A Study of the Impact of Open Source License Selection on Software Technical Debt

Ben Cotton
401 N. Grant St.
CIT-Knoy
West Lafayette, IN 47907
bcotton@funnelfiasco.com

Prof. Kevin Dittman
401 N. Grant St.
CIT-Knoy
West Lafayette, IN 47907
kcdittman@purdue.edu

Prof. Jeff Brewer
401 N. Grant St.
CIT-Knoy
West Lafayette, IN 47907
jbrewer@purdue.edu

Prof. Jeff Whitten
401 N. Grant St.
CIT-Knoy
West Lafayette, IN 47907
jwhitten@purdue.edu

Abstract

Open source software plays an important part in the modern world, powering businesses large and small. However, little work has been done to evaluate the quality of open source software with respect to technical debt. Two different license paradigms exist within the open source world, and this study examines the difference in software quality between them. In this paper, we use technical debt as a measure of software quality.

Eighty open source projects (40 from each paradigm) were downloaded from the popular open source hosting website SourceForge. Using complexity, code duplication, comments, and unit test coverage as inputs to the SonarQube technical debt model, each project was evaluated. The technical debt was normalized based on the cyclomatic complexity and the paradigms were compared with the Mann-Whitney test.

Introduction

The development of open source software has grown from the purview of hobbyist programmers into a major source of revenue. In 2012, Red Hat became the first open source company to record a billion dollars in revenue (Babcock, 2012). Red Hat has seen steady growth in revenue in the past decade and reported net revenue above $100 million in 2011 and 2012 (Red Hat, 2013). Other companies such as Oracle also generate revenue from support of their open source offerings. Many large Internet corporations such as Google, Facebook, and Amazon make heavy use of open source software to run their business. Small businesses especially rely on the open source Word Press and MySQL projects for their web presence(Hendrickson, Maguolas, and O'Reilly, 2012).

Researchers (Kuan, 2002, and Mockus, Fielding, and Herbsleb 2002) have investigated the quality of open source projects in comparison to their proprietarily-developed counterparts. Some efforts have been made to determine the effects on quality of governance practices in open source projects (Capra, Francalanci, and Merlo, 2008). Additional work has examined the role of license selection in developer and user engagement (Subramaniam, Sen, and Nelson, 2009). To date, no one has researched the effect of license selection on software quality. Because "quality" is a broad term with many possible measures, this study focuses on technical debt as a measure of quality.

There are two broad license paradigms in the open source world. One requires authors of derivative works to make their source available under the same license. The other permits derivative works to be relicensed under any terms. This work investigates if this difference between the legal implications of licenses results in a difference in technical debt.

Copyleft licenses

Copyleft licenses, as typified by the GNU General Public License (GPL), focus on preserving the freedom of users (Williams, 2011). Anyone who receives software licensed under a copyleft license may edit and re-distribute the software, provided that the derivative work is also licensed under the same terms (Carver, 2005). This requirement ensures that software initially released under an open source license cannot be re-licensed under a license that restricts the four freedoms laid out by the Free Software Foundation (2013). Critics decry the coincident effect of making all software that includes copyleft-licensed code necessarily copyleft, arguing that it is a threat to intellectual property (Mundie, 2001). The legal and political merits of copyleft licenses are beyond the scope of this work.

Raymond (1999) famously noted "given enough eyes, all bugs are shallow." If this is true, it follows that reducing the pool of potential contributors potentially weakens a project’s ability to produce quality software. Developers who wish to use a non-copyleft license would avoid projects that use a copyleft license and would not make derivative works of copyleft projects. Thus, employing a copyleft license might reasonably be expected to diminish the quality of a software project. Research has indeed shown that use of copyleft licenses is associated with lower developer interest (Subramaniam et al., 2009).

Conversely, because copyleft licenses compel derivative works to use the same license, any defects fixed downstream are available to all users, including the original developer. Proponents could argue that copyleft promotes quality software by preventing fixes from being "hidden". However, the original developer is under no obligation to accept any bug fixes or enhancements created by downstream projects.

Permissive licenses

Permissive licenses still meet the Open Source Initiative’s definition (n.d.) of "open source", but lack the same-license requirement that is the hallmark of copyleft. Anyone who receives software licensed under a permissive license may edit and re-distribute the software, under any license they choose (Carver, 2005). This leads to the possibility that software that was formerly open source may become proprietary. For those who view free software as a moral imperative (Williams, 2011), such licenses are unacceptable.

From a practical standpoint, permissive licenses are associated with higher levels of developer interest (Subramaniam et al., 2009). One may expect that this is due to the fact that permissive licenses maximize developer freedom instead of user freedom. Again using Raymond's (1999) "enough eyes" aphorism, it is reasonable to expect that a broader developer pool will result in higher quality software.

However, Subramaniam et al. (2009) found an association between permissive licenses and lower interest from users. One may expect a lower degree of user feedback (e.g. bug reports) as a result, providing fewer opportunities for developers to notice and fix defects. Since derivative works can include fixes not available to the original authors, it can be seen how permissive licenses might hinder the quality of the original software.

Technical Debt Model

Cunningham (1992) was the first to use the term ”technical debt.” He used the analogy of a mortgage loan: ”not right code” is used to ship a product more quickly in the same way a mortgage is used to buy a house more quickly. As a mortgage must be paid with interest, so too must the ”not right code” be paid with rework. The greater the technical debt incurred upfront, the greater the payment required in the future. In effect, technical debt may be understood as selling quality to purchase a shorter initial development time.

Although Cunningham is credited with introducing the term to the literature, it is easy to trace the roots of the idea back further. Crosby (1979) wrote ”every penny you don’t spend on doing things wrong, over, or instead becomes half a penny right on the bottom line.” He called the expense of scrap, rework, testing, inspection, et cetera the ”cost of quality.” Crosby’s arguments for saving actual cash can be taken as an argument in favor of reducing technical debt

Project selection

Software packages analyzed for this study come from the open source hosting site SourceForge.com. The packages were selected by performing a search of the "Most Popular" projects for the most popular licenses as given in the Open Source Initiative's (2006) License Proliferation Report, listed in Table 1. The selected packages and their license paradigm are listed in Table 2. All packages were downloaded on March 29, 2014. The most recent available release was downloaded; in-development branches were not considered except when a stable release was not presented.

License	Paradigm
Apache	Permissive
BSD 2-Clause	Permissive
BSD 3-Clause	Permissive
Common Development and Distribution License	Copyleft
Eclipse Public License	Copyleft
GNU “Lesser” General Public License	Copyleft
GNU General Public License	Copyleft
MIT (a.k.a. “X11”)	Permissive
Mozilla Public License	Copyleft

Table 1: The open source licenses used in this study.

Permissive	Copyleft
CoCEd	Angry IP Scanner
FlacSquisher	Dcm4chee
PDFMerge	Div/er
dom4j	DocFetcher
DrJava	Eclipse Checkstyle Plug-in
DSpace	EclipseFP
FreeMarker	FreeMind
FreeTTS	GURPS
Geotag	HAPI
HyperSQL Database Engine	Hattrick Organizer
Joda-Time	JasperReports Library
Json-lib	JFreeChart

KoLmafia	jpcap
ksar	jTDS
lwjgl	jVi
OpenGTS	Mondrian
OWASP Zed Attack Proxy	Pentaho Platform
picard	PyDev
Sahi	RSS Owl
VietOCR	SAP Netweaver Plugins for Eclipse
Adminer	ShellEd
HybridAuth	SMC
ISPConfig	SoapUI
magmi	Squirrel SQL
Mibew Messenger	SweetHome 3D
NagiosQL	TikiOne Steam Cleaner
Open Source Point of Sale	TuxGuitar
phpseclib	VASSAL
Simple HTML DOM Parser	Vuze
Ganglia (gmetad-python)	Weka
PyFFI	phpMyAdmin
pyparsing	The Bug Genie
PyUSB	gns3
SciPy	OpenLP
Scons	RPy
SimpleCV	SABnzbd+
WikiPad	TaskCoach

Table 2: The software projects examined in this study.

Metrics Collected

This study collected technical debt as measured by the SQUALE technical debt plugin for the SonarQube software analysis system. Measurements were collected for each of the software packages. The values fed into SQUALE are given in Table 3. Thresholds are drawn from Eisenberg (2012). Absent a well-established methodology in the scholarly literature, Eisenberg's approach is straightforward enough for use here. Each programming language uses a language-specific plugin, so some measures are not available in all languages.

Measure	**Python**	**C#**	**Java**	**PHP**
Class complexity	60	60	60	60
File complexity	60	60	60	NA
Duplicate blocks	Yes	Yes	Yes	Yes
FIXME/TODO comments	Yes	Yes	Yes	FIXME only
Comment density	25	25	25	25
Branch coverage in unit tests	80	80	80	80
Line coverage in unit tests	80	80	80	80

Table 3: The metrics used in this study.

The size of a project reasonably impacts the potential for the accumulation of technical debt. It therefore makes sense to normalize the technical debt reported by SonarQube. Two measures of size are readily available from SonarQube analysis: lines of code (LOC) and cyclomatic complexity. The line count for a section of code is a strict count of the non-blank, non-comment lines. As such, it is subject to the coding style of the developer. Cyclomatic complexity, as introduced by McCabe (1976), is based on the structure of the code elements and is therefore consistent regardless of visual style. For this study, the technical debt of a project is reported as days per thousand units of complexity.

The distribution of technical debt for the two license paradigms were be compared using the Mann-Whitney (1947) test. This non-parametric test was selected due to its minimal assumptions about the distribution of the data. Arcuri and Briand (2011) recommend this test for testing the differences in two groups. Due to the relative novelty of this study, a significance of 0.10 was chosen for the test.

Data analysis

The distribution of normalized debt is shown in Figure 1. A visual inspection of the normalized technical debt values for the two paradigms, as seen in the figure suggests we should see a difference between them. Using the Mann-Whitney test confirms the visual analysis. The normalized technical debt in copyleft-licenced programs is higher than in permissively-licensed programs (U = 531, p = .010).

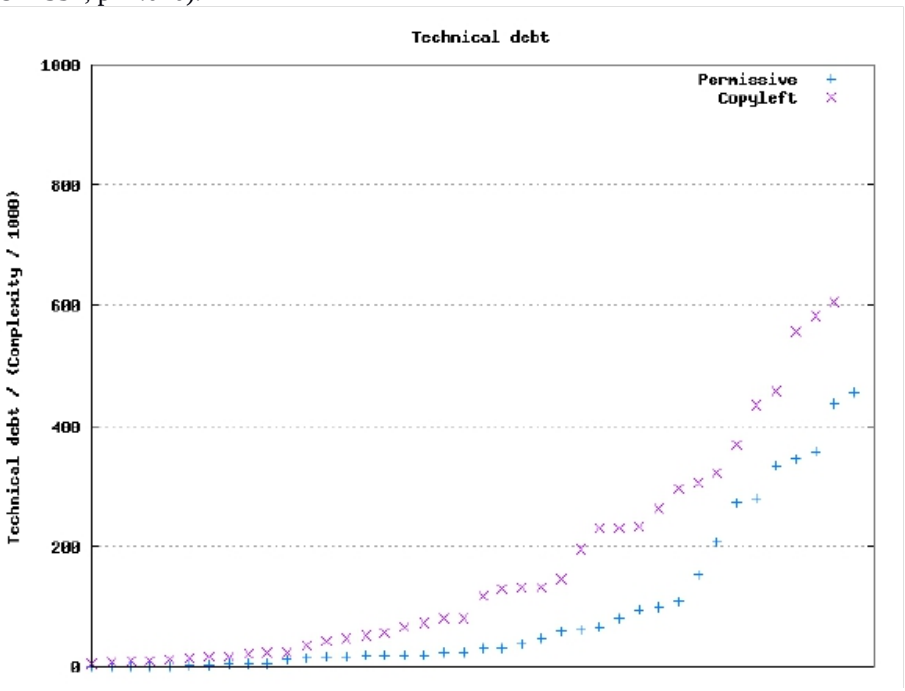

Figure 1.0 Technical debt of projects analyzed in this study

As Zhao and Elbaum (2000) noted, the language used in a project can have an impact on the resulting quality. For both licensing paradigms, Java was the dominant language. However, many of the most popular (as listed on SourceForge) copyleft programs are written in C or C++. Since a free SonarQube plugin for those languages was not available at the time of this study, those programs could not be included. Illustration shows the distribution of programming languages for both licensing paradigms.

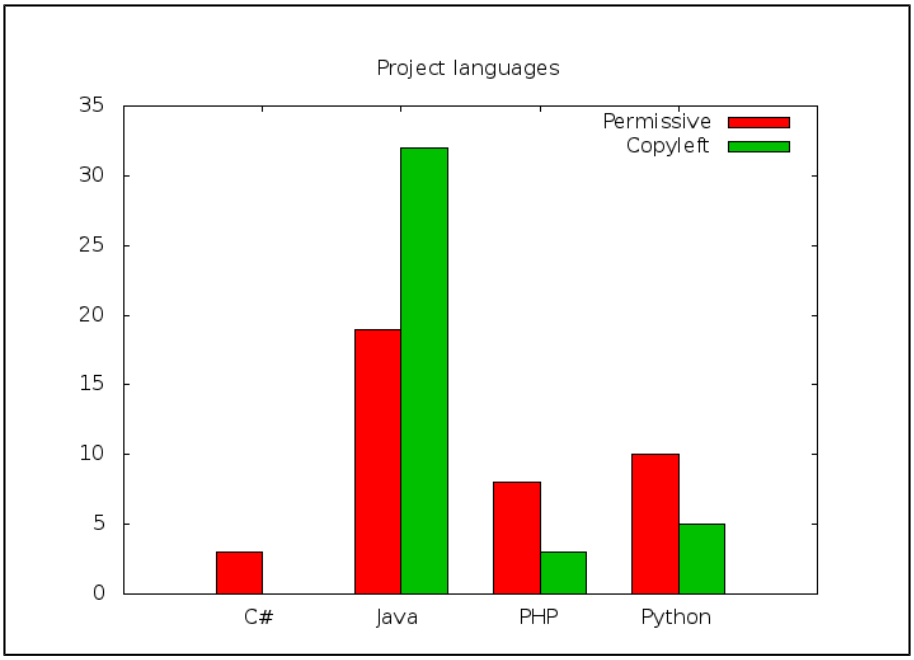

Figure 2.0 Distribution of programming languages in this study

Due to the predominance of Java in this study, the test was re-run with only the Java projects. The Mann-Whitney test does not require the sample sizes to be equal, but it does lose power with greater inequality (Mann-Whitney, 1947). In this case, although the visual analysis (as shown in Illustration suggests copyleft Java programs have lower technical debt, the statistical results are inconclusive (U = 258, p = .370).

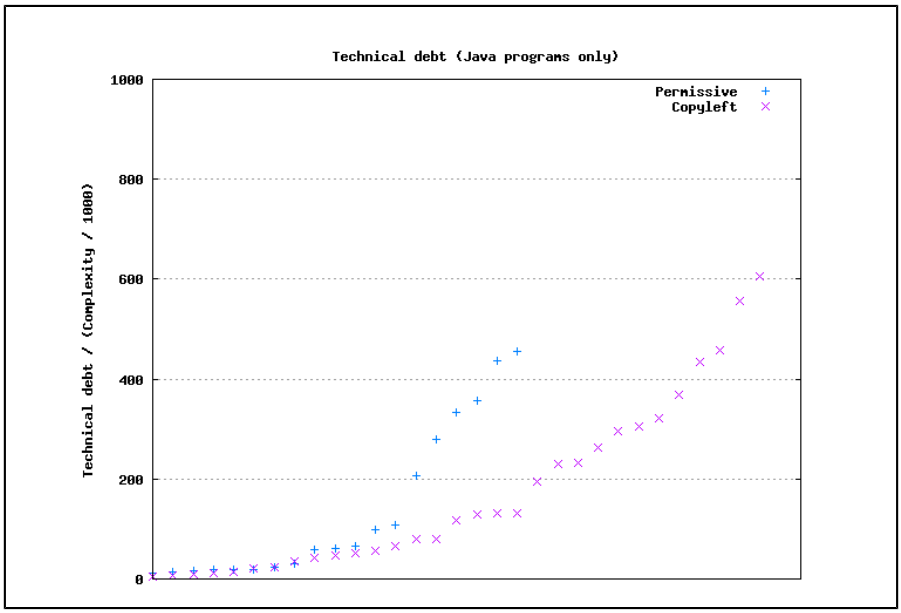

Figure 3.0 Normalized technical debt of Java programs in this study.

Indeed, averaging the normalized technical debt by language shows a higher mean debt for copyleft Java programs. The same is true for Python and not PHP, but the total count of the projects in those languages are small enough to render statistical tests meaningless. Nonetheless, it reinforces the findings of Zhao and Elbaum (2000) and offers avenues for further exploration. Table 4 lists the mean debt by language and paradigm.

Language	Permissive	Copyleft
C#	53.5	(none)
Java	7.2	1.8
PHP	8.7	1.8
Python	8.3	21.5

Table 4: Mean normalized technical debt of projects by language and paradigm.

Conclusion

In conclusion, we have established that a difference exists in the technical debt, of copyleft- and permissively-licensed software projects. Permissive licenses correspond to lower technical debt. Because permissive licenses have higher levels of developer engagement (Subramaniam et al., 2009), more developer involvement may be understood to correspond to lower levels of technical debt.

Absent other considerations, the results of this study suggest that a project manager looking to develop or incorporate open source software should look toward permissive licenses. This is a simple method for improving the likely overall quality of the project. By using higher-quality inputs, the project's quality management efforts can be used to address other areas.

It is important to remind the reader that this study is novel. While published literature on technical debt exists, much of the discussion is qualitative rather than quantitative. Furthermore, comparisons of the license paradigms within open source software development appears to be a largely unexplored field. Literature discussing open source software quality is largely confined to comparison against proprietary programs.

Due to the novelty of this study, the reader must be careful to not overextend the conclusions. The p value of the Mann-Whitney test was small enough that the null hypothesis would have been rejected with a less conservative significance. Nonetheless, this study is hardly a definitive statement on the matter. The results presented here serve to indicate that further study is warranted.

Because this study is so novel, a great deal of work remains to be done. First, a larger study that involves projects written in C and C++ should be conducted. These languages are widely used in software projects; TIOBE.com's August 2014 index has C as the most popular language and C++ as the 4th most popular language. Both languages have averaged in the top 5 for over two decades (TIOBE, 2014). Inclusion of these languages would give additional credibility to the results.

Furthermore, enlarging the sample to enable single-language comparisons would eliminate any language-specific effects. As Zhao and Elbaum (2000) observed, and this study's results confirmed, quality is not consistent across languages. A set of studies, each focused on one language, would provide more robust results.

Knowing that developer and user engagement differ based on the license paradigm (Subramaniam et al, 2009), further study of the communities around projects is warranted. Is the observed difference in technical debt due to the effects of the license, or are coincident governance factors responsible? While some studies have touched on governance, further study is warranted. In particular, a good taxonomy is needed to serve as a basis for study of governance within projects.

Another avenue for further work is to improve technical debt models. Mapping technical debt to other quality measures will help validate the use of technical debt and allow for consistent measurement across projects. In addition, more rigorous technical debt models will allow project managers and developers to guide decisions to reduce the accumulation of technical debt.

References

A. Arcuri and L. Briand. *A practical guide for using statistical tests to assess randomized algorithms in software engineering. In Software Engineering (ICSE), 2011 33rd International Conference* on, pages 1– 10. IEEE, 2011.

C. Babcock. Red Hat: First $1 billion open source company. Retrieved February 16, 2013, from: http://www.informationweek.com/development/open-source/red- hat-first-1-billion-open-source-comp/232700454, March 2012.

J. L. Brewer and K. C. Dittman. *Methods of IT project management*. Prentice Hall, 2010.

N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, et al. Managing technical debt in software-reliant systems. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 47–52. ACM, 2010.

E. Capra, C. Francalanci, and F. Merlo. An empirical study on the relationship between software design quality, development effort and governance in open source projects. *Software Engineering, IEEE Transactions* on, 34(6):765–782, 2008.

B. W. Carver. Share and share alike: Understanding and enforcing open source and free software licenses. *Berkeley Tech. LJ*, 20:443, 2005.

P. B. Crosby. *Quality is free: The art of making quality certain*, volume 94. McGraw-Hill New York, 1979.

W. Cunningham. The WyCash portfolio management system. In *ACM SIGPLAN OOPS Messenger*, volume 4, pages 29–30. ACM, 1992.

R. J. Eisenberg. A threshold based approach to technical debt. *ACM SIGSOFT Software Engineering Notes*, 37(2):1–6, 2012.

J. Feller, B. Fitzgerald, et al. *Understanding open source software development*. Addison-Wesley London, 2002.

Free Software Foundation. What is free software? Retrieved February 16, 2013, from: http://www.gnu.org/philosophy/free-sw.html, 2013.

M. Hendrickson, R. Magoulas, and T. O'Reilly. *Economic impact of open source on small business: A case study*. O'Reilly Media, July 2012.

J. Kuan. Open source software as lead user's make or buy decision: a study of open and closed source quality. Stanford Institute for Economic Policy Research, Stanford University, 2002.

H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, 18(1):50–60, 1947.

T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, (4):308–320, 1976.

A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(3):309– 346, 2002.

C. Mundie. Speech transcript. Retrieved February 16, 2013, from: http://www.microsoft.com/en-us/news/exec/craig/05-03sharedsource.aspx, December 2011.

E. Raymond. The cathedral and the bazaar. *Knowledge, Technology & Policy*, 12(3):23–49, 1999.

Red Hat. Annual reports. Retrieved February 17, 2013, from http://investors.redhat.com/annuals.cfm, 2013.

TIOBE Software. TIOBE Index for August 2014. Retrieved August 31, 2014 from http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html, Aug. 2014.

C. Subramaniam, R. Sen, and M. L. Nelson. Determinants of open source software project success: A longitudinal study. *Decision Support Systems*, 46(2):576–585, 2009.

S. Williams. *Free as in freedom: Richard Stallman's crusade for free software*. O'Reilly Media, 2011.

L. Zhao and S. Elbaum. A survey on quality related activities in open source. *ACM SIGSOFT Software Engineering Notes*, 25(3):54–57, 2000.