On the Technical Debt Prioritization and Cost Estimation with SonarQube tool

Andrej Katin[1], Valentina Lenarduzzi[2], Davide Taibi[3], and Vladimir Mandić[1]

University of Novi Sad, Novi Sad, Serbia[1]
katin@uns.ac.rs, vladman@uns.ac.rs
LUT University, Lahti, Finland[2]
valentina.lenarduzzi@lut.fi
Tampere University, Tampere, Finland[3]
davide.taibi@tuni.fi

Abstract. Software developers commonly faced with situations to compromise internal quality ta=hat achieve short term goals, e.g. time-to-market. In software engineering, such compromises are described with Technical Debt (TD) concept. TD implies cost of additional rework—usually expressed as effort—and when the code is compromised, it is called code debt. One of the most popular tools for identifying TD items and estimating effort for solving them is SonarQube. However, there is still a need for empirical validations of remediation times that SonarQube estimates. The objective of this research is to validate the usefulness of the tool for junior developers in terms of understanding SonarQube's categorization of the criticality levels of TD issues and the accuracy of estimated remediation times. We designed and conducted an empirical study with 185 students in a context of university software engineering courses in Finland and Serbia. The estimates provided by SonarQube are more pessimistic, i.e. only in 3% of cases the actual fixing time was higher that the estimated one. Our results also indicate that participants' perception of the criticality levels of the TD issues is misaligned with SonarQube's classification and prioritization.

Keywords: Technical debt · Empirical study · Technical debt management · SonarQube

1 Introduction

Commonly, software developers are faced with situations to compromise internal quality to achieve short term goals, e.g. time-to-market. In software engineering, such compromises are described with Technical Debt (TD) concept [1]. TD implies cost of additional rework—usually expressed as effort—and when the code is compromised, it is called code debt [18].

The first mention of the TD concept is related to late 90's when Cunningham defined TD as a metaphor that will help to make some aspects of internal software quality visible [14, 17]. TD contextualizes the problem of outstanding work in software development tasks (e.g. pending refactoring) as a debt that provides short-term benefits in software development that will be paid much harder in the later stages of development, i.e. paid with interest [1, 18]. The debt metaphor reminds developers about the choices that can be made with design flaws [5].

The concept of TD can be particularly useful when used by junior developers, who nowadays represent the large majority of developers. However, some empirical investigations indicate that junior developers are less familiar with the concept of TD than their more experienced colleagues [11, 15].

Different Automated Static Analysis Tools (ASAT) can be used to identify TD [2]. SonarQube (SQ)[1] is one of the first ASAT tools that provides estimates of the accumulated TD in source code. It utilizes different code metrics and a large number of coding rules that are defined for different programming languages [12].

In this research, we aim to determine how accurate SQ's time estimates are when used by junior developers, as well as whether their perception of TD issues criticality matches SQ's categorization. For this purpose, we designed and conducted an empirical study with 185 junior developers in academic setting.

Results of this study can be of interest to industry practitioners who are planning to adopt the SQ on their software projects to better interpret estimated remediation times, especially when remediation tasks are assigned to junior developers.

This paper is structured as following. Section 2 presents a literature review on this topic. Section 3 describes the empirical design of the research. Section 4 presents the results obtained by the research. Section 5 identifies the threats to the validity of this paper and Section 6 defines the conclusion and future work.

2 Related Work

In this section we introduce all relevant aspects of the SonarQube for this study, alongside with an overview of the literature regarding empirical investigations of TD estimations with SQ. There are not many contributions on the topic of the use of different ASAT tools for improving the quality of developers work.

2.1 SonarQube

SQ identifies three basic types of TD issues. (1) **Code smells** are points in the code that can be difficult to maintain. Also, issues can be classified as security (2) **Vulnerabilities** or (3) **Bugs** that will cause the breaking of code. After identifying an issue, SQ estimates *remediation effort* or the time needed to remove that issue. [13]

SQ provides users with categorization of identified issues according to the level of critically at five categories [19]. *Blocker* represents most critical level of SQ issue for bugs with a high probability to affect program misbehavior. *Critical* are issues that represents bugs with low probability to impact the program or a

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

major security flow. *Major* are problems that can significantly affect the productivity of developers. *Minor* are issues that have minimal impact on productivity [19]. *Info* is issue that provides information to developers [19].

2.2 Empirical Investigations of TD estimates

The first empirical papers on the topic of TD identification using the SQ tool were written in the mid 2010s when SQ began to be used [6, 14]. The papers mainly focused on improving the metrics and rules used by SQ. Characteristically, research has been done on open-source projects. [6, 23, 24]

Researches in 2019 and 2020 is aimed at improving the list of SQ rules [7, 21, 12, 13, 16]. During these years, the papers have been extended to academic purposes, which includes students as the main respondents [13, 16]. Machine Learning is introduced as one of the solutions to emerge improving performance (e.g. use of SZZ algorithm) [12, 3, 10].

Some papers are also focused on estimating the time to solve the identified SQ issues. The researches have come to the conclusion that SQ gives overestimated remediation times and that the percentage of accurate estimates is even less than 20 percent. That can easily affect inexperienced developers productivity. [20, 7, 8]

This research is focused on the time estimate provided by SQ and prioritization of SQ issues which is very interesting topic that no specific research has yet been done as far as we know.

3 Study Design

In this section, we present the research question that are the main focus of this paper, protocol for data collecting, and execution of the study.

3.1 Research Questions

RQ1. How accurate are SQ's remediation times in the case of junior developers? Answering this question should inform us what to expect when remediation tasks are assigned to junior developers with respect to the estimated effort, i.e. cost.

RQ2. How do junior developers categorize criticality of TD issues identified by SQ? This RQ investigates whether the perceived criticality of identified TD issues by junior developers alignes with SQ's criticality levels.

3.2 Data Collection

In this Section, we give an overview of the case study protocol, the selection of the participants, and the follow-up survey.

Case study protocol The research was conducted in a context of university courses. As a part of the student assignments, i.e. projects, SQ was used as a tool for static code analysis. Students had to correct all generated issues and record the actual time required to resolve them. Upon the completion of assignments, a follow-up survey was conducted.

Subjects We selected participants for the two replications from two University in the context of a software engineering courses. Students are divided into teams of six to eight people. Teams were formed in the following way. The first members of the teams were picked by instructors, afterward each picked member was instructed to pick one next member from remaining students until the groups are formed.

Follow-up survey The follow-up survey was in English and it contained a list of up to ten the most frequent TD issues that students had to evaluate the criticality for each issue in the survey. A four-point ordinal scale from *Not concerned at all* to *Very concerned* was offered for the answer to each question and each team was given a list of approximately ten issues that were most common in their project.

3.3 Execution of the study

Authors from Finland conducted the research as a part of a course at Tampere University, while authors from Serbia conducted the research at the University of Novi Sad.

The Case Finland implied that each team of students participating in the research must select one Java open-source project. Further work is focused on the analysis performed by SQ and the correction of identified errors (e.g. refactoring). Students were instructed to use SQ (community Version 7.4 (build 18908)).

The Case Serbia meant that students divided into teams start the development of the project from scratch. The project assignments included a high-level specification of an e-commerce system and instructions for the development process. The projects were done in C# programming language. Students were instructed to use the same version of SQ as in Finland.

3.4 Analysis and Interpretation

Analysis and Interpretation for RQ1. We collected actual remediation times from participants and calculated the following quantitative indicators to better describe differences between SQ's estimates and actual times. The usage of different indicators allows us to obtain a more complete picture of the estimation accuracy.

Mean(RE) is the mean of Relative Errors (REs), $Mean(RE) = \frac{1}{n}\sum_{i=1}^{n} RE_i$, where n represents the number of SQ's estimations, while RE_i is the RE of the i-th estimation. Relative error is defined as $RE_i = \frac{actual - estimate}{actual}$ for each i-th estimation [4]. This metric is useful to provide insight weather estimated values are on average larger than actual times, i.e. overestimated, and in that case the $sgn(MeanRE) = -1$; while for underestimated values $sgn(MeanRE) = +1$.

In order to better analyse the magnitude of the differences between actual times and estimated values by SQ we calculated *the mean magnitude of RE (MMRE)* and *the median magnitude of RE (MdMRE).*

MMRE is calculates as $MMRE = \frac{1}{n}\sum_{i=1}^{n} MRE_i$, where n represents the number of SQ's estimations, while MRE_i is the MRE of the i-th estimation. Magnitude of relative error is defined as $MRE_i = \frac{|actual-estimate|}{|actual|}$ for each i-th estimation [4]. If the value of $MMRE$ is small, then we can say that SQ should give us on average good estimations.

MdMRE. The given n SQ's estimations, $MdMRE$ is computed as the median of the MREs of estimations [9]. The lower the $MdMRE$ value is, the better SQ's estimations are.

Analysis and Interpretation for RQ2. In order to assess the perceived criticality of TD issues by junior and novice developers we used the results of the follow-up survey. The criticality assessment of a TD issue was performed based on the response frequencies given by the students where all team members rated an identical group of TD issues.

Each TD issue that SQ identifies has a certain level of criticality, we used the following mapping between the responses from the survey and criticality levels. TD issues marked by Sonar as *Info* are considered as an issue that participants were *Not concerned at all*, *Minor* coincides with the answer from the survey *Not very concerned*, *Major* is equivalent to the answer *Concerned*, while *Blocker* and *Critical* correspond to *Very concerned.*

4 Results

In Case Finland, the study involved 133 students who were divided into teams of 4 to 6 members. We collected data from 26 different open-source projects which only 11 projects provided valid data. In Case Serbia, there were a total of 52 students divided into 6 teams and all of 6 projects provided valid data.

4.1 RQ1. How accurate are SQ's remediation times in the case of junior developers?

Table 1 shows the summary of the data analysis. The actual remediation times for the majority of TD items were significantly smaller than SQ estimates (the sign -1 indicates overestimated values). Interestingly, overestimated values does not depend on the type of the issue nor on the criticality level. Only in the case of two blocker issues (Case Serbia), SQ estimates were smaller than actual.

Regarding the magnitude of the difference in the estimated and actual values, in Case Finland, the difference is significantly larger compared to Serbian data. A plausible reason for this is that in Serbia the projects were done from the scratch by students, while in Finland the students had the task to choose an open-source project whose scope is significantly larger.

Table 1. Relative error indicators for TD issue remediation times grouped by Type and Severity for Finland and Serbia Cases.

TD Items		Finland				Serbia			
		N	$SGN(\cdot)$	MmRE	MdMRE	N	$SGN(\cdot)$	MmRE	MdMRE
Type	Bug	122	-	525%	495%	10	-	78%	78%
	Code Smell	3703	-	456%	435%	475	-	70%	69%
	Vuln.	1256	-	629%	650%	3	-	80%	80%
Severity	Minor	1557	-	378%	339%	153	-	76%	75%
	Major	1945	-	581%	584%	312	-	63%	63%
	Critical	465	-	472%	452%	21	-	69%	68%
	Blocker	58	-	436%	445%	2	+	70%	70%

4.2 RQ2. How do junior developers categorize criticality of TD issues identified by SQ?

Figure 1 depicts the results of the follow-up survey. Due to space limitation, here we present results for the top ten of the most frequent TD issues that occured.

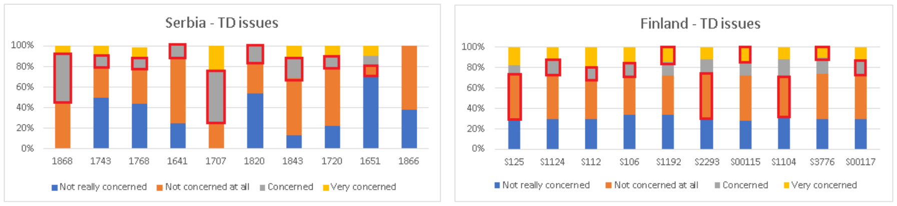

Fig. 1. Percived TD issue criticality level vs. SQ's categorization for the top-10 TD issues. Red-box indicate SQ's categorization for the given TD issue.

The red rectangle indicates the percentage of students whose perception of criticality coincides with SQ's assessment. The Figure 1 shows that only in the case of 5 issues from both Cases around 40% of students rate the criticality level identical to SQ.

5 Threats to Validity

The empirical research presented in this paper is prone to validity issues that we tried to address.

Construct validity: The main factor is socio-behavioral phenomena [22], which was reduced in the research by the fact that students did not even know that they were participating in the research, but performed all tasks within the project. As part of the follow-up survey, only the students were introduced to the goal of the research.

Conclusion validity: The activity that could have undermined the success of the research was the erroneous entry of time by the students because they entered the time independently. However, each student received a large number of issues and the probability of disrupting the research is minimal.

Internal validity: Instrumentation and maturation are two threats to internal validity [22]. Instrumentation, in our case the survey is comprised of direct questions only, resulting with as minimal as possible interpretations. While maturation indicates that participants might act differently as time passes, which can happen if a questionnaire is too long [22].

External validity: Threats to external validity limit the generalizability of the findings [22]. The focus of the research is on junior developers but we have achieved diversity in different project contexts and by having two comparative Cases.

6 Conclusion and Future Work

In this paper we presented results of an empirical study conducted in a context of two university courses in Finland and in Serbia.

The results indicated that SQ's remediation times to address TD issues are mostly greater than actual times reported by junior developers, i.e. SQ's cost estimates are higher. Such finding is unexpected because the focus group was junior developers who had minimal programming experience. Also, we found that SQ's categorization of the criticality levels is misaligned with participant's perception of the criticality levels.

Future work on this topic should be aimed at increasing the generalizability of this work by increasing the number of respondents.

References

1. Avgeriou, P., Kruchten, P., Ozkaya, I., Seaman, C.: Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162). Dagstuhl Reports **6**(4), 110–138 (2016)
2. Avgeriou, P., Taibi, D., Ampatzoglou, A., Fontana, F.A., Besker, T., Chatzigeorgiou, A., Lenarduzzi, V., Martini, A., Moschou, N., Pigazzini, I., Saarimäki, N., Sas, D., de Toledo, S.S., Tsintzira, A.: An overview and comparison of technical debt measurement tools. IEEE Software (2020)
3. Baldassarre, M.T., Lenarduzzi, V., Romano, S., Saarimäki, N.: On the diffuseness of technical debt items and accuracy of remediation time when using sonarqube. Information and Software Technology **128** (2020)
4. Conte, S.D., Dunsmore, H.E., Shen, V.Y.: Software effort estimation and productivity. In: Advances in Computers, vol. 24, pp. 1–60. Elsevier (1985)
5. Fowler, M.: Technical debt quadrant, 2009. URL: http://martinfowler.com/bliki/TechnicalDebtQuadrant. html (2009)
6. García-Munoz, J., García-Valls, M., Escribano-Barreno, J.: Improved metrics handling in sonarqube for software quality monitoring. In: Omatu, S., et al. (eds.) Distributed Computing and Artificial Intelligence, 13th International Conference. pp. 463–470. Springer International Publishing, Cham (2016)

7. 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. In: 7th International Workshop on Computer Science and Engineering, WCSE. pp. 171–175 (2017)
8. Gustafsson, T.: Machine learning and sonarqube kpis to predict increasing bug resolution times (2019)
9. Jorgensen, M.: Experience with the accuracy of software maintenance task effort prediction models. IEEE Transactions on software engineering **21**(8), 674–681 (1995)
10. Lenarduzzi, V., Lomio, F., Huttunen, H., Taibi, D.: Are sonarqube rules inducing bugs? In: 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER). pp. 501–511 (2020)
11. Lenarduzzi, V., Mandić, V., Katin, A., Taibi, D.: How long do junior developers take to remove technical debt items? In: 14th International Symposium on Empirical Software Engineering and Measurement. ACM/IEEE (2020)
12. Lenarduzzi, V., Saarimäki, N., Taibi, D.: The technical debt dataset. In: Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering. p. 2–11. Association for Computing Machinery, New York, NY, USA (2019)
13. Lenarduzzi, V., Saarimäki, N., Taibi, D.: Some sonarqube issues have a significant but small effect on faults and changes. a large-scale empirical study. Journal of Systems and Software **170**, 110750 (2020)
14. Letouzey, J., Ilkiewicz, M.: Managing technical debt with the sqale method. IEEE Software **29**(6), 44–51 (2012)
15. Mandic, V., Taušan, N., Ramac, R.: The prevalence of the technical debt concept in serbian it industry: Results of a national-wide survey. In: International Conference on Technical Debt (TechDebt '20), Seoul, Republic of Korea, p. 10 (2020)
16. 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. In: International Conference on Program Comprehension (ICPC). pp. 209–219 (2019)
17. Nugroho, A., Visser, J., Kuipers, T.: An empirical model of technical debt and interest. In: Workshop on Managing Technical Debt. p. 1–8 (2011)
18. Rios, N., de Mendonça Neto, M.G., Spínola, R.O.: A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners. Information and Software Technology **102**, 117–145 (2018)
19. SA, S.: https://docs.sonarqube.org/latest/. Tech. rep., SonarSource S.A, Switzerland (01 2020)
20. Saarimaki, N., Baldassarre, M.T., Lenarduzzi, V., Romano, S.: On the accuracy of sonarqube technical debt remediation time. In: 2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). pp. 317–324 (2019)
21. Saarimäki, N., Lenarduzzi, V., Taibi, D.: On the diffuseness of code technical debt in java projects of the apache ecosystem. In: Second International Conference on Technical Debt. pp. 98–107. TechDebt '19, IEEE Press (2019)
22. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., Wesslén, A.: Experimentation in software engineering. computer science (2012)
23. Zazworka, N., Izurieta, C., Wong, S., Cai, Y., Seaman, C., Shull, F., et al.: Comparing four approaches for technical debt identification. Software Quality Journal **22**(3), 403–426 (2014)
24. Zazworka, N., Spínola, R.O., Vetro', A., Shull, F., Seaman, C.: A case study on effectively identifying technical debt. In: 17th International Conference on Evaluation and Assessment in Software Engineering. p. 42–47 (2013)