Demystifying Sonar Tool Estimates in the Contexts of Familiar and Unfamiliar Software Projects: An Empirical Study with Junior Developers

Andrej Katin [1] [ORCID 0000-0001-9755-3733], *Nebojša Taušan* [2] [ORCID 0000-0001-9663-7951], *Valentina Lenarduzzi* [3] [ORCID 0000-0003-0511-5133], *Vladimir Mandić* [1] [ORCID 0000-0001-6996-2222]

[1] Faculty of Technical Sciences, University of Novi Sad, Novi Sad, Serbia
[2]Faculty of Economics, University of Novi Sad, Novi Sad, Serbia
[3]University of Oulu, Oulu, Finland

Abstract: *Developing a high-quality software product requires complete familiarity with the software product requirements and constraints. However, developers are often assigned to projects they are unfamiliar with, making project properties hard to understand for them. Sonar tool is one of the most popular Automated Static Analysis Tools that nowadays has become an integral part of software development process. The objective of this study is to evaluate the usefulness of the Sonar tool in the context of familiar and unfamiliar projects. More specifically to investigate the accuracy of Sonar's estimates while the remaining three assess the perceived refactoring difficulties, Sonar issue and fixing instructions as possible reasons for differences in estimates. A repeated measurements experiment in the context of a university course was designed and involved two sessions during which the subject refactored Sonar issues identified in familiar and unfamiliar projects. In total 60 students completed assignments, who in two sessions combined refactored 5,179 Sonar issues by violating 138 different Sonar rules. This study showed that estimates provided by Sonar are less accurate for familiar projects. Plausible explanations are that the refactoring and understandability of Sonar tool issue description seem to be more challenging in unfamiliar, compared to familiar projects.*

Key words: Empirical Software Engineering, Technical Debt, SonarQube

1. INTRODUCTION

In order for developers to deliver high-quality software, their familiarity with the software project is essential. For developers to be *familiar* with projects, they need to understand the context and decisions made from the beginning of the project. Including decisions regarding software design, architecture, and requirements [1]. Thus, the concept of familiarity surpasses the understanding of the code itself. However, developers are often assigned to projects they are unfamiliar with, making project properties hard to understand for them. This requires developers to familiarize themselves with the project by reviewing the code base and available documentation.

Various software tools can be used to aid developers while reviewing code base and identifying potential issues in projects. Automatic static analysis tools (ASAT) provide feedback in the form of analysis reports [2]. Sonar[1] as one such tool has proven to be very useful for identifying various quality issues [2, 3]. It provides a set of *code smell* detection rules for different programming languages and various metrics that are used together as means for identifying and estimating the cost of remedying issues in a code base [4]. Furthermore, Sonar's reports aid developers in identifying potential suboptimal software design and implementation choices [2. 5]. This phenomenon is conceptualized as technical debt (TD) [6]. Benefits of the Sonar tool are that it provides effort estimates, issue descriptions and instructions for refactoring the source code to remediate the identified TD issues. Such instructions can significantly affect the productivity of developers, especially if they are novice or junior [7, 8]. Where novice developers are often unfamiliar with projects assigned to them.

In the past years, several studies have been conducted that have focused on defining the accuracy of estimates provided by Sonar tool [3. 7, 8, 9]. The general conclusion is that the Sonar tool gives an overestimated remediation time and that the level of criticality identified for each issue does not match the perception of junior developers [3, 8].

In practice it is very common to have new members joining ongoing projects. For them the code base is unknown—*unfamiliar project*—while for other team members it is a known code base or *familiar project*. Managing such teams requires clear expectations regarding Sonar's estimates and how they translate to

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

the newly joined team members vs others. However, to the best of our knowledge there are no studies that intentionally compared the use of Sonar tool in the contexts of familiar and unfamiliar projects.
The objective of this research is to investigate the usefulness of Sonar's analysis reports in the contexts of familiar and unfamiliar software projects. We investigated the accuracy of the provided estimates and the perceived understandability of Sonar's feedback by junior developers in the context of these two types of projects. We designed and conducted an experiment within a university course. The experiment consisted of two separate sessions, i.e. *repeated measurements design* [10, 11], where students in the first session were tasked with refactoring identified TD issues in prior unknown to them open-source projects (unfamiliar projects) while for the second session they were first instructed to designed and implement projects from scratch and generate Sonar's report with TD issues used for refactoring in the second session (familiar projects).

2. RELATED WORK

In this section, we selected important papers for the research we conducted. All papers can be divided into two subgroups: (1) empirical research that dealt with the use of ASAT tools in projects that were familiar to the respondents, and (2) empirical research that dealt with the use of ASAT tools in projects with which respondents were not familiar.
Regarding empirical research in the context of familiar projects, Tan et al [18] designed an empirical study in which they used 20 Python and 16 Java projects, observing their first 2000 commits on repositories, with the aim of trying to estimate in what percentage the identified TD is resolved independently. They used the Sonar tool for analysis. As the initial commits on the repositories were examined, it exactly corresponds to the way of work that we also checked in our research where projects were created from the very beginning. In the research, the results were obtained that almost half of the Sonar issues are self-fixed.
While in the context of unfamiliar projects, Nunes et al. [19] analyzed the impact of two different ASAT tools on projects implemented in modern development environments. The focus was on web applications and how ASAT tools affect the identification of potential vulnerabilities. Sonar tool was not one of them. Vassallo et al. [20] included 176 open-source projects in their research in order to investigate whether and in what way developers accept the results of the ten different ASAT tools. The results showed that more than 60 percent of developers pay attention to the results of the analysis and are ready to accept them as a regular practice. In another paper, Lenarduzzi et al. [23] used unknown projects up to twelve years of age. The research focused on evaluating the usefulness of Sonar when translating a system architecture from monolithic to microservices. The results showed that after a short period of time, TD in the microservice architecture is significantly less compared to monolithic systems and that Sonar proved to be very useful.
With this brief review of papers that focused on examining the performance of ASAT tools in different areas of their application, we can also find the basic motivation for conducting such research. Namely, we see great scope in examining the impact of ASAT tools on the work of developers, depending on their familiarity with the project.

3. METHODOLOGY

The methodology is designed following the guidelines from *Wohlin et al.*[11], while the replicability of the study is supported with a publicly accessible replication package (https://bit.ly/3p70R7u}).
The goal of this study is formally expressed using GQM template [27]: A*nalyze* the Sonar tool analysis report *for the purpose of* evaluation *with respect to* usefulness *from the point of view of* junior developers *in the context of* unfamiliar and familiar software projects. Following this goal, the study focused on four aspects that were embodied in the following hypothesis:

- *H_1*: Remediation estimates provided by Sonar tool are more accurate in the context of familiar projects compared to unfamiliar project context.
- *H_2*: Issue descriptions provided by Sonar tool are easier to understand in familiar compared to unfamiliar project context.
- *H_3*: Fixing descriptions provided by Sonar tool are easier to understand in familiar compared to unfamiliar project context.
- *H_4*: Familiar projects are easier to refactor compared to refactoring unfamiliar projects.

The experimental sessions were designed as one factor with two treatments. The experiment relied on a within subject repeated measurements design with two objects. Figure 1 summarizes the experimental design, while the remaining paragraphs further elaborate the concepts presented in the figure.

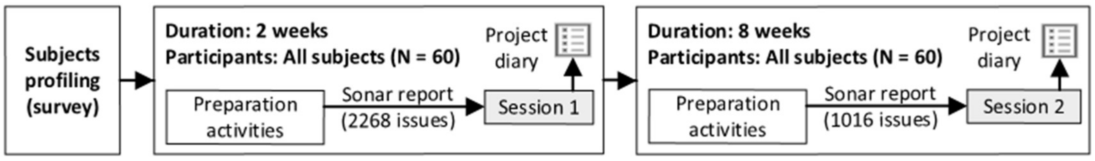
Subjects profiling (survey)
Duration: 2 weeks
Participants: All subjects (N = 60)
Project diary
Preparation activities
Sonar report (2268 issues)
Session 1
Duration: 8 weeks
Participants: All subjects (N = 60)
Project diary
Preparation activities
Sonar report (1016 issues)
Session 2

Figure 1: Experimental design

Subjects are senior year students of Information systems engineering curriculum. The subjects were sought as a proxy for junior developers since. The object of the study is the Sonar analysis report of the unfamiliar projects, for the first session, and of familiar projects, for the second session. The analysis report consists of identified Sonar issues that are characterized with estimated time for remediation, issue description, fixing description and issue categorization. Independent variable is *project type* with two levels: *unfamiliar* projects and *familiar* projects. Dependent variables are calculated times for fixing Sonar issues (H_1), perceived understandability of Sonar's issue description (H_2), perceived understandability of Sonar's issue fixing instructions (H_3), and the perceived refactoring difficulty (H_4).

The session protocol presents three groups of activities, which are: (a) *The initial activities* included training on the usage of the Sonar tool and project diary, which was one of the data collection instruments, (b) *Session preparation activities* differed for the first and second session. During the preparations for the first session, students selected, and lecturers checked and approved an open-source project found in the public repository (https://github.com). During the preparations for the second session, subjects used the provided software specification, to design and develop a software solution using .NET/C\# technology. This solution was seen as a familiar project that is analyzed during the second session. Finally, in (c) *experiment sessions* subjects had to remediate the identified issues and to record the data to project diary.

Data collection was realized using three data collection instruments: the profiling survey, Sonar tool and project diary. Profiling survey was implemented as an on-line (web) survey, and it was executed at the beginning of the course, thus before the experiment sessions. The survey contained a total of six different closed-typed questions, and there was no possibility to skip any of the questions. Survey questions probed subjects' programming experience, familiarity with code refactoring, as well as with ASAT tools and with the concept of technical debt. Subject used the project diaries to record the data. Within the project diary, each of the remediated issues had to be characterized with data such as issue ID, estimated and actual remediating time, assessment of the perceived refactoring difficulty, understandability of issue description and fixing instructions.

The data analysis of the Sonar tool's remediation estimates, and test the H_1, the mean magnitude of relative error (MMRE) and mean relative error metrics (MeRE) were used. These metrics are defined in the literature [3], and it is calculated for each subject as:

$$MMRE = \frac{1}{n}\sum_{i=1}^{n} MRE_i \qquad MRE_i = \frac{|T_A(i) - T_E(i)|}{T_A(i)} \qquad MeRE = \frac{1}{n}\sum_{i=1}^{n} RE_i \qquad RE_i = \frac{T_A(i) - T_E(i)}{T_A(i)}$$

where *n* is the total number of Sonar issues remediated by subject. MRE_i represents the magnitude of the relative error for the *i*-th issue, RE_i represents a relative error for the *i*-th issue, $T_{A(i)}$ stands for actual time the subject used to remediate the *i*-th issue, and $T_{E(i)}$ stands for the time estimate for remediating the *i*-th issue. Low *MMRE*, *MRE*, *MeRE* and *RE* values means that the differences between estimated and actual times for remediating the issues are also small, thus making the estimates more accurate. The formulas were, therefore, used to calculate: (a) are there any differences between the measurements in two sessions, and (b) which session has higher mean relative error. Consequently, the H_1 is tested using two statistical hypotheses $H_{\{1.1\}}$ that relies on *MMRE*, and $H_{\{1.2\}}$ that relies on *MeRE*. The remaining hypothesis were tested based on subjects' assessment of the perceived understandability of the Sonar's issue description (*PID*) for H_2, the perceived understandability of the Sonar's fixing instructions (*PFI*) for H_3, and the perceived refactoring difficulty (*PRD*) to eliminate each identified issue for H_4.

4. RESULTS

Demographics: The research was done in the context of a software engineering university course with fourth year undergraduate students at the University of [*Anonymized for review*] enrolled at Information Systems Engineering study program. In total *72* students attended the course and *60* of them participated in the experiment as subjects. Most of the subjects have between *2* to *4* years of programming experience (Figure 2.Q1). Also, about one fourth of participants (Figure 2.Q1a) have worked in industry. The profiling survey analysis revealed that participants are largely unfamiliar with TD concept and ASAT. A summary of the demographics data is presented in Figure 2.

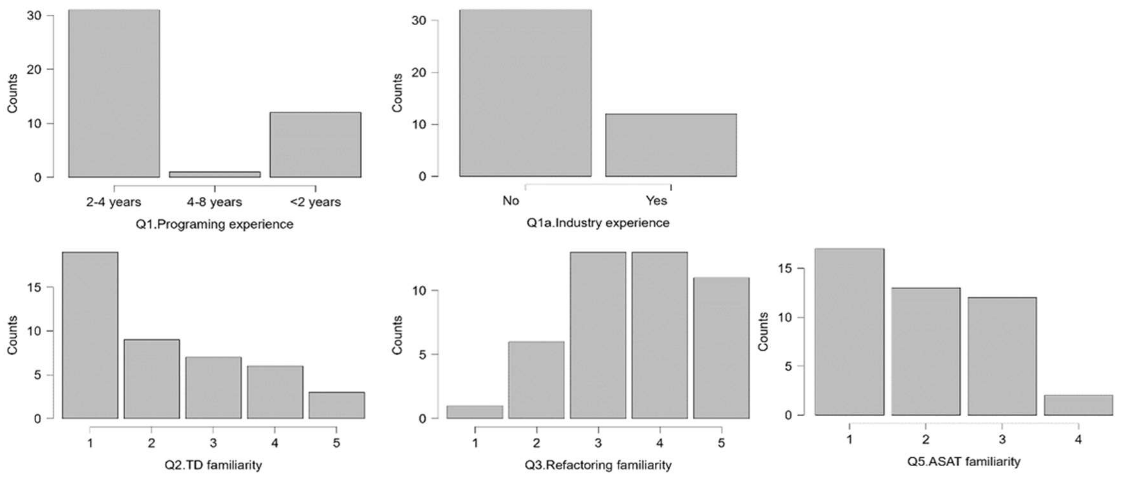

Figure 2. Demographics: Q1. Years of programming experience, Q1a. Experiences working in industry, Q2. Familiarity with the TD concept, Q3. Familiarity with refactoring practices, and Q5. Familiarity with ASAT tools.

The dataset consists of *2268* refactored Sonar issues in the first session and *1016* refactored issues in the second session. Out of these issues, *135* unique Sonar rules were violated in the first and *43* unique rules were violated in the second session. In total, *40* unique (or overlapping) rules were present in both sessions. Rule category wise, the majority of the violated rules are categorized as *bug* and *code smell.* Consequently, it can be asserted that the complexity of all the remediated issues, in both sessions, is highly similar.

Hypotheses testing relied on the following statistics: *Shapiro-Wilk* test was used to check the normality assumption of the data. The *Wilcoxon signed-rank* test was used as a non-parametric alternative to paired *t-test* to test the differences between the paired values from two sessions. For all tests we set the significance criterion as *0.05*. Finally, *Rank-Biserial Correlation* (r_B) was used to test the effect size and interpreted according to Goss-Sampson [28]. Calculated *W* statistics and *p*-values for each variable MMRE, MeRE, PID, PFI, and PRD, and for each session: unfamiliar projects, and familiar projects. Based on the test results the normality assumption can be accepted for *MMRE* and *MeRE* measured in first session ($p > 0.05$), while rejected for the remaining variables ($p < 0.05$). Thus, a non-parametric Wilcoxson Signed Rank test was used to assess the differences.

Hypothesis H_1 is tested using two variables, *MMRE* for testing the differences between the measurements using two-sided test. And variable *MeRE* for assessing the higher relative error in estimated time for the issues remediation. It can be concluded that the Sonar Tool overestimates the time needed to remediate the identified issues in both the unknown and known project context—hence the negative *MeRE* values. It can also be concluded that developers need more time to remediate the issues when working on unfamiliar projects, compared to time needed for working on familiar projects. Test of hypothesis H_2 indicates that developers perceived the Sonar issue descriptions less understandable in session s_1 (unfamiliar projects) compared to s_2 (familiar projects). The third hypothesis is tested using the *PFI* variable and both one and two-sided test, suggesting that perceived understandably of fixing instructions provided by Sonar tool does not differ session wise. The fourth hypothesis was tested by comparing the *PRD* variables using one-sided, paired test, for which the null hypothesis was rejected, leading to conclusion that the perceived refactoring difficulty in unfamiliar project context is more difficult, compared to refactoring of familiar projects. Figures 3 and 4 present the box plots for tested variables.

Table 1: Hypothesis test results

Hypothesis	Wilcoxon Signed Rank Test (p < 0.05)	Decision
$H_{1.1}^0 : \mu_{s1}(MMRE) = \mu_{s2}(MMRE)$ $H_{1.1}^A : \mu_{s1}(MMRE) <> \mu_{s2}(MMRE)$	V = 204, p = 1.691e $^{-7}$ (two-sided)	Rejected
$H_{1.2}^0 : \mu_{s1}(MeRE) = \mu_{s2}(MeRE)$ $H_{1.2}^A : \mu_{s1}(MeRE) <> \mu_{s2}(MeRE)$	V = 1657, p = 2.399e $^{-8}$ (one-sided)	Rejected
$H_2^0 : \mu_{s1}(PID) = \mu_{s2}(PID)$ $H_2^A : \mu_{s1}(PID) <> \mu_{s2}(PID)$	V = 148, p = 0.009226	Rejected
$H_3^0 : \mu_{s1}(PFI) = \mu_{s2}(PFI)$ $H_3^A : \mu_{s1}(PFI) <> \mu_{s2}(PFI)$	V = 129, p = 0.6247 (two-sided)	Accepted
$H_3^0 : \mu_{s1}(PFI) = \mu_{s2}(PFI)$	V = 129, p = 0.3123 (one-sided)	Accepted

$H_3^A: \mu_{s1}(PFI) > \mu_{s2}(PFI)$		
$H_4^0: \mu_{s1}(PRD) = \mu_{s2}(PRD)$ $H_4^A: \mu_{s1}(PRD) <> \mu_{s2}(PRD)$	V = 434, p = 2.022e $^{-05}$	Rejected

Finally, the **effects sizes** are calculated for hypotheses using Rank-biserial correlation. For all hypotheses the effect size is *large* with (*95%* confidence interval), except for H_3 the effect size is small ($r_β$=0.12).

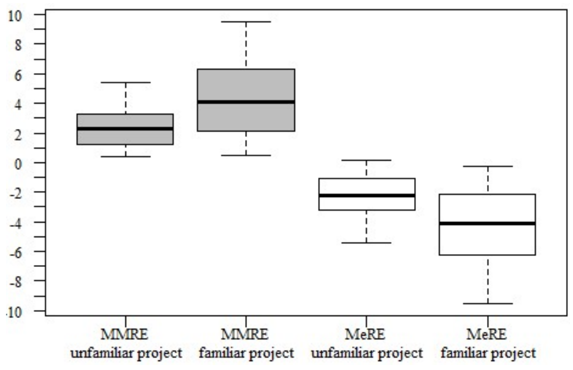

Figure 3: Box plots for MMRE and MeRE.

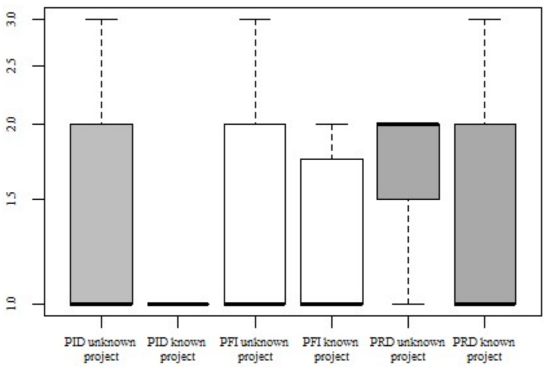

Figure 4: Box plots for PID, PFI, and PRD.

5. CONCLUSIONS

This paper presents the findings of an empirical study that was conducted in the context of a university course, with an objective to investigate the impact of the project familiarity on the usefulness of feedback generated by Sonar tool. Our findings confirm previous studies that Sonar overestimates remediation effort [3, 8], which is a case for both types of projects as well. However, the estimates are more accurate for the unfamiliar projects than for the familiar ones. Furthermore, it seems that the understandability of the fixing instructions provided by Sonar is not impacted by project type, while Sonar's descriptions of identified issues were significantly less understandable for unfamiliar projects. The plausible explanation is that fixing instructions are decontextualized and sufficiently self-explanatory, while understanding issues descriptions require understanding of the targeted code base for which issues were identified. Finally, the data showed that developers tend to understand easier the Sonar's issue descriptions when they are involved in the project from the start. Also, this study showed that estimates provided by Sonar tool are less accurate for familiar projects. One plausible explanation is that in such projects *refactoring* seems to be less challenging.

ACKNOWLEDGMENT

The research presented in this paper is partially supported by the project "Implementation of the results of scientific research work in the field of Industrial Engineering and Management in DIIM teaching processes with the aim of their continuous improvement", at the Department of Industrial Engineering and Management, Faculty of Technical Sciences, University of Novi Sad, Republic of Serbia.

REFERENCES

V. Mandić, M. Oivo, P. Rodríguez, P. Kuvaja, H. Kaikkonen, and B. Turhan, "What is flowing in lean software development?" in Lean Enterprise Software and Systems: First International Conference, LESS 2010, Helsinki, Finland, October 17-20, 2010. Proceedings. Springer, 2010, pp. 72–84.

P. C. Avgeriou, D. Taibi, A. Ampatzoglou, F. Arcelli Fontana, T. Besker, A. Chatzigeorgiou, V. Lenarduzzi, A. Martini, A. Moschou, I. Pigazzini, N. Saarimaki, D. D. Sas, S. S. de Toledo, and A. A. Tsintzira, "An overview and comparison of technical debt measurement tools," IEEE Software, vol. 38, no. 3, pp. 61–71, 2021.

V. Lenarduzzi, V. Mandí c, A. Katin, and D. Taibi, "How long do junior developers take to remove technical debt items?" in International Symposium on Empirical Software Engineering and Measurement, 2020.

A. Katin, V. Lenarduzzi, D. Taibi, and V. Mandić, "On the technical debt prioritization and cost estimation with sonarqube tool," Int. Sci. Conf. on Industrial Systems Industrial Innovation in Digital Age, 2020.

S. Freire, N. Rios, B. Pérez, C. Castellanos, D. Correal, R. Ramač, V. Mandić, N. Taušan, G. López, A. Pacheco, M. Mendonça, D. Falessi, C. Izurieta, C. Seaman, and R. Spínola, "Software practitioners' point of view on technical debt payment," Journal of Systems and Software, vol. 196, p. 111554, 2023.

R. Ramač, V. Mandić, N. Taušan, N. Rios, S. Freire, B. Pérez, C. Castellanos, D. Correal, A. Pacheco, G. Lopez, C. Izurieta, C. Seaman, and R. Spinola, “Prevalence, common causes and effects of technical debt: Results from a family of surveys with the it industry,” Journal of Systems and Software, vol. 184, 2022.

V. Lenarduzzi, N. Saarimäki, and D. Taibi, “Some sonarqube issues have a significant but small effect on faults and changes. a large-scale empirical study,” Journal of Systems and Software, vol. 170, 2020.

A. Katin, V. Lenarduzzi, D. Taibi, and V. Mandĭ c, “On the technical debt prioritization and cost estimation with sonarqube tool,” in Proceedings on 18th International Conference on Industrial Systems–IS’20: Industrial Innovation in Digital Age. Springer, 2022, pp. 302–309.

M. T. Baldassarre, V. Lenarduzzi, S. Romano, N. Saarimäki, “On the diffuseness of technical debt items and accuracy of remediation time when using sonarqube,” Inf. and Software Technology, vol. 128, p.2020.

N. Juristo and A. M. Moreno, Basics of software engineering experimentation. Springer Science & Business Media, 2013.

C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, Experimentation in software engineering. Springer Science & Business Media, 2012.

G. A. Campbell and P. P. Papapetrou, SonarQube in action. Manning Publications Co., 2013.

P. H. de Andrade Gomes, R. E. Garcia, G. Spadon, D. M. Eler, C. Olivete, and R. C. M. Correia, “Teaching software quality via source code inspection tool,” IEEE Frontiers in Education Conference, 2017, pp. 1–8.

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

I. Tollin, F. A. Fontana, M. Zanoni, and R. Roveda, “Change prediction through coding rules violations,” in Proceedings of the 21 Int. conference on evaluation and assessment in software engineering, 2017,

J.-L. Letouzey, “The sqale method for evaluating technical debt,” in 2012 Third International Workshop on Managing Technical Debt (MTD). IEEE, 2012, pp. 31–36.

T. D. Oyetoyan, B. Milosheska, M. Grini, and D. Soares Cruzes, “Myths and facts about static application security testing tools: an action research at telenor digital,” 19th Int. Conference on Agile Processes in Software Engineering and Extreme Programming, Portugal, 2018,Springer International Publishing

J. Tan, D. Feitosa, and P. Avgeriou, “Does it matter who pays back technical debt? an empirical study of self-fixed td,” Information and Software Technology, vol. 143, p. 106738, 2022.

P. Nunes, I. Medeiros, J. Fonseca, N. Neves, M. Correia, and M. Vieira, “An empirical study on combining diverse static analysis tools for web security vulnerabilities based on development scenarios,” Computing, vol. 101, pp. 161–185, 2019

C. Vassallo, S. Panichella, F. Palomba, S. Proksch, H. C. Gall, and A. Zaidman, “How developers engage with static analysis tools in different contexts,” Empirical Soft. Engineering, vol. 25, pp. 1419–1457, 2020

R. Alfayez, R. Winn, W. Alwehaibi, E. Venson, and B. Boehm, “How Sonarqube-identified technical debt is prioritized: An exploratory case study,” Information and Software Technology, p. 107147, 2023.

V. Lenarduzzi, A. Martini, D. Taibi, and D. A. Tamburri, “Towards surgically-precise technical debt estimation: Early results and research roadmap,” 3rd ACM SIGSOFT International workshop on machine learning techniques for software quality evaluation, 2019, pp. 37–42.

V. Lenarduzzi, F. Lomio, N. Saarimäki, and D. Taibi, “Does migrating a monolithic system to microservices decrease the technical debt?” Journal of Systems and Software, vol. 169, p. 110710, 2020

I. Ahmed, U. A. Mannan, R. Gopinath, and C. Jensen, “An empirical study of design degradation: How software projects get worse over time,” in 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). IEEE, 2015, pp. 1–10

C. F. Kemerer and S. Slaughter, “An empirical approach to studying software evolution,” IEEE transactions on software engineering, vol. 25, no. 4, pp. 493–509, 1999

N. Rachatasumrit and M. Kim, “An empirical investigation into the impact of refactoring on regression testing,” in 2012 International conference on software maintenance, IEEE, 2012, pp. 357–366.

V. R. Basili, “Software modeling and measurement: the goal/question/metric paradigm,” Institute for Advanced Computer Studies, University of Maryland, Tech. Rep., 1992.

M. Goss-Sampson, “Statistical analysis in jasp: A guide for students,” 2020.