The Impact of Continuous Code Quality Assessment on Defects

Rolf-Helge Pfeiffer
IT University of Copenhagen
Copenhagen, Denmark
ropf@itu.dk

***Abstract*—Continuous Code Quality Assessment (CCQA) tools promise that increasing code quality leads to fewer defects, i.e., that software quality from the user view can be increased by increasing quality of the product. Currently, there is limited evidence on that application of CCQA tools, such as, SonarCloud (SC), during software development actually reduces the amount of defects over time. In this paper we study five open-source projects that adopt SonarCloud (SC) for CCQA and we compare frequencies of defect reports before and after adoption of SC. For only one project (Apache Ratis), we find a statistically significant decrease of defects after adoption of the tool. After closer investigation we find, that this decrease is likely just a coincidence and not caused by the adoption of SC and adherence to its code quality recommendations. In general, we find no evidence for that application of a CCQA tool increases product quality.**

I. Introduction

Delivering high-quality software is usually the goal of organizations and stakeholders that are involved in software development. For example, the Apache Software Foundation (ASF) has the declared goal *"to create high quality software that leads the way in its field"*[1], practitioners strive to identify ways of ensuring software quality [1], and managers show an increasing interest in improvement of software quality [2].

However, creating quality software is not easy and there are many aspects to consider. Previous studies demonstrate the impact of various aspects on software quality. For example, organizational aspects, such as, organizational structure [3], code ownership [4], or adherence to processes [5], [6] impact defect rates. Socio-technical aspects, such as, code review coverage and code review participation [7], applied branching strategies during development [8], or deployment schedules, hardware configurations, and software platforms [9] impact defect rates, requests for assistance, or user-perceived quality. Technical aspects, such as, size of software [10] or the application of certain design patterns [11] impact defects or customer perceived software quality.

To make things more complicated, there is not *one* kind of software quality but multiple perspectives on it [12]. Kitchenham et al. [12] describe five perspectives on *software quality*. The two perspectives that are relevant in this paper are the *"user view [that] sees quality as fitness for purpose"* and the *"product view [that] sees quality as tied to inherent characteristics of the product"*.

Recently, CCQA tools like SonarQube (SQ) promise to increase software quality from the user view by increasing quality in the product view [13]. That is, it is promised that increasing code quality (product view) will result in fewer defects (user view). Practitioners apply such tools under the same assumption[2,3]. However, it is unclear to which degree application of CCQA tools during development actually impacts the amount of reported defects over time. Previous work on static analysis tools, such as, FindBugs for Java [14], [15] suggest that such tools are potentially suitable to decrease defects. But other studies that actually link the product view (code quality) with user view (defect rates) show either no impact of such tools [16] or they argue for their limited suitability since *"at least 45 percent of all defects [are] undetectable using code analysis"* [17].

The goal of this paper is to investigate empirically the research question (RQ): Does the application of a CCQA tool (SC) in five open-source projects actually reduce the number of defects that are reported for software products? We present initial results (Sec. IV) for the five ASF projects Daffodil, Groovy, Hadoop Ozone, Karaf, and Ratis, which recently adopt SC.

Only for two projects (Hadoop Ozone and Ratis) we find a decrease of the number of reported defects from the year before introduction of SC to the year after. Only for Ratis this decrease is statistically significant.

A closer inspection of the SC quality issues that Ratis developers rectify together with associated tickets and commits suggests, that the statistically significant decrease of defects is not due to application of CCQA but due to an unknown factor. Generally, our results do not provide evidence for that adoption of the CCQA tool SC leads to increased software quality from the user's view (Sec. VI).

The main contribution of this paper is the –to a large degree automated– empirical case study of the impact of SC to defect rates in five ASF projects. A reproduction kit with all code and data is available online[4].

II. Background

Terminology: In this paper, *defects* are software failures that are experienced and reported either by end-users or by other developers. Developers can be users too, e.g., in case of libraries, programming languages, etc. Therefore, we consider all tickets in a project's issue tracker that are labeled as *bug* to be *defects*.

SonarQube (SQ) is a Continuous Code Quality Assessment (CCQA) tool, a kind of automatic static analysis (ASA) tool that is applied continuously to a software repository. SonarCloud (SC) is an SQ instance that is hosted in the "cloud". The Apache Infrastructure team operated a SQ instance[5] for all ASF projects until mid-2019. That SQ instance is now deactivated and ASF projects switched to SC. SC assesses quality *issues* in the categories *code smell*, *vulnerability*, and *bug*. SC defines a *bug* as *"An issue that represents something wrong in the code. If this has not broken yet, it will, and probably at the worst possible moment. This needs to be fixed. Yesterday."*[6] In the remainder, we call these *SC-bugs*. SC-bugs are not *defects*. Instead, they are patterns in code that match certain static code analysis rules which are aggregated under a label *bug*[7]. These rules are configurable, since developers may disagree if matching instances in code actually constitute a problem. *Code smells* in SC are considered to be maintainability-related issues in the code, which are expected to increase maintenance effort and potentially provoke errors. *Vulnerabilities* are security-related issues that open backdoors for attackers.

In the remainder, we refer to issues in an issue tracker as *tickets* and we call the issues that SC reports (bugs, vulnerabilities, and code smells) *quality issues*.

Related Work: Prior work presents inconsistent results regarding the impact of application of ASA tools to defects. For example, Wagner et al. [16] apply `FindBugs` and `PMD` in two industrial case studies. They demonstrate that the tools cannot detect field defects that are stored in an issue tracker. Also Cuoto et al. [18] do not find a correspondence between ASA issues and field defects when applying `FindBugs` to Rhino, the AspectJ compiler, Lucene, and 30 other ASF projects. However, they find a moderate correlation between ASA issues and defects, i.e., projects with more static analysis issues have usually more defects. Contrary, Plösch et al. [19] who apply `PMD` and `FindBugs` to the sources of the Eclipse SDK, report statistically significant correlations between ASA issues and defects from issue trackers. Boogerd et al. [20] report mixed results from an industrial case study where they investigate the impact of the MISRA C coding standard via the `QA-C` tool. They find that a small subset of static analysis rules can predict fault locations but they also find that application of the default ASA ruleset may actually increase defect probability. Similarly, Zheng et al. [15] find that the number of ASA issues can indicate fault-prone modules. However, their results do not allow to confirm a positive impact of application of ASA tools to increased product quality from the user perspective. Finally, Lauesen et al. [17] report that at least 45% of all defects are undetectable with ASA tools, since these defects are connected to requirements that are un- or wrongly-specified. The authors state that the case company can only find less than 10% of reported defects with the help of ASA tools.

Others study the impact of ASA tools on pre-release defects, i.e., defects that are revealed during testing (product view). For example, when applying proprietary ASA tools during development of Windows Server 2003, Nagappan et al. [21] find a correlation between the amount of ASA issues of components and the number of pre-release defects that are identified via testing. When applying `FindBugs` to multiple hundreds of academic Java projects Vetro et al. [22], find that a small subset of the ASA rules can predict pre-release defects.

Only Digkas et al. [23] study how developers address quality issues from SC in 57 ASF projects. However, they do not study the impact of this remediation work to defects.

Since we cannot identify studies that assess the impact of CCQA tools, such as, SC, to defects, we conduct this study to extend the prior body of knowledge.

III. Method

In this section we describe how we select suitable software projects and how we assess the impact of CCQA on quality from the user perspective.

Many ASF projects apply SC for CCQA. A list of these projects is available online.[8] From this list, we select all those projects that: *a)* are neither incubator nor retired projects, *b)* for which at least one SC code quality assessment is available, *c)* that have exactly one SC assessment dashboard, and *d)* that use either Apache's Jira or Bugzilla[9] as issue trackers.

SC assessments are usually conducted on `Git` repositories. With criterion *c)*, we exclude projects from our study that organize their sources in more than one repository. For example, the REST framework Sling[10] organizes its more than 300 components in separate repositories[11], which leads to more than 300 assessment dashboards on SC. That hinders direct comparison of issue tracker data (one issue tracker per project) and quality assessment data.

From the 455 projects that are listed on SC (Jun. 1st 2021), 20 fulfill the four criteria above. We retain only those projects, which do not apply SQ prior to SC, since quality assessments from ASF's previous SQ instance are not accessible anymore for our study. Commit messages that match the regular expression `[Ss]onar` with a commit date more than a month before the first SC assessment are considered a sign prior use of SQ.

The following five projects (alphabetical order) fulfill all criteria above and adopt CCQA first with SC [12]:

Daffodil is an implementation of the Data Format Description Language (DFDL) [24], which converts fixed format data to, e.g., XML or JSON. It is mainly written in Scala with ca. 161KLOC.[13]

Groovy is a dynamic and optionally typed JVM language. It is mainly written in Groovy and Java and ca. 444KLOC in size.

Hadoop Ozone is a distributed object store for Hadoop, with ca. 356KLOC of mainly Java.

Karaf is an application container and runtime, with ca. 182KLOC of mainly Java.

Ratis is a ca. 63KLOC (mainly Java) implementation of the *Raft* consensus algorithm [25].

For these five projects, we clone the respective Github repositories[14] and export all tickets labeled as *bug*, i.e. defects, from the respective issue trackers[15]. From that data, we compute the weekly defect creation rates (WDCRs) per project. WDCRs, are the number of newly created defect reports per week in the respective issue tracker.

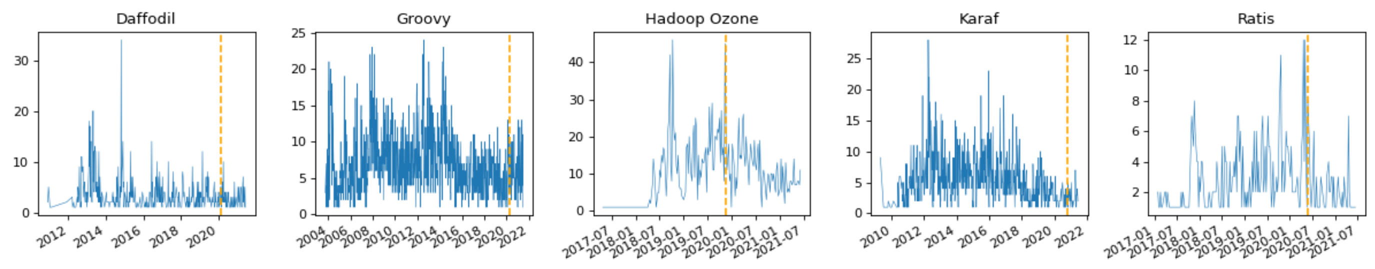

Figure 1: Weekly defect creation rates per project over time. Orange lines indicate introduction of SonarCloud.

Per project, we automatically scrape quality assessments including timestamps and frequencies of quality issues from SC. From that data, we determine the *SC adoption date* as the date of the first accessible quality assessment. Based on adoption dates, we compute statistics over the WDCRs per project in the year before and after SC adoption. We compute arithmetic means –with corresponding standard deviations (σ)– of WDCRs in the year before (μ_b) and after (μ_a) adoption of SC. For the same periods we compute the medians of WDCRs ($\tilde{x}_b$ and $\tilde{x}_a$ respectively), see Tab. I.

Since CCQA tools promise to improve software quality (user view) by increasing code quality (product view), see Sec. I, we hypothesize that introduction of a CCQA tool and work on code quality improvement will reduce the amount of defect reports over time. We compute the Kruskal-Wallis H-test [26][16] (with significance level of 5%) on WDCRs with the null hypothesis that medians of WDCRs are equal in the two consecutive years before and after adoption of SC and the alternative hypothesis that these medians diverge. We choose this test since it does not assume a certain distribution in the underlying data.

We generate statistics over tickets to understand how developers work with SC and how they address the reported quality issues. Per project, we count the number of tickets in the issue trackers that mention the terms *sonar*, *smell*, *bug*, or *vulnerability* via a corresponding regular expression over ticket descriptions.

Semi-automatically, we examine commits, tickets, and quality issues for projects with decreasing WDCRs and for which the Kruskal-Wallis H-test yields statistically significant results. The purpose is to understand if the rectified SC quality issues can explain the decrease in reported defects, i.e., the increase in quality from user perspective. The following results are based on data that is collected and interpreted on the Jun. 14th 2021, the *reference date*.

Table I: Descriptive statistics of weekly defect creation rates.

	μ_b	μ_a	$\tilde{x}_b$	$\tilde{x}_a$		H	p
Daffodil	3.0 σ : 1.8	2.6 σ : 1.9	2	2	$\rightarrow$	2.26	0.13
Groovy	4.5 σ : 3.0	6.1 σ : 2.7	4	5	$\boldsymbol{\uparrow}$	9.43	0.002
Hadoop Ozone	15.1 σ : 8.1	12.3 σ : 5.8	15	12	$\downarrow$	2.88	0.09
Karaf	2.6 σ : 1.3	2.8 σ : 1.5	2	2	$\rightarrow$	0.23	0.63
Ratis	4.0 σ : 2.8	2.2 σ : 1.4	3	2	$\boldsymbol{\downarrow}$	15.42	8.5×10^{-5}

IV. Results

Fig. 1 illustrates the development of weekly defect creation rates (WDCRs) for the five projects (blue lines). Orange lines indicate the SC adoption dates, see Tab. II (SC Adopt. Date).

Per project, Tab. I lists the arithmetic means (μ_b and μ_a) with standard deviations (σ), medians ($\tilde{x}_b$ and $\tilde{x}_a$), and the Kruskal-Wallis H statistic (H) with corresponding p-values (p) for the WDCRs for the years before and after adoption of SC. Arrows symbolize in-/decreasing or unchanged median values of both periods. Bold arrows indicate statistically significant changes with p-values below 0.05.

Only for Hadoop Ozone and Ratis the median of WDCRs decrease from the year before adoption of SC to the year after. For the other three projects, they either increase (Groovy) or remain unchanged (Daffodil and Karaf). We reject the null hypothesis of equal medians WDCRs only for Ratis and Groovy. Both possess p-values smaller than 0.05 (8.5×10^{-5} and 0.002 respectively). For the remaining three projects we do not reject the null hypothesis, i.e., WDCRs do not change in a statistically significant way.

Tab. II lists statistics over SC related tickets. The first block lists the SC adoption dates and the second block lists the number of tickets in Jira that mention the terms *sonar*, *smell*, *bug*, or *vulnerability* respectively (#*x* Tickets), how many of these are resolved at the reference date (#*x* Tickets Resolved), and how many tickets are created since adoption of SC (#Tickets since Adopt.).

Obviously, the amount of SC related tickets is low for Daffodil, Groovy, and Karaf (Tab. II). Manual inspection of these tickets reveals that they are either about set-up and configuration of SC or false-positives, e.g., code examples like `toJSONArray` that match our case-insensitive regular expression (`sonar`).

Table II: Statistics over SC related tickets and commits.

	Daffodil	Groovy	H. Ozone	Karaf	Ratis
SC Adopt. Date	02/27/20	03/22/20	11/14/19	10/05/20	05/28/20
#SC Tickets	4	4	119	2	20
#SC Tickets Resolved	3	4	96	2	12
#Smell Tickets	0	1	5	0	3
#Smell Tickets Resolved	0	0	3	0	0
#Bug Tickets	31	65	129	15	23
#Bug Tickets Resolved	17	34	101	11	17
#Vuln. Tickets	0	3	1	0	3
#Vuln. Tickets Resolved	0	2	1	0	2
#Tickets since Adopt.	247	660	2842	304	433

Deeper Analysis: Given the relatively high numbers of SC related tickets, it seems that only Hadoop Ozone and Ratis developers are actively addressing the quality issues reported by SC. Since only Ratis shows a statistically significant decrease in WDCR, we focus our following detailed analysis only on it. Due to constrained space, all mentioned tickets and commits are linked in our protocol online.[17]

For Ratis, we manually inspect all tickets from Tab. II. We find, that the 20 SC related tickets (#SC Tickets) are all about SC, i.e., no false-positives. Five of these (R940, R948, R1306, R1311, and R1314), are about setup and configuration of SC and the remaining 15 are about SC quality issues, mainly SC-bugs (9). Our manual inspection reveals four new tickets that do not match any of the search terms from Sec. III. These are sub-tasks of R1054 that describe four different SC-bugs. The three tickets that mention code `smell`s (#Smell Tickets: R955, R956, R1314) also mention the term `sonar`, i.e., they are included in the 20 tickets above. Two of the three tickets that mention the term vulnerability mention also the term `sonar` (R953, R954). Ticket R1342, is newly identified. It is not about a SC quality issue but about updating a dependency due to a detected vulnerability. Of the 23 tickets that mention a `bug` (#Bug Tickets), nine also mention the term `sonar`. Of the remaining, 13 describe defects, i.e., faults or unintended behavior, not SC-bugs. Ticket R1232 is a false-positive, an improvement that describes adjustment of *debug* levels.

After deduplication of these tickets and after adding the manually identified sub-tasks of R1054, we identify in total 24 tickets from Ratis that are either about configuration of SC or about quality issues. Twelve of these are resolved and describe rectification of quality issues. We identify 11 commits that link via a ticket identifier in a commit message to these 12 tickets. These 11 commits rectify 105 SC quality issues. The majority of rectified SC quality issues (ca. 75%) are in exception handling, e.g., in 40 cases handling of `Throwable` objects[18] is rectified, and in 39 cases handling of interrupted threads[19] is rectified.

To understand if the rectified quality issues cause the decrease of WDCRs, we automatically identify all commits from the year before SC adoption that apply a change similar to the commits that rectify the 79 SC quality issues. In the year before SC adoption, there are no defects that are resolved via commits that correct exception handling of `Throwable` objects or incorrect handling of interrupted threads.

Discussion: Based on the results presented above, we do not believe that the rectified quality issues are the cause for decreasing WDCR in Ratis. There are no defects in the year before SC adoption that are caused by a corresponding quality issue. To substantiate our belief, we ask the Ratis developers via their mailing list on their opinion on the effect of adoption of SC to WDCRs or if they are aware of alternative explanations for decreasing WDCRs. Unfortunately, we did not receive feedback on our questions yet.

It seems as if only Hadoop Ozone and Ratis developers work actively with SC and the reported quality issues. For the other three systems, it seems as if developers just enable SC but do not really take its quality reports into consideration (see Tab. II), which would be inline with previous results that indicate that CCQA is not really applied in practice [27]. It is noteworthy, that Hadoop Ozone and Ratis are the two projects with decreasing WDCR in the year after SC adoption. The readiness of the developers of Hadoop Ozone and Ratis (the youngest projects in our study) to work with SC and the reported quality issues might indicate that the developers of these projects are in general more aware about software quality and thereby produce code that results in fewer defects in the long run.

V. Threats to Validity

For calculation of statistics of development of WDCR (Fig. 1 and Tab. I), we rely on that tickets are correctly labeled as *bugs* in the issue trackers. We believe that this problem is negligible, since the amount of defects per project is quite large (Daffodil: 1403, Groovy: 6571, Hadoop Ozone: 2055, Karaf: 3104, Ratis: 533). Consequently, we believe that a certain amount of wrongly labeled tickets do not affect the trends in the data.

We identify SC related tickets (Tab. II) via case-insensitive regular expressions for the terms `sonar`, code `smell`, `vulnerabilit[iy]`, or `bug`. Based on these we identify corresponding commits via ticket identifiers in commit messages. Thereby, we likely do not identify all relevant tickets and commits. That is, all presented values represent lower bounds. Via our manual inspection (Sec. IV), we identify relevant tickets (20%) that are linked as sub-/super-tasks to any of the automatically identified ones. Additionally, we contact the Ratis developers via their email list on Jun. 15th 2021. We ask if there are relevant tickets and commits that we should have included in our study. Unfortunately, we do not receive their reply prior to the paper deadline.

For Karaf, our data covers only ca. nine months of data – instead of a year– (Tab. II). That might impact our results and it may explain why there are only two tickets (#SC Tickets) that explicitly mention `sonar`. However, we decide to include Karaf in our study since it fulfills all selection criteria, see Sec. III. In future, we will execute the study again on updated data and will report diverging results.

Since we conduct our study in a non-lab environment, we cannot account for possible effects of other ASA tools that projects may apply. For example, during our manual investigation, we find that Ratis also relies on `FindBugs`. In future, we plan to extend this study to isolate potential effects of other tools.

We believe that we adequately represent the user view on software quality via defect reports in issue trackers. Prior work on ASA [16], [18], [19] does so too.

VI. Conclusions & Future Work

In this paper, we study the impact of applying the Continuous Code Quality Assessment tool SonarCloud in five ASF projects. We compare statistically the numbers of reported defects (weekly defect creation rates (WDCRs)) in the years before and after SC adoption. To understand if the decrease is caused by the developers' work on SC's quality issues, we inspect tickets and commits for the only project with statistically significant decrease of WDCR (Ratis).

We conclude that adoption of SC in the five studied ASF projects either does not lead to a decrease of WDCR (Daffodil, Groovy, Karaf), does not lead to a statistically significant decrease of WDCR (Hadoop Ozone), or in case of a statistically significant decrease (Ratis) our thorough inspection of tickets and commits does not reveal evidence for that the decrease is actually caused by addressing SC quality issues. Consequently, we cannot find evidence for that increasing code quality (product view) leads to an increase of software quality from the user's view, as promised by SC [13].

In future, we plan to collect structured feedback from the developers, especially from Ratis, on their opinions about impact of SC on software quality. Also, we aim to identify the underlying reason for Ratis' decreasing WDCR. Additionally, we want to study systematically to which degree defects occur that are identified by any of SC's static analysis rules and we plan to extend our thorough investigation to Hadoop Ozone.

References

[1] "How to develop high-quality software," Vector Informatik GmbH, Tech. Rep., 05 2020. [Online]. Available: https://www.coderskitchen.com/wp-content/uploads/2020/05/How-to-Develop-High-Quality-Software_CK.pdf

[2] J. Luftman, R. Kempaiah, and E. H. Rigoni, "Key issues for it executives 2008." *MIS Quarterly Executive*, vol. 8, no. 3, 2009.

[3] N. Nagappan, B. Murphy, and V. Basili, "The influence of organizational structure on software quality," in *2008 ACM/IEEE 30th International Conference on Software Engineering*. IEEE, 2008, pp. 521–530.

[4] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code! examining the effects of ownership on software quality," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 4–14.

[5] J. Herbsleb, A. Carleton, J. Rozum, J. Siegel, and D. Zubrow, "Benefits of cmm-based software process improvement: Initial results," CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, Tech. Rep., 1994.

[6] M. Diaz and J. Sligo, "How software process improvement helped motorola," *IEEE software*, vol. 14, no. 5, pp. 75–81, 1997.

[7] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 192–201.

[8] E. Shihab, C. Bird, and T. Zimmermann, "The effect of branching strategies on software quality," in *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, 2012, pp. 301–310.

[9] A. Mockus, P. Zhang, and P. L. Li, "Predictors of customer perceived software quality," in *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.* IEEE, 2005, pp. 225–233.

[10] M. Lavallée and P. N. Robillard, "Why good developers write bad code: An observational case study of the impacts of organizational factors on software quality," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 677–687.

[11] F. Khomh and Y.-G. Guéhéneuc, "Do design patterns impact software quality positively?" in *2008 12th European Conference on Software Maintenance and Reengineering*. IEEE, 2008, pp. 274–278.

[12] B. Kitchenham and S. L. Pfleeger, "Software quality: the elusive target [special issues section]," *IEEE software*, vol. 13, no. 1, pp. 12–21, 1996.

[13] G. A. Campbell and P. P. Papapetrou, *SonarQube in action*. Manning Publications Co., 2013, ch. 1, p. 4.

[14] D. Hovemeyer and W. Pugh, "Finding bugs is easy," *Acm sigplan notices*, vol. 39, no. 12, pp. 92–106, 2004.

[15] J. Zheng, L. Williams, N. Nagappan, W. Snipes, J. P. Hudepohl, and M. A. Vouk, "On the value of static analysis for fault detection in software," *IEEE transactions on software engineering*, vol. 32, no. 4, pp. 240–253, 2006.

[16] S. Wagner, F. Deissenboeck, M. Aichner, J. Wimmer, and M. Schwalb, "An evaluation of two bug pattern tools for java," in *2008 1st International Conference on Software Testing, Verification, and Validation*. IEEE, 2008, pp. 248–257.

[17] S. Lauesen and H. Younessi, "Is software quality visible in the code," *IEEE software*, vol. 15, no. 4, pp. 69–73, 1998.

[18] C. Couto, J. E. Montandon, C. Silva, and M. T. Valente, "Static correspondence and correlation between field defects and warnings reported by a bug finding tool," *Software Quality Journal*, vol. 21, no. 2, pp. 241–257, 2013.

[19] R. Plosch, H. Gruber, A. Hentschel, G. Pomberger, and S. Schiffer, "On the relation between external software quality and static code analysis," in *2008 32nd Annual IEEE Software Engineering Workshop*. IEEE, 2008, pp. 169–174.

[20] C. Boogerd and L. Moonen, "Evaluating the relation between coding standard violations and faultswithin and across software versions," in *2009 6th IEEE International Working Conference on Mining Software Repositories*. IEEE, 2009, pp. 41–50.

[21] N. Nagappan and T. Ball, "Static analysis tools as early indicators of pre-release defect density," in *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.* IEEE, 2005, pp. 580–586.

[22] A. Vetro, M. Morisio, and M. Torchiano, "An empirical validation of findbugs issues related to defects," in *15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011)*. IET, 2011, pp. 144–153.

[23] 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 *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2018, pp. 153–163.

[24] R. E. McGrath, "Data format description language: Lessons learned, concepts and experience," Tech. Rep., 2011.

[25] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, 2014, pp. 305–319.

[26] W. H. Kruskal and W. A. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American statistical Association*, vol. 47, no. 260, pp. 583–621, 1952.

[27] C. Vassallo, F. Palomba, A. Bacchelli, and H. C. Gall, "Continuous code quality: are we (really) doing that?" in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 790–795.

Referenced URLs

[1] https://apache.org/

[2] https://svenbayer.blog/2015/08/09/maintaining-high-code-quality-with-sonarqube/

[3] https://www.triology.de/en/blog-entries/statistical-code-analysis-with-sonarqube

[4] https://github.com/HelgeCPH/ccqa_effect_study

[5] https://cwiki.apache.org/confluence/display/INFRA/SonarQube+Analysis

[6] https://docs.sonarqube.org/latest/user-guide/concepts/

[7] https://rules.sonarsource.com/java/type/Bug/

[8] https://sonarcloud.io/organizations/apache/projects

[9] https://issues.apache.org/jira, https://bz.apache.org/bugzilla

[10] https://github.com/apache/sling-aggregator

[11] https://github.com/apache/sling-aggregator/blob/master/docs/modules.md

[12] https://daffodil.apache.org, https://groovy.apache.org, https://ozone.apache.org, https://karaf.apache.org, https://ratis.apache.org

[13] size and languages assessed via the Succinct Code Counter (SCC) tool (version 2.13.0) https://github.com/boyter/scc

[14] https://github.com/apache/daffodil (d0cb60), https://github.com/apache/groovy (b3bba1), https://github.com/apache/ozone (d45819), https://github.com/apache/karaf (394dff), https://github.com/apache/ratis (a73155)

[15] https://issues.apache.org/jira/projects/DAFFODIL, https://issues.apache.org/jira/projects/GROOVY, https://issues.apache.org/jira/projects/KARAF, https://issues.apache.org/jira/projects/HDDS, https://issues.apache.org/jira/projects/RATIS

[16] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kruskal.html

[17] https://github.com/HelgeCPH/ccqa_effect_study/blob/master/notebooks/Analysis.ipynb

[18] https://rules.sonarsource.com/java/RSPEC-1181

[19] https://rules.sonarsource.com/java/RSPEC-2142