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:

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

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

[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.