Beyond Low-Hanging Fruit: Seeking the Next Generation in FLOSS Data Mining

Megan S. Conklin
Elon University, Department of Computing Sciences, Elon, NC 27244
mconklin@elon.edu,
WWW home page: http://www.cs.elon.edu

Abstract. This paper will discuss the motivations and methods for collecting quantitative data about free, libre and open source (FLOSS) software projects. The paper also describes the current state of the art in collecting this data, and some of the problems with this process. Finally, the paper outlines the challenges data miners should look forward to when trying to improve the usefulness of their quantitative data streams.

1 Introduction

It is surprisingly difficult to obtain and compare timely, quantitative data in order to answer even simple questions about the free/libre/open source software (FLOSS) world: How many open source projects are there? How many developers? How many users? How much does each developer contribute? Which projects are dead, which are flourishing? Which projects are popular? How are development teams structured, and which team structures are the most successful?

FLOSS project teams are self-organized, widely-distributed geographically, and use many different programming languages and software development methodologies. Teams are organized in an ad hoc, decentralized fashion. Projects can be very hard to track, and changes can be difficult to follow. Because developers primarily use the Internet for communication, and because they are organized around the idea that anyone can join a team, it is usually easy to get data about FLOSS project teams, but difficult to rely upon or standardize this data.

This is in direct contrast to the way proprietary projects are most often structured. Empirical software engineering researchers have, in the past, typically used metrics from a single company or a single proprietary project. This data was collected systematically and distributed in a tightly controlled manner, consistent with the proprietary nature of the software being developed.

Whereas data analysis about proprietary software practices was primarily a problem of scarcity (getting access and permissions to use the data), collecting and analyzing FLOSS data becomes a problem of abundance and reliability (storage, sharing, aggregation, and filtering of the data). To this end, this paper discusses the motivations and methods for collecting FLOSS data, contrasting these with traditional software engineering methods. We then outline some challenges data miners should look forward to when trying to improve the usefulness of their quantitative data streams.

Please use the following format when citing this chapter:

Conklin, M.S., 2006, in IFIP International Federation for Information Processing, Volume 203, Open Source Systems, eds. Damiani, E., Fitzgerald, B., Scacchi, W., Scotto, M., Succi, G., (Boston: Springer), pp. 47-56

2 Motivations

2.1 Importance of Metrics in Software Engineering

The collection and aggregation of real-world and historical data points are critical to the task of measurement in software engineering. Interesting measures of the software process can vary depending on the goals of the research [1], but they could include things like the number of errors in a particular module, the number of developers working in a particular language or development environment, or the length of time spent fixing a particular code defect [2]. Software engineering metrics can be used to avoid costly disasters [3], efficiently allocate human and financial capital [4], and to understand and improve business processes.

There are hundreds of these examples in the software engineering literature about how important metrics are for studying proprietary projects, but where are the metrics and measurements for studying FLOSS development practices? We know that FLOSS projects are fundamentally different from proprietary projects in several important ways: they are primarily user-driven as opposed to driven by a hierarchically-organized, for-profit corporation [5]. These user-programmers work in loosely defined teams, rarely meet face-to-face, and coordinate their efforts via electronic media such as mailing lists and message boards [1]. These are all fundamentally different arrangements than the way proprietary software is traditionally developed.

2.2 Importance of Metrics in FLOSS

Recognizing this unique separation between proprietary and FLOSS software engineering traditions, and building on a strong foundation of measurement in software engineering literature, there are then several compelling reasons to collect, aggregate, and share data about the practice of FLOSS software development. First, studying FLOSS development practices can be useful in its own right, in order to educate the larger research and practitioner communities about an important new direction in the creation and maintenance of software [6]. FLOSS researchers have noticed that many of the practices of FLOSS teams are not well-understood [7, 8] or, when they are, they seem to directly oppose traditional wisdom about how to build software [9]. At the very least, this situation indicates something interesting is afoot, and in the best case will foreshadow an important methodological shift for software development.

Additionally, the lessons learned through studying FLOSS development teams are applicable to many other fields. Much research has been conducted on the economic [10, 11] and policy aspects of FLOSS development, especially as the reason for various licensing choices [12] or about their implications for intellectual property [13–16]. Additional research has been conducted on the motivations of FLOSS developers [11, 17, 18], which is an interesting question to consider since these developers are working without pay. There are also implications for other types of distributed teams and computer-mediated group work [19, 20], such as gaining a

better understanding of the role of face-to-face meetings in highly distributed work teams, or analyzing the leadership hierarchies that work best for distributed teams.

3 Difficulties

FLOSS data appears to be highly available, and appears easier to access for research than proprietary data. While this means that it is possibly more appealing to use than proprietary data, FLOSS data has its own very long list of collection difficulties.

3.1 Questions of Accessibility

Researchers who wish to test a quick hypothesis about the use of a particular software module, or who wish to study adoption rates of various programming languages know that, in theory, they should have access to this information via FLOSS project data, since the code is free and open to everyone. Therefore, it is no longer necessary to find a corporation willing to provide researchers access to their development databases and source code control systems. Much of the FLOSS project data is stored inside large, public source code repositories such as [21–24]. However, the difficulties in gathering FLOSS data from these repositories in an automated fashion are numerous and on-going [25, 26]. Gaining control over this "free" and "open" data is actually a hugely inefficient process for a researcher. If each isolated research team is taking on this tedious responsibility of gathering the same data, this will quickly result in redundancy in the collection effort, which prolongs and denigrates the data analysis effort.

3.2 Questions of Accuracy and Reproducibility

Another significant problem with isolated researchers attempting to collect and analyze FLOSS data is one of validation and reproducibility of results [27]. There are numerous examples in the FLOSS literature that reflect on this general problem with collecting, validating, and reproducing data and results. Some studies have addressed their difficulties with collecting data by limiting their studies to a single public repository, and then to draw on samples that are easy to collect, but which were created for entirely different purposes [28, 29]. In addition, the demands of traditional publication may mean that the data collection methodologies are not fully described. This makes them impossible to reproduce, which slows down the compounding effects [30] started by good research [31]. The tradition of scientists working together to solve a hard problem [32] is an important tradition to continue, but how is this to happen if each isolated research team must start from square one?

3.3 Questions of Quantity

In software engineering data analysis, this massive project cross-referencing and metadata creation is a problem probably unique to FLOSS. Rarely would empirical

software engineers studying proprietary systems need to study hundreds of disparate project teams stored in dozens of unique data models (repositories) with thousands of data attributes. The amount of raw data available for collection in FLOSS software is greater than that of proprietary software by orders of magnitude, both in terms of project team counts and in developer counts. For each developer and each project there are thousands of additional attributes that can also be mined for interesting insights.

However, much of the FLOSS research to date closely emulates the research methods used to study proprietary software: the research follows a single project and extrapolates some lesson or advancement which can then be applied to other projects. Examples include [33–36, 18]. Some other projects have used surveys or other instruments to collect information about a small number of FLOSS projects. For example, [37] was based on a survey of 684 developers on 287 FLOSS projects. [7] was based on ethnographic research principles, and involved a dozen software projects in four different research areas. [38] studies four open source projects all related to the same coordinating company. [10] studied four different open source projects, some of which also appear in other studies [35, 36]. [39] surveys 81 developers working on an unspecified number of open source projects. The 2000 Orbiten data [40] includes 12706 (identifiable) developers and 3149 projects. Within the corpus of previously published FLOSS literature, the Orbiten project data can be considered large. However, we know that these numbers represent less than 3% of the total activity in FLOSS development [27].

3.4 Questions of Reliability

Another problem with relying on published-but-proprietary data sources for research is that type of data can disappear. For example, the Orbiten project mentioned above is no longer in active development. Though the original article [40] links to a web site intended to provide both the software and the data, this site is no longer operational. A researcher wishing to duplicate or validate the methods of Orbiten would be at a loss to do so. Thus, there is really no way to build upon or extend the metrics published in the original article (i.e. further this valuable FLOSS research). Using FLOSS development methodologies such as project handoff [11,16] would have reduced this tendency for information to exist only in one place.

4 Future Challenges

As an answer to these goals described above and expressed by the FLOSS research community, the FLOSSmole project [41] was designed to be accessible, accurate, reproducible, compatible, comprehensive, and reliable [27, 42]. In its current state, FLOSSmole serves the greater FLOSS research community by providing a collection of software tools (database schemas, code libraries, scripts, source code) that mines code repositories and provides the resulting data and summary analyses as open

source products. The project is hosted on Sourceforge [24], a public, open-source code repository. The code, data, and schemas are all open-sourced and free for other researchers to use and modify. FLOSSmole has been successful in its role as a basic data gathering and reporting tool for research.

However, FLOSSmole and other quantitative FLOSS data gathering projects could be better; in this section we propose improvements to the data-gathering community research infrastructure. Though we have FLOSSmole in mind while writing, these ideas are based on general ideas, and could therefore be applied to many other projects designed to collect and aggregate quantitative FLOSS data.

4.1 Exploit Low-Hanging Fruit

The primary activity of our community data repository is to collect and store data. In FLOSSmole, we currently pull data from two open source code repositories (also called "forges"), and have historical data from a third repository. These forges represent the low-hanging fruit of FLOSS data: even though there is relative difficulty [26] involved in getting data from the forges, they are still the easiest places to get large amounts of data quickly. So, one of the most important things we can provide the community is to pull data from a broader range of forges. There are dozens of independent open source forges that host important projects, but we do not currently collect this significant quantity of data. This also represents a step in the right direction for promoting collaboration and sharing between communities and between development efforts and research groups.

Moreover, as FLOSS researchers in the true spirit of collaboration, we should expect our data to become the low-hanging fruit for other projects. The SWIK project [43], an independent effort by programmers at Sourcelabs, is a wiki-based database of open source projects. Each open source project has an entry in the Swik system, and Swik users can annotate and tag each project page with keywords or descriptors. This entire project was created in one month, using data made public by FLOSSmole. Swik is a great example of why it is important to make data easily accessible. Developers and researchers should be able to find, interpret and use quantitative data quickly and painlessly. However, despite how easy it is to download FLOSSmole data, it is not as easy to query the database or interpret results. FLOSSmole data is available to the research community in two formats: massive text ("raw") database dumps, and summary reports. There is also a nice query tool. But the most important thing the research community has asked us for is for more reporting tools (better visualizations, more graphs/charts, an online, interactive graphing tool), and for fuller descriptions of the data we are making available (more metadata). Both of these items would go a long way to improving the usability of the data in our community data repository.

4.2 Seek High-Hanging Fruit

In the same way that FLOSS development is a collaborative process, FLOSS research is also collaborative at its nature. Thus, any FLOSS data repository will need to

integrate both donated data sets and historical research data. We occasionally have access to data from now-defunct projects, and from previously published FLOSS research studies – both of these sources of data are valuable for historical analyses, and may be able to be integrated into the existing (and active!) community database. Even if this donated or historical data were complete, clean, and well-labeled, integrating it could still be problematic: different repositories store different data elements, different forges can have projects with the same names, different developers can have the same name across multiple forges, the same developer can go by multiple names in multiple forges. In addition, forges have different terminology for things like developer roles, project topics, and even programming languages.

What is the best way to extract knowledge from published research? What is the best way to express the quantitative knowledge in a domain and integrate multiple sources of this knowledge? How will we create sufficient metadata about each data source so that the results can be used together? Can any of this be done in an automated fashion? What query tools should be used so that the user can fully explore both data sets? These are big questions with no easy answers; these are the rare and exceptional fruits, located higher up in the tree.

Assuming we are able to successfully meld multiple data sources and create this richer, more interesting multi-repository structure, we must also consider privacy issues. There is some vigorous debate in the research community about breaching developer privacy in a large system of aggregated data like ours [44]. For example, if we aggregate several code repositories and are now able to show in a colorful graph that Suzy Developer is ten times more productive than Bob Coder, does this violate Bob's privacy? If we can show that Suzy's code changes are five times more likely to cause errors than Bob's, does that violate Suzy's privacy? The next generation of community repositories like FLOSSmole should have the ability to hash the unique keys indicating a developer's identity. This effort will have to be researched, implemented, and documented for our community.

5 Conclusions

This paper first reviews why quantitative data is useful in software engineering, including some ways in which the FLOSS and proprietary software data gathering processes are different. Next we point out some common problems with the FLOSS data gathering process. Finally, we pass on the benefit of our experience creating FLOSSmole by posing questions about what the next steps should be for creating a truly valuable and transformative community data repository.

Reflecting on our initial successes creating data repositories for quantitative FLOSS data, it is clear that simply gathering public repository data (the "low-hanging fruits" of FLOSS data collection) is interesting and useful, but not sufficient. This type of data does not capitalize on some of the most interesting aspects of FLOSS movement: its focus on collaboration, its respect for individual privacy issues. In

order to provide truly meaningful and useful data, we must reach beyond these low-hanging fruits.

6 References

1. R.E. Park, W.B. Goethert, and W.A. Florac. (1996) Goal Driven Software Measurement – A Guidebook. CMU/SEI-96-BH-002. Carnegie Mellon U. 1996.

2. E. Yourdon, E. *Decline and Fall of the American Programmer* (Prentice-Hall: Englewood Cliffs, New Jersey, 1993).

3. J.-M. Jezequel and B. Meyer, Design by contract: The lessons of Ariane, *Computer* **30** (1),129-130 (1997).

4. F. Brooks, *The Mythical Man-Month*, rev ed. (Addison-Wesley: Reading, Massachusetts, USA, 1995).

5. E. von Hippel, Innovation by user communities: Learning from open-source software. *Sloan Management Review* (Summer). 82-86 (2001).

6. J. Feller, Thoughts on Studying Open Source Software Communities. In *Realigning Research and Practice in Information Systems Development: The Social and Organizational Perspective*, edited by N.L. Russo, et al. (Kluwer Academic Publishers, Dordrecht, 2001).

7. W. Scacchi, Understanding the requirements for developing Open Source Software systems, *IEE Proc. on Software*, **149** (1). 24-39 (2002).

8. E. von Hippel, Exploring the Open Source Software Phenomenon: Issues for Organization Science. *Organization Science* **14** (2), 209-223 (2003).

9. J.D. Herbsleb and R.E. Grinter, Splitting the organization and integrating the code: Conway's law revisited. In *Proc. of the Intl Conf. on Soft. Eng.* (1999).

10. J. Lerner, and J. Tirole, (2002). Some simple economics of open source, *Journal of Industrial Economics* **L**, 197-234 (2002).

11. E. Raymond, *The Cathedral and the Bazaar*. (O'Reilly, Sebastopol, CA, 1999).

12. L. Rosen, *Open Source Licensing: Software Freedom and Intellectual Property Law* (Prentice Hall, Upper Saddle River, New Jersey, 2004).

13. C. DiBona, S. Ockman, and M. Stone, *Open Sources: Voices from the Open Source Revolution* (O'Reilly, Sebastopol, CA, 1999).

14. B. Kogut, and A. Meitu, Open-source software development and distributed innovation, *Oxford Review of Economic Policy* **17**, 2. 248-264 (2001).

15. J. Lerner, and J. Tirole, The open source movement: Key research questions, *European Economic Review* **45**, 819-826 (2001).

16. S. Weber, *The Success of Open Source* (Harvard U. Press, Cambridge, 2004).

17. L. Torvalds, FM interview with Linus Torvalds: What motivates free software developers? *First Monday* **3**(3) (March, 1998).

18. Y. Ye and K. Kishida, Toward an understanding of the motivation of open source software developers. In *Proc. of the 25th Intl. Conf. on Soft. Eng.* (2003).

19. K. Crowston, H. Annabi, J. Howison, and C. Masango, Effective work practices for Soft. Eng.: Free/libre/open source software development, *WISER Workshop on Interdisciplinary Soft. Eng. Research* (2004).

20. K. Crowston, H. Annabi, J. Howison, and C. Masango, Effective work practices for FLOSS development: A model and propositions, *Proc. of the Hawai'I Intl. Conf. on System Science* (2005).

21. Bugzilla. Apache Foundation, (March 1, 2006); http://issues.apache.org/bugzilla/

22. Freshmeat (March 1, 2006); http://www.freshmeat.net

23. Savannah (March 1, 2006); http://savannah.nongnu.org/

24. Sourceforge (March 1, 2006); http://www.sf.net

25. D.M. German, Mining CVS repositories: The Softchange experience. In *Proc. of the Workshop on Mining Software Repositories* (2004).

26. J. Howison and K. Crowston, K. (2004). The perils and pitfalls of mining Sourceforge. In *Proc. of the Workshop on Mining Software Repositories* (2004).

27. M. Conklin, J. Howison, and K. Crowston, K. Collaboration Using OSSmole: A Repository of FLOSS Data and Analyses, *Proc. of the Workshop on Mining Software Repositories* (2005).

28. S. Krishnamurthy, Cave or community? An empirical examination of 100 mature open source projects, *First Monday* **7**(6), (June, 2004).

29. I. Samoladas, and I. Stamelos, Assessing free/open source software quality. TR Aristotle University of Thessaloniki, Thessaloniki, Greece. 2003 (unpublished).

30. K.S. Louis, L.M. Jones and E.G. Campbell, Sharing in science. *American Scientist* **90** (4), 304-307 (2002).

31. D.R. Krathwohl, *Methods of Education and Social Science Research: An Integrated Approach* (Longman: New York, 1998).

32. R.K. Merton, *Social Theory and Social Structure*, (Free Press: New York, 1968).

33. M.S. Elliott and W. Scacchi, Communicating and Mitigating Conflict in Open Source Software Development Projects. *Projects and Profits* **10**(4), 25-41 (2004).

34. D.M. German, Decentralized open source global software development, the GNOME experience. *J. of Soft. Process: Imp. and Practice*, **8** (4), 201-215 (2004).

35. S. Koch and G. Schneider, Results from software engineering research into open source development projects using public data, *Diskussionspapiere zum Tät igkeitsfeld Informationsverarbeitung und Informationswirtschaft*, 22, (2000).

36. A. Mockus, R.T. Fielding, and J. Herbsleb, A case study of open source software development: The Apache server. In *Proc. of the 22nd Intl. Conf. on Soft. Eng.*, 263-272 (2000)

37. K. Lakhani and R.G. Wolf, Why hackers do what they do: Understanding motivation effort in free/open source software projects. WP 4425-03. Sloan School of Management, MIT, 2003 (unpublished).

38. K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye, Evolution patterns of open-source software systems and communities. In *Proc. of the Intl. Workshop on Software Evolution* (2002).

39. A. Hars and S. Ou, Working for free? - Motivations of participating in open source projects. In *Proc. of the 34th Hawaii Intl. Conf. on System Sciences*, (2001).

40. R.A. Ghosh and P.P. Prakash, The Orbiten free software survey. *First Monday* **5**(7), (July, 2000).

41. FLOSSmole Project (March 1, 2006); http://ossmole.sf.net

42. J. Howison, M. Conklin, and K. Crowston, OSSmole: A Collaborative Repository for FLOSS Research Data and Analyses. In *Proc. of the First Intl. Conf. on Open Source Systems* (2005).

43. Swik (March 1, 2006); http://swik.sourcelabs.com

44. G. Robles, Developer identification methods for integrated data from various sources. In *Proc. of the Intl. Workshop on Mining Software Repositories* (2005).