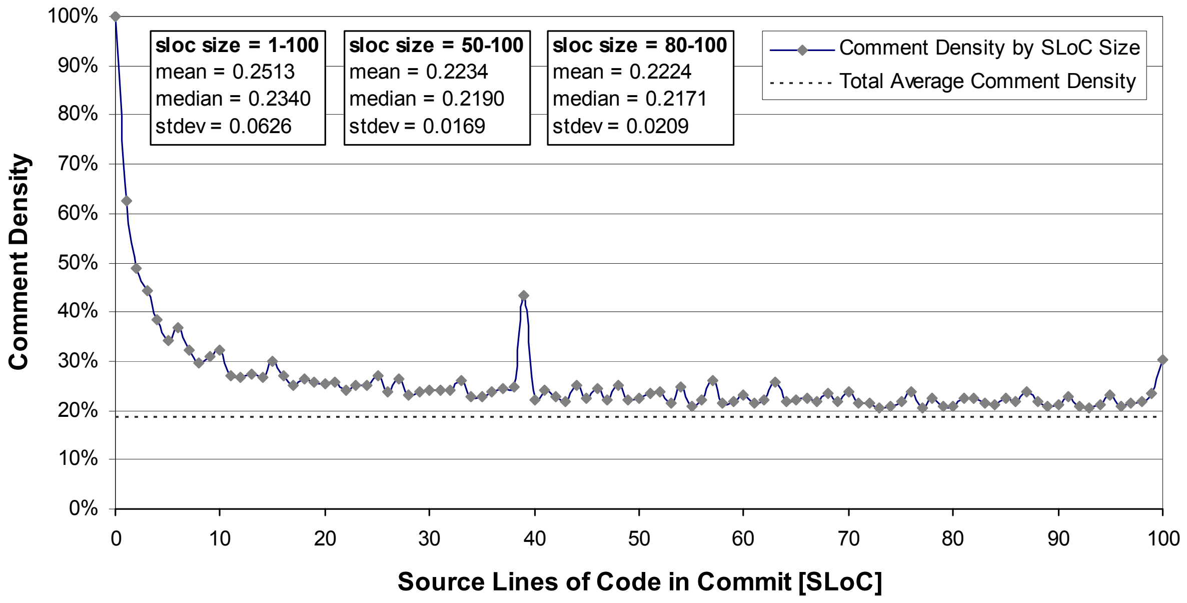


Figure 3: Comment density as a function of source code lines in a given commit.

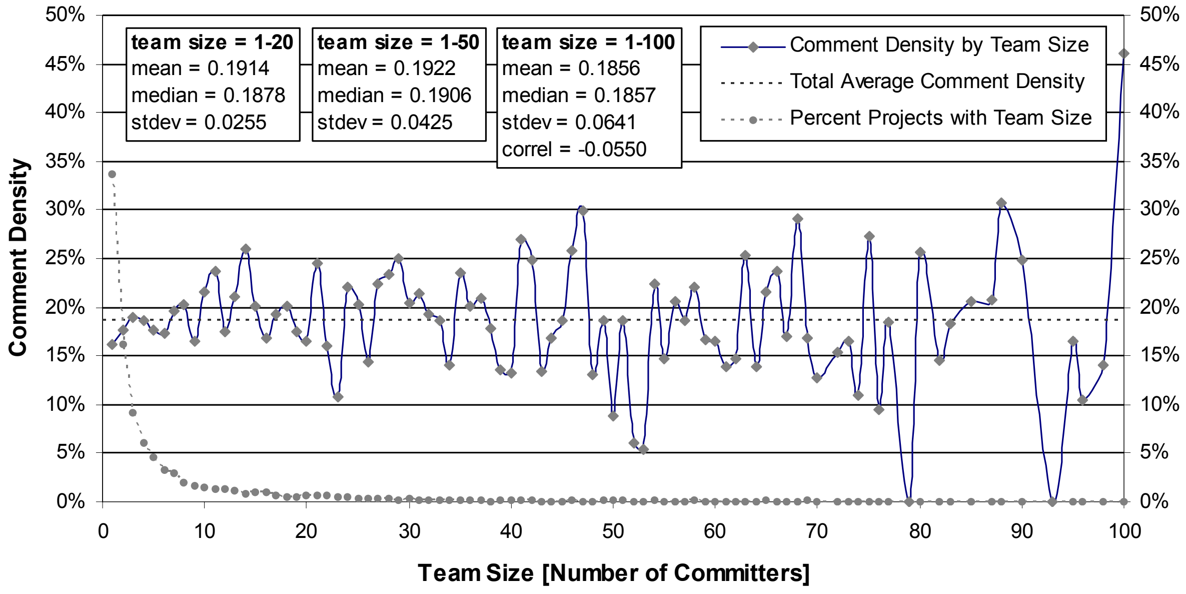


Figure 4: Comment density as a function of team size of open source projects.

Next, for those commits with one SLoC, the comment density is 62.50%. In other words, for every one-liner (of source code) contributed, on average 1.667 comment lines are contributed. Or more poignantly, when developers are making minimal source code changes, they thoroughly document them with comment lines. For commits with two SLoC, the comment density falls to 48.90%, meaning that for almost every source code line there is a comment line.

Finally, we can see that for increasing source code lines in a commit, the comment density keeps falling, approaching asymptotically the total average comment density of 18.67% graphed by the dashed line. The high value at 39 SLoC is caused by a single commit with 364,438 comment lines; the sample size for the 39 SLoC commits bin is 16,534.

Thus, we conclude that successful open source projects like those in our sample population follow a practice of ongoing and integrated documentation of their code base. This activity is bipolar in that developers both perform documentation as a separate housekeeping activity as well as integrated with regular source code lines (SLoC) contributions.

4. Functional dependencies

This Section discusses the influence of several variables of open source software projects on their comment density relevant to scaling up projects. Specifically, this Section looks at the relationship between comment density and project size, team size, and age of project.

4.1 Project size and comment density

Figure 1 already displays the comment density as a function of project size. The comment density remains constant at 18.67% for most project sizes but those of the largest projects. Also, the correlation between project size and comment density is -0.0079, suggesting they are independent of each other.

For large projects, the comment density appears to be decreasing. However, the data for large projects (> 10 million SLoC) is getting sparse. We only have 18 such projects in our dataset, out of 5,229, representing 0.3% of the total population. Thus, variation in comment density for these few select projects may unduly distort the model. Also, some of the large projects have unusual properties. For example, the Debian distribution of Linux is mostly generated code, repeating the same patterns over and over.

Thus, for all practical purposes, we conclude that the comment density is independent of project size and that its average remains a constant over a wide range of project sizes.

4.2 Team size and comment density

Figure 4 shows the comment density as a function of team size. We define team size as the number of committers to a given project. The committers are those people who have write access to the code repository and have made a contribution at least once.

The average comment density for team sizes 1-20 is 19.14%, for team sizes 1-50 is 19.22% and for team sizes 1-100 is 18.56% with standard deviations between 2.6% and 6.4%. Of all projects, projects with team sizes 1-10 represent 80.99% and projects with team sizes 1-20 represent 89.96%. Projects with team sizes 101 and higher represent 1.32%. Thus, the bulk of projects are in the 1-20 people team range and the comment density of these projects dominates the average comment density.

However, despite the dominance of the smaller teams, we find little variation around the average comment density of 18.67%. The mean for projects of team sizes 20-100 is 18.40%, the median is 18.55% and the standard deviation is 7.16%. While the variance for the comment density of projects run by larger teams is going up due to the increasing sparseness of data points, the average comment density is roughly staying the same. In addition, the correlation between team size and comment density is -0.0550, suggesting independence of the two variables.

We conclude that in open source software the comment density is by and large independent of team size. This suggests that successful open source projects are capable of maintaining a commenting discipline as their teams grow larger.

5. Discussion of findings

We summarize our finding and discuss potential threats to their validity. We then discuss future work.

5.1 Conclusions

We have found and demonstrated that commenting source code is a consistently followed practice of successful open source projects. It has led to an average comment density of about 19%. This density is maintained by dedicated commenting activities (about 2.5% of all code contributions) as well as a regular part of on-going software development. We have also found that the average comment density varies by programming language but remains constant on several other dimensions.

In particular, we have found that the average comment density is independent of team size and project size, suggesting that as teams and projects get larger, successful open source projects maintain their commenting discipline.

5.2 Threats to validity

Our sample population represents about 30% of the total population of active open source projects. Our database was initially seeded with popular projects by Ohloh, Inc. After this, it was opened up for community editing. There is no apparent bias in the selection of projects; however, Ohloh's crawler can only cope with the configuration management systems CVS, Subversion, and Git. These are the most popular configuration management systems, so we feel that this does not unduly bias the overall sample.

We only count physical lines and do not analyze their contents. Thus, in terms of actual comment contents our numbers may be misleading, in particular if a large number of comments in open source software was auto-generated or if the comments refer to the license that is used within the project for example. We do not feel that this is a major issue right now, however, with features like comment stub generation in modern Java IDEs this issue may become more important in the future.

In all analyses that rely on counting the exact number of lines affected in a commit, there is a risk of miscounting these lines because it is not possible, a posteriori, to determine whether a line was changed or whether it was removed and then added. Given our statistic over these changes and the large sample size of commits in our population, we believe that this problem is not a serious issue [12].

5.3 Future work

Future work might include a more thorough analysis of the semantic content of comment lines to see whether the differences in comment densities between programming languages reflect real content or were created by auto-generation features of IDEs.

We intend to compare the comment density of open source projects with those of closed source projects found at SAP. We are currently preparing such a comparison. Our hope is that such comparison and the resulting insight can help us better define corporate code metrics that in turn aid in the management of software development projects.

We have yet to correlate comment density with project success. We started out with successful projects ignoring unsuccessful projects. It would be interesting to look at the comment density of failed projects and analyze to what extent commenting behavior of software developers can be a predictor of project success of failure.

6. Related work

We did not find many studies of comment densities in open source or software development at all. We found no study that assesses comment density on the level of scale as presented in this paper.

Prechelt reports about a controlled experiment performed from 1997-1999 [4] [13]. 91 teams implemented the same set of requirements using different programming languages, including C, C++, Java, Perl and Python. The goal was the comparison of scripting languages with non-scripting languages. In contrast to our results in Section 3, Prechelt found that the scripting language solutions were significantly better documented (had a higher comment density) than the non-scripting language solutions. Values for the comment densities were in the 20-30% range. Our main explanation for the differences is that the study is just too different from ours. Prechelt's subjects were students, and the programs were throw-away exercises. The study is over 10 years old and has a much smaller sample size. May be most importantly, the implementers of the C, C++, and Java versions were paid, while the implementers of the Perl and Python solutions volunteered.

In his 2001 M.S. Thesis, Sundbakken assess the comment density of maintenance phase code contributions to components of four open source projects [5]. Sundbakken observes in his data that consistent commenting correlates highly with maintainability of the components. The measured comment density, however, is much lower than what we have found: It ranges from 0.09% for poorly maintainable components to 1.22% for highly maintainable components. We mostly attribute this discrepancy to the small sample size of his study.

In a study on the comment density of a closed-source compiler project in its maintenance phase, Siy and Votta find a consistent comment density around 50% [6].

In another study of 100 Java open source classes, Elish and Offutt find an average comment density of 15.2% with a standard deviation of 12.2% [8]. Again, while closer to our numbers, the small size makes it hard to compare this study with our work.

Spinellis assesses the comment density for four operating system kernels, namely FreeBSD, Open Solaris, Linux, and the Windows Research Kernel [2]. His data is not comparable with our data nor the data of any of the other studies, as he uses a semantic (statement) based definition of comment density and not a line-based one. The comment density of the four kernels varies widely.

Fluri et al. assess three open source projects (Azureus, ArgoUML and JDT Core) and describe how code and comments co evolve [17]. Specifically, they observe whether the comment density remains stable over time and whether developers maintain a strong commenting discipline over a project's lifetime. They also find that open source developers consistently comment their code base as 97% of all common changes between source code and comments are in the same revision. Regarding the comment ratio over a project's lifetime they find that it does not stay at a consistent value. In one case they observe a significant upwards trend while they find a significant downwards trends in the two remaining projects. However, the small sample size of three projects makes it hard to compare this study with our work.

We did not find any work that discusses how the comment density of open source projects correlates to other relevant variables of the involved projects.

7. Conclusions

This paper shows that successful open source projects are consistently well documented with an average comment density of 18.67%. We have found that this comment density varies by programming language but remains invariant with respect to team size and project size (as measured in source code lines). Maybe most importantly, we have found that commenting source code is an integrated activity in the development of open source software and not a separate activity or an afterthought. These results shed further light on how open source software is being developed. In future work we will relate it to closed source software development to improve corporate software development processes.

8. References

[1] Amit Deshpande, Dirk Riehle. "The Total Growth of Open Source." In Proceedings of Fourth Conference on Open Source Systems. Springer Verlag, 2008. Page 197-209.

[2] Diomidis Spinellis. "A Tale of Four Kernels." In *Proceedings of the 2008 International Conference on Software Engineering* (ICSE '08). IEEE Press, 2008. Page 381-390.

[3] Lutz Prechelt. "Are Scripting Languages any Good? A Validation of Perl, Python, Rexx, and Tcl against C, C++, and Java." *Advances in Computers* 57 (2003). Page 207-271.

[4] N. E. Fenton. *Software Metrics: A Rigorous and Practical Approach.* Thomson Computer Press, 1996.

[5] Marius Sundbakken. *Assessing the Maintainability of C++ Source Code.* M.S. Thesis, Washington State University, 2001.

[6] Harvey Siy, Lawrence Votta. "Does the Modern Code Inspection have Value?" In *Proceedings of the 17th IEEE International Conference on Software Maintenance* (ICSM '01). IEEE Press, 2001. Page 281-290.

[7] Carlo Daffara. "How Many Stable and Active Libre Software Projects?" See http://flossmetrics.org/news/11.

[8] Mahmoud Elish, Jeff Offutt. "The Adherence of Open Source Java Programmers to Standard Coding Practices." In *Proceedings of the 6th IASTED International Conference Software Engineering and Applications* (SEA '02). Page 193-198.

[9] Ohloh, Inc. See http://www.ohloh.net.

[10] Ohloh, Inc. Ohloh API. See http://www.ohloh.net/api.

[11] Ohloh, Inc. ohcount. See http://labs.ohloh.net/ohcount.

[12] Philipp Hofmann, Dirk Riehle. "A Statistic for Calculating Commit Size Probabilities in Open Source Projects." *Technical Report*, forthcoming.

[13] Lutz Prechelt. "An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program." *Technical Report 2000-5*, Universität Karlsruhe, Fakultät für Informatik, Germany, March 2000.

[14] Amit Deshpande, Dirk Riehle. "Continuous Integration in Open Source Software Development." In *Proceedings of the Fourth Conference on Open Source Systems* (OSS 2008). Springer Verlag, 2008. Page 273-280.

[15] Barry W. Boehm. "A spiral model of software development and enhancement." Computer vol. 21, no. 5 (May 1988). Page 61-72.

[16] Kent Beck. *Extreme Programming Explained: Embrace Change.* Addison Wesley, 1998.

[17] B. Fluri, M. Wümrsch, and H.C. Gall, "Do Code and Comments Co-evolve? On the Relation between Source Code and Comment Changes," Proc. 14th Working Conf. Reverse Eng., IEEE CS Press, 2007, pp. 70–79.

[18] Oliver Arafat, Dirk Riehle. In Companion to Proceedings of the 31st International Conference on Software Engineering (ICSE 2009). IEEE Press, 2009. Page 195-198.