Measuring the Impact of Different Metrics on Software Quality: a Case Study in the Open Source Domain

Valentino Sartori, Birhanu Mekuria Eshete, Adolfo Villafiorita
Fondazione Bruno Kessler
via Sommarive, 18
38123 Trento, Italy
valentino.sartori@tin.it, eshete@fbk.eu, adolfo.villafiorita@fbk.eu

***Abstract* — Knowledge about the expected impact of different project and technological choices is fundamental for project planning, resource allocation, and quality of the final software product. The latter property, in particular, is essential to gain users' trust and confidence. In this paper we present some preliminary results about a study we are conducting on open source web applications available in SourceForge. The ultimate goal is providing tools to support project managers and team in making choices that, being all other factors the same, increase the probability of delivering higher quality software products.**

***Keywords* - Software Robustness; Software Metrics; Software Quality; Project Metrics.**

I. Introduction

The last decade has seen a steady growth of web applications and services. Their popularity is due to many factors, among which we mention making content on the web easier to update (e.g., Content Management Systems), enabling forms of remote collaboration (e.g., with Wikis), delivering in a more efficient ways e-Government services to citizens (e.g., with web portals), and providing a new way to deploy and make available complex applications (e.g., Google Docs).

The development of web applications, however, is rather complex, since it nearly always requires the integration and harmonization of code written in different programming languages. A typical web application could, for instance, be written in PHP, use a MySQL database for storage, and deliver information to the user with pages written with HTML, CSS, and JavaScript. On top of that, the programming languages used to code the applications' logic (e.g., PHP, Ruby, Perl) do not have features, such as, for instance, type and range checking, that help programmers spot and correct errors before the application is deployed. As a result, several applications have vulnerabilities that could be exploited to, e.g., expose or steal sensitive data. Although one could claim that these problems could be mitigated using languages with more stringent syntax checks, such as Java, practical issues often make the development of web applications with these technologies unfeasible (e.g., lack of trained resources) or less attractive (the vast majority of service providers, for instance, do not offer deployment of Java applications).

Growing popularity of web applications and the flexibility granted by the different technological layers that can be used to deliver web applications have resulted in a rich array of different frameworks. We mention, as an example, ASP, C#, Java, Ruby, PHP, Perl, Python, and Javascript. When starting the development of a new web application, thus, the project team might be faced with the necessity of choosing one among the various frameworks and programming languages available for development. In such a scenario, knowledge about the impact of different technological choices could become a strategic tool to guide the selection of the technology to adopt.

This paper presents some preliminary results related to a study we have conducted on web applications made available by SourceForge. We are, in particular, interested in understanding relationship between some technological choices (e.g., the main programming language used for the development of the web application) and the quality of the corresponding product. To do so, we collected data and historical data about several applications under the "web application" category of SourceForge and tried to link the data to the quality of the product. We need to remark and emphasize that this paper is a first step toward a more systematic and complete analysis of the data. More in depth analyses, therefore, will be needed to further validate the interpretation of (some) data and (some) results we present here; the extension of the results to a wider class of applications and to a wider set of variables will help consolidate the assumptions made in this paper.

The paper is structured as follows. Section II presents the data sources and the data collection tools we used. Section III defines the goal of this work. The actual results are shown in Section IV, where we characterize the applications we have included in our study and Section V, where we show the data we have obtained. Finally, Section VI presents some related work and Section VII draws some conclusions.

II. SourceForge and Data Collection

SourceForge [1] is a repository of open source software that provides tools for managing a software development project and distributing applications for free. Tools made available by SourceForge to support project teams include versioning and bug tracking systems, wikis, forums, and repositories to distribute different releases of a system. In the words of the owners, "SourceForge.net is the world's largest open source software development web site. As of August 2010 more than 240,000 software projects have been registered to use our services by more than 2.6 million registered users, making SourceForge.net the largest collection of open source tools and applications on the net."

Given availability of data and number of projects, SourceForge is a great opportunity for researchers to analyze trends in (open source) software development. Matter of fact SourceForge has been used in the past for analyses and studies by several authors: we mention [2], [3], [4], [5], [6], and [7].

For various metrics, such as the number of downloads and team size, however, the data made available by SourceForge is the most recent value. Moreover extraction of the data requires to parse the HTML web pages of the SourceForge website. To simplify the analysis work, repositories containing dumps of the SourceForge database are available to researchers (see, e.g., [8] and [9]).

We used the SourceForge Research Data Archive (SRDA) [10]. The SRDA repository, available to registered users, provides a web form that allows to query a database containing monthly dumps of the SourceForge database. From SRDA one can get a vast amount of descriptive and statistical data about SourceForge projects and users [11]. Not all information is however made available by SRDA. In particular, no data about source code metrics, necessary for our work, was available when we performed the analyses.

To support our data collection needs, that requires downloading big amounts of data and integrating information from different sources, we developed a small system, whose architecture is depicted in Fig. 1. The left hand side of the picture shows the data sources we use, namely SourceForge (through the web pages made available on the Internet) and SRDA (through the web form made available to registered users). The right hand side of the picture shows the tools we developed:

- a *Parser*, written in Java and based on *Jtidy* [12] and on a DOM inspector, that we use to extract information from SourceForge's web pages.
- A *Repo Client* that we use to automate calls to *cvs* and *svn*, to download the source code of the projects we analyze. The source code is analyzed using *cloc* [13], a tool to compute basic metrics about source code (size, expressed in lines of code and comments).
- A *database* (*Local DB*, in Fig. 1), which we use to locally integrate and store all the information we need. The database is then queried by users to perform analyses.

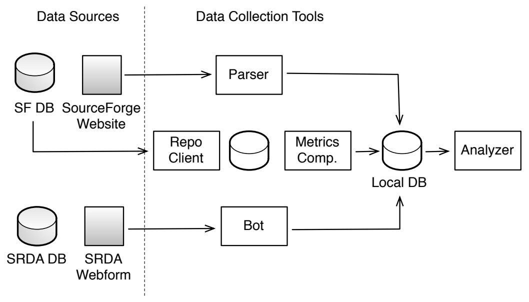

Figure 1. The system architecture

III. Goal of the Analysis and Domain

We performed analyses on the SourceForge database to achieve the following goals:

- Goal 1. Provide a characterization of Source Forge systems available in the "web application" category. The goals in this area include: understanding what categories of applications are most represented, what programming languages are used, and their growth over time; understanding whether there is a relationship between technology adopted and system size, measured both in terms of lines of code and in terms of Function Points [11].
- Goal 2. Provide a characterization of the quality of systems available in the "web application" category of SourceForge. Software quality is a difficult property to measure. Various approaches have been proposed emphasizing various dimensions, that include internal properties (e.g., software maintainability) and external properties (e.g., usability, reliability, availability, security). However, an assessment involving all these aspects can hardly be automated. We decided, therefore, to limit our attention to the number of bugs, their evolution over time, and the time taken by the project team to fix bugs. Since security bugs are of particular interests for web applications, we also distinguished and computed specific data for them.
- Goal 3. Highlight patterns between some of system characteristics (e.g., system size, programming language used) and software "quality" (in the sense of the previous goal).

The data we analyzed include all projects under the category "web based" which had released at least one version from January 2006 to May 2010 and for which there is at least one bug filed in the project's bug tracking system. This screening is necessary to select projects which have had some active development. Various projects in SourceForge, in fact, are just "placeholders" for ideas that never get developed or that will be developed in the future.

IV. A Characterization of SourceForge's WebApps

Fig. 2 and Fig. 3 provide a simple characterization of web applications in SourceForge. The data, collected from SRDA, spans from January 2006 (the first snapshot

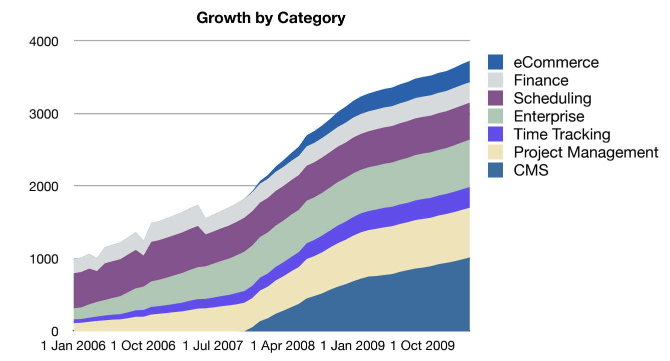

Figure 2. Growth of web applications by category

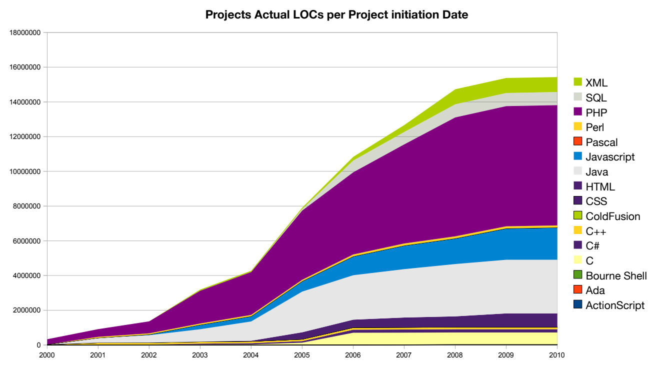

Figure 3. Source Lines of Code by Technology and Project

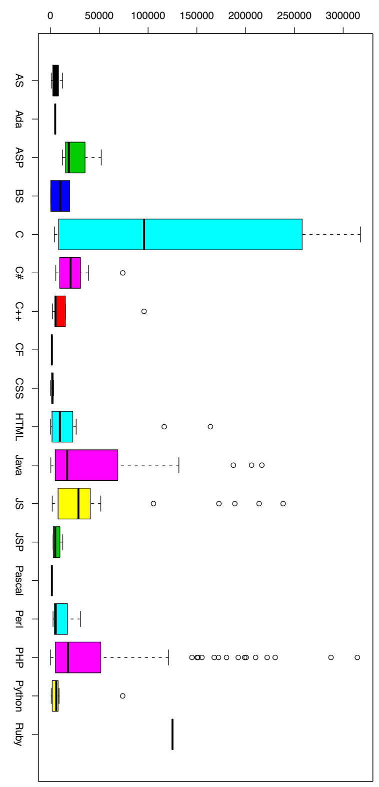

Figure 4. Source Lines of Code by Technology and Project

available) to May 2010 (the last snapshot available when we performed the analyses).

All projects in SourceForge are assigned by the project leader to one or more categories. The graph in Fig. 2 shows the growth, over time, of the categories under the "web based" umbrella. (Notice that the total sum in the graph is bigger than the actual number of projects, given the fact that one application can be assigned different categories.) Maybe not surprisingly the categories collecting the highest number of applications are CMS (Content Management Systems), Project Management, and Enterprise applications.

Fig. 2 weights all projects equally, independent from size and complexity. Applications in SourceForge, however, range from simple scripts no bigger than a few hundred lines of code to complex applications in the range of hundred of thousands of lines of code. A more accurate measure of growth, therefore should take into account also the size. This is shown in Fig. 3 where we measure, for each project, the size (in SLOC) as of May 2010 of the different technologies used in the projects. It has to be remarked that some of these technologies (e.g., CSS, XML) are not programming languages, although they can still be the "targets" of bugs and bug reports. The data of each project is shown on the date the project was started. Values accumulate over time. Thus, for instance, the data on the year 2000 (the first value on the x axis) shows the size (in SLOC) reached in May 2010 by all the projects that were started in that year. The data in 2001, as a second example, shows the size reached in May 2010 by all the projects started in 2001 together with those started in 2000. This explains the asymptotic nature of the graph, since projects started later will have had less time to evolve (and, hence grow).

The graph clearly shows that a wide variety of languages are used for developing web applications. Among them PHP is, by far, the programming language of choice, followed by Java, and Javascript. The graph omits some environments, such as Ruby and Python, which are scarcely represented in SourceForge. Notice also that the data refers to SourceForge only and that it might not be representative of the overall popularity of programming environments. Ruby on rails applications, for instance, have in RubyForge (a provider alternative to SourceForge) their repository of choice. Thus Ruby will tend to be underrepresented in SourceForge. Notice also that some of the programming languages represented in the figure, such as C and C++, come both from CGI-based web applications and from projects related to the development of desktop applications that, on the side, provide also some kind of web interface or service. Finally, we remark that the total number of lines of code we measured is about 16 million.

The technologies chosen to develop an application depend upon many factors, among which training and skills, legacy, and availability of libraries, to name a few. Fig. 3 shows the "popularity" of different technologies in SourceForge, but it does not tell us anything about whether there is a consistent usage of certain programming languages given some specific project characteristics, such as, for instance, system's size. This is shown in Fig. 4, where, we measure the size of projects for each different programming language. Data is presented with a box plot, that allows us to show the median value (the bold vertical line in the box), the

TABLE I. DENSITY OF BUGS AND SECURITY BUGS PER (MAIN) PROGRAMMING LANGUAGE

Project's Main Programming Language	*Bugs per KSLOC*	*Security Bugs per KSLOC*
Javascript	0.2	0
SQL	0.5	0.1
JSP	0.7	0
Java	0.9	0
C#	1.4	0.1
Perl	2.2	0
C	3.4	0.4
C++	4.9	0
Bourne Shell	6.6	3
Action Script	8.1	0
PHP	16.6	7

TABLE II. TIME REQUIRED TO CLOSE A BUG

Project's Main Programming Language	*Average Time to close a Bug*	*Average Time to close a Security Bug*
Javascript	74	46
SQL	54	8
JSP	9	7
Java	61	76
C#	31	3
Perl	156	201
C	322	25
C++	149	2
Bourne Shell	15	16
Action Script	48	90
PHP	76	124

are where the majority of the population lies (the box), the minimum and the maximum values (the "T"s at the extremes of the box plot). By looking at the diagram, C seems to be the most "flexible" language, since it appears in a wide variety of projects, ranging from small applications to systems in the range of 300K SLOC. Java and PHP are closely related, with similar patterns with respect to the size of projects in which these programming languages are used. In both cases the vast majority of applications written in PHP or Java is below 100K SLOCs, with PHP being the language with the most exceptions.

V. WHERE ARE THE BUGS?

Projects in SourceForge can use the SourceForge's bug tracking system to maintain track of the bugs discovered during development or usage of the system. Like many other similar systems, the bug tracker in SourceForge allows one to assign a description, a priority, a status, a category, a person responsible for the resolution, among other things. In the second part of our work we tried to correlate information about bugs and, more specifically, security bugs, with the technologies used to develop an application. We had, however, to face the following two issues related to data quality and availability:

1. Not all fields in SourceForge's bug tracker are compulsory and many projects do not file information about the category of the bug. To distinguish between security-related and non-security related bugs, when the category is not available we used a simple classification algorithm that measures the presence of specific words in the bug description. In particular, if the bug description contains some (key)words typically associated to security problems, we classify the bug as a security bug. The (key)words include, for instance: login, logout, session, phishing, penetration. The set of keywords is synthesized based on Sans Security Terms Glossary [14]. The approach is similar to [18]. See Section VI for more details.
2. The information about the file in which a bug is located is not available in the bug tracking system. Moreover, commit messages in several cases do not mention the bug they fix. As a result it is often impossible to assign a bug to a specific file and, hence, to a specific programming language. All projects, however, have a main programming language. To allocate bugs to a specific technology, therefore, we made the (strong) hypothesis that all bugs reported in a project refer to the main programming language used. Thus, for each project, we identified the main programming language (that is the programming language with the greatest number of SLOCs) and assumed all bugs referred to it.

The results of the analyses are shown in Table 1, where we report the number of bugs per thousand lines of code. Some software engineers estimate the defect density of well-written code to be between 3 and 6 per thousand lines of code [15]. Our data shows quite a few values outside the predicted range. Although one explanation could be that we refine the work in [15], a more likely explanation is due to the "noise" in our data (not all bugs refer to the main programming language) and to the great variety of applications hosted by SourceForge (which include both high and low quality software). That said, the table seems to show that Java, a language with a rather strict syntax, shows a lower density of bugs than languages with a relaxed constructs, such as PHP and Bourne Shell. There are some exceptions: Javascript seems to perform better than Java; C++, in spite of being object oriented, worse than C.

Table 2, finally, reports the average time required to close both non-security and security related bugs. The table shows the elapsed time and not the actual effort spent on fixing the bug. Thus the values in the table should be interpreted more as the combination of priority and complexity, rather than a simple measure of complexity.

Also in this case results are not definitive. The table shows that security bugs tend to be fixed in a shorter time for some technologies, but not for all of them. The interpretation is not clear: the most likely reasons could include complexity in fixing certain security bugs and some differences in the data available (e.g., the huge difference between the time take to fix bugs and security bugs in C++ could be due to the fact that there are few projects written in C++). Further analysis is needed.

VI. Related Work

In [16] the authors analyze the correlation among different object-oriented metrics. The goal is identifying dependent metrics to reduce the burden of metrics computation and to define statistically significant quality threshold for Java software. The analysis, conducted on 146 open source Java projects downloadable from SourceForge, for a total amount of over 70,000 classes and over 11 million lines of code. The author show a strong correlation among metrics in five different cases and, in the process, identify actual ranges of values for several metrics. Our work, by contrast, focuses on correlation between project choices (such as the programming language) and the corresponding quality of the end system.

In [17] the authors examine the code base of the OpenBSD operating system to determine whether its security is increasing over time. They do so by measuring the rate at which new code has been introduced and the rate at which vulnerabilities have been reported over the last 7.5 years and fifteen versions. Some of the questions the authors try to answer include aspects related to whether legacy code influences security and whether software and software development practices are leading to the development of more secure software. The authors show that the majority of security bugs are in foundational code (that is code released with the first versions of a system).

In [18] the authors use text-mining techniques to classify some bugs as security bugs. The results of the classification is then validated with software engineers yielding a 77% of correct classification. Our method for the classification of security bugs is inspired by that of the authors, although simpler in scope and lacking the manual validation phase.

Finally, in [19] the authors report on data collected during corrective maintenance and refactoring of a complex system to improve software quality. In the case of [19] the association between bugs fixed and changes to the code was possible due to the practices adopted by the development team, that required to state in the commit messages that issues being addressed.

VII. Conclusions

Knowledge about the expected impact of different project and technological choices is fundamental for project planning, resource allocation, and quality of the final software product. Open Source Repositories, such as SourceForge, not only deliver high-value services to support teams and individuals interested in open source development, but they also provide a wealth of information about software projects and development practices.

In this paper we have presented a study we conducted to understand whether some simple technological choices, such as the programming language adopted to develop an application, provide a clear advantage to control the complexity of development and increase a system's quality. We chose to analyze web applications hosted by SourceForge. The choice was made for various reasons, among which complexity of web application development and the wide choice of technologies to develop them. To understand whether some technologies consistently outperform others, we used some crude indicators, such as the density of bugs and the time required to fix bugs. The results we got, in our opinion, provide some preliminary insights.

Further work is needed to consolidate the results presented in this paper. The directions include: the input domain, the interpretation of some metrics, and the consolidation of the analyses. Concerning the first point, two obvious areas of improvement are the enlargement to a wider set of applications (e.g. by including other repositories, such as RubyForge) and the reduction of "noise" from the data (e.g., removal of projects that are not active). Concerning the second point (the interpretation of some metrics), we could extend analyses to other metrics usually related to software quality (e.g., inner quality metrics). Concerning the third point, more work is needed to systematically analyze the correlation among the different variables characterizing (SourceForge) projects.

These are some of the necessary steps to build a solid ground upon which we could eventually come with a set of rules of the thumb to guide technological choices to increase the quality of software artifacts.

References

[1] SourceForge website, Available at http://sourceforge.net. Last accessed December 20, 2010.

[2] English, R. and Schweik, C. M. "Identifying success and tragedy of floss commons: A preliminary classification of sourceforge.net projects". In *FLOSS '07: Proceedings of the First International Workshop on Emerging Trends in FLOSS Research and Development* (Washington, DC, USA, 2007), IEEE Computer Society, p. 11.

[3] Grechanik, M., McMillan, C., DeFerrari, L., Comi, M., Crespi, S., Poshyvanyk, D., Fu, C., Xie, Q., and Ghezzi, C. "An empirical investigation into a large-scale java open source code repository". In *ESEM '10: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2010), ACM, pp. 1–10.

[4] Li, Y., Tan, C.-H., Teo, H.-H., and Mattar, A. T. "Motivating open source software developers: influence of transformational and transactional leaderships". In *SIGMIS CPR '06: Proceedings of the 2006 ACM SIGMIS CPR Conference on Computer Personnel Research* (New York, NY, USA, 2006), ACM, pp. 34–43.

[5] Robles, G., and Gonzalez-Barahona, J. M. "Geographic location of developers at sourceforge". In *MSR '06: Proceedings of the 2006 International Workshop on Mining Software Repositories* (New York, NY, USA, 2006), ACM, pp. 144–150.

[6] Van Antwerp, M., and Madey, G. "The importance of social network structure in the open source software developer community". In *HICSS '10: Proceedings of the 2010 43rd Hawaii International Conference on System Sciences* (Washington, DC, USA, 2010), IEEE Computer Society, pp. 1–10.

[7] Gao, Y. and Madey, G. R. “Towards understanding: a study of the SourceForge.net community using modeling and simulation”. In *SpringSim* (2) (2007), M. J. Ades, Ed., SCS/ACM, pp. 145–150.

[8] J. Howison, M. Conklin, and K. Crowston. “Flossmole: A collaborative repository for FLOSS research data and analyses”. In *International Journal of Information Technology and Web Engineering*, 1(3), pp 17–26, 2006.

[9] C. Daffara and J. Gonzalez-Barahona. “Flossmetrics Project”, 2007. Available at http://www.flossmetrics.org/. Last accessed December 20, 2010.

[10] Van Antwerp, M. and Madey, G., “Advances in the SourceForge Research Data Archive (SRDA)”, The *4th International Conference on Open Source Systems - (WoPDaSD 2008)*, Milan, Italy, September 2008. Also available at http://www.nd.edu/~oss/Papers/srda_final.pdf, last accessed December 20, 2010.

[11] Albrecht, A. J., “Measuring Application Development Productivity,” *Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium*, Monterey, California, October 14–17, IBM Corporation (1979), pp. 83–92.

[12] JTidy - An HTML Parser and Pretty Printer in Java. Available at http://jtidy.sourceforge.net/howto.html. Last accessed December 20th, 2010.

[13] CLOC - Count Lines of Code. Available at http://cloc.sourceforge.net/. Last accessed December 20, 2010.

[14] Glossary of security terms. Available at: http://www.sans.org/security-resources/glossary-of-terms/. Last accessed December 20, 2010.

[15] Hatton, L. “Re-examining the fault density - component size connection”. *IEEE Software* 14, 2 (1997), pp. 89–97

[16] Barkmann, H., Lincke, R., and Lowe, W. “Quantitative evaluation of software quality metrics in open-source projects”. In *WAINA ’09: Proceedings of the 2009 International Conference on Advanced Information Networking and Applications Workshops* (Washington, DC, USA, 2009), IEEE Computer Society, pp. 1067–1072.

[17] Ozment, A. and Schechter, S., “Milk or Wine: Does Software Security Improve with Age?" *Proceedings of the 15th Usenix Security Symposium*, *Usenix*, 2006, pp. 93–104.

[18] Gegick, M., Rotella, P., and Xie, T. “Identifying security bug reports via text mining: An industrial case study”. In *Proceedings of the 7th International Working Conference on Mining Software Repositories, MSR 2010* (Co-located with ICSE), Cape Town, South Africa, May 2-3, 2010, pp. 11–20.

[19] Longo, F., Tiella, R., Tonella, P., and Villafiorita, A. “Measuring the impact of different categories of software evolution”. In *Software Process and Product Measurement, International Conferences: IWSM 2008, Metrikon 2008, and Mensura 2008*, Munich, Germany, November 18-19, 2008. Proceedings (2008), vol. 5338 of Lecture Notes in Computer Science, Springer, pp. 344–351.