Information and Software Technology 000 (2017) 1–5

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

The relation between technical debt and corrective maintenance in PHP web applications

Theodoros Amanatidis[a], Alexander Chatzigeorgiou[a,*], Apostolos Ampatzoglou[b]

[a] Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
[b] Department of Mathematics and Computer Science, University of Groningen, Groningen, The Netherlands

ARTICLE INFO

Article history:
Received 3 October 2016
Revised 11 May 2017
Accepted 13 May 2017
Available online xxx

Keywords:
Technical debt
Corrective maintenance
Interest
PHP
Case study
Empirical evidence

ABSTRACT

Context: Technical Debt Management (TDM) refers to activities that are performed to prevent the accumulation of Technical Debt (TD) in software. The state-of-research on TDM lacks empirical evidence on the relationship between the amount of TD in a software module and the interest that it accumulates. Considering the fact that in the last years, a large portion of software applications are deployed in the web, we focus this study on PHP applications.
Objective: Although the relation between debt amount and interest is well-defined in traditional economics (i.e., interest is proportional to the amount of debt), this relation has not yet been explored in the context of TD. To this end, the aim of this study is to investigate the relation between the amount of TD and the interest that has to be paid during corrective maintenance.
Method: To explore this relation, we performed a case study on 10 open source PHP projects. The obtained data have been analyzed to assess the relation between the amount of TD and two aspects of interest: (a) corrective maintenance (i.e., bug fixing) frequency, which translates to *interest probability* and (b) corrective maintenance effort which is related to *interest amount*.
Results: Both interest probability and interest amount are positively related with the amount of TD accumulated in a specific module. Moreover, the amount of TD is able to discriminate modules that are in need of heavy corrective maintenance.
Conclusions: The results of the study confirm the cornerstone of TD research, which suggests that modules with a higher level of incurred TD, are costlier in maintenance activities. In particular, such modules prove to be more defect-prone and consequently require more (corrective) maintenance effort.

1. Introduction

In recent years, Technical Debt Management (TDM) has become a popular research field in software engineering. The majority of TDM approaches are based on the two pillars of Technical Debt (TD) quantification, namely ***principal*** (i.e., the effort needed to refactor the system in order to address existing inefficiencies) and ***interest*** (i.e., the additional effort needed in performing maintenance, due to the existence of the principal). According to Alves et al. [1], interest can be perceived as a risk for software development, and therefore its quantification should be assessed based on two components: ***interest probability*** (i.e., how possible is that one module that holds TD will need maintenance) and ***interest amount*** (i.e., the amount of additional effort). According to Ampatzoglou et al. [2] interest is incurred while performing two types of maintenance activities: (a) bug-fixing (namely ***corrective maintenance***), and (b) adding new features (namely ***perfective maintenance***).

In the literature, one can identify several studies that have investigated the relation between low levels of design-time qualities (e.g., coupling, bad smells, etc.) that constitute proxies of modules' ***TD amount***—i.e., principal plus interest—and the maintenance intensity on these modules [3–9][1]:

- All studies agree that the more flaws a file is involved in, the higher the likelihood to undergo defect-related changes.
- MacCormack and Sturtevant have found evidence on 2 industrial projects that source files with higher levels of coupling are associated with more extensive corrective maintenance [3].
- Feng et al. [4] and Nord et al. [5] have found evidence that files participating in architectural flaws (especially in unstable interfaces) are highly correlated with bugs and changes.

* Corresponding author.
E-mail addresses: tamanatidis@uom.edu.gr (T. Amanatidis), achat@uom.gr (A. Chatzigeorgiou), apostolos.ampatzoglou@gmail.com (A. Ampatzoglou).

[1] Due to space limitations, a separate related work section has been omitted. The state-of-the-research on the subject is thoroughly presented in a recent secondary study [1].

http://dx.doi.org/10.1016/j.infsof.2017.05.004
0950-5849/© 2017 Elsevier B.V. All rights reserved.

- In an earlier study in 2013 [6], Zazworka et al. suggested that dispersed coupling, god class symptoms, modularity violations and multithread correctness issues are located in classes with higher defect-proneness.
- Another work by Li et al. [7] suggests that two modularity metrics are strongly correlated with commit density: IPCI (Index of Package Changing Impact) & IPGF (Index of Package Goal Focus). Strong correlation was also found between corrective maintenance and fan-out, file size and frequency of changes of file in a study by Schwanke et al. in 2013 [8].
- An interesting study has been carried out by Oliva et al. [9], who searched for symptoms of increased rigidity and fragility on a degraded software system (Apache Maven 1.x) which was completely rewritten to Maven 2.x. The authors found signs of increased fragility (i.e. tendency of a system to break when changes are performed), but no definite evidence of increased rigidity (i.e. difficulty in performing changes due to ripple effects).

The results of these studies, despite the fact that some of them are only indirectly related to TD, have produced some evidence about the relation between maintenance effort and TD. However, the following limitations have been identified:

- Almost all studies quantify TD by means of few metrics, whereas TD manifests itself through a number of parameters in a software project.
- Most studies conducted research on a restricted sample of projects limiting the generalizability of the results (except for [4] and [7] that considered 10 and 13 projects, respectively).
- There is no relevant study that focuses on PHP web applications, which form the majority of operating code in Web today.
- There is no study that focuses on the interest that incurs when performing corrective maintenance.

Based on the abovementioned limitations, the purpose of this study is to provide insights into the relation between the accumulated amount of TD in a module and the maintenance effort spent on corrective activities. In particular, we investigate the relation between TD amount and: (a) frequency of corrective maintenance activities (***interest probability***), and (b) the effort spent in these activities (***related to interest amount***). To over-come the limitations mentioned in the previous paragraph, we: (a) calculate TD amount with SonarQube[2] that assesses TD based on seven axes of code quality (e.g., code duplications, metrics, styling conventions, etc.), (b) perform our case study on 10 open source PHP web applications, and (c) holistically investigate both interest probability and interest amount.

2. Case study design

In this section, we present the case study design, based on the guidelines reported by Runeson et al. [10].

2.1. Goal and research questions

The goal of this study is to examine whether the frequency and the effort spent on corrective maintenance activities of a specific module, is related to the amount of its TD. Based on this goal, the main research question of this study can be formulated as follows: "*Is the amount of TD in a software module related to the frequency and extent of corrective maintenance activities performed in it?*" To ease the reporting of the case study, from this main question, we derived two research questions:

[2] Available at: http://www.sonarqube.org

Table 1
Analyzed Projects.

Project	#stars	#releases
CodeIgniter	12K	27
Symfony	12K	209
Composer	8K	24
Yii2	8K	13
Guzzle	7K	108
Slim	7K	74
Laravel (kernel)	6K	192
Piwik	6K	429
PHPunit	5K	402
Twig	3K	86

RQ_1: *Is the TD amount of a file related to the number of times that it underwent corrective maintenance?*

RQ_1 aims at investigating whether files with higher amount of TD are associated with more problems and therefore require more frequent corrective maintenance. The presence of such an association would imply that TD can serve as an indicator for prioritizing maintenance and testing activities. Moreover, such a finding would validate the importance of TD as a crucial parameter to be taken into account during software development.

RQ_2: *Is the TD amount of a file related to the extent of modification that it underwent during corrective maintenance?*

Although the number of times a file undergoes corrective maintenance is a solid indicator of interest probability, to investigate whether modules with high TD produce more interest, in RQ_2, we focus on the extent of maintenance effort, as captured by the number of modified lines.

It should be clarified that the proposed case study design does not support the investigation of causal relationships between the TD incurred in one revision and the amount of corrective maintenance in subsequent revisions. Such an analysis is an interesting research topic but should be properly performed, since it would be extremely difficult to associate changes in a specific commit, to the TD as measured in one out of the many past revisions.

2.2. Cases and units of analysis

Our study focuses on web applications developed with PHP. The motivation for focusing on PHP is that it holds the lion's share of operating Web applications today. The criteria for selecting the projects are:

- the source code should be publicly available (data was retrieved via GitHub's API)
- projects should be actively maintained (until the date on which this paper is written)
- projects should have at least 10 releases denoting jumps in functionality or the addition of significant fixes[3] in their history to justify evolution analysis
- projects should be popular (among the projects with most stars in GitHub)

The list of the investigated projects (i.e., cases) is presented in Table 1. This study is an embedded multiple-case study, because we analyze every project at the file level (unit of analysis), whereas the results are presented at the case (i.e., project) level. We used files as a unit of analysis, since we include both object-oriented and non-object-oriented code. Thus, the use of any other type of

[3] The rationale for this choice is that any project with a history spanning more than 10 significant releases underwent substantial adaptive maintenance and is highly probable to have been the subject of corrective maintenance as well.

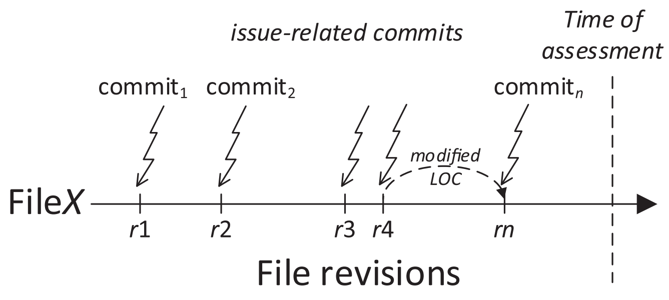


Fig. 1. Corrective maintenance at file level.

module (e.g., class) would not be possible. An alternative to this study design could be to perform a per-version analysis, i.e. by considering the TD amount of each file for every project version and the corrective maintenance between successive versions, as a unit of analysis. However, in such a case the TD of each version would be correlated to the TD of previous versions, thus rendering the data points of our study not independent.

2.3. Data Collection

For each unit of analysis (file), we recorded three variables. To facilitate the following description of variables we illustrate related concepts in Fig. 1:

- *V1, Average TD of each file*. All projects were analyzed with SonarQube and TD of each file (in minutes) was retrieved via the SonarQube. SonarQube calculates a file's TD by summing up the technical debt of every violation found on that file, which is the estimated time to fix that violation. The debt for each file represents various aspects of TD quantification, ranging from programming convention violations (e.g., lack of comments) to structural characteristics of the software (e.g., method complexity), which can affect the maintainability and comprehensibility of files. Since several revisions of each file have been analyzed, the TD for each file is obtained as the average of the TD corresponding to the file after each issue-related commit:

$$TD_{FileX} = \frac{1}{n} \cdot \sum_{i=1}^{n} TD(r_i)$$

  The average for a file's TD offers the advantage of obtaining a relatively accurate estimate, compared to alternatives, as it characterizes the entire history of the file. On the contrary, assuming that TD remains relatively stable and considering only the TD of the initial or the last revision would not be accurate since by nature software systems are evolving and TD changes over time.
- *V2, Number of times each file is modified due to corrective maintenance***.** Variable V2 corresponds to the total number of issue-related commits (for the examined file) from the initial revision to the time of assessment.
- *V3, Number of modifications (modified LOC) each file undergoes during corrective maintenance*. Variable V3 corresponds to the average number of modified LOC (for the examined file) from the initial to the last issue-related commit prior to the time of assessment.

To calculate [V2] and [V3] we retrieved commit and issue data for each project via the GitHub API. For each project's issue we tracked the commit by which the issue was closed and eventually found the files that were modified and the number of modified lines in that file. GitHub identifies issue-related commits by recognizing in the commit message the keywords 'fixes', 'resolves' and 'closes' when accompanied by a hash-tagged issue id. The tool for analyzing GitHub data was developed by the first author.[4]

2.4. Data analysis

To answer the research questions stated in Section 2.1, using the data described in Section 2.3, we performed correlation analysis and hypothesis testing. For both questions, we perform exactly the same analysis, but on different variables. For RQ_1 the testing variable is [V2], whereas for RQ_2 the testing variable is [V3]. An overview of the data analysis strategy is presented in Table 2.

Additionally, the Mann-Whitney U Test (we did not use the independent sample *t*-test, since variables do not follow the normal distribution) is able to investigate the discriminative power of the TD amount as an indicator of corrective maintenance frequency and effort (RQ_1 and RQ_2, respectively). In other words, we investigate if modules with high levels of TD amount present more frequent and more intense corrective maintenance activities, compared to modules with lower TD amounts. We note that in order to answer RQ_2, we needed to transform [V3] from a continuous to a binary variable. As low (high) TD files are characterized the ones that have technical debt that falls below (higher than) the median TD amount across all files for that project.

The aforementioned tests are fitting ways to assess the consistency/correlation and discriminative power of metrics, as described by 1061:1998 IEEE Standard for Software Quality Metrics.[5]

3. Results

Table 3 lists the results of the conducted Spearman's correlation analysis for each project for both RQs. Concerning RQ_1, in all ten projects there is a statistically significant positive correlation between TD amount of a file and the number of times that file underwent corrective maintenance (***interest probability***). Regarding RQ_2, in 8 out of 10 projects there is a statistically significant positive correlation between the amount of TD of a file and the extent of modification that the file underwent during corrective maintenance (related to ***interest amount***).

To allow a visual interpretation of the results, in Fig. 2 we depict the two indicators of the required effort (times that a file undergoes defect-related changes and the extent of changes in terms of lines of code) for each project, by differentiating between low and high TD files. As it becomes evident from the box plots, the required maintenance is always (except for one case in Fig. 2(b)) larger for high TD modules. This finding is also supported by the results of the Mann-Whitney U test which suggest that [V2] and [V3] in high-TD files are statistically different from [V2] and [V3] in low-TD files ([*V*2]: p-value=~0.00, [V3]: p-value=~0.00). On average, the number of times that a high TD file is modified is **1.9** times larger than the number of times a low TD file is changed. In terms of the extent of change, the corresponding ratio is **2.4** to 1.

4. Threats to validity

The results of the study are subject to external validity threats since the investigation has been performed on 10 PHP projects. Further studies on other projects or languages would be valuable in assessing the relation between TD amount and interest probability / amount in different contexts. Moreover, the assessment of interest amount through the extent of modification poses a threat to construct validity, since interest should be ideally quantified as

[4] https://github.com/theoAm/githubGrabber

[5] 1061-1998 IEEE Standard for a Software Quality Metrics Methodology, IEEE Standards, IEEE Computer Society, 31 December 1998 (re-affirmed 9 December 2009).

Table 2
Data Analysis.

RQ	**Analysis Strategy**
RQ_1	Spearman Correlation [V1] and [V2] Mann-Whitney U Test for [V2] grouped by [V1]
RQ_2	Spearman Correlation [V1] and [V3] Mann-Whitney U Test for [V3] grouped by [V1]

Table 3
Spearman's correlation results.

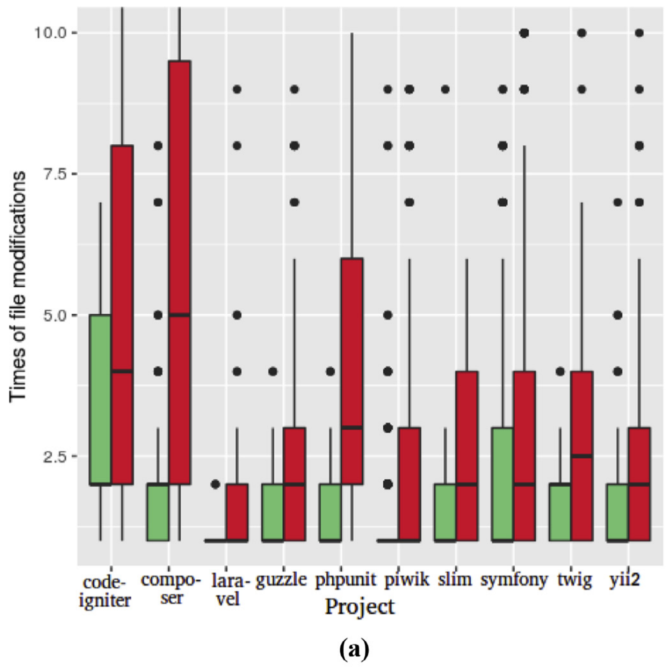
Project	RQ1		RQ2	
	p	r	p	r
CodeIgniter	0.00	0.293	0.08	−0.124
Symfony	0.00	0.301	0.00	0.280
Composer	0.00	0.544	0.00	0.310
Yii2	0.00	0.278	0.00	0.262
Guzzle	0.00	0.366	0.01	0.178
Slim	0.00	0.409	0.00	0.591
Laravel (kernel)	0.00	0.481	0.17	0.148
Piwik	0.00	0.363	0.00	0.204
PHPunit	0.00	0.626	0.00	0.290
Twig	0.00	0.366	0.00	0.433

the difference between the nominal effort for fixing an issue (i.e. in case no TD were present) and the actual effort spent. The former effort is unfortunately unknown. However, the findings observed when high TD modules are contrasted to low TD ones, imply that increased frequency and extent of modification are often encountered in files with increased interest amount.

A second threat to construct validity stems from the fact that not all reported issues point to errors, but some of them might contain a feature request or suggestion for performance improvement. As a result any actions to handle this issue would constitute adaptive or perfective maintenance rather than corrective one. Another threat of the same category, is that bug-related commits, which indeed fix an issue, but do not employ the keywords sought by GitHub, will be missed. This threat implies that there might be other bug-related commits which have been neglected in the study.

Another threat pertaining to the construct validity of the study stems from the fact that TD amount and the two employed indicators of corrective maintenance are aggregated over multiple revisions, possibly accounting for a significant period of time. As a result, especially in the case of variations of TD or corrective maintenance during that time, it cannot be safely assumed that the measured levels of corrective maintenance correspond to the measured TD. For example, an observed high level of corrective maintenance in a module with high level of TD, could in fact be due to a particular sub-period in which the module had low TD.

Finally, the present study does not investigate whether the two interest-related variables of the research questions (i.e., *frequency of modifications* and *extent of modification* due to corrective maintenance) might be affected by the propagation of errors. In particular, we study the relation between the two aforementioned variables and the TD principal of the files in which errors have been fixed, possibly neglecting the TD of the originating files (i.e., those from which errors might have propagated). This treatment poses a threat to construct validity and constitutes an interesting research direction for future work.



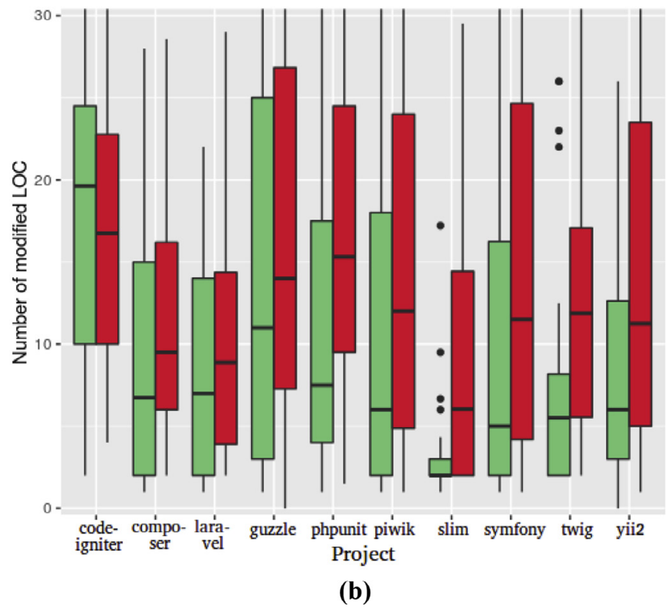


Fig. 2. Discriminative power of TD amount (left/right bars correspond to low/high TD files, respectively).

5. Discussion and conclusions

The results of this study suggest that TD amount is indeed correlated with maintenance effort. In particular, developers appear to spend more time on fixing issues in files with high levels of accrued technical debt, compared to files that present less TD. Therefore, project managers should take quality-oriented decisions to deter the appearance of software units with increased technical debt.

With respect to practitioners, the results provide additional evidence that TD undermines software maintenance and that it should be taken under consideration before any design and implementation decision. Moreover, the domain of the study suggests that TD appears to be important in a web context as well. Software engineers can take advantage of such empirical evidence to convince management about the importance and need to manage TD. From a research perspective, since there is sufficient empirical evidence of the impact of TD amount on corrective maintenance, the need to devise a framework for assessing the associated risk and costs of managing TD becomes essential.

References

[1] N.S.R. Alves, T.S. Mendes, M.G. de Mendonça, R.O. Spínola, F. Shull, C. Seaman, Identification and management of technical debt: a systematic mapping study, Inf. Softw. Technol. 70 (Feb. 2016) 100–121.
[2] A. Ampatzoglou, A. Ampatzoglou, P. Avgeriou, A. Chatzigeorgiou, A financial approach for managing interest in technical debt, in: B. Shishkov (Ed.), Business Modeling and Software Design, Springer International Publishing, 2015, pp. 117–133.
[3] A. MacCormack, D.J. Sturtevant, Technical debt and system architecture: the impact of coupling on defect-related activity, J. Syst. Softw. 120 (Oct. 2016) 170–182.
[4] Q. Feng, R. Kazman, Y. Cai, R. Mo, L. Xiao, Towards an architecture-centric approach to security analysis, in: 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA), 2016, pp. 221–230.
[5] R.L. Nord, I. Ozkaya, E.J. Schwartz, F. Shull, R. Kazman, Can knowledge of technical debt help identify software vulnerabilities? 9th Workshop on Cyber Security Experimentation and Test (CSET 16), 2016.
[6] N. Zazworka, A. Vetrò, C. Izurieta, S. Wong, Y. Cai, C. Seaman, F. Shull, Comparing four approaches for technical debt identification, Softw. Qual. J. 22 (3) (Apr. 2013) 403–426.
[7] Z. Li, P. Liang, P. Avgeriou, N. Guelfi, A. Ampatzoglou, An empirical investigation of modularity metrics for indicating architectural technical debt, in: Proceedings of the 10th International ACM Sigsoft Conference on Quality of Software Architectures, New York, NY, USA, 2014, pp. 119–128.
[8] R. Schwanke, L. Xiao, Y. Cai, Measuring architecture quality by structure plus history analysis, in: Proceedings of the 2013 International Conference on Software Engineering, Piscataway, USA, 2013, pp. 891–900.
[9] G.A. Oliva, I. Steinmacher, I. Wiese, M.A. Gerosa, What can commit metadata tell us about design degradation? in: Proceedings of the 2013 International Workshop on Principles of Software Evolution, New York, NY, USA, 2013, pp. 18–27.
[10] P. Runeson, M. Host, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley & Sons, 2012.