SoHist: A Tool for Managing Technical Debt through Retro Perspective Code Analysis

Benedikt Dornauer*
Michael Felderer†‡
benedikt.dornauer@uibk.ac.at
michael.felderer@uibk.ac.at
University of Innsbruck
Innsbruck, Austria

Johannes Weinzerl
Mircea-Cristian Racasan
jweinzerl@cccom.at
mracasan@cccom.at
c.c.com Moser GmbH
Graz, Austria

Martin Hess
martin.hess@softwareag.com
Software AG
Darmstadt, Germany

ABSTRACT

Technical debt is often the result of Short Run decisions made during code development, which can lead to long-term maintenance costs and risks. Hence, evaluating the progression of a project and understanding related code quality aspects is essential.

Fortunately, the prioritization process for addressing technical debt can be expedited with code analysis tools like the established SonarQube. Unfortunately, we experienced some limitations with this tool and have had some requirements from the industry that were not yet addressed.

Through this experience report and the analysis of scientific papers, this work contributes: (1) a reassessment of technical debt within the industry, (2) considers the benefits of employing SonarQube as well as its limitations when evaluating and prioritizing technical debt, (3) introduces a novel tool named *SoHist* which addresses these limitations and offers additional features for the assessment and prioritization of technical debt, and (4) exemplifies the usage of this tool in two industrial settings in the ITEA3 SmartDelta project.

CCS CONCEPTS

• **Software and its engineering** → **Software evolution**; **Software maintenance tools**; • **General and reference** → *Evaluation.*

KEYWORDS

SoHist, technical debt, software quality evolution, SonarQube

ACM Reference Format:
Benedikt Dornauer, Michael Felderer, Johannes Weinzerl, Mircea-Cristian Racasan, and Martin Hess. 2023. *SoHist:* A Tool for Managing Technical Debt through Retro Perspective Code Analysis. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE '23), June 14–16, 2023, Oulu, Finland.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3593434.3593460

*Also with University of Cologne.
†Also with German Aerospace Center (DLR), Institute for Software Technology.
‡Also with University of Cologne.

EASE '23, June 14–16, 2023, Oulu, Finland

ACM ISBN 979-8-4007-0044-6/23/06.
https://doi.org/10.1145/3593434.3593460

1 INTRODUCTION

Nowadays, software development projects tend to focus more on the quick delivery of new features to compete in the highly competitive software market rather than on delivering high-quality code. While this approach may yield increased revenue in the *Short Run*, it often results in high maintenance costs in the *Long Run*. These costs are known as Technical Debt (TD) and are a severe problem in software development projects, as exemplified in Figure 1. Numerous studies have substantiated these trends in several years of research [2, 8, 13], highlighting the detrimental impact of such circumstances on quality assurance.

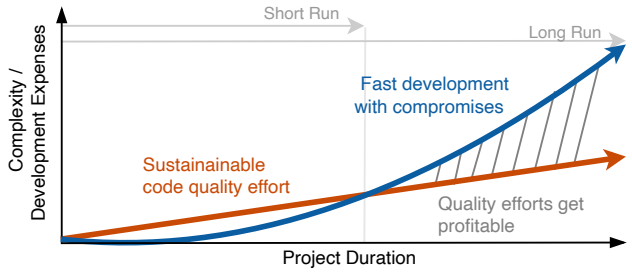

Figure 1: In a *Short Run*, more features of software systems can emerge and lead to a better turnover. But if the code quality approaches are not "sustainable", it may result in extra effort and overhead in the *Long Run*, which could lead to total project failure.

To counteract this trend, various factors have to be taken into account when evaluating, measuring, and finally addressing TD, which depend on the individual requirements and priorities of the project. Some projects may prioritize security, while others focus on performance scalability or test coverage to catch potential bugs and errors before they are deployed to production. Unfortunately, research on prioritizing TD is still in an early stage and lacks consensus on identifying important factors of TD and determining appropriate measurement methods [1].

In the ITEA3 project *SmartDelta*[1], e.g., we observed that some use case providers require a deeper understanding of how their projects have evolved over time in order to analyze and understand TD. Among those are *c.c.com Moser GmbH* and *Software AG*. For this reason, we were looking for a tool that is capable to analyze code quality but also supports the historical analysis of code commits

[1] www.smartdelta.org with partners www.softwareag.com and www.cccom.at

and deltas to investigate the code quality trend over time and to identify those code parts requiring the most maintenance effort.

2 SONARQUBE: ADVANTAGES AND LIMITATIONS FOR ANALYZING CODE EVOLUTION

Various code quality analysis tools have been developed to support the improvement of reliability and quality in software systems. In 2018 Lenarduzzi et al. [7] compiled a list of the most popular code analysis tools. Based on their findings, we adopted their methodology and investigated the current popularity of those tools in 2023. Our analysis [6] shows that SonarQube[2] is the most popular among those (see Figure 2).

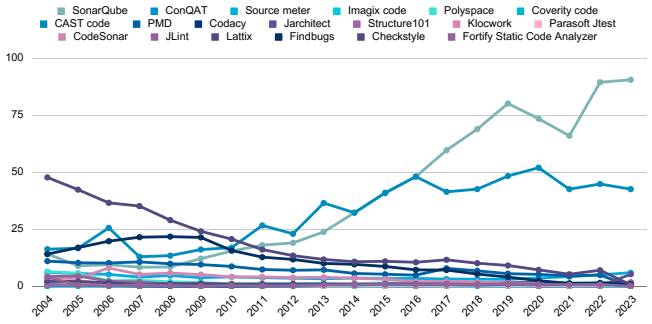

Figure 2: Google Search interest level in percent measured for popular code analysis tools since 2004. SonarQube shows the highest Google Search interest level since 2016 [6].

With more than 200 000 companies, as claimed by Sonar Source [11], the *SonarQube Community Edition* is widely established in the industry [Adv.1]. SonarQube gives companies, such as Software AG and c.c.com Moser GmbH, the opportunity to inspect code quality and code security in their software engineering pipeline, supporting over 19 programming languages in the free version and further 5 in the developer edition [Adv.2][10].

The classification system for issues used by SonarQube can be grouped into three main types. The first category is *Code Smells*, which are issues that can increase change-proneness and maintenance efforts. The second category is *Bugs*, which are issues that could result in an error or unexpected behaviour. The third category is *Vulnerabilities*, which identifies problems that can compromise the security of the software [9]. All these issues compiled into one tool are a convenient and time-saving solution for industry [Adv.3].

However, SonarQube also has some limitations, which we will discuss further:

[Lim.1] Focus on the latest project version
SonarQube analyses the modifications made to a project from the point of its initial integration. Consequently, the previous project's evolution might not be accessible, preventing the user from analysing the project's quality trend over time [12]. Specifically, this could be of interest if someone is tasked with taking over a new project. Notably, there is a workaround, but it is complicated to implement, requires detailed knowledge, and is far from being fully automated.

[Lim.2] Limited analysis options
Another limitation of SonarQube Community Edition is the need for additional filtering options for more detailed assessments. For instance, it is not possible to analyze a different branch than the main branch or to focus the analysis on specific developers.

[Lim.3] Lacking comparability between SonarQube versions
Quality metrics and rules may change between different versions of SonarQube [3]. Consequently, such discrepancies can result in different outcomes and a distorted picture of the TD evolution depending on the SonarQube versions used during the project's lifetime.

[Lim.4] Reduced visualization capabilities
SonarQube offers some basic visualizations supporting three metrics to be shown at the same time. This requires, however, that the values of the selected metrics are in the same range because otherwise, smaller distinctions are no longer visible. Another aspect to consider is the need for a single metric for interpreting TD, which may vary depending on a case-by-case basis (see Section 1). Unfortunately, SonarQube does not currently offer such a visualization.

3 SOHIST

Considering the industrial relevance of SonarQube in 2023 and the demand for extended (historical) analysis functionality, we decided to develop a new code analytics tool called *SoHist* that addresses the limitations of SonarQube and provides extended historical code analysis capabilities such as the evolution of TD over time. SoHist is open-source and available on GitHub `bdornauer/sohist`[3].

3.1 The Concept of SoHist

SoHist is a toolset available in a containerized format and can be deployed using *Docker*. The system's architectural design encompasses a SonarQube instance along with a database, the SonarScanner to trigger the code analysis, an interface to Git, as well as a web service hosting SoHist's user interface. Figure 3 illustrates the automated procedure for conducting a historical analysis of a Git repository history [Lim.1].

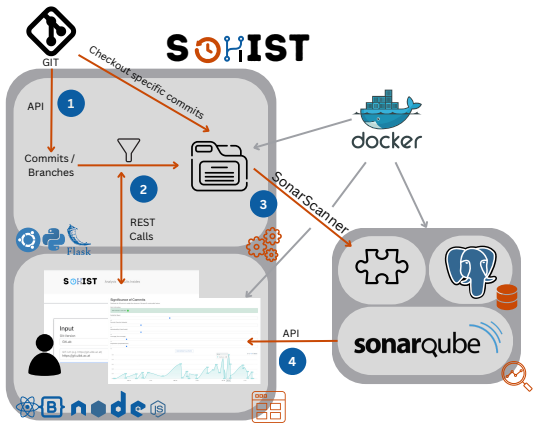

Figure 3: Conceptual structure of SoHist.

[2] www.sonarsource.com/products/sonarqube

[3] https://github.com/bdornauer/sohist

① In the first step, the user must enter some required parameters to connect SoHist to the version control system. This includes, e.g., the URL of the target GitLab instance, the target project name, and the user's access token. ② After connecting SoHist to the specified GitLab, the user can define the parameters for the historical code analysis of the selected project. This includes the time range of the analysis, the desired committers (and their corresponding commits), and a Git branch of their choice [Lim.2]. ③ Subsequently, SoHist analysis can be triggered by the user, which automatically executes individual SonarQube analysis runs for each change. By using the same SonarQube versions and thus identical metrics and rules for each run, SoHist ensures that the results for the changes are comparable [Lim.3]. If a newer SonarQube version is available, the retro perspective analysis with SoHist can be executed again, which could be cumbersome, but ensures comparability. ④ After the execution of SonarQube analysis, the user can access the code quality history and use the two available SoHist visualizations[4] – *Code Evolution* and *Weighted Code Evolution Significance*. This enables the user to track the evolution of the code quality over time and evaluate the impact of individual changes.

3.2 Outcomes and Visualization [Lim. 4]

SoHist provides the user with the capability to view multiple metrics[4] simultaneously (Visualization 1: *Code Evolution*). When the user moves the cursor over a specific time, the corresponding timestamp is highlighted across all metric charts. This allows the user to compare the various metrics against each other.

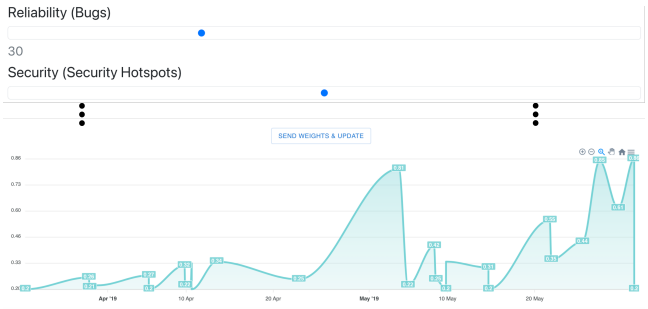

Figure 4: The user can prioritize specific metrics of interest in assigning weights. In this particular scenario of a university teaching project, Security issues were assumed to be critical (50%), followed by Bugs (30%) and Code Smells (20%). Through this visualization, it can be observed that during the initial phase, when only experienced researchers were involved in setting up the repository (skeleton), there were fewer issues. Later on, as students joined, the number of issues focusing on security increased comparatively.

The second visualization called *Weighted Code Evolution Significance*[4] introduces a novel approach to address the challenge of individual project demands and prioritization of specific SonarQube's main metrics, as outlined in Section 2. This approach allows users to focus their analysis on multiple specific main metrics of their choice (Reliability, Security, Maintainability, Test Coverage and Duplicated Lines) by assigning individual metric weights. Based on this weighting and the data obtained from the individual SonarQube runs, a visual representation of the project's life cycle is shown, as demonstrated in Figure 4, highlighting those changes that have had the greatest impact according to the chosen weights. The higher the *Weighted Code Evolution Significance*, the more significant the change related to the weighted category is. For a detailed description of the calculations, take a look at [5]. Further, a detailed analysis can be conducted using the first visualization.

[4]Videos for demonstration: https://doi.org/10.5281/zenodo.7713782

4 INDUSTRIAL CHALLENGES AND USAGE OF SOHIST

4.1 c.c.com Moser GmbH - Logistics and Personal Mobility

Background: One of the solutions provided by c.c.com Moser GmbH are their BLIDS sensors, which enable the measurement of traffic flow on the street using Bluetooth, WiFi and Bluetooth Low Energy data. In temporary deployment scenarios, these sensors typically rely on battery power and require periodic replacement. Consequently, c.c.com Moser GmbH aims to reduce the energy drain of the sensors.
Scope: Given the high cost associated with hardware exchange, c.c.com Moser GmbH has undertaken efforts to address energy consumption at the software level (design decisions, data compression, etc.). The company has sought to establish potential correlations between software quality metrics and physical properties, such as specific energy consumption. Over the years, c.c.com Moser GmbH has amassed a wealth of physical measurements that will be used in the upcoming analysis tasks. On the software level, c.c.com Moser GmbH still needs to collect software quality metrics over time and has needed a tool that facilitates historical code analysis with support for multiple programming languages.
Usage of SoHist: Therefore, c.c.com has initiated an analysis of the firmware repository. For every historical update of the sensor software, c.c.com collects energy-related code metrics via the SonarQube API. This is only feasible with SoHist's retrospective analysis, as no previous measurements are available. Metrics of particular interest include, for instance, the *McCabe Cyclomatic Complexity* or *Lines of Code* as prior research by Corrêa et al. [4] has demonstrated their potential impact on the energy consumption of embedded systems. In the ongoing activities of SmartDelta, c.c.com has the historical energy-related code metrics available and can investigate if specific changes (related to the code metrics) impacted the energy consumption. With SoHist, c.c.com Moser GmbH can now address its correlation efforts.

4.2 Software AG – Enterprise Software

Background: Enterprises are highly dependent on the availability and reliability of their software in today's world. Any malfunction resulting in unavailable or severely slowed down software systems may severely impact their business. Continuously improving the codebase is thus critical for Software AG as a leading supplier of enterprise software to deliver high-quality products and increase customer satisfaction.

Scope: The more complex a software product gets, the more time and effort must be invested in code maintenance, leaving less time for feature development. Thus, identifying which parts of the code are causing problems – such as security or performance issues – or require increased maintenance is critical to efficiently solving issues and managing efforts and costs. Software AG Research uses SonarQube for the detection of common bugs, security issues and for analyzing the overall code quality. However, SonarQube does not give direct insights into the historical evolution of the codebase. This information would enable the analysis of TD and could lead to a better understanding of a software development project's progression over time.

Usage of SoHist: Software AG Research used SoHist to assess the code quality trend within one of their research projects. The SoHist analysis and especially the novel *Weighted Code Evolution Significance* view, focusing on Bugs and Code Smells, revealed some interesting insights. Figure 5 presents the analysis results of the Master branch. The increased number of Bugs and Code Smells shown for the Master branch in the project's early to mid-stage (1) are indicators for technical debt. Further analysis revealed that during this time frame, the team focused on developing new features rather than improving the code quality on the Master branch. In the middle of March, the team incorporated a new process for addressing code quality issues. This led to a substantial reduction in code quality issues. Thanks to the new process, the code quality issues observed at the end of May (2) have been addressed and solved faster this time. Through SoHist, Software AG Research was able to reproduce this time frame and assess the actual benefits of the new quality assurance process.

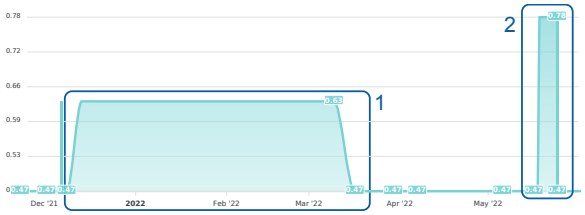

Figure 5: Software AG Use Case: Historical analysis of the master branch of a project, focusing on bugs and code smells.

5 CONCLUSION

Maintaining a balance between developing new features and ensuring software quality is essential for the success of software development projects. Achieving this balance requires precise monitoring and management of TD overtime. To support this effort, automated tools are necessary to analyse changes and identify areas requiring high maintenance. One of these widely used tools in the industry is SonarQube.

In this paper, we highlighted some limitations of SonarQube and proposed a new tool, named SoHist, to address these limitations and perform historical code quality analysis. In addition, SoHist introduces a new visualization approach, which allows the weighting of code quality aspects according to individual project needs.

The tool has demonstrated its benefits in two industrial settings, making it a good candidate for implementation in other industrial projects. Nevertheless, SoHist is still a prototype and thus has some limitations. At present, SoHist only shows the timestamp of the commits, but further details, such as the committer, still need to be included. Similarly, it is currently not possible to directly open the corresponding commit in GitLab or issue description in SonarQube. These usability issues may limit the overall root cause analysis of a project. Besides that, a potential risk to the long-term viability of SoHist is the possibility of restrictions imposed by SonarQube.

In future work, we will conduct further assessments to evaluate the usability, including performance and comprehensibility issues, in the industry domain. Furthermore, we plan to make improvements, such as automated plugin integration or disposal of current limitations.

ACKNOWLEDGMENTS

This work has been supported by and done in the scope of the ITEA3 SmartDelta project, which has been funded by the Austrian Research Promotion Agency (Grant No. 890417) and the German Federal Ministry of Education and Research (Grant No. 01IS21083A).

REFERENCES

[1] Maria Teresa Baldassarre, Valentina Lenarduzzi, Simone Romano, and Nyyti Saarimäki. 2020. On the Diffuseness of Technical Debt Items and Accuracy of Remediation Time When Using SonarQube. *Information and Software Technology* 128 (Dec. 2020), 106377. https://doi.org/10.1016/j.infsof.2020.106377

[2] R. Baskerville, L. Levine, J. Pries-Heje, and S. Slaughter. 2001. How Internet Software Companies Negotiate Quality. *Computer* 34, 5 (May 2001), 51–57. https://doi.org/10.1109/2.920612

[3] Hendrik Buchwald. 2022. How Often Are Builtin Rules Updated in SQ. https://community.sonarsource.com/t/how-often-are-builtin-rules-updated-in-sq/69493

[4] Ulisses Brisolara Corrêa, Luis Lamb, Luigi Carro, Lisane Brisolara, and Júlio Mattos. 2010. Towards Estimating Physical Properties of Embedded Systems using Software Quality Metrics. In *2010 10th IEEE International Conference on Computer and Information Technology*. IEEE, Marrakech, 2381–2386. https://doi.org/10.1109/CIT.2010.409

[5] Benedikt Dornauer. 2023. Computations behind the Weighted Code Evolution Significance. https://doi.org/10.5281/ZENODO.7713698

[6] Benedikt Dornauer. 2023. Trend of Code Analysis Tools 2004 -2023. https://doi.org/10.5281/ZENODO.7713953

[7] Valentina Lenarduzzi, Alberto Sillitti, and Davide Taibi. 2020. A Survey on Code Analysis Tools for Software Maintenance Prediction. In *Proceedings of 6th International Conference in Software Engineering for Defence Applications*, Paolo Ciancarini, Manuel Mazzara, Angelo Messina, Alberto Sillitti, and Giancarlo Succi (Eds.). Vol. 925. Springer International Publishing, Cham, 165–175. https://doi.org/10.1007/978-3-030-14687-0_15

[8] Robert Ramač, Vladimir Mandić, Nebojša Taušan, Nicolli Rios, Sávio Freire, Boris Pérez, Camilo Castellanos, Darío Correal, Alexia Pacheco, Gustavo Lopez, Clemente Izurieta, Carolyn Seaman, and Rodrigo Spinola. 2022. Prevalence, Common Causes and Effects of Technical Debt: Results from a Family of Surveys with the IT Industry. *Journal of Systems and Software* 184 (Feb. 2022), 111114. https://doi.org/10.1016/j.jss.2021.111114

[9] SonarSource. 2023. Metric Definition. https://docs.sonarqube.org/latest/user-guide/metric-definitions/

[10] SonarSource. 2023. SonarQube. https://www.sonarsource.com/products/sonarqube/

[11] SonarSource. 2023. SonarQube - Downloads. https://www.sonarsource.com/products/sonarqube/downloads/

[12] Paul Spencer. 2022. Have SonarQube Create Historical Data. https://community.sonarsource.com/t/have-sonarqube-create-historical-data/60492

[13] June Verner, Jennifer Sampson, and Narciso Cerpa. 2008. What Factors Lead to Software Project Failure?. In *2008 Second International Conference on Research Challenges in Information Science*. IEEE, Marrakech, 71–80. https://doi.org/10.1109/RCIS.2008.4632095