Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

Further Investigation of the Survivability of Code Technical Debt Items

Ehsan Zabardast*[1] | Kwabena Ebo Bennin[2] | Javier Gonzalez Huerta[1]

[1]Software Engineering Research Lab (SERL), Blekinge Institute of Technology, Blekinge, Sweden
[2]Information Technology Group, Wageningen University and Research, Wageningen, The Netherlands

Correspondence
*Ehsan Zabardast, Software Engineering Research Lab (SERL), Blekinge Institute of Technology, Campus Gräsvik, Valhallavägen 1, Karlskrona, Sweden .
Email: ehsan.zabardast@bth.se

Summary

Context: Technical Debt (TD) discusses the negative impact of sub-optimal decisions to cope with the need-for-speed in software development. Code Technical Debt Items (TDI) are atomic elements of TD that can be observed in code artifacts. Empirical results on open-source systems demonstrated how code-smells, which are just one type of TDIs, are introduced and "survive" during release cycles. However, little is known about whether the results on the survivability of code-smells hold for other types of code TDIs (i.e., bugs and vulnerabilities) and in industrial settings.

Goal: Understanding the survivability of code TDIs by conducting an empirical study analyzing two industrial cases and 31 open-source systems from Apache Foundation.

Method: We analyzed 144,476 code TDIs (35,372 from the industrial systems) detected by Sonarqube (in 193,196 commits) to assess their survivability using survivability models.

Results: In general, code TDIs tend to remain and linger for long periods in open-source systems, whereas they are removed faster in industrial systems. Code TDIs that survive over a certain threshold tend to remain much longer, which confirms previous results. Our results also suggest that bugs tend to be removed faster, while code smells and vulnerabilities tend to survive longer.

KEYWORDS:
Survivability, Code Technical Debt Items, Code Smells, Bugs, Vulnerabilities

1 | INTRODUCTION

There is an ever-increasing pace in the size and complexity of software systems as they evolve, as Lehman[1,2] formulated in his Laws of Software Evolution, also known as Lehman's Laws. As a consequence of this ever-increasing size and complexity, software systems accumulate Technical Debt as they are developed and evolve[3]. Technical Debt (TD)[4] is a metaphor commonly used to discuss the negative impact of sub-optimal design decisions, often taken to cope with the need for speed in the development. As a software system evolves, these sub-optimal design decisions taken in order to be able to deliver the product in time, can potentially hinder its maintainability, and even our ability of delivering future releases of the product[3].

Sub-optimal design decisions are often not visible, however, they might manifest in the form Technical Debt Items (TDIs), which are the manifestations of TD[3]. TDIs can take the form of vulnerabilities, and code smells[5,6,7], whilst some of them might be visible and they might materialise in terms of bugs or defects[6]. TD items are "atomic elements of TD" that connect a set of artefacts (e.g., code) with the consequences of the quality[3] and have been introduced as a way to quantify or visualise TD. Bugs, code smells, and vulnerabilities are some examples of code Technical Debt Items (TDIs)[3,8,9]. These code TDIs go a long way to affect the development process and evolution of the software system, thus creation friction[3] and increasing the maintenance effort[10]. Adding new functionality to the existing system is also challenging when code TDIs are present in the system[11].

Several empirical studies acknowledge the negative effects of code TDIs on software quality, e.g., [12,13]. The repaid code TD (removing code TDI) not only improves the maintainability of the software system but also reduces the maintenance costs [10]. Studying the life cycle of code TDIs (bugs, code smells, and vulnerabilities) can therefore help prioritise the maintenance activities [14]. Understanding the life cycle of code TDIs is essential in building support tools as well. The empirical analysis of the evolution and survivability (i.e., the time that the code TDIs remain in the system) of the code TDIs thus have crucial implications for software development teams.

Prior research [15,5,12,16,17,18,19,14] have studied the survivability of code TDIs in software systems - mainly focusing on – some – code smells. While there are studies that investigate the effects of code TDIs on codebase and code entities, few empirical studies have been conducted on industrial settings. Additionally, a more in-depth investigation and insights into how development and maintenance activities impact the survivability of code TDIs are yet to be considered by prior studies. By taking inspiration and following a similar approach of the study by Tufano et al. [14], we conduct a large-scale empirical but parallel study where we focus on a different scope, a much wider set of code TDIs, and different research questions.

This sample study aims at assessing the survivability of code TDIs in large-scale industrial and open-source systems by using robust statistical tests. We conduct an empirical study to investigate further how long TDIs in codebases survive. To achieve this goal, we conduct a longitudinal study of two industrial systems and 31 open-source systems from the Apache Foundation. We analysed 144,476 code TDIs in 193,196 commits in total to investigate *the survivability of different code TDI types: bugs, code smells, and vulnerabilities.*

More specifically, this study aims at answering the following research question:

- **RQ.** *What are the differences in the survivability of the types of code TDI, namely bugs, code smells, and vulnerabilities?*

This paper makes the following contributions:

- Provides insights about the survivability of code TDIs (bugs, code smells, and vulnerabilities);
- Provides a replication package for reproducible results and analysis by other researchers[1].

The rest of this article is organized as follows: Section 2 summarizes the related works of studies. Section 3 describes the data collection and analysis process. The results are described in Section 4. Section 5 provides a discussion of the implications of the results. Lastly, the threats to validity and conclusions of this study are presented in Section 6 and Section 7 respectively.

2 | RELATED WORK

Several studies have focused on the detection and understanding of when and how the introduction of specific types of code TDI (i.e., code smells) impact software maintenance and quality (e.g., [15,20]). These studies use historical data to evaluate the lifespan of code smells and help improve maintenance activities and code quality. Moreover, the empirical analysis of the evolution of code smells during the product life-cycle has been addressed in several research studies [15,17]. Chatzigeorgiou and Manakos [12] investigated the presence and evolution of code smells through an exploratory analysis of past versions of a software system. With a focus on when and how code smells are introduced and removed from software systems, the authors examined the evolution of three types of code smells in two open-source systems. Their results indicated that most code smells last as long as the software system operates, and refactoring does not necessarily eliminate code smells. This is not unusual as a study by Peters and Zaidman [20] showed that developers, although being very aware of the presence of code smells in their code, tend to ignore the impact of those code smells on the maintainability of their code. This observation ignited interest in assessing the impact of code smells on maintenance activities within the research community, and several empirical studies have been conducted since then. Marcilio et al. [21] investigated the usage of an automatic static analysis tool (ASAT), i.e, Sonarqube, to examine its usage by the developers. They report that practitioners can benefit from using ASATs if they are properly configured, i.e., using relevant rules. Their results show that only 8.76% of the code TDIs are *fixed* from the detected code TDIs among which code smells and major issues are more prevelant

A study by Yamashita and Moonen [22] consisting of an empirical study of four Java-based systems before entering the maintenance phase and observation and interview of 14 developers who maintained the Java systems identified 13 maintainability factors that are impacted by code smells. Further empirical studies by the same authors (Yamashita and Moonen [23]), which focused on the interaction between code smells and their effect on maintenance effort revealed that some inter-smell relations were associated with problems during maintenance and some inter-smell relations manifested across coupled artefacts. A study by Sjøberg et al. [10] demonstrated that the effect of code smells on maintenance effort was limited.

More studies have recently empirically analysed different types of technical debt items in several other software projects intending to understand the evolution of code smells, when and how they are introduced into systems. Tufano et al. [19] conducted an extensive empirical study on 200 open-source systems and investigated when bad smells are introduced. Comprehensive analysis of over 0.5M commits and manual analysis of 9164

[1] https://github.com/ehsanzabardast/code_tdi_survivability

smell-introducing commits revealed that smells are not introduced during evolutionary tasks. A further study by the same authors [14] contradicts common wisdom, showing that the majority of code smells are introduced when an artefact is created and not during the evolution process where several changes are made to software artefacts. They also observed that 80% of smells are not removed, and they survive as long as the system functions, confirming previous results by Chatzigeorgiou and Manakos [12].

Additionally, some research has been conducted on how much attention is dedicated to technical debt items such as code smells, bugs, and others by software developers and how these TDIs are resolved during the evolution process of a software system. A recent work by Digkas et al. [16] analysed the life cycles of code TDIs in several open-source systems. The authors reported a case study focusing on the different types of code TDIs fixed by developers and the amount of technical debt repaid during the software evolution process. The study analysed the evolution (weekly snapshots) of 57 Java open-source software systems under the Apache ecosystem. The study revealed that allocating resources to fix a small subset of the issue types contributes towards repaying the technical debt. Similarly, the recent study of Saarimäki et al. [9] aimed to comprehend the diffuseness of TD types, how much attention was paid to TDIs by developers and how severity levels of TDIs affected their resolution. The authors observed that code smells are the most introduced TDIs, and the most severe issues are resolved faster. To understand the needs of software engineers with regards to technical debt management, Arvanitou et al. [24] surveyed 60 software engineers from 11 companies. The authors observed that developers were mostly concerned with understanding the underlying problems existing in source code, whereas managers cared most about financial concepts.

Prior studies have, in general, focused on studying the introduction and evolution of code smells in open-source software systems. To the best of our knowledge, only the work by Digkas et al. [16] and our previous work ([25]) analysed TD repayment. Digkas et al. [16] analysed TD repayment by focusing on a broader set of TDIs, but with a coarse granularity (weekly snapshots) whilst we analyse the effect at commit level. In our previous work [25], we analysed the impact of different activities on TD, i.e., whether each activity contributed to the accumulation or repayment of TD whilst in this paper, we focus on TDIs. We present a statistical models on the survivability of code TDIs.

This study aims to extend the findings of previous studies in literature, especially the study by Tufano et al. [14]. We aim to verify and complement the results obtained in the study of Tufano et al. [14] with regards to the survivability aspects, by extending the scope of the analysis considering a bigger set of code smells and other categories of code TDI that can appear in code-bases such as *bugs*, and *vulnerabilities*. We thus analyse a similar set of code TDI as the one considered by Digkas et al. [16] by focusing on individual commits.

3 | RESEARCH METHODOLOGY

We have conducted a sample study to address the research questions defined in Section 1. Purpose of a sample study is to "study the distribution of a particular characteristic in a population (of people or systems), or the correlation between two or more characteristics in a population. [26]" In the subsections below we provide details on the data collection and analysis.

3.1 | Context Selection

We selected two industrial systems for this study and 31 open source systems (OSS) from the Apache Software Foundation, all developed in Java. The industrial systems were selected by convenience, due to their availability and because they are a long-lived, large-scale system that are still in production and continuously evolving, and have the following characteristics:

- **Industrial 1.** The selected system has over one million lines of code. We analyzed 9,331 commits that contained 33,974 code TDIs.
- **Industrial 2.** The selected system has over 60,000 lines of code. We analyzed 8,414 commits that contained 1,398 code TDIs. The developers are using Sonarqube in this system.

Table 1 summarises the historical information of the OSS analysed systems. All the OSS systems are hosted in `git` repositories[2] and the Sonarqube data was collected through a web API available online[3]. The data collection for the industrial cases was performed using the corresponding SonarQube API in their on-site SonarQube installations.

[2]https://github.com/apache
[3]https://sonarcloud.io/organizations/apache/projects

TABLE 1 General description of the analysed open-source systems.

			Number of			
	System Name	**System Size**	**Commits**	**TDIs**	**Classes**	**Contributors**
1	Ant	12503	14698	9351	1321	53
2	Commons Compress	26974	3133	772	377	56
3	Commons Geometry	16483	431	99	355	10
4	CXF	427463	16155	10000†	7555	154
5	Groovy	189114	18181	10000†	1724	307
6	Hadoop Ozone	132255	3202	3531	2182	104
7	IoTDB Project	120051	4611	2082	1685	94
8	Isis (Aggregator)	165713	15676	5532	5232	40
9	Jackrabbit FileVault	47692	8859	2053	3128	23
10	JSPWiki	56647	8853	3598	618	13
11	Karaf	123212	8482	6567	1582	141
12	PDFBox	141971	9694	1876	1348	6
13	POI	259765	10679	10000†	3512	15
14	Ratis	38622	1131	957	607	41
15	ServiceComb Pack	17494	1568	614	458	55
16	ServiceComb Toolkit	15262	237	331	639	5
17	Shiro	32723	2125	1964	724	46
18	Sling - CMS	11409	894	52	287	9
19	Sling Distribution Core	14185	483	621	267	9
20	Sling Launchpad Integration Tests	16180	483	1901	167	17
21	Sling Resource Resolver	10377	710	444	101	24
22	Sling Scripting JSP	27233	275	1929	124	12
23	Incubator Tamaya (Retired)	19056	1618	702	460	10
24	Dolphin Scheduler	58979	4441	2135	912	175
25	Gateway	76495	2455	2241	1479	54
26	Hop Orchestration Platform	483430	1572	10000†	3240	20
27	Jmeter	117352	17331	4362	1386	35
28	Openmeetings	99052	3121	1439	584	12
29	PLC4X	57162	3740	2491	1007	43
30	Roller	66248	4549	3318	610	12
31	Struts 2	110941	6064	8142	2060	51
	Totals	2992043	175451	109104	45731	-

†The limit for fetching the data from API is 10000 code TDIs.

3.2 | Using Sonarqube for Code TDI Detection

While there are alternative tools to detect the TDIs in the codebase of a system such as Codacy[4] and PMD source code analyzer[5], we decided to use Sonarqube[6] because it is widely used in both industrial and open-source systems [27] and has been used in other research studies, e.g., Zabardast et al. [25], Digkas et al. [16], and Guaman et al. [28]. Sonarqube, similar to other static analysis tools, parses the code base, and builds a model for each commit being analysed.

Sonarqube classifies code TDIs into three types of *Bugs*, *Code Smells*, and *Vulnerabilities:*

[4]https://www.codacy.com/
[5]https://pmd.github.io/pmd-6.17.0/index.html
[6]https://www.sonarqube.org/

- **Bug** is a synonym of fault, which is a manifestation of an error in the software, such an incorrect step, process, or data definition in a computer program [29].
- **Code Smells** are surface indications of deeper problems in the system [30,31], and they are "sniffable" at code level [32]. Code-Smells are sometimes also referred to as Code Anti-Patterns, which are common re-occurring solutions to a problem, which generate negative consequences [33].
- **Vulnerability** is a weakness in an information system, system security procedures, internal controls, or implementation that could be exploited by a threat source [34]. A vulnerability "does not cause harm in itself as there needs to be a threat present to exploit it [35]".

3.3 | Data Analysis

To understand the survivability of each code TDI's in the studied systems, we analyse the existence of code TDIs from their introduction in the codebase until they are marked as "closed". Similarly, the code TDIs that are still remaining in the system are marked as "open". In our analysis we include *all* the code TDIs present in the system, i.e., we include the code TDIs marked as closed and open. The details are provided in the rest of this subsection.

In other words, we use the creation time and the updated time – when the TDI was closed – of each TDI in the collected data. The extracted information is used to calculate the number of survived days and the number of survived commits for each case.

We use *The Number of Survived Days* and *The Number of Survived Commits* similar to the study designed by Tufano et al. [14] as complementary metrics to capture the views of code TDI survivability. Considering either of these metrics, individually, might be misleading since a project can be inactive for months (i.e., nothing committed for a while). The processed data will contain code TDI with no removal time, i.e., the code TDIs that are not marked as "closed" in the collected data. To interpret this, we use the same approach as [14] and mark such code TDI as "*Censored Data*".

We analyse the data using Survival analysis. Survival analysis is a statistical method that analyses and models the duration of events until other events happen [36], i.e., code TDI removal in this case. The survival function for code TDI, $S(t) = Pr(TDI > t)$ indicates that a code TDI exists longer than a specific time t. The survival analysis creates a Survivability Model based on historical data. Using this model we can generate survival curves illustrating the survival probability as a function of time. The model can handle both *complete data* (observations with an ending event) and *incomplete data* (observations without an ending event) if the data is marked properly. We create survival models based on the number of survived days, and the number of survived commits for each type of code TDI namely bugs, code smells, and vulnerabilities. The analysis is done in `R` using the `survival`[7] and `survminer`[8] packages. The `Surv` function is used to generate the survival model, and the `survfit` function is used to estimate survival curves. Additionally, similar to [14], we utilise the Kaplan-Meier estimator [37] in the analysis to estimate the removal time for the *incomplete* observations, therefore we also included "censored" data-points.

4 | RESULTS

In this section, we report the results of our study. We focus on the three types of code TDI (i.e., code smells, bugs, and vulnerabilities), as done in [16]. The raw experimental results, datasets extracted from open-source systems and the source code for implementing the experiments is provided in our replication kit online[9].

To address the research question, we analyzed the survivability of the code TDIs collected from two industrial systems and 31 open-source system from the Apache Foundation. We have analyzed the survivability both in terms of the survived days and survived commits. Figure 1 illustrates the box plots for the distribution of the survived days (left) and commits (right) for the detected code TDIs. The first and second rows belongs to the industrial systems and the third row belongs to the open-source systems. The plots are presented on a log-scale to make the representation easier to read. We observe that the median of the distributions are not significantly different across different types of code TDI for the number of survived days and the number of survived commits. However they are much lower in the industrial systems for all types of code TDIs.

Table 2 summarizes the descriptive statistics for the number of survived days and commits for all systems. The table distinguishes between the industrial systems and open-source systems for each row labeled as *Industry* and *Open-Source*. We use the median as the measure of central tendency to minimize the effect of outliers [38]. We use median values as the references of analysis, as summarized in Table 2.

[7] https://cran.r-project.org/web/packages/survival/index.html
[8] https://cran.r-project.org/web/packages/survminer/index.html
[9] https://github.com/ehsanzabardast/code_tdi_survivability

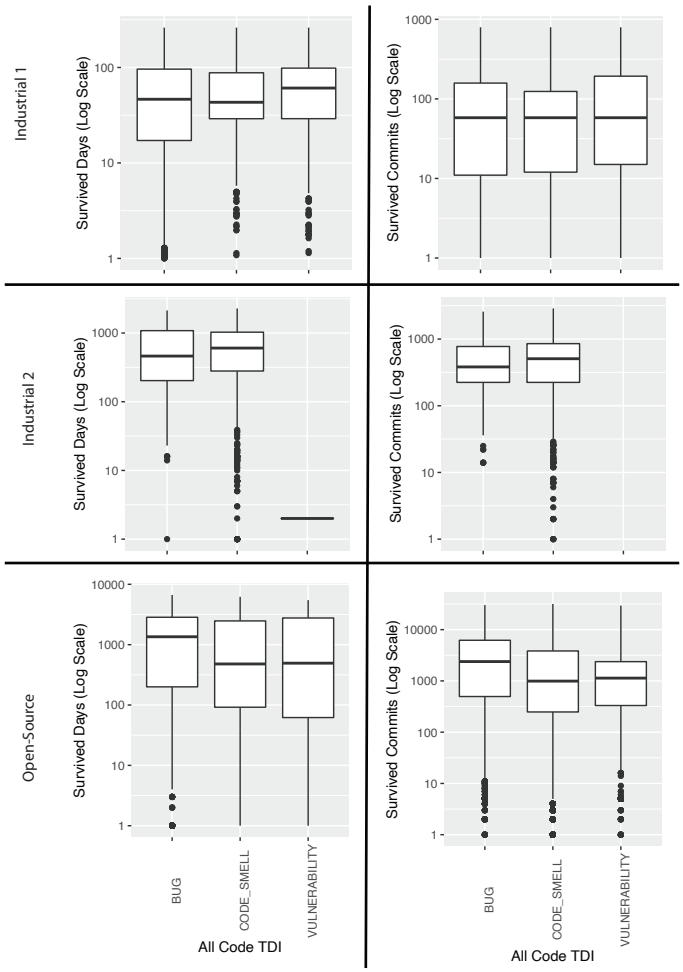

FIGURE 1 Distribution of the number of days (left) and commits (right) that code TDIs survived. The first row belongs to the industrial system and the second row belongs to the open-source systems.

We observe that the majority of the code TDIs detected in the industrial 1 are removed before the 43^{th} day (before 58^{th} commit) and the majority of the code TDIs detected in the industrial 2 are removed before the 580^{th} day (before 496^{th} commit). In open-source systems, the majority of the code TDIs are removed before the 398^{st} day (before 722^{th} commit).

Figure 2 illustrates the calculated survival probability curves for all the systems (survived days on the left and survived commits on the right). The first row belongs to the industrial 1 system, the second row belongs to the industrial 2 system, and the third row belongs to the open-source systems.

The first thing we observe is that survivability of code TDIs vary in the systems under investigation, both in terms of number of days and in number of commits. Considering these calculated probability curves, the survival probability of code TDIs in terms of survived days is highest in open-source systems where as the survival probability of code TDIs in the industrial systems are lower. Having higher survivability probability in terms of days means that for the same number of days, code TDIs in open-source systems have higher probability of surviving as compared to the probability of a code TDI surviving with the same number of days in the industrial systems. In other words, we have observed that in the analyzed systems, code TDIs tend to be removed faster, in terms of number of days, in industrial settings.

We also observe that the survival probability of code TDIs in terms of survived commits is higher in open-source systems. Having higher survivability probability in terms of commits means that for the same number of commits, code TDIs in open-source systems have higher probability of surviving as compared to the probability of a code TDI surviving with the same number of commits in the industrial systems. In other words, we have observed that in the analyzed systems, code TDIs tend to be removed faster, in terms of number of commits, in industrial settings.

TABLE 2 Descriptive statistics for the number of survived days and commits that code TDIs survived in the systems.

Case	Min	1stQu.	Median	Mean	3rdQu.	Max
			Survived Days			
Industry 1 - All Code TDI	0	12	43	58.70	95	262
Industry 2 - All Code TDI	0	221	580	678.01	1,007	2,283
Open-Source - All Code TDI	0	21	398	1,255.80	2,370	6,656
Industry 1 - Bug	0	11	43	58.37	96	262
Industry 2 - Bug	1	203	462	665.81	1,083	2,134
Open-Source - Bug	0	168	1,287	1,662.78	2,767	6,656
Industry 1 - Code Smell	0	15	43	62.32	83	252
Industry 2 - Code Smell	0	221	584	679.16	989	2,283
Open-Source - Code Smell	0	18	398	1,238.73	2,338	6,211
Industry 1 - Vulnerability	0	23	49	1,340.52	98	261
Industry 2 - Vulnerability	2	2	2	2	2	2
Open-Source - Vulnerability	0	26	415	1,340.52	2,767	5,470
			Survived Commits			
Industry 1 - All Code TDI	1	11	58	131.72	155	795
Industry 2 - All Code TDI	1	200	496	603.29	813	2,881
Open-Source - All Code TDI	1	68	722	3,154.40	3,492	31,690
Industry 1 - Bug	1	11	58	131.11	158	795
Industry 2 - Bug	1	224	375	545.30	773	2577
Open-Source - Bug	1	255	1,719	4,443.13	4,950	30,185
Industry 1 - Code Smell	1	12	58	143.38	124	695
Industry 2 - Code Smell	1	198	498	606.80	813	2,881
Open-Source - Code Smell	1	67	707	3,114.26	3,311	31,690
Industry 1 - Vulnerability	1	15	58	138.74	193	790
Industry 2 - Vulnerability	1	1	1	1	1	1
Open-Source - Vulnerability	1	186	1,134	2,448.88	2,376	29,493

We use 100 days as point of reference to present the results of survivability models. However, instead of using 10 commits as in Tufano et al. work, we use 100 commits in our analysis as the point of reference. This is owing to the fact that the 10 commit threshold might be too short for our datasets and the results might turn inconclusive, since the probabilities of survivability for 10 days are very high and very similar among systems.

We observe that, in general, code TDIs are removed faster in the investigated industrial system as compared to the investigated open-source systems. Table 3 summarizes the probability of code TDIs surviving up to 100 days and commits for *Bug*, *Code Smell*, and *Vulnerability* separately for the industrial system and open-source systems. Note that there was only one observation as *vulnerability* in the Industrial 2, therefore we cannot presenting any survival probability in Table 3.

From the survivability models for all code TDIs in the investigated systems in Table 3, we observe that:

- **In the industrial 1 system:** There is a higher chance for code TDIs to be removed before 100 days (100 commits) than when the they stay in the system after 100 days (100 commits), i.e., many of the code TDIs are removed relatively soon.
  - *Bug:* There is a 18.476% chance that the investigated code TDIs survive until 100 days (38.312% chance that the investigated code TDIs survive until 100 commits).
  - *Code Smell:* There is a 13.850% chance that the investigated code TDIs survive until 100 days (37.180% chance that the investigated code TDIs survive until 100 commits).

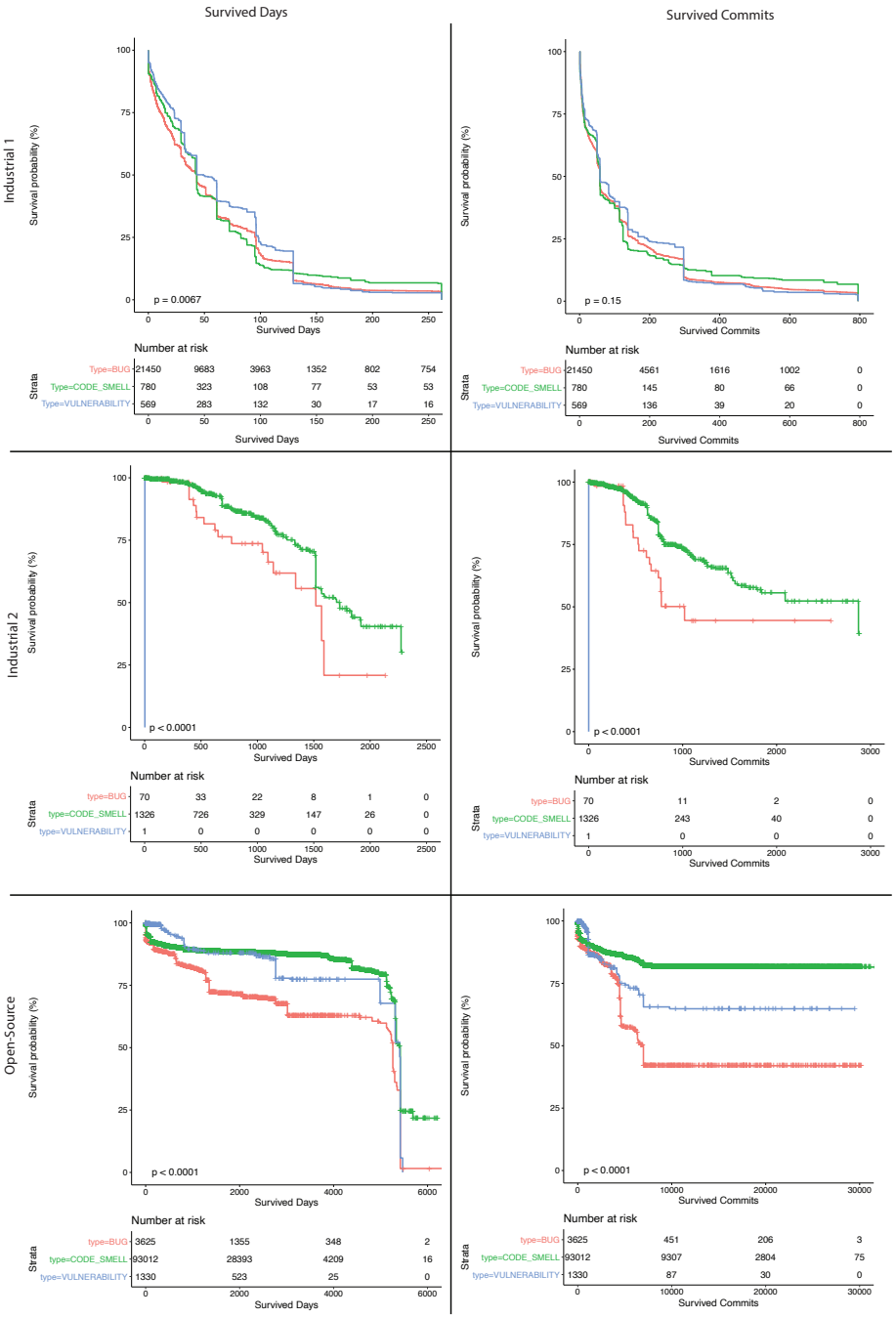

FIGURE 2 Distribution of the number of days (left) and commits (right) that code TDIs survived in all systems.

- *Vulnerability*: There is a 23.200% chance that the investigated code TDIs survive until 100 days (39.890% chance that the investigated code TDIs survive until 100 commits).

- **In the industrial 2 system:** There is a lower chance for code TDIs to be removed before 100 days (100 commits) than when the they stay in the system after 100 days (100 commits).

  - *Bug*: There is a 100% chance that the investigated code TDIs survive until 100 days (98.360% chance that the investigated code TDIs survive until 100 commits).

TABLE 3 The probability of code TDIs surviving up to 100 days and commits.

Case	Survival Probability	Standard Error	Lower 95% CI	Upper 95% CI
		Survived Days		
Industry 1 - Bug	18.476%	0.00265	0.17963	0.19002
Industry 2 - Bug	100.000%	0	1	1
Open-Source - Bug	91.838%	0.00474	0.90913	0.92773
Industry 1 - Code Smell	13.850%	0.01240	0.11620	0.16500
Industry 2 - Code Smell	99.410%	0.00221	0.98982	0.998448
Open-Source - Code Smell	94.470%	0.00084	0.94306	0.94635
Industry 1 - Vulnerability	23.200%	0.01770	0.19980	0.26940
Industry 2 - Vulnerability	-	-	-	-
Open-Source - Vulnerability	99.381%	0.00236	0.98920	0.99844
		Survived Commits		
Industry 1 - Bug	38.312%	0.00332	0.37667	0.38968
Industry 1 - Bug	98.360%	0.01630	0.95230	1.0000
Open-Source - Bug	92.619%	0.00447	0.91747	0.93499
Industry 1 - Code Smell	37.180%	0.01730	0.33940	0.40730
Industry 2 - Code Smell	99.320%	0.00238	0.98862	0.99793
Open-Source - Code Smell	95.227%	0.00078	0.95074	0.95380
Industry 1 - Vulnerability	39.890%	0.02050	0.36070	0.44130
Industry 2 - Vulnerability	-	-	-	-
Open-Source - Vulnerability	99.683%	0.00158	0.99374	0.99994

  - *Code Smell:* There is a 99.410% chance that the investigated code TDIs survive until 100 days (99.320% chance that the investigated code TDIs survive until 100 commits).
  - *Vulnerability*: There are not enough observations to calculate survival probabilities.

- **In open-source systems:** Similar to the industrial system, there is a higher chance for code TDIs to be removed before 100 days (100 commits) than when the they stay in the system after 100 days (100 commits), i.e., many of the code TDIs are removed relatively soon but if they are not removed, they can stay in the system for as long as the system is in production.
  - *Bug*: There is a 91.838% chance that the investigated code TDIs survive until 100 days (92.619% chance that the investigated code TDIs survive until 100 commits).
  - *Code Smell:* There is a 94.470% chance that the investigated code TDIs survive until 100 days (95.227% chance that the investigated code TDIs survive until 100 commits).
  - *Vulnerability*: There is a 99.381% chance that the investigated code TDIs survive until 100 days (99.683% chance that the investigated code TDIs survive until 100 commits).

We analyzed the different types of code TDIs individually for the industrial system and open-source systems. Individual box plots for the distribution of the survival days (left) and commits (right) for the different types of code TDI detected in the industrial systems and open-source systems are presented in Figure 1. We observe that the median of the distribution for bug, code smell, and vulnerability is significantly higher in the open-source systems when compared to the industrial systems both for the number of survived days and the number of survived commits.

5 | DISCUSSION

5.1 | General Findings

Our empirical analyses reveal the unpredictable nature of the survivability of code TDIs in software systems. In general, practitioners aim to remove TDIs as soon as possible as demonstrated by our analysis. The survivability of the issues range from 0 to $2,283$ days for the industrial systems and 0 to $6,656$ days for open-source systems.

The median for the industrial system stay below 100 days; therefore, the TD mitigation strategies to remove code TDIs seem to be more effective in industrial systems as compared to open-source systems. Our results also suggest that, for open-source systems, when code TDIs remain in the system after 100 days, they can survive for a long time in the system. A similar behavior can be observed when analyzing the number of survived commits, which ranges from 1 to $2,881$ commits for the industrial systems and 1 to $31,690$ commits for open-source systems.

TD mitigation strategies seem to be more effective removing TDIs in industrial settings in terms of number of survived commits as well. As discussed before, regardless of the system under study, code TDIs that survive past the 100^{th} day threshold might survive much longer, confirming the finding in the study by Tufano et al.[14].

However, we observed patterns that contrast what was found in previous research studies. We found that:

- After 100 days:
  - The survival probability for the industrial 1 system is as follows: *Bug* 17.48% - *Code Smell* 13.85% - *Vulnerability* 23.20%.
  - The survival probability for the industrial 2 system is as follows: *Bug* 100.00% - *Code Smell* 99.41%.
  - The survival probability for the open-source systems is as follows: *Bug* 91.84% - *Code Smell* 94.47% - *Vulnerability* 99.38%.
- After 100 commits:
  - The survival probability for the industrial 1 system is as follows: *Bug* 38.31% - *Code Smell* 37.18% - *Vulnerability* 39.89%.
  - The survival probability for the industrial 2 system is as follows: *Bug* 98.36% - *Code Smell* 99.32%.
  - The survival probability for the open-source systems is as follows: *Bug* 92.62% - *Code Smell* 95.23% - *Vulnerability* 99.68%.

The study by Tufano et al.[14] found that it was after 1000 days when the survival probability achieved similar values. This might be owing to the fact that we have extended the scope of our analysis to include a much wider set of code smells (the study by Tufano et al.[14] only studied five code-smells), and we also analyze bugs and vulnerabilities, which were not in the scope of previous research studies. We hypothesize that some of the five code smells analyzed in the the previous study by Tufano et al.[14] might not be the priority for developers in their maintenance activities, in line with what is found in[39]. Furthermore, we have observed a completely different behavior when it comes to code-smells. Chatzigeorgiou and Manakos[12] observed that code smells are never removed and stay as long as the software system operates. Similarly, Marcilio et al.[21] observed that a low percentage (8.76%) of the code TDIs, including bugs, code smells, and vulnerabilities, are removed suggesting that not all code TDIs detected by Sonarqube are relevant to the developers.

5.2 | Implications of the Results

Technical Debt (TD) management has recently been the focus of attention in both academic and industrial communities[40]. The research on TD is in its initial phase, with researchers focusing on a few types of debt[41,42]. Empirical evaluation of TD management activities, especially the evidence from the software industry, is essential to shedding light on technical debt prioritization activities[40]. The results and the analysis methods provided by our study can help the research and industrial communities to better understand what is the lifespan of different types of TDIs, not only code smells, and invite researchers tofurther investigate TD prioritization and the activities related to it.

Sonarqube use and its impact on the survivability of the code TDI

The use of automatic static analysis tools has become popular in the last few years[21]. Given the fact that developers working on the Industrial 2 system have been using Sonarqube during the last year of the development, we have examined if the use of such tools impacts the survivability of code TDIs by manually investigating the code TDIs that were created in the last year. It seems that the survivability of the code TDIs, both in terms of the number of survived days and commits, are not to impacted by using Sonarqube and follow a similar pattern to what is portraited in Figure 2 for Industrial 2 system. The code TDIs that were introduced in the last year during the period where developers used Sonarqube have similar number of survival days and commits.

System type impacts the survivability of the code TDIs.

Our study suggests that the survival probability of the code TDIs, both in terms of the number of survived days and commits, varies throughout the software systems under investigation. By comparing the density distribution for all code TDIs of survived days with the density distribution for all code TDIs of survived commits presented in Figure 1, we observe that the industrial and open-source systems have different distributions, but these distributions follow a similar trend. The code TDIs have a similar distribution in terms of the number of days but different survival probabilities in terms of the number of commits. This might be owing to the fact that the open-source systems have more frequent commits as compared to the industrial systems. Therefore, the code TDIs are addressed in later commits, whereas in industrial systems, the code TDIs tend to be addressed and resolved closer to the point when they are introduced in the system.

Code TDIs in the industrial systems are removed faster in terms of number of commits.

Our analysis reveals that code TDIs survive longer in open-source systems as compared to industrial projects. A viable reason can be that the quality standards in the industrial systems prevent code TDIs to stay in the system for longer periods. There are other factors that might affect how long the code TDIs survive in different systems such as the development practices put in place, the domain, the business model, the product maturity, and the expertise of the development team, and as discussed above, the usage of static analysis tools like SonarQube. The circumstances of each system might affect the survivability of the code TDIs, e.g., the industrial systems might have more rigorous development process including more stringent code reviews and test processes before code is pushed into production.

Before drawing firm conclusions, each system should be analyzed in isolation, considering additional factors. These factors might include the development process, developers' experience (in general and in the system), team's culture, product maturity, specific refactoring policies, developers' perception of whether the code contains code TDIs or not, and the willingness of developers' to fix code TDIs.

6 | THREATS TO VALIDITY

In this section, we present the potential threats to validity that might affect the results and findings of this study. We discuss below the threats to the construct, internal, and external validity of the study.

The main threat to validity is the *Construct Validity*, i.e., the relationship between the theory and observation. The construct validity threats comprise of the errors and imprecision in measurement procedure adopted during the data collection process and whether the measurements actually reflect the construct being studied. We use Sonarqube, a widely used tool for measuring TD, as a way to detect code TDIs in the system. We also use the categories defined by SonarQube, and employed in other research studies (e.g., [16]), to categorize the types of code TDIs. We identify code TDIs in the systems using the default profile for `Java` by Sonarqube. We acknowledge the problems that might arise due to the use of a particular tool, i.e., Sonarqube, which includes the thresholds, measurements, and rules used to detect code TDIs and the possibility of having false positives and false negatives in the collected data.

Another threat to validity is regarding the detection of the bugs and vulnerabilities. Bugs and vulnerabilities that are detected now might not have been detected when the systems were being developed and there was no way for the developers to be aware of their existence. Therefore, no tool could warn the developers to remove the flaws when they were introduced.

Internal Validity refers to the degree to which the presented evidence can support a cause and effect relationship within the context of the study.

Our results are based on the `R` packages used to calculate the survivability curves. Different implementations of the same survivability analysis might lead to obtaining different results. The fact that one of the systems, (i.e., Industrial 1) was developed following the principles of clean-code [43] might have an influence on the results of the analysis of how code TDIs are removed, especially when it comes to the analysis of code smells and bugs being removed due to the development of new features.

External Validity refers to the degree to which the results can be generalized. In this study, we analyzed five systems, which are mainly Java-based software systems, two industrial systems, and three open-source systems. We limit the results of this study to the systems under investigation. We understand and acknowledge that the generalizability of the results is limited. We can only claim that the results are applicable to the analyzed commits in the systems under investigation.

We acknowledge that the removal of the code TDIs and the categories are highly dependent on each system, i.e., the development process, the developers' experience both in the software system project and their overall experience, the team's culture, the stage of development, and other factors that affect the survivability of code TDIs given a system [14]. Our analysis and findings are thus impacted by the system type under consideration.

7 | CONCLUSIONS

In this paper, we present the results of a sample study on the survivability of code TDIs in software systems. This paper presents a study on the change history of five software systems, and it aims to understand the survivability of code technical debt items (TDIs) in the codebase. Furthermore,

this study aims to extend the results of prior studies by including in the analysis not only code smells, but also other types of code TDIs such as bugs, and vulnerabilities. We have therefore focused on examining and assessing the differences in survivability among the three categories of detected code TDI.

We have conducted comprehensive empirical sample study using data gathered from other software systems, including two large industrial software systems and 31 open-source systems from the Apache Foundation. As illustrated by the survival curves, most of the code TDI in the investigated systems are removed rather quickly, i.e., there is a 23.20% chance that they survive until 100 days. Any code TDI that survives this threshold has a higher probability of surviving longer in the system.

When the system type and commit activities are taken into account, the code TDIs in the systems which have a bigger size tend to have longer survival duration. On the other hand, the code TDIs that survive past the median threshold tend to stay in the system for a long time.

Our findings opens the door for further studies. Our results can be strengthened by digging into the other factors that affect the systems. Additionally, we believe replications are needed to strengthen the results and study whether the results might be generalizable to similar systems.

ACKNOWLEDGEMENTS

This research was supported by the KKS foundation through the SHADE KKS Hög project with ref: 20170176 and the KKS S.E.R.T. Research Profile with ref. 2018010 project at Blekinge Institute of Technology, SERLSweden.

References

1. Lehman M. On understanding laws, evolution, and conservation in the large-program life cycle. *Journal of Systems and Software* 1979; 1: 213–221.

2. Lehman M. Laws of software evolution revisited. *European Workshop on Software Process Technology* 1996: 108–124.

3. Kruchten P, Nord R, Ozkaya I. *Managing Technical Debt: Reducing Friction in Software Development*. Pearson . 2019.

4. Cunningham W. The WyCash portfolio management system. *ACM SIGPLAN OOPS Messenger* 1993; 4(2): 29–30.

5. Brown N, Cai Y, Guo Y, et al. Managing technical debt in software-reliant systems. *Proceedings of the FSE/SDP workshop on Future of software engineering research* 2010: 47–52.

6. Kruchten P, Nord RL, Ozkaya I. Technical debt: From metaphor to theory and practice. *IEEE Software* 2012; 29(6): 18–21.

7. Lim E, Taksande N, Seaman C. A balancing act: What software practitioners have to say about technical debt. *IEEE software* 2012; 29(6): 22–27.

8. Li Z, Avgeriou P, Liang P. A systematic mapping study on technical debt and its management. *Journal of Systems and Software* 2015; 101: 193–220.

9. Saarimäki N, Lenarduzzi V, Taibi D. On the diffuseness of code technical debt in Java projects of the apache ecosystem. *Proceedings of the Second International Conference on Technical Debt* 2019: 98–107.

10. Sjøberg DI, Yamashita A, Anda BC, Mockus A, Dybå T. Quantifying the effect of code smells on maintenance effort. *IEEE Transactions on Software Engineering* 2012; 39(8): 1144–1156.

11. Neamtiu I, Xie G, Chen J. Towards a better understanding of software evolution: an empirical study on open-source software. *Journal of Software: Evolution and Process* 2013; 25(3): 193–218.

12. Chatzigeorgiou A, Manakos A. Investigating the evolution of bad smells in object-oriented code. *2010 Seventh International Conference on the Quality of Information and Communications Technology* 2010: 106–115.

13. Menzies T, Greenwald J, Frank A. Data mining static code attributes to learn defect predictors. *IEEE transactions on software engineering* 2006; 33(1): 2–13.

14. Tufano M, Palomba F, Bavota G, et al. When and why your code starts to smell bad (and whether the smells go away). *IEEE Transactions on Software Engineering* 2017; 43(11): 1063–1088.

15. Arcoverde R, Garcia A, Figueiredo E. Understanding the longevity of code smells: preliminary results of an explanatory survey. *Proceedings of the 4th Workshop on Refactoring Tools* 2011: 33–36.

16. Digkas G, Lungu M, Avgeriou P, Chatzigeorgiou A, Ampatzoglou A. How do developers fix issues and pay back technical debt in the apache ecosystem?. *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)* 2018: 153–163.

17. Lozano A, Wermelinger M, Nuseibeh B. Assessing the impact of bad smells using historical information. *Ninth international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting* 2007: 31–34.

18. Rapu D, Ducasse S, Gîrba T, Marinescu R. Using history information to improve design flaws detection. *Eighth European Conference on Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings.* 2004: 223–232.

19. Tufano M, Palomba F, Bavota G, et al. When and why your code starts to smell bad. *Proceedings of the 37th International Conference on Software Engineering-Volume 1* 2015: 403–414.

20. Peters R, Zaidman A. Evaluating the lifespan of code smells using software repository mining. *2012 16th European Conference on Software Maintenance and Reengineering* 2012: 411–416.

21. Marcilio D, Bonifácio R, Monteiro E, Canedo E, Luz W, Pinto G. Are static analysis violations really fixed? a closer look at realistic usage of sonarqube. *2019 IEEE-ACM 27th International Conference on Program Comprehension (ICPC)* 2019: 209–219.

22. Yamashita A, Moonen L. Do code smells reflect important maintainability aspects?. *2012 28th IEEE international conference on software maintenance (ICSM)* 2012: 306–315.

23. Yamashita A, Moonen L. Exploring the impact of inter-smell relations on software maintainability: An empirical study. *Proceedings of the 2013 International Conference on Software Engineering* 2013: 682–691.

24. Arvanitou EM, Ampatzoglou A, Bibi S, Chatzigeorgiou A, Stamelos I. Monitoring Technical Debt in an Industrial Setting. *Proceedings of the Evaluation and Assessment on Software Engineering* 2019: 123–132.

25. Zabardast E, Gonzalez-Huerta J, Šmite D. Refactoring, bug fixing, and new development effect on technical debt: An industrial case study. *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* 2020: 376–384.

26. Stol KJ, Fitzgerald B. The ABC of software engineering research. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 2018; 27(3): 1–51.

27. Saarimaki N, Baldassarre MT, Lenarduzzi V, Romano S. On the accuracy of sonarqube technical debt remediation time. 2019: 317–324.

28. Guaman D, Sarmiento P, Barba-Guamán L, Cabrera P, Enciso L. SonarQube as a tool to identify software metrics and technical debt in the source code through static analysis. *7th International Workshop on Computer Science and Engineering, WCSE* 2017: 171–175.

29. ISO/IEC/IEEE . Systems and software engineering ISO/IEC/IEEE 24765 — Vocabulary. tech. rep., 2010.

30. Fowler M, Beck K, Brant J, Opdyke W, Roberts D. *Refactoring: Improving the Design of Existing Code*. Addison Wesley. 1st ed. 1999

31. Fowler M. *Refactoring: improving the design of existing code*. Addison-Wesley Professional . 2018.

32. Fowler M. CodeSmell. 2006.

33. Brown WJ, Malveau RC, Mowbray TJ, Wiley J. *AntiPatterns: Refactoring Software , Architectures, and Projects in Crisis*. 3. Wiley . 1998.

34. NIST . NIST Special Publication 800-30 Revision 1 - Guide for Conducting Risk Assessments. Tech. Rep. September, 2012

35. ISO/IEC . IEC 27005:2018 Information technology–security techniques–information security risk management. Tech. Rep. 0, 2018.

36. Miller Jr RG. *Survival analysis*. 66. John Wiley & Sons . 2011.

37. Kaplan EL, Meier P. Nonparametric estimation from incomplete observations. *Journal of the American statistical association* 1958; 53(282): 457–481.

38. Leys C, Ley C, Klein O, Bernard P, Licata L. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology* 2013; 49(4): 764–766.

39. Palomba F, Bavota G, Penta MD, Fasano F, Oliveto R, Lucia AD. On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation. *Empirical Software Engineering* 2018; 23(3): 1188–1221. doi: 10.1007/s10664-017-9535-z

40. Rios N, Mendonça Neto dMG, Spínola RO. A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners. *Information and Software Technology* 2018; 102: 117–145.

41. Li Z, Avgeriou P, Liang P. A systematic mapping study on technical debt and its management. *Journal of Systems and Software* 2015; 101: 193–220.

42. Alves NS, Mendes TS, Mendonça dMG, Spínola RO, Shull F, Seaman C. Identification and management of technical debt: A systematic mapping study. *Information and Software Technology* 2016; 70: 100–121.

43. Martin RC. *Clean Code*. Prentice Hall . 2008.

How to cite this article: Zabardast E., K.E. Bennin, and J. Gonzalez Huerta (2021), Further Investigation of the Survivability of Code Technical Debt Items, *JSME*, *2021;00:1–6*.