

Research Friendly Software Repositories

Israel Herraiz
DSIC
Complutense University of
Madrid
herraiz@fdi.ucm.es

Gregorio Robles
GSyC/Libresoft
Universidad Rey Juan Carlos
grex@gsysc.urjc.es

Jesus M.
Gonzalez-Barahona
GSyC/Libresoft
Universidad Rey Juan Carlos
jgb@gsysc.urjc.es

ABSTRACT

What is the future of software evolution? In 1974, Meir M. Lehman had a vision of software evolution being driven by empirical studies of software repositories, and of a theory based on those empirical results. However, that scenario is yet to come. Software evolution studies are often based on a few cases, because the needed information is scarce, dispersed and incomplete. Their conclusions are not generalizable, slowing down the progress of this research discipline. Libre (free / open source) software supposes an opportunity to alleviate this situation. In this paper we describe the existing approaches to provide research datasets that are mining libre software repositories, and propose an agenda based on the concept of research friendly software repositories, which provides finer granularity and integrated data.

Categories and Subject Descriptors

D.2 [Software]: Software Engineering

General Terms

Management

Keywords

research friendly, software repositories, software evolution, FLOSSMetrics, FLOSSMole

1. INTRODUCTION

In 1974 Lehman presented the laws of software evolution in his inaugural lecture at the Imperial College [12]. The laws were based in some metrics of the OS/360 system development process that he obtained while working at IBM, and were formulated based on the statistical properties of those metrics. In that lecture he mentioned some of the statistical approaches that should be used to study software evolution, like time series analysis, that has been shown successful in the field of software evolution [7].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

IWPSE-Evol'09, August 24–25, 2009, Amsterdam, The Netherlands.
Copyright 2009 ACM 978-1-60558-678-6/09/08 ...\$10.00.

That study and others at the time, that gave birth to software evolution as a research field, could be nowadays classified in the area of Mining Software Repositories (MSR), which studies software repositories to extract knowledge that can be applied to improve the software development process.

The intersection of MSR and software evolution when studying libre (free / open source) software has raised some controversial results, like the study by Godfrey and Tu [4], that was labeled by Lehman as an anomaly [13]. The study by Godfrey and Tu showed that the Linux kernel's evolution could not be explained using the laws of software evolution.

Certainly, the appearance of anomalies may be due to the small scale of those studies, and of the majority of the software evolution studies, because of the scarcity of research data and the difficulties to gain access to software projects information. With the growth of the libre software phenomenon, a large amount of open repositories is now available for research purposes, enabling large scale statistical studies of software evolution like those originally proposed by Lehman in 1974. Examples of these have started to appear [11, 6].

In spite of these rich availability of software repositories, the heterogeneity of the data makes it difficult to apply studies at a large scale, although some research projects, like FLOSSMole [10] or FLOSSMetrics [8] are addressing these issues and aim to provide datasets about thousands of libre software projects for research purposes. Other commercial initiatives with similar goals have also appeared, like the case of Ohloh¹.

Heterogeneity is just an example of the drawbacks of software repositories when applied to research. Because these repositories were not designed with research in mind, they are not research friendly, and this slows down the pace of software evolution as a research field. However, there is an increasing trend in the research-friendliness of software repositories in the last years. In this paper, we highlight the main problems that researchers must face when dealing with current software repositories, review this evolutionary trend, and show how by increasing the research-friendliness of software repositories, some advantages for the software development process are also achieved.

The rest of the paper is as follows. Section 2 shows how software repositories have evolved in the scope of libre software development, and how that evolution converges with the needs of software evolution research. Section 3 shows the different already existing approaches that are mining public software repositories to provide datasets that can be used

¹See <http://www.ohloh.net/>

for software evolution research. Next, although the latest software repositories are implementing features that make them more useful for research, Section 4 discusses some of the features that are still missing, and that future repositories should have to make them more research friendly. Finally, Section 5 concludes this paper, with some considerations about how software evolution research should be in the future.

2. EVOLUTION OF SOFTWARE REPOSITORIES

Libre software projects are usually developed in communities that communicate through the Internet. The project's members often have not met in person, come from different cultures and speak different native languages, and yet they manage to coordinate, collaborate and deliver quality products. Because of this distributed nature, libre software projects crucially rely on software repositories to achieve their goals, and are a good case study to analyze how software repositories have evolved to fit the increasing needs of these communities.

About ten years ago, the most popular source code management (SCM) system was CVS, the Concurrent Versions System, that is still used by some well-known projects (for instance, Eclipse). This system allowed developers to work in a distributed and coordinated fashion over the same sources tree, and kept the history of all the changes made to the files managed by the system. These changes are known as commits in the CVS' terminology. These commits were not atomic, that is, if a change affected several files, it appeared as several changes in the system. This apparently trivial feature was a subject of research in the field of software evolution, with several proposals to reconstruct the actual changes from the commit records of CVS [3, 15]. This might be a positive issue from a purely researchers' productivity perspective, but in practice it is a loss of information about the evolution of the software project, and a waste of research resources on problems caused by tools that should be solving problems instead of causing them.

In 2000, another version control system became popular in the libre software community: Subversion. It was born with the idea of fixing many of the features missing in CVS, like the atomic operations mentioned above, or the lack of support of moving files under version control. Its popularization in the libre software world caused its increased use for research purposes. Like CVS, Subversion follows a traditional client-server scheme, and so the repository is centralized in a host, with whom developers communicate through a client. In the case of libre software projects, these centralized repositories are often mined for research purposes; repositories contain interesting information for researchers, like the `log` command with a list of the change records, of the `blame` that shows who was the last developer that modified a line and allows for line-level analysis of the evolution [1], although these operations are very demanding in terms of bandwidth and CPU in the server side, and as a consequence sometimes libre software projects ban these operations. However, it lacks some other features, like branching, with some workarounds adopted by practitioners, being the most usual copying new *branches* to a directory called *branches*.

Some of these Subversion's missing features have been highlighted by key developers in the libre software community, like Linus Torvalds², that advocated for some innovations in the field of SCMs, and that developed a new system called Git, that is gaining popularity in the libre software community. The main difference between Git and Subversion or CVS is that this system is distributed, there is no a central repository. Even though Git is not the first distributed SCM, it is another example of how the libre software community has dedicated efforts to the improvement of software repositories. Git includes some other advantages over Subversion, like the difference between committer and author. In libre software projects, the committer status is only held by a limited amount of people that usually get their statuses after gaining some merits. However, many of the contributions are done by other members of the project without that status, in the form of patches submitted for the consideration of committers. These contributors can not commit themselves the changes to the repository, so they need a committer to do so. In Subversion and CVS, this would appear in the history as a contribution of the committer, and not of the original author, unless the committer indicates her name somehow in the log text. But this solution is heterogeneous, because it depends on the practices that each project adopts, making it difficult to recover this information for a broad range of software projects.

Unlike Subversion or CVS, in distributed SCM the repository is not in a single machine, but in every machine of every developer in the project. This is, when you get a copy of the repository, you do not get just a snapshot, but the whole repository itself. This characteristic implies that much of the development process may occur behind the scenes, because public commit is not required to leave a trace of the work done in the repository, and so developers' may have private branches whose history can be tweaked before merging with the public branch of the project (if it ever gets merged). Therefore some history may be lost before it becomes public.

In the other hand, this distributed nature is a step towards the research-friendliness of software repositories, because replicating repositories is just a matter of obtaining a copy of the public repository from the project, which is a cheap operation, and therefore researchers can easily mine the repositories without bothering and consuming the software projects resources.

In summary, these three examples (CVS, Subversion and Git) show an evolutionary converging trend in the features of software repositories with their *research friendliness*, that have been fostered by the needs of developers, like the need of atomic commits, or improved merging and branching. Interestingly, this trend shortens the distance between research and software practice, and makes software repositories friendlier towards research activities. If it was traditionally assumed that there is a tradeoff between developers' comfort and needs and researchers needs to gather data and measure software development activities, it now seems that these two extremes are converging as software repositories evolve.

²In a talk at Google on May 3rd 2007

3. RESEARCH DATASETS FOR SOFTWARE EVOLUTION

In spite of the increasing friendliness of open software repositories, studying software evolution in the large scale is still a problematic task, because of the heterogeneity of the data sources, making it necessary to dedicate too many efforts to data gathering and filtering. Fortunately, there exist research projects committed to this endeavor, which are providing datasets that can be easily used, and that allows for third party verification and validation of studies based on them.

FLOSSmole [10], formerly OSSMole, is a research project that gathers information about thousands of libre software projects, extracting data from different websites about libre software development (SourceForge.net, Freshmeat, Rubyforge, Objectweb, Free Software Foundation, Debian, and Source Kibitzer), by means of HTML output parsing. Most of these sites are known as *forges*, which are centralized hosts for libre software projects hosting, offering different tools, like SCM, wikis, etc. FLOSSMole parses the web interface of these sites, to extract information about libre software projects. For instance, in the case of the summary page of a project in SourceForge.net, it includes information about the license of the project, intended audience, number of developers of the project, and so forth. The dataset is provided in several formats, being one of them MySQL dumps that are easily handled using the MySQL database management system.

One of the main problems that FLOSSMole faces is keeping up to date with HTML output of these sites. Sometimes, these sites include improvements, change layouts or reorganize project options, turning the FLOSSMole parsers useless. Although from another point of view, this is an advantage, as FLOSSMole is exclusively based on public information, and thus it does not impose further restrictions on the kind of activities that can be done using their datasets, unlike other datasets containing similar information, like the NotreDame SourceForge dataset [2] that is only available under a license.

However, FLOSSMole lacks evolutionary information, it contains data obtained using the latest information available in the upstream sources, and it can not be used for most of the studies often found in software evolution research, although it can be at least used to characterize libre software projects, that are common case studies in software evolution research.

Another similar dataset exclusively based on public data sources is FLOSSMetrics [8] (FM3 in the following), a 6th Framework Programme funded research project, that unlike FLOSSMole, it is suitable for software evolution research. FM3 contains data about thousands of libre software projects, that is gathered directly from the software repositories of those projects. Thus it does not rely on any kind of hub, as FLOSSMole does. It mainly contains information about three different kind of repositories:

- Source code management systems
- Bug tracking systems
- Mailing list archives

These sources are mined from their original locations, parsed and transformed into MySQL dumps. It is intended

to provide a web site with summary information about the projects stored in the dataset, and to provided an API to gather information using a web service, however those features are still under development at the time of writing this.

Being based on the original repositories allows to include the whole history of the projects in the datasets, and some additional information like source code metrics. Because these repositories are quite stable, the parsers do not have to be frequently changed. In the other hand, keeping track of all the repositories belonging to a project may be a cumbersome task in some cases.

In any case, these two datasets are ideal for research purposes, because they hide all the nasty details of doing empirical research at a large scale. Researchers do not have to worry about a hub like SourceForge changing its layout and breaking the parsers, or about a project being dispersed across different sites or repositories. These projects take care of that kind of tasks, and provide integrated and filtered datasets. Any study based on these datasets can be verified and validated by third parties, because the datasets are public, or extended to more cases as soon as they get included in the datasets. Another possibility is keeping research studies up to date as software projects evolve, because these datasets are updated in a periodic fashion, and so the results can be automatically obtained every time there is a new version of the dataset available. In summary, using these datasets ensures crucial aspects of any experimental discipline, like repeatability, traceability.

In spite of these clear advantages, these approaches are still dependent on public interfaces that are offered by software repositories, and that often are not intended for research purposes but for software development. The future of datasets gathering is research friendly repositories; distributed SCMs are an example of research friendly repository because they allow for an easy replication of the whole repository without consuming the resources of the projects offering their repositories. In the next section we propose some of the desirable features for the software repositories of the future.

4. THE SOFTWARE REPOSITORIES OF THE FUTURE

Up to this point, we have shown the evolution in software repositories, and how the information provided by these repositories can be used for research purposes, and to provide useful datasets for researchers. However, current software repositories are still far from being friendly enough for researchers, and this fact is pacing down the progress of empirical software disciplines, like the software evolution discipline. In this section we discuss two features that we feel are missing in current repositories: granularity and integration.

Granularity

Current software repositories discard much of the information about software development processes. Let's take the example of SCMs. When a developer commits a change, the SCM records the changes in the code and some additional information, but all the information that has led to the change is lost. A developer might have been adding and removing code several times before committing the change, but all the information that is recorded is the final decision that she took. Although that information might not be

useful from a strict software development point of view, it is very interesting for research, because it provides a complete picture of the process that has led to a change including all the alternatives that were considered before taking a final decision. But more interestingly, the way the SCM works shape the way developers work [5]. That information is discarded because recording it increases the possibilities of conflicts between developers editing the same files, and a sophisticated algorithm to deal with merging in such conditions is needed. If work in parallel is difficult because with merging conflicts arise, developers will tend to avoid working in parallel, and will set fictitious locks in files to avoid merging with someone else's code and hence to avoid conflicts. A SCM tool that keeps a complete record of all the software development changes and is capable of dealing with parallel development (like Syde [5]), will also help developers to work in a less constrained way, without having to adopt workarounds for missing functionalities.

Integration

Current software repositories are not integrated. Developers work in different repositories: commit code to the SCM, talk to other developers in the mailing lists, report, fix and discuss bugs in the bug tracking system. Software artifacts are also mentioned and referred in different repositories. However, there is no mechanism to recover all the activity of a developer across repositories, or to recover all the activity with regards to a particular artifact, not to mention to track software artifacts or developers across different projects. Some projects adopt good practices, like tagging commits with particular keywords when they are related to a bug report, or attaching patches to bug reports before adding the code to the SCM, but these practices differ from project to project, which makes it very difficult to do integrated research studies. There are some approaches to overcome some of the difficulties of the current software repositories [14], but the solution to this problem has to be implemented in the software repositories themselves. After all, every software project presents a common structure of repositories: a SCM to keep track of all the history of the code, mailing lists (or any other similar tool) to communicate, an issue tracking system to report and trace bugs, etc., and it should not be difficult to integrate that information.

We need standards to make all these tools to interoperate between them, so developers and artifacts can be referred in a coherent way in all the repositories of the project. This is obviously interesting for software projects, because it makes maintenance easier, but it is even more interesting for research, because thanks to the availability of research datasets, these integrated studies could be done at a large scale without additional efforts.

Finally, software repositories must take in account research activities. With the availability of thousands of public repositories thanks to the popularization of libre software, researchers have found a rich mine of data to test their hypotheses. However, research activities are often very demanding, and libre software projects are eager to dedicate their resources for this purpose. Software development and research converge, and features that make software repositories research friendly will also help developers. The case of SCM is the clearest one. The birth of distributed SCM responded to software development needs, but as a side effect has made it possible to replicate repositories in a straight-

forward way, so research activities do not consume projects' resources. In general, providing ways to replicate repositories, and providing the information in structured manners (like RDF, OWL [9]) will make a difference in what regards to research.

5. CONCLUSIONS

In 1974 Lehman suggested to use a statistical approach to study software evolution [12], with the hope of obtaining an universal theory of software evolution. Decades later, software evolution studies are often based only on a few case studies, and conflicting studies are common in this field.

The availability of thousands of software repositories thanks to the popularization of libre software has made it possible for the first time in history to study software evolution as Lehman originally suggested. However, current software repositories lack many desirable features, constraining the possibilities of research. Some of these problems are that data sources are very heterogeneous and poorly integrated, discouraging large scale studies.

The integration issue should not be hard to solve. From a technical point of view, using semantic web technologies [9] should make it possible for different programs to communicate. The logical side should take in account that there are two entities that are common to all the repositories, in every software project: people and files. It should be possible to trace a developer across the repositories, by means of an unique identifier. The same applies to files. Any activity regarding a file should be logged and properly tagged. For instance, with bug reports, once they are solved, the change may affect many files. With the current technologies, there exist some workarounds to recover which files where affected by a bug. However, that kind of information should be included by design in software repositories. The granularity issue has already being addressed. For instance, the Syde [5] control version system writes a change record every time a developer saves a file. It also includes mechanisms to facilitate parallel development and merging when several developers are working on the same files.

There are already initiatives that overcome, or more precisely, alleviate these difficulties with the currently available technologies, like FLOSSMole [10] or FLOSSMetrics [8], which offer datasets containing information about thousands of libre software projects. However, the research possibilities are still constrained by the quality of the data, like the aforementioned lack of integration between different repositories.

Empirical software evolution studies should be as general as possible, and validate results at a large scale, ensuring repeatability, traceability and third-party independent verification and validation. Every time a study done with only a few cases, or that does not allow for repetition, validation or verification, software evolution is going a step back as a research discipline. That could be understandable in other environment or time, where access to software data was difficult and scarce, but it can not be justified in the present research environment.

Finally, the needs of software developers in what regards to repositories are the needs of researchers. Developers need better repositories, that do not constraint the programming process, and so do researchers need. Software evolution as a discipline should give value to research devoted to provided better and more research friendly repositories.

6. REFERENCES

- [1] G. Canfora, L. Cerulo, and M. D. Penta. Identifying changed source code lines from version repositories. In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 14, Washington, DC, USA, 2007. IEEE Computer Society.
- [2] S. Christley and G. Madey. Collection of activity data for SourceForge projects. Technical Report TR-2005-15, Dept. of Computer Science and Engineering, University of Notre Dame, 2005. http://www.nd.edu/~oss/Papers/activity_data.pdf.
- [3] D. M. German. Mining CVS repositories, the softChange experience. In *Proceedings of the International Workshop on Mining Software Repositories*, Edinburgh, UK, 2004.
- [4] M. W. Godfrey and Q. Tu. Evolution in Open Source software: A case study. In *Proceedings of the International Conference on Software Maintenance*, pages 131–142, San Jose, California, 2000.
- [5] L. Hattori and M. Lanza. Mining the history of synchronous changes to refine code ownership. In *Proceedings of the 6th International Working Conference on Mining Software Repositories*, pages 141–150, 2009.
- [6] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles. Towards a theoretical model for software growth. In *International Workshop on Mining Software Repositories*, pages 21–30, Minneapolis, MN, USA, 2007. IEEE Computer Society.
- [7] I. Herraiz, J. M. Gonzalez-Barahona, G. Robles, and D. M. German. On the prediction of the evolution of libre software projects. In *Proceedings of the International Conference on Software Maintenance*, pages 405–414, Paris, France, 2007. IEEE Computer Society.
- [8] I. Herraiz, D. Izquierdo-Cortazar, F. Rivas-Hernandez, J. M. Gonzalez-Barahona, G. Robles, S. D. nas Dominguez, C. Garcia-Campos, J. F. Gato, and L. Tovar. FLOSSMetrics: Free / libre / open source software metrics. In *Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR)*. IEEE Computer Society, 2009.
- [9] J. Howison. Cross-repository data linking with RDF and OWL. In *Proceedings of the Workshop on Public Data About Software Development*, 2008.
- [10] J. Howison, M. Conklin, and K. Crowston. FLOSSMole: a collaborative repository for FLOSS research data and analyses. *International Journal of Information Technology and Web Engineering*, 1(3):17–26, July–September 2006.
- [11] S. Koch. Evolution of Open Source Software systems - a large-scale investigation. In *Proceedings of the International Conference on Open Source Systems*, Genova, Italy, July 2005.
- [12] M. M. Lehman. Programs, Cities, Students: Limits to Growth?, 1974. Inaugural lecture, Imperial College of Science and Technology, University of London.
- [13] M. M. Lehman, J. F. Ramil, and U. Sandler. An approach to modelling long-term growth trends in software systems. In *International Conference on Software Maintenance*, pages 219–228, Florence, Italy, 2001. IEEE Computer Society.
- [14] G. Robles and J. M. Gonzalez-Barahona. Developer identification methods for integrated data from various sources. In *Proceedings of the International Workshop on Mining Software Repositories*, pages 106–110, St. Louis, Missouri, USA, May 2005.
- [15] T. Zimmermann and P. Weissgerber. Processing CVS data for fine-grained analysis. In *Proceedings of the International Workshop on Mining Software Repositories*, Edinburg, Scotland, UK, 2004.