Short Paper

The Developer's Dilemma: Factors affecting the Decision to Repay Code Debt

Theodoros Amanatidis[1], Nikolaos Mittas[2], Alexander Chatzigeorgiou[1], Apostolos Ampatzoglou[1], Lefteris Angelis[2]
[1]Department of Applied Informatics, University of Macedonia, Greece
[2]Department of Computer Science, Aristotle University of Thessaloniki, Greece
tamanatidis@uom.edu.gr, nmittas@csd.auth.gr, achat@uom.gr, apostolos.ampatzoglou@gmail.com, lef@csd.auth.gr

ABSTRACT

The set of concepts collectively known as Technical Debt assume that software liabilities set up a context that can make a future change more costly or impossible; and therefore repaying the debt should be pursued. However, software developers often disagree or even dislike an automatically generated list of improvement suggestions, which they consider not fitting or important for their own code. To shed light into the reasons that drive developers to adopt or reject refactoring opportunities (i.e., technical debt repayment), we have performed an empirical study on the potential factors that affect the developers' decision to agree with the removal of a specific Technical Debt liability. The study has been addressed to the developers of four well-known open-source applications. To increase the response rate, a personalized assessment has first been sent to each developer, summarizing his/her own contribution to the technical debt of the corresponding project. Responds have been collected through a custom built web application that presented code fragments suffering from violations as identified by SonarQube along with information that could possibly affect their level of agreement to the importance of resolving an issue. These factors include data such as the frequency of past changes in the module under study, the number of bugs, the type and intensity of the violation, the level of involvement of the developer and whether he/she is a contributor in the corresponding project. Multivariate statistical analysis methods have been used to understand the importance and the underlying relationships among these factors and the results are expected to be useful for researchers and practitioners in Technical Debt Management.

CCS CONCEPTS

Software and its engineering → Software creation and management → *Software post-development issues***:** Maintaining software

KEYWORDS

Technical Debt Management, refactoring, empirical study

1 INTRODUCTION

According to A. Hunt and D. Thomas many developers are reluctant to start 'ripping up' their code (a.k.a. refactor) just because it isn't quite right [1]. As they vividly put it, going to a boss or client and saying that a working piece of code needs another week to refactor it, would probably cause a response that cannot be printed. However, deferring a refactoring might incur technical debt (TD) requiring greater time investment to fix the problem down the road.

Previous studies have shown that developers perceive and handle TD in different ways [2] and have distinct motivations for applying refactorings [3]. To shed light into the factors that drive developers to accept or reject automated suggestions for TD removal we have carried out a study targeting developers of open-source PHP projects with the following two main characteristics: (a) to increase their motivation for participating in the study we have provided to each participant, prior to requesting his feedback, a personalized report on the TD that he/she has incurred to the project, and (b) to facilitate the collection of data a web application has been implemented to present individual code fragments suffering from an identified TD issue along with information on the parameters that might affect the developers decision to repay the TD or not.

2 RELATED WORK

There is a limited number of studies providing insights on the developer's perception regarding the urgency to resolve code violations, which in turn lead to the accumulation of TD. In a recent study [4], the authors sent surveys to explore whether issues involving architectural elements lie among the most significant sources of TD. A number of 536 respondents replied (leading to a response rate of 29%) and the results showed that architectural issues are the greatest source of TD. Such issues are difficult to cope with and they dragged on for many years. Another explored subject was the existence of effective tools for managing TD. Respondents claimed that existing tools do not capture the key areas of accumulated problems related to TD.

In a 2014 study [5], the authors investigated which bad smells are considered by the developers as the most harmful. The developers were given code snippets from three systems with twelve kinds of bad smells and were asked to rank the severity of the smells. Both original developers from the systems and outsiders (industrial developers) were included in the survey (a response rate of 40% was achieved). The results suggested that smells related to complex code are considered an important threat by developers. In another exploratory study [6], comments of four large open-source systems were used to identify self-admitted TD. The authors found that more experienced developers introduce most of

the self-admitted TD while time pressure and code complexity do not relate to the amount of self-admitted debt.

In another study in 2013 [7], 20 developers were interviewed in order to investigate the reasons why static analysis tools are not used during development process. Participants claimed that static analysis tools are beneficial, but false positives, poorly presented output and low customizability deter their use. Spinola et al. [8] chose 14 statements regarding TD and asked from 37 practitioners if they agree with them. The statement that says "Not all technical debt is bad" lies among the statements with the maximum consensus. In other words developers believe that there is a healthy level of TD in every software system.

Kim et al. [9] conducted a survey to examine developers' perception regarding code refactoring. Participants responded that refactoring hides substantial cost and risks and further support is needed beyond automated refactoring within IDEs. However, a case study on Windows 7 highlighted the benefits of refactoring. The results showed that "*refactored modules experienced higher reduction in the number of inter-module dependencies and post-release defects than other changed modules*".

In another study which debated developers' position about code refactoring [10], 20 refactoring practitioners were interviewed. Participants recognized the added value of a refactoring (code reusability), however if too much effort is needed they may be reluctant to make refactoring decisions. Mäntylä and Lassenius [11] studied the refactoring decisions made by 37 students on a small Java application. According to participants' responses, 'Long method' was the top driver for refactoring decision and poor readability along with poor understanding of the code were also among the most important drivers.

3 STUDY DESIGN

The purpose of the current study is to shed light on the factors that drive developers to resolve TD Items (TDIs) identified in their own code. To achieve this, four PHP open source projects on GitHub were analyzed to obtain commit activity and code debt information. Specifically, Composer, CakePHP, Laravel[1] and Yii2 were included. Composer (composer/composer) is a dependecy manager for PHP, CakePHP (cakephp / cakephp) is a framework to build PHP applications and Laravel (laravel / framework) and Yii2 (yiisoft / yii2) are also well known PHP frameworks. The criteria for selecting the aforementioned projects are as follows:

- Projects had to be open source so as to have direct access to their code base.
- Projects had to be actively maintained up until the time of this study.
- Projects had to be widely used by the PHP community: Composer has 5 millions downloads, CakePHP has 2 millions, Laravel has 6 millions and Yii2 has 1 million.
- Projects had to be maintained by many contributors: Composer has 600+ contributors, CakePHP has 500, Laravel has 400 and Yii2 has 800.
- Projects had to be widely recognized by the PHP community: Composer has 11k stars on GitHub, CakePHP has 7k, Laravel has 35k and Yii2 has 11k.

The history of the commit activity of the projects was retrieved via the GitHub API and their code base was analyzed by SonarQube [2] in order to measure their TD at every commit snapshot. It should be noted that the commit history includes the last year's commit data of the projects for two main reasons: The aim of the study is to track the most recent developers' activity in order to ask currently active developers to evaluate the importance of the code violations that SonarQube detected. For example, it would not be reasonable to approach a developer that pushed some commits two or three years ago without any recent activity, since he may be currently inactive. The second reason is that the analysis process with SonarQube is costly in terms of time and resources, especially in cases when the TD is measured for every single commit of the project.

As in any similar study, the major challenge was to retrieve sufficient responses as previous experience has shown that people outside the academic community are not always willing to spare time to contribute to academic studies. To increase the likelihood of obtaining a response, the developers have been approached in a way that could potentially attract their interest, as described next.

3.1 Providing a personalized report to participants

Prior to the request for participating in the evaluation of TD items (even just a single one), developers have been provided with a personalized report of their current activity including their commit density, contribution to the overall TD of the project (relatively to the rest of the developers) and the top-five code violations they insert into the code. The report for a random developer (with anonymized information) for project Yii2 is shown in Figure 1. The obtained response rate in our study was 35%.

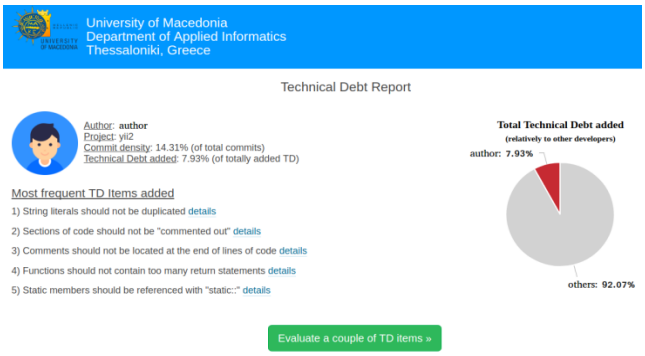

Figure 1: Anonymized TD report of a developer in Yii2.

3.2 Set-up of the study

At the end of the report each developer was asked to evaluate TD items detected in the project under study. In the evaluation screen the developer was presented with a code violation (assessed TD

[1] Laravel core (laravel/framework)

[2] https://www.sonarqube.org/

item) as detected by SonarQube along with some information regarding the violation itself and the file in which the violation was found, so as to provide a spherical view of the TD item before answering. In particular, the evaluator was given the following information regarding the violation (see Figure 2):

- Short description of the TD item
- Suggested solution of the TD item
- Tag categorization (serving as keywords of the TD item)
- Severity of the TD item
- Estimated time to fix the TD item
- The name of the file in which the TD was detected
- The revision of the file
- The code snippet where the TD item was detected
- [3]The change frequency of the file (as percentage)
- [4]The issue fixing frequency of the file (as percentage)
- [5]The total technical debt of the file (as percentage)

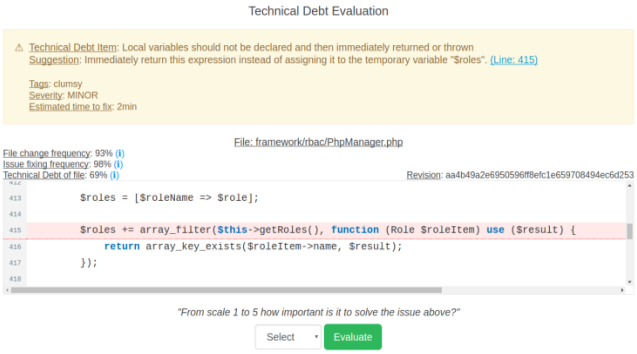

Figure 2: Evaluation screen for a TD item in project Yii2.

At the bottom of the screen the developers were asked to evaluate the urgency of the TD item to be fixed in a Likert scale (from 1 to 5), with 1 meaning "*no need to solve it*" and 5 corresponding to "*it is urgent to solve it*". The developers' response to this question served as the dependent variable in the statistical analysis.

4 RESULTS AND DISCUSSION

4.1 Statistical Analysis

In this subsection we present the descriptive statistics on the involved variables and inferential statistics regarding the relationship between the factors that have been considered (explanatory variables) and the agreement of a developer on the resolution of a TDI (dependent variable).

Table 1 summarizes the distributions for the categorical variables of the study, whereas Table 2 provides the univariate descriptive statistics of the continuous variables, in which results were expressed as mean (M), standard deviation (SD), median (Mdn), minimum (min) and maximum (max). (Developer Participation indicates whether the participant was a contributor to the project in which the TD item was found, or not).

Table 1: Frequency distributions for categorical variables

		N	%
Developer Evaluation (Dependent)	Very low	74	27.2
	Low	38	14
	Moderate	60	22.1
	High	49	18
	Very high	51	18.8
Severity	Info	15	5.5
	Minor	58	21.3
	Major	189	69.5
	Critical	10	3.7
Debt characterization	Changeability	14	5.3
	Maintainability	157	59.2
	Reliability	70	26.4
	Security	5	1.9
	Testability	19	7.2
	Missing	7	
Developer Participation	No	105	38.6
	Yes	167	61.5

Table 2: Descriptive statistics for continuous variables

	N	*M*	*SD*	*min*	*max*
Time to fix (in min)	272	10.71	14.28	1	60
TD file (in min)	272	274.63	441.30	2	2828
File modifications ranking	272	84.97	17.47	6	100
File corrections ranking	272	36.18	40.91	0	100

In order to examine the relationship between explanatory variables and outcome responses (Developer Evaluation), the *Generalized Estimation Equations* (GEE) approach is adopted. GEE introduced by Liand and Zeger [12] can be considered as the extension of the Generalized Linear Model, suitable for taking into account the dependence among observations. As in our survey eighteen developers provided their evaluations, each one for one up to eighty-three TD items, there is an imperative need to handle the inherent dependence (or "developer effect"), stemming from the evaluations of the same developers to TD items.

Describing briefly, consider a random sample of observations from n subjects (responses on TD items). Let $\boldsymbol{O}_i^T = (O_{i1}, \dots, O_{in_i})^T$ be the column vector of ordinal responses provided by subject $i = \{1, \dots s\}$ where O_{ir} takes values in $\{1, \dots, C\}$. Also let $\boldsymbol{X}_i = (\boldsymbol{X}_{i1}, \dots, \boldsymbol{X}_{in_i})^T$ be a $n_i \times p$ dimensional matrix of repeated p covariates for subject i. Then, the model describing the correlation between the set of covariates and the conditional probabilities of each ordinal response is given by

$$l[P(O_{ir} \le c | \boldsymbol{X}_{ir} = \boldsymbol{x}_{ir})] = \beta_{0c} + \boldsymbol{x}_{ir}\boldsymbol{\beta}_1^T \qquad (1)$$

For $c = 1, \dots, C-1$, β_{0c} the threshold parameter for level c, $\boldsymbol{\beta}_1$ the row vector of regression coefficients corresponding to covariates and with l we denote a known link function (*logit* function in our case). The selection of the explanatory variables was based on a backward elimination.

The backward elimination procedure indicated that the covariates *File Modifications Ranking*, $\chi^2(1) = 0.030$, $p = 0.863$, *Time to Fix (in minutes)*, $\chi^2(1) = 0.512$, $p = 0.474$ and *TD Files (in minutes)*,

[3] This indicates how often the file gets modified, relatively to other files. A percentage of 100% means that the file is the most frequently modified file.
[4] This indicates how often the file gets modified for issue fixing, relatively to other files. A percentage of 100% means that the file produces the most issues.
[5] This indicates the technical debt of the file, relatively to other files. A percentage of 100% means that the file has the highest technical debt.

$\chi^2(1) = 1.482$, $p = 0.223$ do not present a statistically significant main effect on responses and for this reason they were dropped out from any further analyses. The final model, after omitting insignificant predictors, indicated that *Severity*, $\chi^2(3) = 15.625$, $p = 0.001$, *Debt Characterization*, $\chi^2(4) = 12.669$, $p = 0.013$, *Developer Participation (Binary)*, $\chi^2(1) = 6.625$, $p = 0.009$ and *File Corrections Ranking*, $\chi^2(1) = 3.418$, $p = 0.064$ presented statistically significant main effects on the developer evaluation for TD items. The parameters of the final model are presented in Table 3, in which the reference categories for factors *Severity*, *Debt Characterization* and *Developer Participation* are "Critical", "Maintainability" and "Yes", respectively. Interpreting the parameter estimates of the model for the factor Severity, the coefficient for the level Info (b = -3.070, SE = 1.374) indicates that the ordered logit for Info TD items, being into a higher evaluation response is -3.070 ($\chi^2(1) = 4.990$, $p = 0.025$) less than the reference category (Critical TD items). In other words, the odds for a Critical TD item to be evaluated into a higher category are 21.5 ($1/e^{-3.070}$) times higher compared to an Info TD item. In addition, the model reveals a statistically significant difference between the odds ratio (OR) of Minor and Critical Severity, $\chi^2(1) = 7.407$, $p = 0.006$. Regarding Debt Characterization, the findings suggest that Testability debt (b = 1.363, SE = 0.539) is 3.9 times more likely to be evaluated into higher categories compared to Maintainability debt, $\chi^2(1) = 6.391$, $p = 0.011$. In addition, the parameter of the binary predictor Developer Participation, (b = 1.120, SE = 0.430) indicates that TD items presented to developers that have not participated in the project under study at all are almost 3 times more likely to be evaluated to higher categories compared to TDIs presented to developers who contributed to the project. Finally, the coefficient for the covariate File Corrections Ranking, (b = 0.007, SE = 0.004) indicates a marginally significant positive correlation between File Corrections Ranking and Developer Evaluation, $\chi^2(1)=3.418$, p=0.06.

4.2 Discussion of the results

The distribution of developer responses to the question on whether they agree with the need to resolve a particular TD item are rather uniform, as in 41% of the violations their level of agreement was 'very low' or 'low', in 22% of the cases their level of agreement was 'moderate', while in 36% of the cases they agreed on the need to apply a refactoring for resolving an issue (level of agreement was 'high' or 'very high').

According to results of the *Generalized Estimation Equations* approach developers appear to be largely influenced by the severity of a TD issue (i.e. Critical, Major, Minor and Info as no Blocking issues were identified). For example, it is 21.5 times more probable that a Critical issue will be classified as needing resolution compared to an Info issue. This finding is reasonable, as the categorization of severity by SonarQube already distinguishes between issues. In other words, it is reasonable that a Critical code issue like "*String literals should not be duplicated*" is perceived as more urgent to be resolved than an Info code issue like "*Comments should not be located at the end of lines of code*".

Table 3: Parameters of the final model

Parameter		*b*	*SE*	Hypothesis Test			*OR*	95% OR	
				χ^2	*df*	*p*		*Lower*	*Upper*
Threshold[6]	*Very low*	-2.103	1.154	3.318	1	0.069	0.122	0.013	1.173
	Low	-1.323	1.126	1.381	1	0.240	0.266	0.029	2.419
	Moderate	-0.223	1.044	0.046	1	0.831	0.800	0.103	6.191
	High	0.861	1.053	0.669	1	0.413	2.366	0.301	18.615
Severity: *Info*		-3.070	1.374	4.990	1	0.025	0.046	0.003	0.686
Severity: *Minor*		-2.984	1.096	7.407	1	0.006	0.051	0.006	0.434
Severity: *Major*		-1.409	0.963	2.144	1	0.143	0.244	0.037	1.612
DebtCharacterization: *Changeability*		0.481	0.290	2.742	1	0.098	1.617	0.915	2.857
DebtCharacterization: *Testability*		1.363	0.539	6.391	1	0.011	3.908	1.358	11.241
DebtCharacterization: *Security*		-0.653	1.047	0.389	1	0.533	0.520	0.067	4.049
DebtCharacterization: *Reliability*		0.143	0.266	0.288	1	0.591	1.153	0.685	1.942
Developer Participation: *No*		1.120	0.430	6.783	1	0.009	3.066	1.319	7.123
File Corrections Ranking		0.007	0.004	3.418	1	0.064	1.007	1.000	1.014
Notes: Reference categories Severity: *Critical*, Debt Characterization: *Maintainability*, Developer Participation: *Yes*									

[6] In this type of models threshold parameters that define transition points between adjacent categories are estimated for C-1 levels

The broader characterization of the TD issue also seems to have an effect on the developer's decision. For example, if an issue pertains to Testability (like "*Expressions should not be too complex*") it is 3.9 times more probable to be considered as needing resolution than an issue related to Maintainability (like "*Sections of code should not be "commented out*"). Considering that the scanner employed for identifying rule violations in PHP code relied on static analysis, it is reasonable that issues related to Testability, Changeability and Maintainability are considered as more 'real' compared to security/reliability issues which in order to be accurate require further validation by run-time analysis.

Finally, developers do not tend to accept suggestions for revising their own code: it is 3 times more likely that a developer who has not participated in a project agrees with a suggestion to remove a TD issue, than a developer who is a contributor. This might be related to the particular practices within the community of a software project where certain violations are not considered as harmful because the evolution of the project might have been unaffected by their presence.

On the other hand, developers' decisions appear to be unaffected by factors such as the frequency of modifications to the file under study (reflected in the Files Modifications Ranking variable), the time required to fix an issue and the total TD in the examined file. The last two findings could be related to a latent belief that automated quality analysis tend to overestimate the magnitude of problems and thus these factors might be subconsciously overlooked. The frequency by which a file undergoes modification, under normal circumstances, should be driving factor; for example, for a file that has never been the subject of maintenance there is probably limited urgency to resolve its TD issues. However, it appears that developers tend to focus on the problem per se, rather than the surrounding context. Of course, a relevant threat is related to whether the respondents really understood the concept of the presented variables. These findings can be valuable to researchers and practitioners by guiding the design of more efficient tools that suggest refactorings with a higher probability of being adopted by the developers.

5 THREATS TO VALIDITY

In this section we briefly list major threats to the validity of the present study. With regard to statistical conclusion validity we should stress that the small sample size unavoidably affects the conclusions regarding the extent of the observed relationships between the explanatory and output variables. Further investigation by collecting a larger set of responses is required to increase our confidence in the identified relationships. With regard to the construct validity of the study we should acknowledge that despite our efforts to facilitate the work of the study participants by offering an easy-to-use web application, it is not certain that they have correctly interpreted the presented pieces of information around the examined code fragment and TD issues. Finally, the conclusions should be cautiously generalized to other projects, languages, development models as this kind of studies are subject to external validity threats.

6 CONCLUSIONS

Existing software quality tools can yield extremely long lists of refactoring suggestions, deterring developers from adopting them. Thus, there is a need to determine which refactoring opportunities make sense for the developers depending on their background, nature and importance of the problem, surrounding code context, etc. In this paper we present results from an ongoing study on various factors that potentially drive open-source software developers to accept or reject a suggestion to resolve a TD item.

ACKNOWLEDGMENTS

Work reported in this paper has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 780572 (project SDK4ED).

REFERENCES

[1] A. Hunt and D. Thomas, *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional, 1999.

[2] T. Amanatidis, A. Chatzigeorgiou, A. Ampatzoglou, and I. Stamelos, "Who is Producing More Technical Debt?: A Personalized Assessment of TD Principal," *Proceedings of the XP2017 Scientific Workshops*, USA, 2017, p. 4:1–4:8.

[3] D. Silva, N. Tsantalis, and M. T. Valente, "Why We Refactor? Confessions of GitHub Contributors," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, New York, NY, USA, 2016, pp. 858–870.

[4] N. A. Ernst, S. Bellomo, I. Ozkaya, R. L. Nord, and I. Gorton, "Measure It? Manage It? Ignore It? Software Practitioners and Technical Debt," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, USA, 2015, pp. 50–60.

[5] F. Palomba, G. Bavota, M. D. Penta, R. Oliveto, and A. D. Lucia, "Do They Really Smell Bad? A Study on Developers' Perception of Bad Code Smells," in *2014 IEEE International Conference on Software Maintenance and Evolution*, 2014, pp. 101–110.

[6] A. Potdar and E. Shihab, "An Exploratory Study on Self-Admitted Technical Debt," in *2014 IEEE International Conference on Software Maintenance and Evolution*, 2014, pp. 91–100.

[7] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?," in *35th International Conference on Software Engineering (ICSE)*, 2013, pp. 672–681.

[8] R. O. Spínola, A. Vetrò, N. Zazworka, C. Seaman, and F. Shull, "Investigating technical debt folklore: Shedding some light on technical debt opinion," in *4th International Workshop on Managing Technical Debt (MTD)*, 2013, pp. 1–7.

[9] M. Kim, T. Zimmermann, and N. Nagappan, "A Field Study of Refactoring Challenges and Benefits," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, New York, NY, USA, 2012, p. 50:1–50:11.

[10] Y. Wang, "What motivate software engineers to refactor source code? evidences from professional developers," *IEEE International Conference on Software Maintenance*, 2009, pp. 413–416.

[11] M. V. Mäntylä and C. Lassenius, "Drivers for Software Refactoring Decisions," in *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*, New York, NY, USA, 2006, pp. 297–306.

[12] K.-Y. Liang and S. L. Zeger, "Longitudinal data analysis using generalized linear models," *Biometrika*, vol. 73, no. 1, pp. 13–22, Apr. 1986.