Technical Debt Diffuseness in the Apache Ecosystem: A Differentiated Replication

Dario Amoroso d'Aragona,[1] Fabiano Pecorelli,[1 2] Maria Teresa Baldassarre,[3] Davide Taibi,[1 4] Valentina Lenarduzzi[4]
[1]Tampere University — [2]JADS, Eindhoven University of Technology — [3]University of Bari — [4]University of Oulu
dario.amorosodaragona@tuni.fi; f.pecorelli@tue.nl; mariateresa.baldassarre@uniba.it;
davide.taibi@oulu.fi; valentina.lenarduzzi@oulu.fi;

***Abstract*—Technical debt management is a critical activity that is gaining the attention of both practitioners and researchers. Several tools providing automatic support for technical debt management have been introduced over the last years. SonarQube is one of the most widely applied tools to automatically measure technical debt in software systems. SonarQube has been adopted to quantify the diffuseness of technical debt in projects of the Apache Software Foundation ecosystem. Lenarduzzi et al. [1] found that the vast majority of technical debt issues in the code are code smells and that, surprisingly, developers tend to take more time to remove severe issues than the less-severe ones. While this study provides very interesting insights both for researchers and practitioners interested in technical debt management, we identified some major limitations that could have led to results that do not perfectly reflect reality. This study aims to address such limitations by presenting a differentiated replication study. Our findings have pointed out significant differences with the reference work. The results show that technical debt issues appear much more rarely than what the reference work reported.**

In this study, we implemented a new methodology to calculate the diffuseness of SonarQube issues at project and commit level, based on the reconstruction of the SonarQube quality profile in order to understand how the quality profile has evolved and to compare the number of active rules per category and severity level with the respective number of issues found. The results show that over 50% of rules active in the quality profile, are Code Smell rules and that over 90% of the issues belong to Code Smell category. Furthermore, analyzing the life span of the issues, we found that developers take into account the level of severity of the issues only for the Bug category, thus fixing the issues starting from the most severe, which is not the case for the other categories.

***Index Terms*—Technical debt, SonarQube, Replication study**

I. Introduction

Software is becoming more and more important in our daily lives. From simple daily tasks to complex company management, nowadays everything is built on top of software. In such a scenario, software systems are increasing their size and complexity, hence becoming more demanding to maintain. Moreover, the constant need to adapt them to new environments and match new requirements generates a continuous and constant change process that requires developers to satisfy all requirements in the right way and in the shortest possible time. Consequently, developers often tend to overlook good programming practices and principles to deliver the most appropriate product on time [2]–[4]. This manner of programming causes the introduction of the so-called *technical debt* (TD) [5], i.e., a metaphor from the financial domain that consists in adopting (intentionally or not [6]) a limited solution instead of a better one that would require more effort.

Over the last years, several studies have proven the harmfulness of keeping technical debt alive in software systems and the negative consequences it entails. These studies have demonstrated that technical debt leads to cost increasing [7], [8], product quality decreasing [9], [10], and a slow down in the entire software development process [5], [11], thus being detrimental to software systems.

Therefore, much effort has been spent to measure technical debt, in order to make developers aware of its presence and keep it under control. Nowadays, there is a large availability of automatic static analysis tools (ASATs) for TD measurement: e.g., *Coverity Scan*, *Better Code Hub*, *Checkstyle*, *FindBugs*, *PMD*, just to list a few.

Among them, SonarQube[1] is one of the best known and most frequently adopted [12]. It allows monitoring TD evolution in software repositories and alerts developers when certain specific technical debt types go beyond a specified threshold, thus encouraging the developers' intervention.

In a recent article by Lenarduzzi et al. [1] SonarQube was adopted to investigate the diffuseness of technical debt in Java systems of the Apache Software Foundation (ASF) ecosystem. Major results of this work report that design issues are among the most likely to appear and that developers tend to resolve minor issues faster than major or critical ones. However, we observed some important limitations that could have threatened the validity of the provided findings.

Specifically, the dataset contained spurious data since SonarQube was executed a posteriori. Additionally, we found that the authors discussed their results by simply counting (and comparing) the occurrences of specific types/severities of issues, without considering the actual weight these types/severities had in the systems' quality profiles. Finally, we also observed that different time ranges were considered for different projects in the dataset, this may have affected the results because the number of issues could change significantly over time, e.g., the comparison of Technical Debt items in different projects could be significantly different if we consider the commits in one month against those in 1 year.

[1]http://www.sonarsource.org/

This paper presents a differentiated replication [13] of the study by Lenarduzzi et al. [1] that aims to address the limitations identified. In particular, we rely on a different dataset providing real data about the SonarQube usage and the issues generated considering commits in the same time range for all the projects. We also weighted the count of occurrences by considering how frequent issues of a certain type/severity were with respect to the quality profile the projects rely on.

Our findings show that the vast majority of projects/commits are characterized by a limited number of issue occurrences. Additionally, we found that code smells are significantly more diffused than the other types and that developers tend to prioritize fixing activities only for "bug" issues.

Structure of the paper: Section II describes the related literature. Section III discusses the reference work as well as its limitations. Section IV reports the methodology adopted for our replication. The results of our study are reported and discussed in Section V. Section VI discusses the potential threats to the validity of the study we conducted. Finally, Section VII concludes the article and outlines future research developments.

II. Related Work

This section reports the most relevant literature related to the diffuseness of TD. To our knowledge, the vast majority of articles look into the distribution and growth of code smells, but none of them look into SonarQube violations.

Vaucher et al. [14] looked into God Class code smells, concentrating on whether they harm software systems for lengthy periods and checking if the code smell is associated with refactoring operations.

Olbrich et al. [15] studied the evolution of God Class and Shotgun Surgery. They discovered that the distribution of these code smells is not consistent over time, increasing and decreasing with no relation to the project size.

In contrast, Chatzigeorgiou and Manakos [16], on the other hand, looked into the evolution of a larger number of code smells and discovered that the number of instances of code smells grows steadily over time. Later, Arcoverde et al. [17], corroborated this finding by studying the persistence of code smells.

More recently, Tufano et al. [18] demonstrated that the persistence of code smells in source code might result in a variety of problems. They stated that this could be caused by the fact that code smells affect code from the start of the development process, and that some code smells are introduced during refactoring efforts as well. They also claimed that nearly 80% of code smells are never removed from the code and that the remaining ones are removed by eliminating the smelly artifact rather than by reworking operations.

Digkas et al. [19] looked into the evolution of Technical Debt over five years at the weekly snapshot granularity level on 66 Apache ecosystem open-source software projects. They identified that the size, the number of issues, and the complexity indicators of the evaluated projects all increased significantly over time. However, they noticed that as the project metrics progressed, normalized Technical Debt declined. Digkas et al. [20] also looked into how Technical Debt builds up as a result of software maintenance. As a starting point, they looked at 57 open-source Java software projects from the Apache Software Foundation, analyzing them at the weekly snapshot temporal granularity level and focusing on the categories of issues that were fixed. The findings revealed that a small fraction of issue types are responsible for the majority of technical debt payback.

Amanatidis et al. [21] looked at the accumulation of Technical Debt in PHP applications, focusing on the relationship between the debt level and the interest that must be paid during corrective maintenance actions. They looked at ten open-source PHP projects in terms of corrective maintenance frequency and effort in connection to interest amount and discovered a significant association between interest and the amount of accrued Technical Debt.

Palomba et al. [22] investigated the diffuseness of 13 code smells and their impact on two software qualities: change- and fault-proneness, using 395 versions of 30 different open-source Java apps. They looked at 17.350 examples of 13 code smells that were discovered using a metric-based methodology. As a result, they got that only seven of the 13 code smells were highly dispersed, and their removal would have a significant impact on the software's change proneness. The benefit in terms of fault proneness, on the other hand, was minimal or non-existent. As a result, programmers should keep an eye on these smells and refactor them when necessary to improve the code's general maintainability.

Finally, Lenarduzzi et al. [1] presented a study about the diffuseness of technical debt issues in Apache projects. In this work they counted the occurrences of technical debt issues, analyzing what are the most frequent types and the severities that arise. They also evaluated the time taken by developers to fix issues concerning the different types and severities. As a result, they observed that most of the issues output by SonarQube are related to design concerns and that developers resolve less-severe issues faster than more severe ones.

Our article focuses on this last work, by presenting a replication study aiming to address some limitations that could have conditioned the achieved findings.

III. Reference Work

The reference work of our replication is the case study by Lenarduzzi et al. [1]. The work analyzes 33 projects from the Apache Software Foundation (ASF) ecosystem to study (i) the diffuseness of technical debt issues and (ii) the time required to fix such issues. The term *diffuseness* refers to the presence of Technical Debt (TD) items, i.e., how many TD items occur and how they are distributed among projects, level of severity and categories.

The next sections provide an overview of the methodological details of reference work and discuss the major limitations we identified.

A. Context

The context of the reference work consisted of 33 projects of different ages, sizes, and domains from the ASF ecosystem. In particular, the author's selected projects that (i) were developed in Java, (ii) were older than three years, (iii) had more than 1000 commits, (iv) contained more than 100 classes, and (v) used an issue tracking system with at least 100 issues.

B. Design

Intending to identify the diffuseness of Technical Debt issues in the source code, concerning the type and the severity level of SonarQube issues, Lenarduzzi et al. [1] designed a case study [23] revolved around three research questions.

First, Lenarduzzi et al. [1] aimed to assess the diffuseness of SonarQube issues in Apache software systems (RQ_1). Then, they conducted a more in-depth analysis to study the diffuseness of technical debt for the type and the severity level of TD issues (RQ_2). Finally, they also investigated the time needed to resolve TD issues, according to type and severity (RQ_3).

C. Data Collection & Analysis

All the projects in the dataset were cloned from GitHub and then each commit was analyzed by running SonarQube. The authors in Lenarduzzi et al. [1] collected information about all the violations, relying on the default rule set, and the time needed to fix each rule. Then, they counted the occurrences for each rule at a project and commit level and also examined the correlations between the occurrences of technical debt and the number of classes, methods, and lines of code in a commit by relying on the Spearman correlation coefficient (RQ_1).

In the context of RQ_2, they grouped the issues by type and severity to compare and comment on the diffuseness of different type/severity groups.

Finally, in RQ_3, they examined the number of days needed to resolve issues by inspecting the SonarQube analyses of subsequent commits.

D. Limitations

In the work presented by Lenarduzzi et al. [1] we identified three major limitations that we aim to address in this paper. First, in their study, they relied on the violations identified by SonarQube to measure the diffuseness of technical debt issues. However, to perform their study they launch SonarQube a posteriori on the whole dataset but, unfortunately, this approach does not allow to track the issues very precisely. As a matter of fact, in some cases they found duplicated issues, referring to the same lines of code and the same problem but with different resolution times. In addition, launching SonarQube a posteriori does not take into account the customization and the usage of a custom quality profile, that allows the developer to enable/disable certain rules.

A quality profile in Sonar is a set of rules that are used to analyze the code. SonarCloud has a default quality profile for each programming language, called SonarWay, which is not editable. Users can define their quality profile by choosing which rules to activate/deactivate. In other words, a quality profile defines the rules used to find the violations in the code.

In the context of RQ_2, they counted issues' occurrences grouped by type and severity. We think that such a count could be not accurate enough to represent the actual weights of specific type/severity groups. In particular, we think that the results are conditioned by the actual quality profiles the systems are using, i.e., if a quality profile includes a low number of issues of a certain type/severity, such type/severity will likely result in a lower number of occurrences. Therefore, comparing the number of occurrences in different types/severity groups to each other could not be fair enough. Finally, we also found that the authors of the reference article relied on different time ranges for different projects. This could have led to inaccurate findings when comparing the issues' occurrences among the different projects (RQ_2).

Our work aims to address all these limitations by (i) changing the input dataset and, as a consequence, the data collection phase, and (ii) considering weights when counting issues in specific type/severity groups. The next section discusses our methodology in detail.

IV. Differentiated Replication

The goal of our empirical study is **to assess** the sensitivity of the results of the reference work, namely the study by Lenarduzzi et al. [1], **with the purpose** of understanding whether the findings provided are still observable **from the viewpoint of** both researchers and practitioners, **in a realistic context**. We have addressed the perspective of researchers and practitioners as they are both interested in understanding to what extent technical debt is actually diffused in software systems and what is the time usually required to fix specific issue types.

To this aim, we performed a differentiated replication of the reference study by relying exactly on the same three research questions (i.e., the ones reported in Section III). However, we made some methodological changes that were necessary to overcome the major limitations identified in the work by Lenarduzzi et al. [1]. Specifically, we reported major changes in the context selection, the data collection, and the data analysis phases. The resulting research questions are listed below:

> **RQ_1:** *What is the diffuseness of Technical Debt issues in software systems when real usage data are considered?*

This RQ mirrors the RQ_1 from the original work of Lenarduzzi et al. [1]. Lenarduzzi et al.'s answer to this RQ considered different types and levels of severity for each TD item, to determine how the rules are grouped between different values of severity and type, and the relative distribution in the projects of rules among severity levels and types. We replicate this RQ with the same methodology using an updated dataset, to understand if the results of Lenarduzzi et al. are generalizable or not.

RQ_2: *What is the diffuseness of Technical Debt issues in software systems considering weights of different types and severity levels when real usage data are considered?*

This RQ mirrors the RQ_2 from the original work of Lenarduzzi et al. [1]. Lenarduzzi et al. does not consider the number of rules active per types and severity levels. Is easy to verify that the number of issues for a specific type/severity level is proportional to the number of rules active for that types/severity level.

RQ_3: *What is the lifespan of Technical Debt issues when real usage data are considered in the same time interval?*

This RQ mirrors the RQ_3 from the original work of Lenarduzzi et al. [1]. Lenarduzzi et al. in their works used a dataset that includes projects of very different ages and that where SonarQube was run a posteriori as described in section III-D. These limitations affects the results, in particular is hard to determine a real lifespan of issue if the developers are not conscious about the existence of that issue in the code. For this reason we analyzed projects where SonarQube is used by the developers in development process.

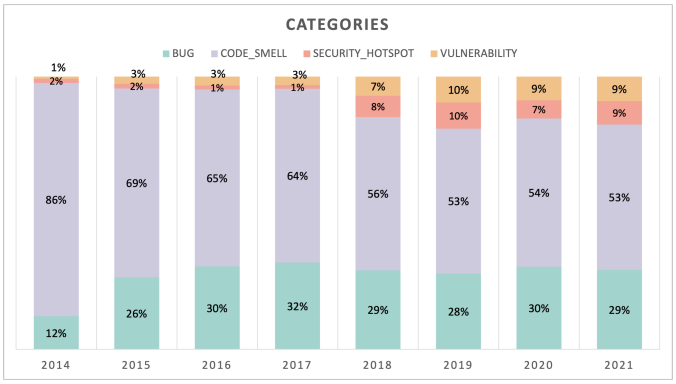
A. Context Selection and Data Collection

One of the major limitations of the reference work was about launching SonarQube analyses a posteriori (see Section III-D). To overcome this issue, we relied on the information available on Pandora [24]. This tool allowed us to access all the *SonarCloud* [2] analyses actually performed on 96 Apache projects [3]. Therefore, we were able to collect actual information about the developer's usage of SonarQube as well as the issues raised by the tool. In this way we were able to reliably reconstruct the history of the analyses, knowing exactly, for each issue, the time range from when the issue has been raised to when it has been closed.

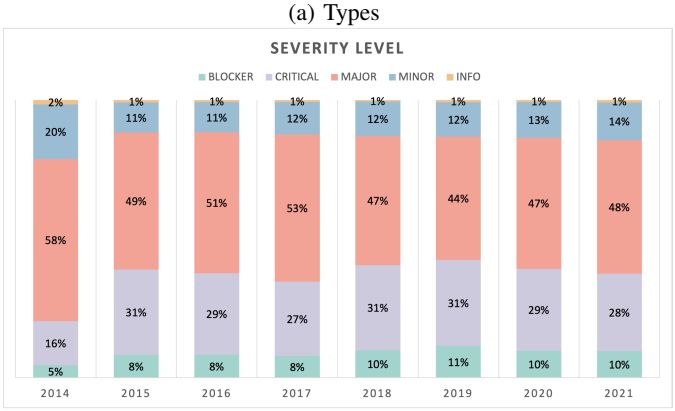
Moreover, to overcome the threat of considering different periods for the different projects, we based our analysis on all the commits from 2018 to 2021 for all the 96 projects. The reason for the selection of this specific time range relies on the evolution of the quality profiles adopted by the projects in our dataset. More details are reported in the following section.

B. Data Analysis

Another important limitation of the reference work was to count and compare occurrences of specific types without considering weights, as explained in Section III-D. Our strategy to overcome this limitation was to give weights to specific type/severity groups, by counting how many issues in each specific group appear in the quality profile the projects rely on. With this respect, we performed a preliminary analysis of the quality profiles of the considered projects. In particular, first, we investigated whether the projects in our dataset used



(a) Types



(b) Severity levels

Fig. 1: Stacked bar plots reporting the percentage of active rules in SonarQube quality profile from 2013 to 2021 per type (a) and severity level (b)

a customized quality profile, hence relying on different sets of rules. As a result, we found that all the projects relied on the default Java quality profile, namely SonarWay [4].

However, SonarWay is in a continuous update process and, unfortunately, there is no track of these updates. Therefore, to calculate weights that are as accurate as possible, we reconstructed the quality profile from 2013 to 2021 using the information available in the changelog and the creation date of each rule in the current quality profile. The result of this preliminary study, reported in Figure 1 shows how the number of active rules for each type and severity-level remains almost constant over the years, in particular considering the period from 2018 to 2021 (we excluded from this analysis 2022 because there are not enough information in the changelog). In addition, the percentages for the year 2021 reported in Figure 1, also reflect the actual quality profile at the time of the study, i.e., Feb 16th 22. Once assessed the stability of the quality profile over time, we calculated the weights for each type and severity level. To this aim we first counted the frequency for each type/severity level; then we calculate the weight as 1 - the resulting percentage. This choice was made to give more importance to the rules that are less likely to appear. In

[2]https://sonarcloud.io

[3]Dataset description available in our replication package [25]

[4]https://docs.sonarqube.org/latest/instance-administration/quality-profiles/

TABLE I: Weights calculated by type and severity level

	Group	Weight
Type	BUG	0.71
	CODE SMELL	0.46
	VULNERABILITY	0.90
	SECURITY HOTSPOT	0.93
Severity	BLOCKER	0.91
	MAJOR	0.56
	CRITICAL	0.80
	MINOR	0.73
	INFO	0.99

particular, we relied on the following formulas. The resulting weights are reported in Table I.

$$severity_{weight}(s) = 1 - \frac{\#rules\,of\,severity\,s}{\#\,rules\,in\,QP} \quad (1)$$

$$type_{weight}(t) = 1 - \frac{\#rules\,of\,type\,t}{\#\,rules\,in\,QP} \quad (2)$$

V. Results and Discussion

This section reports and discusses the results obtained from our investigation.

A. RQ_1: diffuseness of Technical Debt items

Figure 2 reports the distribution of the 40 rules with the highest percentage of commits affected. For each SonarCloud rule-identifier, the figure shows the number of issues generated by each rule on a log scale. As we can observe, the distribution does not vary significantly between items. Table II reports the descriptive statistics of the distributions of the top-40-commit-affect rules for commit and project (the minimum is not reported because it is always zero).

As we can observe from the table, the occurrences are not constant on average. For instance, if we consider the average per commit, the minimum value is 1.72 (suggesting a large number of commits with few issues), and the maximum is 55.44. Moreover, analyzing Figure 4, it is possible to see that the distributions for all rules per commit are skewed to the right, thus indicating that there are many commits/projects with few or zero occurrences.

When comparing our results with those of the reference work, we can observe the first substantial differences. Specifically, in the reference work, the authors found single rules to be way denser in their dataset, both at the commit- and project-level. Similarly, also in the percentage of affected commits, there is a very large difference between our results and the ones of the reference work. The reason behind such a difference could rely on the fact that we considered real SonarQube usage data instead of running the tool externally. Indeed, by going through the raw data, available in our online appendix [25], it is possible to observe that the vast majority of the analyzed commits only contain a limited number of rules among the entire rule list.

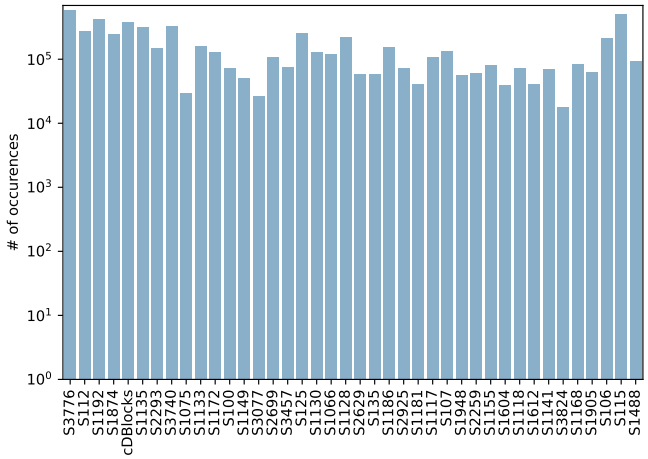


Fig. 2: Distribution of the 40 most common violated rules in the dataset (RQ_1)

Summing Up RQ_1: Considering real SonarQube usage data instead of running the tool externally leads to a different issues distribution with respect to the reference work. Specifically, our study reports a way lower diffuseness of SonarQube issues both at the commit- and project-level.

B. RQ_2: Diffuseness of TD items by type and severity

To answer the second research question (RQ_2), we used the weight formula (1) (2) described in Section (IV-B) to consider real distributions of issues, i.e., taking into account the number of active rules per severity and type.

Table III reports the count of active rules grouped by severity and type, the number of introduced items (the number of issues related to the corresponding severity-level/type), the weighted number of introduced items, the weighted percentage of introduced items. In particular, the *number of introduced items* represents the total number of issues for each type and severity level, instead, the *weighted number of introduced items* is the total number of issues obtained using the weighted methodology described before and the *weighted percentage of introduced items* is the percentage of issues for each type and severity calculated based on the *weighted number of introduced items*.

In Table IV we reported the average instances per commit, the number of max instances in a commit, and the percentage of the affected commits.

From the results, it is possible to observe how the *CODE SMELL* instances are reduced by a factor of ≈0,5 by comparing the *INTRODUCED ITEMS* and the *INTRODUCED ITEMS WEIGHTED* columns, according to the weights reported in Table I. However, even if such a weight is considered, this type of issue still represents the ≈95% of the total issues and arises in 97% of the commits. Way smaller percentages are reported for *BUG* and *VULNERABILITY* issue types that represent, respectively, ≈4% and ≈1% of the total issues. As

TABLE II: Distribution of the 40 most common violated rules among commit and projects (RQ_1)

	General Info			*Per commit*						*Per Project*					
rule	**# instances**	**max instances**	**% affected commits**	**25%**	**mean**	**std**	**50%**	**75%**	**max**	**25%**	**mean**	**std**	**50%**	**75%**	**max**
S3824	17988	1361	19%	0	1.72	17.72	0	0	1361	0	179.88	1798.80	0	0	17988
S3776	580673	46589	52%	0	55.44	591.48	1	6	46589	0	5806.73	58067.30	0	0	580673
S3740	322934	26342	24%	0	30.83	388.58	0	0	26342	0	3229.34	32293.40	0	0	322934
S3457	74748	5809	22%	0	7.14	80.84	0	0	5809	0	747.48	7474.80	0	0	74748
S3077	26735	779	23%	0	2.55	20.71	0	0	779	0	267.35	2673.50	0	0	26735
S2925	73195	6107	20%	0	6.99	83.17	0	0	6107	0	731.95	7319.50	0	0	73195
S2699	107187	8064	22%	0	10.23	112.70	0	0	8064	0	1071.87	10718.70	0	0	107187
S2629	57638	2815	21%	0	5.50	54.96	0	0	2815	0	576.38	5763.80	0	0	57638
S2293	147118	28829	27%	0	14.05	303.48	0	1	28829	0	1471.18	14711.80	0	0	147118
S2259	59918	4941	20%	0	5.72	75.04	0	0	4941	0	599.18	5991.80	0	0	59918
S1948	57119	8939	20%	0	5.45	98.84	0	0	8939	0	571.19	5711.90	0	0	57119
S1905	63074	12259	18%	0	6.02	130.24	0	0	12259	0	630.74	6307.40	0	0	63074
S1874	243630	10672	32%	0	23.26	205.79	0	1	10672	0	2436.30	24363.00	0	0	243630
S1612	40335	2158	19%	0	3.85	36.03	0	0	2158	0	403.35	4033.50	0	0	40335
S1604	39607	2501	20%	0	3.78	34.48	0	0	2501	0	396.07	3960.70	0	0	39607
S1488	92373	9593	17%	0	8.82	142.46	0	0	9593	0	923.73	9237.30	0	0	92373
S135	58630	4231	21%	0	5.60	66.11	0	0	4231	0	586.30	5863.00	0	0	58630
S125	255734	20138	22%	0	24.42	313.12	0	0	20138	0	2557.34	25573.40	0	0	255734
S1192	418124	43517	32%	0	39.92	578.73	0	2	43517	0	4181.24	41812.40	0	0	418124
S1186	155876	15596	21%	0	14.88	184.72	0	0	15596	0	1558.76	15587.60	0	0	155876
S1181	40192	3322	20%	0	3.84	40.75	0	0	3322	0	401.92	4019.20	0	0	40192
S1172	128626	12147	23%	0	12.28	151.87	0	0	12147	0	1286.26	12862.60	0	0	128626
S1168	82657	7087	18%	0	7.89	92.45	0	0	7087	0	826.57	8265.70	0	0	82657
S1155	81319	9257	20%	0	7.76	111.98	0	0	9257	0	813.19	8131.90	0	0	81319
S115	497804	64327	18%	0	47.53	869.74	0	0	64327	0	4978.04	49780.40	0	0	497804
S1149	51178	4163	23%	0	4.89	54.98	0	0	4163	0	511.78	5117.80	0	0	51178
S1141	68798	9909	19%	0	6.57	104.34	0	0	9909	0	687.98	6879.80	0	0	68798
S1135	314021	18075	29%	0	29.98	323.08	0	1	18075	0	3140.21	31402.10	0	0	314021
S1133	156935	17845	23%	0	14.98	217.09	0	0	17845	0	1569.35	15693.50	0	0	156935
S1130	127671	18681	22%	0	12.19	207.56	0	0	18681	0	1276.71	12767.10	0	0	127671
S1128	221197	10782	22%	0	21.12	339.85	0	0	10782	0	2211.97	22119.70	0	0	221197
S112	275502	19397	33%	0	26.30	266.81	0	2	19397	0	2755.02	27550.20	0	0	275502
S1118	73413	4920	19%	0	7.01	83.71	0	0	4920	0	734.13	7341.30	0	0	73413
S1117	106318	10415	20%	0	10.15	141.97	0	0	10415	0	1063.18	10631.80	0	0	106318
S1075	29559	2840	23%	0	2.82	36.72	0	0	2840	0	295.59	2955.90	0	0	29559
S107	131515	21404	20%	0	12.56	277.32	0	0	21404	0	1315.15	13151.50	0	0	131515
S1066	121443	12396	22%	0	11.59	159.09	0	0	12396	0	1214.43	12144.30	0	0	121443
S106	209179	15215	18%	0	19.97	225.82	0	0	15215	0	2091.79	20917.90	0	0	209179
S100	71413	3917	23%	0	6.82	77.78	0	0	3917	0	714.13	7141.30	0	0	71413
cDBlocks	371433	15005	30%	0	35.46	305.62	0	2	15005	0	3714.33	37143.30	0	0	371433

TABLE III: Weighted distribution of issues among type and severity (RQ_2)

TYPE/SEVERITY	**ACTIVE RULES**	**INTRODUCED ITEMS**	**INTRODUCED ITEMS (W)**	**% INTRODUCED ITEMS (W)**
BUG	139	308393	220190.0638	4%
CODE SMELL	261	11761250	5445023.148	95%
VULNERABILITY	50	91773	82331.33333	1%
BLOCKER	43	264942	241500.6296	3%
CRITICAL	95	2106937	1695087.175	20%
MAJOR	215	4882056	2722298.716	33%
MINOR	129	3834150	2816443.519	34%
INFO	4	1073331	1064497	13%

a result, we can observe that when weights are considered to count different types of SonarQube issues, our results are in contrast with the reference work. Indeed, they found all the types of issues to affect more than 90% of the commits.

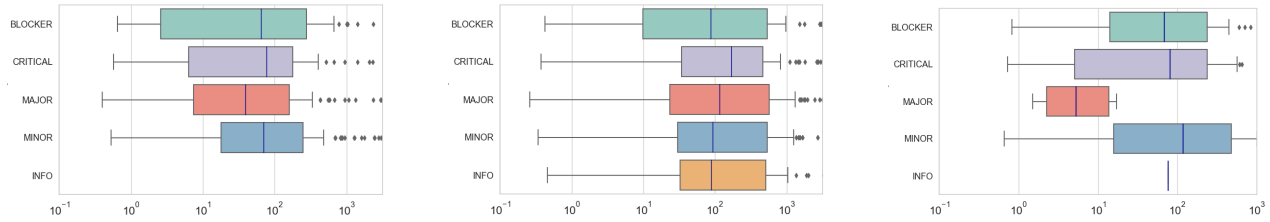
The same discussion can be made for the severity levels: also, in this case, we have very discordant results with respect to the work by Lenarduzzi et al. [1].

While these differences could depend on the fact that we weighted the occurrences of issues per group, we still think that there could have been some measurement errors when running SonarQube a posteriori, as already discussed in Section V-A.

Figure 3 and Figure 4 report the distribution for the issues among types an severities, the former shows the distribution a project-level while the latter the distribution at commit level. At the project-level, the most common issues are *CODE SMELL-INFO*, *CODE SMELL-MINOR*, *CODE SMELL-MAJOR*, *VULNERABILITY-MINOR*, *VULNERABILITY-BLOCKER* issues, while at the commit-level *CODE SMELL-INFO* are the most common issues. The distribution varies largely between types, severities, projects, and commits except for *CODE SMELL* at the project level, where the distribution between the *INFO, MINOR* and *MAJOR* is very similar. Another interesting data to observe is that for *BUG* and *VULNERABILITY* there are no issues with *INFO* severity. This can be related to the fact that these types, identify

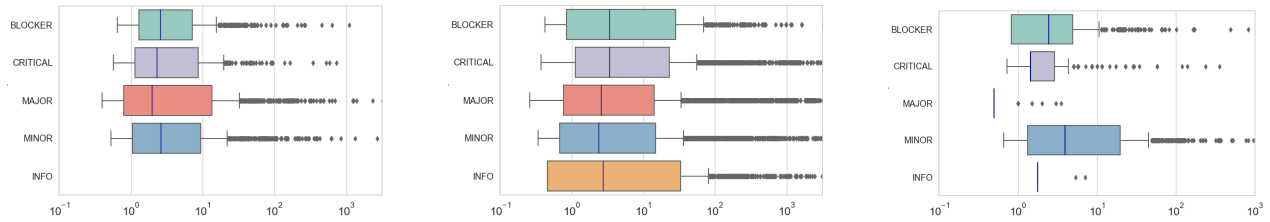
TABLE IV: Distribution of issues at commit level (RQ_2)

TYPE/SEVERITY	AVG INSTANCES PER COMMIT	MAX INSTANCES IN COMMIT	% AFFECTED COMMITS
BUG	29.20	14966	46%
CODE SMELL	1113.44	103209	97%
VULNERABILITY	8.69	2330	24%
BLOCKER	25.08	15143	34%
CRITICAL	199.46	46589	63%
MAJOR	462.18	50323	82%
MINOR	362.98	103209	79%
INFO	101.61	39729	39%



(a) Bug (b) Code Smell (c) Vulnerability

Fig. 3: Issues distribution among projects (RQ_2)



(a) Bug (b) Code Smell (c) Vulnerability

Fig. 4: Issues distribution among commit (RQ_2)

by design more severe rules.

With the aim of studying the average contribution of types/severities issues in our dataset, we also report stacked bar plots reporting the weighted distributions of issues in each project among type and severity. For the sake of space limitation, we only included them in our online appendix [25].

Summing Up RQ_2: When weights are taken into account to count different types/severities of SonarQube issues, our results report results that are in contrast with the reference work. While in Lenarduzzi et al. [1] authors found all the types of issues to affect more than 90% of the commits, in our case this only applies for *Code Smell* issues.

C. RQ_3: What is the lifespan of TD issues

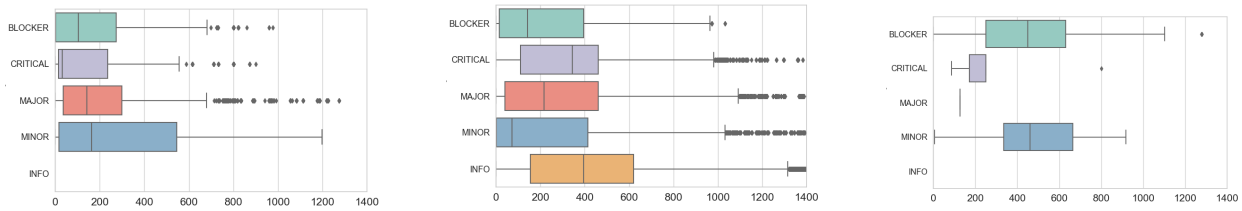
Table V reports the number of days needed to resolve the issues grouped by type and severity. Considering the severity, it seems that the average of days needed to close a *BLOCKER* issue is smaller than the days needed to close a *INFO* issue. *MINOR* issues have the smallest value. By deeper inspecting this aspect, it seems that the developers of the considered projects start first fixing *BUG* issues, then *CODE SMELL* issues, and finally *VULNERABILITY* issues.

Figure 5 reports the time taken to fix issues considering combinations of severities and types. As for *BUG* issues, the results are in line with the expectations, namely, the higher the severity, the faster the resolution. The same does not apply to the other two categories, where the fixing time seems not to be correlated with the severity. Indeed, as for *CODE SMELL* issues, we can observe that *MINOR* issues are solved faster than *MAJOR* and *BLOCKER* ones. Similarly, for *VULNERABILITY* issues, *BLOCKER* issues require more time to be solved with respect to *CRITICAL* and *MAJOR* ones. This result can be explained by the nature of the issue types themselves. Indeed, high-severity bugs can cause relevant issues in the code, hence developers tend to solve them faster. Code smells, instead, are associated with design issues that do not necessarily lead to issues in the short period.

Figure 6a and Figure 6b show the weighted distribution of the issues that were still open at the time of our last analysis (April 22th, 2022). We only considered issues opened from 2018 to 2021. According to the results, *CODE SMELL* issues represent more than 90% of the total open issues. Once again,

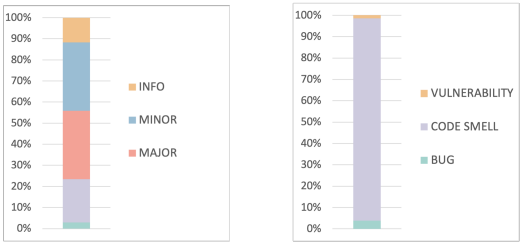
TABLE V: Number of days needed to resolve issues (RQ_3)

TYPE/SEVERITY	AVG	MEDIAN	MAX	STDEV
BLOCKER	242.85	**149.18**	1279.91	252.23
CRITICAL	302.36	344.46	1437.55	227.64
MAJOR	284.82	221.06	1526.40	243.22
MINOR	**201.46**	**84.70**	1557.07	244.87
INFO	**461.11**	395.46	1525.16	370.70
CODE_SMELL	281.17	195.44	1557.07	282.59
BUG	232.11	132.10	1275.27	269.26
VULNERABILITY	394.95	426.75	1279.91	258.33



(a) Bug (b) Code Smell (c) Vulnerability

Fig. 5: Time to resolve issues among category and severity per project (RQ_3)



(a) (b)

Fig. 6: Stacked bar plots reporting the weighted distribution of open issues per severity (a) and type (b) (RQ_3)

the results indicate that developers tend to give less importance to design issues, keeping it into the code and providing fixes only at a later time.

Inspecting the average time to fix an issue we discovered some rules that are never been fixed in these years, thus no one issue raised by the violation of one of these rules has been closed by a developer. In total there are 49 rules never fixed, 22 are *CODE SMELL*, 14 *BUG*, 8 *SECURITY HOTSPOT* and 5 *VULNERABILITY*. Moreover, the issues raised by the violation of these rules have been introduced in a time range that goes from 2018 to 2021, with a peak in 2020 (4600 issues introduced over 10.000 of total issues never fixed until now). Investigating the number of issues fixed and the number of open issues we discover that over 12 million issues, about 11 million are still open, and only 7% of issues have been resolved.

Summing Up RQ_3: Developers tend to fix bug issues faster than vulnerability and code smell issues. For bug issues the severity seems to play an important role, thus more severe issues are solved faster than less severe ones. The same does not apply to code smells and vulnerabilities.

VI. Threats to Validity

This section reports the threats to construct, internal, and external validity of the experimentation we conducted and presented in this study.

Construct Validity. To answer our RQs we focused on the Sonar quality profile and its role in the distribution of the issues among severity and type. Unfortunately, Sonar does not save the history of the quality profile, so we reconstructed it based on the information available (changelog, date of rule creation). We performed the quality profile reconstruction in a very accurate way, however, we are aware that it could still contain some approximation errors.

Furthermore, we are aware that not all issues raised by Sonar are Technical Debt issues, but given the nature of this study, we preferred to be consistent with the replicated work and refer to all issues raised by Sonar as Technical Debt items like Lenarduzzi et al. [1] did in their work.

Internal Validity. We have chosen to analyze commits in a specific date frame (from 2018 to 2021). The reason for this choice regards the consistency of the quality profile over the years. Therefore, we are aware that the analyzes carried out are limited by this factor. Hence, we are conscious that this choice did not allow us to carry out analyzes on a larger scale and therefore consider a longer period of time, which could have returned more precise results.

External Validity. We selected all Java projects belonging to Apache Foundation stored in Pandora [24] system. However, even if these projects present different characteristics, we are aware that other projects, in others organizations, using different Sonar quality profiles could reach different results.

VII. Conclusion

In this work, we have presented a replication of the study by Lenarduzzi et al. [1]. Starting from three major limitations

of the reference study, we addressed them and repeated all the analyses again. The results achieved, pointed out a set of differences with the reference work. As for RQ_1, we found SonarQube issues to be way less diffused than what previously reported. In the context of RQ_2, we obtained similar outcomes, finding that issues of specific types/severities are less likely to appear in projects and commits with respect to the reference study. Finally, in RQ_3 we achieved the same outcome of the reference work only for two out of the three types of SonarQube issues. Specifically, we found that in the case of "bug" issues higher severities lead to faster fixing interventions.

REFERENCES

[1] V. Lenarduzzi, N. Saarimaki, and D. Taibi, "On the diffuseness of code technical debt in java projects of the apache ecosystem," in *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)*. IEEE, 2019, pp. 98–107.

[2] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya *et al.*, "Managing technical debt in software-reliant systems," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*, 2010, pp. 47–52.

[3] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *Ieee software*, vol. 29, no. 6, pp. 18–21, 2012.

[4] F. Shull, D. Falessi, C. Seaman, M. Diep, and L. Layman, "Technical debt: Showing the way for better transfer of empirical results," in *Perspectives on the Future of Software Engineering*. Springer, 2013, pp. 179–190.

[5] W. Cunningham, "The wycash portfolio management system," *ACM SIGPLAN OOPS Messenger*, vol. 4, no. 2, pp. 29–30, 1993.

[6] E. Allman, "Managing technical debt," *Communications of the ACM*, vol. 55, no. 5, pp. 50–55, 2012.

[7] P. Thibodeau, "Counting 'technical debt'," *Information Age*, vol. 64, 2011.

[8] Y. Guo, R. O. Spínola, and C. Seaman, "Exploring the costs of technical debt management–a case study," *Empirical Software Engineering*, vol. 21, no. 1, pp. 159–182, 2016.

[9] A. Yamashita and L. Moonen, "Do code smells reflect important maintainability aspects?" in *2012 28th IEEE international conference on software maintenance (ICSM)*. IEEE, 2012, pp. 306–315.

[10] F. Palomba, G. Bavota, M. D. Penta, F. Fasano, R. Oliveto, and A. D. Lucia, "On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1188–1221, 2018.

[11] E. Lim, N. Taksande, and C. Seaman, "A balancing act: What software practitioners have to say about technical debt," *IEEE software*, vol. 29, no. 6, pp. 22–27, 2012.

[12] V. Lenarduzzi, A. Sillitti, and D. Taibi, "A survey on code analysis tools for software maintenance prediction," in *International Conference in Software Engineering for Defence Applications*. Springer, 2018, pp. 165–175.

[13] M. D. Uncles and S. Kwok, "Designing research with in-built differentiated replication," *Journal of Business Research*, vol. 66, no. 9, pp. 1398–1405, 2013, advancing Research Methods in Marketing.

[14] S. Vaucher, F. Khomh, N. Moha, and Y. Gueheneuc, "Tracking design smells: Lessons from a study of god classes," in *Working Conference on Reverse Engineering*, 2009, pp. 145–154.

[15] S. Olbrich, D. S. Cruzes, V. Basili, and N. Zazworka, "The evolution and impact of code smells: A case study of two open source systems," in *Int. Symp. on Empirical Software Engineering and Measurement*, 2009.

[16] A. Chatzigeorgiou and A. Manakos, "Investigating the evolution of bad smells in object-oriented code," in *Int. Conference on the Quality of Information and Communications Technology*, 2010, pp. 106–115.

[17] R. Arcoverde, A. Garcia, and E. Figueiredo, "Understanding the longevity of code smells: Preliminary results of an explanatory survey," in *Workshop on Refactoring Tools*, 2011, pp. 33–36.

[18] P. F. Tufano, M., G. Bavota, R. Oliveto, M. Di Penta, A. De Lucia, and A. Poshyvanyk, "When and why your code starts to smell bad (and whether the smells go away)," *IEEE Transactions on Software Engineering*, vol. 43, no. 11, pp. 1063–1088, Nov 2017.

[19] G. Digkas, A. C. M. Lungu, and P. Avgeriou, "The evolution of technical debt in the apache ecosystem." Springer, 2017, pp. 51–66.

[20] G. Digkas, M. Lungu, P. Avgeriou, A. Chatzigeorgiou, and A. Ampatzoglou, "How do developers fix issues and pay back technical debt in the apache ecosystem?" in *Int. Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018, pp. 153–163.

[21] T. Amanatidis, A. Chatzigeorgiou, and A. Ampatzoglou, "The relation between technical debt and corrective maintenance in php web applications," *Information and Software Technology*, vol. 90, 2017.

[22] F. Palomba, G. Bavota, M. D. Penta, F. Fasano, R. Oliveto, and A. D. Lucia, "On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1188–1221, Jun 2018.

[23] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[24] H. Nguyen, F. Lomio, F. Pecorelli, and V. Lenarduzzi, "Pandora: Continuous mining software repository and dataset generation," in *EEE International Conference on Software Analysis, Evolution and Reengineering (SANER2022). IEEE*, 2022.

[25] D. Amoroso d'Aragona, F. Pecorelli, M. T. Baldassarre, and V. Lenarduzzi, "Online appendix," 2022. [Online]. Available: https://github.com/darioamorosodaragona-tuni/TD-Diffuseness-in-Apache-A-Replication-Study-Online-Appendix