A Survival Analysis of Source Files Modified by New Developers

Hirohisa Aman[1(✉)], Sousuke Amasaki[2], Tomoyuki Yokogawa[2], and Minoru Kawahara[1]

[1] Ehime University, Matsuyama, Ehime 790–8577, Japan
aman@ehime-u.ac.jp
[2] Okayama Prefectural University, Soja, Okayama 719–1197, Japan

Abstract. This paper proposes an application of the survival analysis to bug-fix events occurred in source files. When a source file is modified, it has a risk of creating a bug (fault). In this paper, such a risk is analyzed from a viewpoint of the survival time—the time that the source file can survive without any bug fix. Through an empirical study with 100 open source software (OSS) projects, the following findings are reported: (1) Source files modified by new developers have about 26% shorter survival time than the others. (2) The above tendency may be inverted if the OSS project has more developers relative to the total number of source files.

Keywords: Open source development · Survival analysis · Time to bug fix

1 Introduction

Open source software (OSS) products have been more and more popular in the IT world. Many users and companies have utilized in their social life or business. As an OSS product has more users or stakeholders, post-release failures occurred in the product have larger impacts [4]. Hence, the quality management of OSS products has gotten a lot of attention recently. A software product usually evolves through its functional enhancements and bug (fault) fixes. Although it is ideal that developers never create any bugs, the bug-free evolution would be hard in reality; some code modifications are also creations of new bugs [6,8]. Nonetheless, frequent bug fixes are always undesirable in software development.

There have been many studies in regard to bug-fix prediction in the past. Rahman et al. [11] reported the trend that recently-bug-fixed source files are likely to be fixed again. Google utilized their results and released the prediction tool working on Git repositories [5]. Bird et al. [3] and Posnett et al. [10] focused on the ownership of source files and reported that a source file having lower ownership is likely to be more fault-prone; low ownership of a source file means that the file has not been developed and maintained by specific core developer(s), i.e., the file has been modified by various developers.

M. Felderer et al. (Eds.): PROFES 2017, LNCS 10611, pp. 80–88, 2017.
https://doi.org/10.1007/978-3-319-69926-4_7

According to the previous work, the change history and ownership of source files are promising data for analyzing the occurrence of bug fixes. However, most previous studies focused on the number of bug fixes or the bug-fix rate; we consider that the time to bug fix would be yet another noteworthy feature to be analyzed. For example, suppose bug fixes were made in two files f_A and f_B. If f_A was modified "one day" ago and f_B was done "one year" ago, we should preferentially examine the precedent modification of f_A than f_B. In order to analyze such a difference, we will focus on the survival analysis [12]. The survival analysis is popular in the medical field, which analyzes the survival rate and the survival time of patients who were received specific treatments. In this paper, we propose an application of the survival analysis to the bug-fix survival time—the time to bug fix. The key contributions of this paper are as follows:

1. An application of the survival analysis to the bug-fix events in source files: To the best of our knowledge, although there have been studies utilizing the survival analysis to the substantivity of OSS projects [2,14], there have not been a study applying it to the time to bug fix.
2. An empirical report with many OSS projects: We collected data from 100 OSS projects, and report the results of the bug-fix survival analysis.

2 Survival Analysis and Its Application to Bug Fix

2.1 Survival Analysis

The survival analysis is a statistical method for analyzing the time to the occurrence of an event (e.g., a patient death in a clinical site).

Let t be the elapsed time from the start of our observation, and $S(t)$ be the probability that the event of interest had not been occurred until t, i.e., a subject survives at t. $S(t)$ is a monotonically decreasing function and called the survival rate function. The expected survival time μ_t is computed as:

$$\mu_t = -\int t \; dS(t). \tag{1}$$

That is to say, μ_t is the area under $S(t)$ (see Fig. 1).

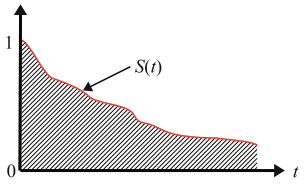

Fig. 1. Example of $S(t)$ and μ_t (area of the hatched part).

Needless to say, there may be a subject which has survived after the end of our event observation. Such a subject is called a "censor sample." If we have a

censor sample, we cannot get true $S(t)$. However, when our event occurrence time is discrete type ($t = t_1, t_2, \ldots, t_i, \ldots$), we can estimate it by using the Kaplan-Meier (KM) method (see [7] for the details). The KM method is a popular non-parametric method for estimating $S(t)$ using the cumulative hazard as:

$$S(t_i) = \prod_{k=1}^{i} \{1 - \lambda(t_i)\}, \tag{2}$$

where $\lambda(t_i)$ is the hazard at t_i and $\lambda(t_i) = e_i/n_i$; n_i is the number of subjects which have survived at least just before t_i and e_i is the number of subjects which died (encountered the event) at t_i, respectively.

2.2 Application of Survival Analysis to Bug Fix

Now we consider another survival analysis in which a subject and an event are a source file and a bug fix, respectively. Then, we can develop a model of the time to bug fix in a source file. While many studies have been done for predicting bug fixes in the past, most of them focused on the number of bug fixes or the occurrence rate. We will focus on yet another point of view in this paper. According to the previous work regarding the file ownership [3,10], modifications made by new developers may have higher risks of causing future bug fixes. By analyzing the survival time to bug fix, we will evaluate such risks. We describe the survival analysis of the time to bug fix in the remainder of this section.

Let f be a source file, and t_E be the time when the observation was finished, respectively. Then, let $t_M (< t_E)$ be the time when f was finally modified (or created) but the modification was not a bug fix. If a bug is fixed in f at t_B, the life time of f to the bug fix, $L(f)$, is computed as $L(f) = t_B - t_M$. If bug fixes occurred in f twice or more, use the oldest bug-fix time as t_B; if no bug fix was observed until t_E, f is a censor sample. Then, we define $L(f) = t_E - t_M$.

Next, we check the developer who made the modification at t_M, and examine whether he/she is a new developer who has never been involved in f before t_M. Then, we categorize f into two types, NEW and CONV—if f is modified by a new developer at t_M, we consider f to be Type NEW; otherwise, f is Type CONV which means that the modification is made by a conventional developer.

Table 1 presents a simple example where eight files f_1, f_2, $\cdots$, f_8 have been developed and maintained by three developers (ID $= 1, 2, 3$). Symbols "A," "M" and "B" in the table signify a creation of new file, a modification of a file, and a bug fix of a file, respectively. Their subscripts denote the developer ID who made those work. For each file, the modification (or the creation) according to its t_M is marked with an asterisk. For example, f_1, f_4 and f_5 are modified by developer 1 at $t = 21$. f_1 is Type NEW because its final modification at $t = 26$ is made by developer 2 and it is the first time for developer 2 to modify f_1 at that time. Similarly, f_5 is Type CONV since its final modification before the bug fix (at $t = 21$) is made by developer 1 and he/she had already been involved in f_5. Since f_1, f_4, f_6 and f_8 have no bug fix within the observation duration ($0 \le t \le t_E$), they are censor samples and their life times are computed as $L(f_i) = t_E - t_M$. The remaining files' life times are obtained as $L(f_i) = t_B - t_M$.

Table 1. Example of source file development history.

file	t											t_E	type	t_B	t_M	$L(f_i)$
	0	5	12	15	18	20	21	25	26	30	31	35				
f_1	A_1	M_1					M_1		M_2^*				NEW	—	26	9
f_2		A_1				M_2^*		B_2					NEW	25	20	5
f_3		A_1		M_2^*	B_1								NEW	18	15	3
f_4				A_2			M_1				M_3^*		NEW	—	31	4
f_5		A_1					M_1^*	B_2					CONV	25	21	4
f_6	A_1		M_2	M_2^*									CONV	—	15	20
f_7			A_2^*					B_2					CONV	25	12	13
f_8										A_3^*			CONV	—	30	5

The KM method can estimates the survival rate function $S(t)$ with Eq. (2). Figure 2 shows the estimated $S(t)$ of each type. We can see that NEW has a shorter life time to bug fix than CONV. The expected survival time of NEW and CONV are 6 and 13.375, respectively[1]. While both type have the same bug-fix rate (50%), they have remarkable differences in terms of expected survival time—6 vs. 13.375: the expected survival time of Type NEW is shorter than the half of Type CONV's survival time.

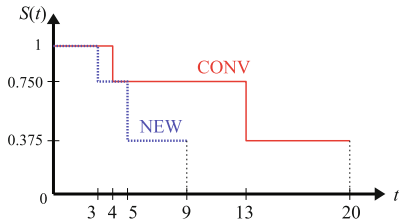

Fig. 2. Estimated $S(t)$'s for NEW and CONV types in Table 1.

3 Empirical Study

3.1 Dataset

We collected 100 local copies of OSS projects' repositories which are available on the GitHub, and obtained data to be analyzed from those repositories. Our subject projects consist of 50 Java projects and 50 C++ ones. The main reason why we selected these projects is their popularities; we believe that a finding derived

[1] NEW: $1 \times 3 + 0.750 \times (5-3) + 0.375 \times (9-5) = 6$; CONV: $1 \times 4 + 0.750 \times (13-4) + 0375 \times (20-13) = 13.375$.

Table 2. Analyzed OSS projects (in decreasing order of "stars").

Java projects	C++ projects
RxJava, elasticsearch, retrofit, okhttp, java-design-patterns, guava, leakcanary, zxing, libgdx, interviews, fastjson, dubbo, Android-CleanArchitecture, realm-java, MaterialDrawer, ExoPlayer, deeplearning4j, BottomBar, spark, vert.x, dagger, presto, junit4, Android-Bootstrap, dropwizard, UltimateRecyclerView, uCrop, jedis, auto, guice, mybatis-3, jadx, metrics, mockito, HikariCP, webmagic, buck, j2objc, jsoup, lombok, rebound, swagger-core, pinpoint, scribejava, okio, android-classyshark, async-http-client, mosby, CoreNLP, dex2jar	folly, imgui, json, libphonenumber, openFrameworks, Catch, proxygen, capnproto, rapidjson, libzmq, libsass, muduo, crow, tiny-dnn, ppsspp, dlib, spdlog, Cpp-Primer, openpose, pybind11, GamePlay, re2, concurrentqueue, envoy, oclint, zopfli, nghttp2, cpprestsdk, websocketpp, osrm-backend, BansheeEngine, swig, i2cdevlib, algorithms, AtomicGameEngine, mlpack, thrust, iaito, glog, cpr, cpp-ethereum, magnum, cppcheck, gosu, phxpaxos, deepdetect, actor-framework, cereal, oryol, cling

from more popular projects is more attractive for more researches and practitioners. These projects have high "stars" scores at GitHub: We performed project searches sorted by "most stars" option, where the search keywords were "Java" and "C++," respectively. Table 2 presents the names of collected projects.

3.2 Procedure

For each project, we conducted our empirical study in the following four steps.

1. Made a copy of the repository, and obtained the set of source files included in the latest version (F). Source files for testings, demos and documents were excluded from F. Let t_E be the time when the repository was copied.
2. For each $f \in F$, extracted its change history from the commit logs, and decided t_B: if the commit message of f contained a bug fix-related keyword, we considered that a bug fix was performed in f at the commit [13].
3. For each $f \in F$, determined t_M and and f's type—NEW or CONV. The identification of developer was performed in accordance with the following rules [1]: if two developers had the same name or the same e-mail address, we considered that they are the same developer.
4. Estimated the survival rate function $S(t)$ for each types, NEW and CONV, using the KM method. Then, computed the expected survival time with Equation (1): let μ_N and μ_C be the expected survival time in NEW and CONV, respectively. To compare their differences across projects, define the following criterion, $\Delta\mu$:

$$\Delta\mu = \frac{\mu_C - \mu_N}{\mu_C}. \tag{3}$$

If $\Delta\mu$ has a larger positive value, Type NEW files have shorter expected survival times than Type CONV ones. While "$\mu_C - \mu_N$" directly shows the difference of two survival times, we considered it is better to normalize the difference with using one of those times because there would be dispersions of survival times among projects; such a raw difference would not be suitable for comparing different projects.

3.3 Results

Table 3 and Fig. 3 show distributions of $\Delta\mu$ values. The median and the average (mean) of $\Delta\mu$ in the 100 OSS products are 0.394 and 0.258, respectively (see Table 3). Moreover, a majority of projects show $\Delta\mu > 0$ (see Fig. 3). In other words, the expected survival times of Type NEW source files tend to be about 26% shorter than that of Type CONV on average.

Table 3. Distribution of $\Delta\mu$ values in analyzed OSS projects.

Min.	25%	Median	Mean	75%	Max.
−1.675	0.048	0.394	0.258	0.628	0.997

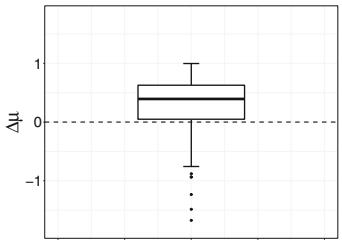

Fig. 3. Boxplot of $\Delta\mu$ values in analyzed OSS projects.

3.4 Discussions

Through the survival analysis, we have understood a major trend of time to bug fix in many OSS projects—how long time a source file tends to take until a bug fix, rather than whether a bug fix would occur or not. A source file modified by a new developer (Type NEW) would have a higher risk of a latent bug and the time to bug fix would be about 26% shorter than another type of source file (Type CONV). This trend seems to support the previous work regarding the file ownership [3,10].

While the above results show an overall trend, some projects had small $\Delta\mu$ values or the opposite trends. To examine if there is a statistically significant

difference between NEW and CONV in terms of $S(t)$, we performed the log-rank test [9] (at a 5% significance level). The test results were as follows: there seem to be significant differences in 65 products (does not in 35 products); in 56 out of 65 products, Type NEW tends to have a shorter survival time (denote it by "NEW < CONV"); the remaining 9 products show the opposite tendency ("CONV < NEW").

To explore a difference between two cases "(a) NEW < CONV" and "(b) CONV < NEW," we examined the numbers of developers ($=n_d$) and source files ($=n_f$) in those projects, and calculated the ratio between them: $r = n_d/n_f$. While the average of r values in case (a) is about 0.079, that in case (b) is 0.246. That is to say, r in case (b) is about 3 times larger than case (a); furthermore, the average r of the outliers shown in Fig. 3 is 3.93 which is about 50 times larger than case (a). Hence, case (b) tends to have more variety in terms of developers relative to the number of source files to be maintained, and they would be projects which more new developers can actively contribute to. We would like to do a further analysis from such a viewpoint as our future work.

Now, we have to notice that we have checked bug "fixing" events but not bug (fault) "inducing" ones. In other words, even if a bug fix occurred after a new developer's commit, it is not always true that the corresponding fault was induced by the new developer. We need to perform a further analysis of fault-inducing commits in the future for an enhanced discussion.

3.5 Threats to Validity

We collected only Java and C++ data from only Git repositories. Since our analysis method is applicable to any other programming languages and version control systems (VCSs) without any change, the limitations of language and VCS are not serious threats to validity. Nonetheless, there is a risk of getting different results in OSS projects other than the ones we examined. To mitigate such a threat, we collected empirical data from a set of many popular projects.

While we focused on whether a source file modification was made by a new developer or not, we did not examine his/her experience of maintaining other source files and expertise. Since our findings in this paper are derived from a lot of samples, impacts of individuals on our results might not be serious threats. A further analysis on individuals is our important future work.

We decided whether a commit is a bug fix or not, by using a keyword matching method [13], and it is a popular way of detecting bug-fix commits in the mining software repository community. However, our analysis was based on the assumption that all commit messages provided proper information in regard to their code changes. The assumption can be a threat to validity. For example, some bug-fix commits might be missed in our dataset because some developers might not appropriately describe their commit messages even if they performed bug fixes. The development of a more accurate detection method is one of our future challenges.

4 Conclusion

For a bug fix of a source file in an OSS development, we focused on the developer who made the last modification of the file before the bug fix—whether the developer had an experience of modifying the file at the time or not. In accordance with the above developer type, we categorized source files into the following two types, NEW and CONV: if a source file's last modification before its bug fix (or the end of our observation) was made by a new developer who had never modified that file, the source file is Type NEW; otherwise, it is Type CONV. Then, we considered to compare these two types in terms of the time to bug fix through a survival analysis, and conducted an empirical study with 100 OSS projects which are available on the GitHub. The empirical results showed that the expected survival time of Type NEW source files is about 26% shorter than that of Type CONV ones. That is to say, when a source file is modified by a new developer who had not been involved in the maintenance of the file, that file is likely to require a bug fix sooner. In such a case, a more careful review would be useful to prevent a quality degradation.

On the other hand, if a project has more developers relative to the total number of source files, the project tends to produce the opposite tendency: Type NEW has a longer expected survival time. Such a project may be easier for more new developers to contribute. A further analysis on those points of view is our future work. Moreover, we plan to take into account the degree of a developer's familiarity with a file and to perform a further analysis toward a just-in-time defect prediction.

Acknowledgment. This work was supported by JSPS KAKENHI #16K00099. The authors would like to thank the anonymous reviewers for their helpful comments.

References

1. Bird, C., Gourley, A., Devanbu, P., Gertz, M., Swaminathan, A.: Mining email social networks. In: Proceedings of International Workshop Mining Software Repositories, pp. 137–143 (2006)
2. Bird, C., Gourley, A., Devanbu, P., Swaminathan, A., Hsu, G.: Open borders? Immigration in open source projects. In: Proceedings of 4th International Workshop Mining Software Repositories (2007)
3. Bird, C., Nagappan, N., Murphy, B., Gall, H., Devanbu, P.: Don't touch my code!: examining the effects of ownership on software quality. In: Proceedings of 19th ACM SIGSOFT Symposium and 13th European Conference Foundations of Software Engineering, pp. 4–14 (2011)
4. Duck, B.: The 2017 open source 360° survey (2017). https://www.blackducksoftware.com/about/news-events/releases/open-source-360-organizations-increase-reliance-open-source
5. Google: Bugspots (2011). https://github.com/igrigorik/bugspots
6. Jones, C.: Applied Software Measurement: Global Analysis of Productivity and Quality, 3rd edn. McGraw-Hill, New York (2008)

7. Kaplan, E.L., Meier, P.: Nonparametric estimation from incomplete observations. J. Am. Stat. Assoc. **53**(282), 457–481 (1958)
8. Li, Y., Li, D., Huang, F., Lee, S.Y., Ai, J.: An exploratory analysis on software developers' bug-introducing tendency over time. In: Proceedings of International Conference Software Analysis, Testing and Evolution, pp. 12–17 (2016)
9. Peto, R., Peto, J.: Asymptotically efficient rank invariant test procedures. J. Roy. Stat. Soc. Ser. A (Gen.) **135**(2), 185–207 (1972)
10. Posnett, D., D'Souza, R., Devanbu, P., Filkov, V.: Dual ecological measures of focus in software development. In: Proceedings of International Conference on Software Engineering, pp. 452–461 (2013)
11. Rahman, F., Posnett, D., Hindle, A., Barr, E., Devanbu, P.: Bugcache for inspections: hit or miss? In: Proceedings of 19th ACM SIGSOFT Symposium and 13th European Conference on Foundations of Software Engineering, pp. 322–331 (2011)
12. Rupert, G., Miller, J.: Survival Analysis. John Wiley & Sons, Hoboken, NJ (2011)
13. Śliwerski, J., Zimmermann, T., Zeller, A.: When do changes induce fixes?. In: Proceedings of International Workshop Mining Software Repositories, pp. 1–5 (2005)
14. Samoladas, I., Angelis, L., Stamelos, I.: Survival analysis on the duration of open source projects. Inf. Softw. Technol. **52**, 902–922 (2010)