An Application of the PageRank Algorithm to Commit Evaluation on Git Repository

Sho Suzuki*, Hirohisa Aman†, Sousuke Amasaki‡, Tomoyuki Yokogawa‡ and Minoru Kawahara†
*Graduate School of Sc. and Eng., Ehime Univ., Matsuyama, Ehime 790-8577, Japan
†Center for Information Technology, Ehime Univ., Matsuyama, Ehime 790-8577, Japan
‡Faculty of Computer Sc. and Systems Eng., Okayama Prefectural Univ., Soja, Okayama 719–1197, Japan

***Abstract*—Many empirical studies have reported notable theories or methods for evaluating or predicting code quality through analyses of code repositories. This paper has yet another point of view: it focuses on "commits" rather than source code. That is to say, this paper proposes to evaluate commits themselves. When an aim of a commit is to fix a bug, there can be another preceding commit which made a reason of the bug fixing. Those commits are linked by a bug fixing-based causal relationship. Then, commits can be modeled as a directed graph model of causal relationships. This paper applies Google's PageRank algorithm to the graph model in order to evaluate commits' influences on the others. Through an empirical study with Git repositories of six open source projects, the following factors are showed to be noteworthy: (1) the number of added files at the commit, (2) the length of commit message, (3) the experience of committing author, and (4) the number of developers who have been involved in the modified files at the commit.**

I. Introduction

Open source software (OSS) products have been more and more popular. For instance, there are a lot of people using Linux, Firefox or Eclipse around the world. Recently, OSS products have also been actively leveraged in business scene. According to a survey report [1], 78% of respondent companies have utilized OSS products in their commercial products or customer environments. More and more companies tend to be positive toward leveraging OSS products.

Source programs of OSS products have been developed and maintained on their code repositories. Therefore, code repositories are significant information source for understanding status of developments and for evaluating and predicting quality of code. An analysis of repository is one of the hottest topics in the software engineering world, which is referred to as "mining software repositories (MSR) [2]." For example, Rahman et al. [3] proposed a bug-prediction algorithm based on change logs in a repository and Google released a tool [4] (an implementation of their algorithm) running on a Git repository. Zimmermann et al. [5] focused on co-change of source files on a repository (a logical coupling) and proposed to utilize the results of logical-coupling analysis in order to support successful code maintenances.

While many MSR studies have developed useful theories and methods, most studies focused on source files rather than commits. Since a commit is a unit of work at a repository, a study on commit itself seems to be important as well as a source file. If a commit caused bug fixes at successive commits, the former commit is an influential one which should have been checked more carefully. Misirli et al. [6] proposed to evaluate the impact of commit by using the amount of code churn, the number of modified files and the number of subsystems where the modified files span. This paper focuses on another perspective in regard to such causal relationships among commits—chains of relationships—and proposes a way of evaluating a commit's level of influence with using the PageRank algorithm [7].

The remainder of this paper is organized as follows: Section II defines causal relationships between commits focusing on bug fixes, and introduces an application of the PageRank algorithm to the commit influence evaluation. Then, Section III reports and discusses the results of empirical study with six popular OSS repositories. Finally, Section IV presents the conclusion of this paper and future work.

II. Evaluation of Commit's Influence

A. Bug Creation Commit and Bug fixing Commit

Suppose a part of a source file was changed at a commit, and the part was changed again for fixing a bug at another (successive) commit. In this paper, we call the former commit and the latter one as a "bug creation commit" and a "bug fixing commit," respectively.

Figure 1 presents a simple example of three commits (commit A, B and C) for a source file, where the aim of commit A is to change a part of the source file, and the aims of commit B and C are to fix bugs. At first, we focus on the relationship between commits A and B: the modified part for fixing a bug at commit B was changed at the preceding commit A. Thus, commit A is a bug creation commit and commit B is a bug fixing commit. Next, we look at the relationship between commits B and C: the fixed part at commit B is changed again at commit C for fixing a bug. That is to say, commit B is not only a bug fixing commit but also a bug creation commit.

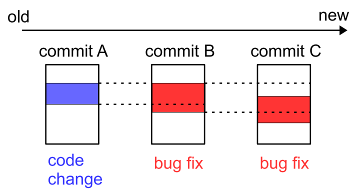

Fig. 1. A simple example of three commits for a source file.

B. Causal Relationship between Commits

In general, a bug fixing commit is caused by its bug creation commit. This is a causal relationship between commits. We will consider a directed graph model representing the causal relationships between commits, where each node is corresponding to each commit, and each causal relationship is expressed as an directed edge from the bug fixing commit to the bug creation commit: Figure 2 shows the model corresponding to the case of Fig. 1. In the case of Fig. 1, commit C does not seem to be independent of commit A because commit A caused commit B which caused commit C. Such a chain of causal relationships is also expressed as a path on the graph model (see Fig. 2). In other words, commit A has influences on not only commit B but also commit C.

Needless to say, our graph model can be more complex in the real repository because one commit may have more changes on more files and their causal relationships would be more complicated. Figure 3 presents another example. In Fig. 3, commit F is caused by commits D and E, and D and E are also caused by B. Thus, commit B seems to have a high level of influence on commits D, E and F. Moreover, commit B is caused by commit A, so A has also a high influence on other commits except for commit C. In order to make such an evaluation of commit automatically, Google's PageRank algorithm [7] can be used, with regarding a node and an edge as a Web page and a hyperlink, respectively.

C. PageRank Algorithm

The PageRank algorithm evaluates a Web page based on the following two simple concepts:

- A page liked by more pages has a higher value;
- A page liked by a high-valued page has also a high value.

The PageRank values of Web pages are computed as follows (see [7], [8] for the details): Let n be the number of Web pages; let H and A be $n \times n$ matrices, respectively; let $\boldsymbol{\pi}$ be n-dimensional column vector; let α be a scalar parameter in $[0, 1]$. H is the transition probability matrix whose (i, j) element signifies the probability of transition from the i-th page to the j-th page. A is an matrix for adjusting the impacts of dangling pages which have no hyperlink: if the i-th page is a dangling page, the i-th column of A is filled with $1/n$; otherwise, the i-th column is filled with zero. Then, the following Google matrix, G is defined:

$$G = \alpha H + \alpha A + (1 - \alpha)\frac{1}{n}\mathbf{1} \ ,$$

where $\mathbf{1}$ is $n \times n$ matrix whose all elements are 1.

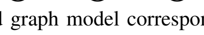
Fig. 2. Directed graph model corresponding to Fig. 1.

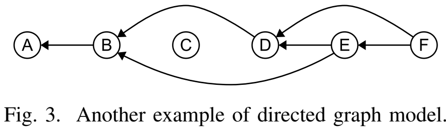
Fig. 3. Another example of directed graph model.

Intuitively, H expresses a page transition by clicking a hyperlink, A presents a restart of net-surfing since the dangling page has no hyperlink, and $\mathbf{1}$ correspond a totally-randomized net-surfing. The parameter α expresses a balance between a Web page-based surfing ($H + A$) and a random surfing ($\frac{1}{n}\mathbf{1}$).

The PageRank values are given as the elements of $\boldsymbol{\pi}$ which holds the following equation:

$$\boldsymbol{\pi}^T G = \boldsymbol{\pi}^T \ ,$$

where $\boldsymbol{\pi}^T$ is the transpose of $\boldsymbol{\pi}$. That is to say, the vector of PageRank values ($\boldsymbol{\pi}^T$) is the eigen vector of the Google matrix G; α is empirically given as 0.85 according to the literature [7]. The above eigen vector problem can be solved by a simple power iteration method [7], [8].

We will apply the above algorithm to evaluate the influence of commit.

III. Empirical Study

A. Aim and Dataset

The aim of this study is to apply the PageRank algorithm to actual commits in OSS repositories and to analyze the features of influential commits.

We examined repositories of six OSS projects from various domains: DrJava[1], Hibernate[2], JabRef[3], eXo Platform[4], PMD[5] and SQuirreL SQL Client[6]. We selected them because 1) their source files are maintained with Git, 2) they are developed in Java, and 3) they have been maintained over a long period.

Reason 1) is for ease of data processing since Git allows making a local copy of repository and provides powerful functionality for analyzing code changes. Reason 2) is from our data analysis tools. Reason 3) is for enhancing our reliability of data; if we examine a younger project, we can collect only a short history and small-sized graph of causal relationships. Then, our empirical results would not be useful for discussing general tendencies.

B. Procedure

We conduct our empirical study in the following procedure for each OSS repository.

1) Collect features of each commit using 12 metrics shown in Table I. We used these metrics because they were easy to automatically collect from a commit log.
2) Determine if each commit is a bug fixing commit or not: check whether the commit log contains a bug-fix-related keyword or not [9].
3) Determine if each commit is a bug creation commit or not: for each line modified at each bug fixing commit, detect the last commit which made a modification at the line by using the git blame command. Notice that comments and blank lines are omitted in the detection.

[1]http://www.drjava.org/
[2]http://hibernate.org/
[3]http://www.jabref.org/
[4]https://www.exoplatform.com/
[5]https://pmd.github.io/
[6]http://squirrel-sql.sourceforge.net/

TABLE I
COMMIT METRICS

metric	description
IM	whether the aim of commit is a modification
ML	commit message length (character count)
FA	# of added files
FD	# of deleted files
FM	# of modified files
LA	lines of added code
LD	lines of deleted code
CAP	# of commits by the commit author (ca) in the past
FAP	# of files committed by ca in the past
FMN	# of modified files which are new for ca
FMP	# of modified files which had been done by ca in the past
AIF	# of unique authors who have been involved in committed files

4) Compute each commit's influence (PageRank value) using the PageRank algorithm with the graph model.
5) Analyze impacts of metrics on commits' influences through a multiple linear regression analysis whose dependent variable is the commit's influence and independent variables are the above metrics: for a commit c,

$$y(c) = a_0 + \sum_{i=1}^{12} a_i x_i(c) \ ,$$

where $y(c)$ is the influence of c and x_i is the i-th metric values of c (for $i = 1, \dots, 12$); a_j is a constant (partial regression coefficient) (for $j = 0, 1, \dots, 12$).
In order to avoid the risk of misjudging bug creation commits, the regression analysis is performed using only the older 50% of commits because a newer non-causal commit might become a bug creation commit after a new bug fixing commit occurs in the future.
6) Examine the prediction power of the above linear regression model through the leave-one-out cross-validation.
At first, let a "highly-influencing commit" to be a bug creation commit having a high influence (PageRank value) which is ranked in the top 20% of them; let n be the number of highly-influencing commits. Next, randomly select n non-highly-influencing commits from the repository. That is to say, we have $2n$ sample commits in which the half of them are highly-influencing commits and the remaining half are non-highly-influencing ones. If we use all commits, the rate of non-highly-influencing commits is 80%, and such an imbalance may affect the regression model. This is the reason why we single out non-highly-influencing commits as many as the highly-influencing ones.
Then, iterate the following two steps for each sample commit, c_i (for $i = 1, \dots, 2n$):
a) Using data of $2n - 1$ commits except for c_i, build the multiple linear regression model, m_i.
b) Predict if c_i is highly-influencing commit or not by m_i.
Finally, compute the rate of correctly predicted commits.

C. Results

1) Regression Analysis:

Table II presents the results of regression analysis: standardized partial regression coefficients and their statistical significances. As the results, the following four metrics showed tendencies common to the majority of products.

1) ML (commit message length): In five out of six projects, ML values have positive impacts on influences of those commits: a commit having a longer commit message is more likely to be more influencing commit. It would be natural that developers tend to record more comments on more important commits at which they made notable changes.
2) FA (number of added files): FA values also have positive impacts on commits' influences in five out of six projects. A commit adding more new files would be worthwhile in regard to the causal relationships in this study. In general, a new source file tends to be more fault-prone, thus the commit has a higher risk of being a bug creation commit. Moreover, those new files might be renamed ones at the commit. While renamed files themselves do not seem to be fault-prone, the renaming possibly causes faults in other source files.
3) CAP (number of commits by the commit author in the past): In five out of six projects, CAP values show negative impacts on influences of those commits. A higher CAP value means that the commit was made by an author who has more experiences in the project. That is to say, a commit made by experienced author tends to have a lower risk of being a bug creation commit.
4) AIF (number of unique authors who have been involved in committed file): For all projects, AIF values have positive impacts on commits' influences. A higher AIF value is obtained when more source files are committed at the time and moreover those files have been developed or maintained by various developers. That is to say, if an author touched more files made by others, the commit is riskier of being a bug creation one.

From the above results, we can say the following three findings: (1) when we have to add new source files or need to record a long, detailed commit message, we should be more careful of the quality of their source code and modifications; (2) the author's experience is also a notable feature of commit, so it is better to review commits made by less-experienced authors; (3) we should more carefully modify programs if they have been developed and maintenance by others.

Since the standardized partial regression coefficients of AIF are positive and relatively-large for all projects, AIF seems to be the most impactive metric in this study, so the third finding has the highest priority in the above three points.

2) Cross Validation:

In order to examine the above regression models, we performed the leave-one-out cross-validation. Table III shows the rate of correctly predicted commits. As the results, the accuracies of commit predictions by the regression models are about 62 – 84%. While not all projects showed so high-level accuracies, they would be useful; Because the rate of highly-influencing commits vs. the rate of non-highly-influencing

TABLE II
RESULTS OF REGRESSION ANALYSIS: STANDARDIZED PARTIAL REGRESSION COEFFICIENTS.

project	IM	ML	FA	FD	FM	LA	LD	CAP	FAP	FMN	FMP	AIF
DrJava	0.034	0.214^*	0.394^*	0.039	0.419^*	-0.012	0.044	0.005	-0.131^*	-0.267^*	-0.057	0.202^*
Hibernate	0.046^*	0.099^*	2.190^*	-1.701^*	0.197	-1.039^*	0.893^*	-0.122^*	0.125^*	-0.190	-0.065	0.117^*
JabRef	0.008	0.068^*	0.437^*	-0.505^*	-2.500	-0.417^*	0.446^*	-0.013	-0.031	0.236	0.502	0.185^*
eXo	0.018	-0.002	0.992^*	0.345^*	0.379^*	-0.119^*	0.300^*	-0.036^*	0.034^*	-0.896^*	-0.253^*	0.188^*
PMD	0.048^*	0.151^*	0.211^*	-0.116	-0.408	0.105^*	-0.029	-0.141	0.138	0.059	0.486	0.242^*
SQuirreL	-0.010	0.108^*	-11.10^*	-0.989^*	-7.366^*	0.889^*	-0.243^*	-0.105^*	0.189^*	10.78^*	7.244^*	0.158^*

(* : p-value < 0.05)

TABLE III
ACCURACIES OF PREDICTIVE DISCRIMINATION.

projects	accuracy
DrJava	64.3%
Hibernate	65.2%
JabRef	81.1%
eXo Platform	84.1%
PMD	71.0%
SQuirreL SQL Client	62.4%

commits in our cross validation dataset is $50 : 50$, the expected accuracy is 50% if the model is not useful.

D. Threats to Validity

Since the causal relationships between commits are based on “bug fixings,” the reliability of our results depends on the detection algorithm of bug fixing commits. We adopted a keyword-matching algorithm proposed by Śliwerski et al. [9]. While their algorithm is simple, we believe it can work appropriately since it has been widely used in the MSR community in the past.

In our empirical study using a regression analysis, we miss detailed analysis of relationships among metrics and their distributions: we have just tried to capture impacts of metrics using a simple linear regression model. Toward a more practical application of our approach to the defect prediction such as Just-In-Time defect prediction [10], we need to perform a more sophisticated analysis of commit features in the future.

If we encounter a project which has a few or no bug fixes, our graph model has an extremely loose coupled structure. Then, the regression model might have a poor power in evaluating commits. Since we examined only OSS projects which are popular and well-maintained for long periods, such a threat is mitigated in this empirical study.

While we analyzed data of commits in different repositories, all of them are from OSS projects and not from commercial projects. Commercial projects might have own rules for checking code changes before their commits, and our results may not be suitable to such projects. Further analyses of project's organization and rules are our important future work.

IV. CONCLUSION AND FUTURE WORK

We focused on bug fixing-based causal relationships among commits and proposed to evaluate commits' influences on their successive commits. Since there can be chains of causal relationships among commits, we applied Google's PageRank algorithm by replacing a Web page and a hyperlink with a commit and a causal relationship, respectively.

We conducted an empirical study with six popular OSS repositories, and analyzed features of commits causing bug fixes in their successive commits through a multiple linear regression analysis. As the results, we obtained the following commits are riskier of being bug creation commits:

1) a commit adding more new source files or having a long, detailed commit message;
2) a commit made by a less-experienced author;
3) a commit modifying source files which have been developed and maintenance by other developers in the past.

Moreover, the leave-one-out cross-validation showed that the regression models can predictively discriminate highly-influential commits with 62 – 84% accuracy.

Since our focus in this paper was on a feature analysis of commit through an application of PageRank algorithm and regression analysis, an empirical study to utilize our findings for Just-In-Time defect prediction is our significant future work. Furthermore, we plan to perform another analysis focusing on the ownership of source files [11] as well.

ACKNOWLEDGMENT

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

REFERENCES

[1] Black Duck Software, “The ninth annual future of open source survey,” https://www.blackducksoftware.jp/future-of-open-source, Apr. 2015.
[2] A. E. Hassan, “The road ahead for mining software repositories,” in *Proc. Frontiers of Softw. Maintenance*, 2008, pp. 48–57.
[3] F. Rahman, D. Posnett, A. Hindle, E. Barr, and P. Devanbu, “Bugcache for inspections: Hit or miss?” in *Proc. 19th ACM SIGSOFT Symp. and 13th European Conf. Foundations of Softw. Eng.*, 2011, pp. 322–331.
[4] C. Lewis and R. Ou, “Bug prediction at google,” http://google-engtools.blogspot.jp/2011/12/bug-prediction-at-google.html, 2011.
[5] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, “Mining version histories to guide software changes,” *IEEE Trans. Softw. Eng.*, vol. 31, no. 6, pp. 429–445, June 2005.
[6] A.T. Misirli, E. Shihab, and Y. Kamei, “Studying high impact fix-inducing changes,” *Empirical Softw. Eng.*, vol. 21, no. 2, pp. 605–641, Apr. 2016.
[7] A. N. Langville and C. D. Meyer, *Google's PageRank and Beyond*. Princeton, NJ: Princeton University Press, 2006.
[8] D. Austin, “How google finds your needle in the web's haystack,” http://www.ams.org/samplings/feature-column/fcarc-pagerank, Dec. 2006.
[9] J. Śliwerski, T. Zimmermann, and A. Zeller, “When do changes induce fixes?” in *Proc. Int'l Workshop Mining Softw. Repositories*, May 2005, pp. 1–5.
[10] T. Fukushima, Y. Kamei, S. McIntosh, K. Yamashita, and N. Ubayashi, “An empirical study of just-in-time defect prediction using cross-project models,” in *Proc. 11th Working Conf. Mining Softw. Repositories*, May 2014, pp. 172–181.
[11] P. D. Daryl Posnett, Rassa D'Souza and U. Filkow University of California Davis, “Dual ecological measure of focus in software development,” in *Proc. 35th Int'l Conf. Softw. Eng.*, May 2013, pp. 452–461.