TEAMSCALE: Tackle Technical Debt and Control the Quality of Your Software*

Roman Haas
CQSE GmbH
Munich, Germany
haas@cqse.eu

Rainer Niedermayr
CQSE GmbH, University of Stuttgart
Munich, Germany
niedermayr@cqse.eu

Elmar Juergens
CQSE GmbH
Munich, Germany
juergens@cqse.eu

***Abstract*—TEAMSCALE is a software intelligence platform, that is, it creates transparency on code quality and the underlying software development process. This makes it possible for developers, testers and managers to better understand and control technical debt of their systems. In this paper, we give an overview of TEAMSCALE and how this tool can be used in practice to control and lower technical debt in the long run. We explain which code analyses can be used to identify and address technical debt. TEAMSCALE is available for free for research and teaching purposes at www.teamscale.io.**

***Index Terms*—Technical Debt, Software Quality, Quality Control**

I. SOFTWARE INTELLIGENCE

Hassan and Xie [6] defined *Software Intelligence* as concepts and techniques that offer software practitioners up-to-date and pertinent information to support their daily decision-making processes. TEAMSCALE is a software intelligence platform, which analyzes source code and raw data from software development and the underlying process. So, the analysis results help to better understand the quality of software and make it possible to control and lower technical debt.

II. OVERVIEW OF TEAMSCALE

The basic idea behind TEAMSCALE is to gather all relevant data at a central place, and immediately provide analysis results from this data using incremental analysis [1]. TEAMSCALE processes data from version control systems, issue trackers, and various third-party analysis tools like language-specific code checkers or test coverage tools (see Fig. 1). The results are available for the whole version history and all development branches in the web client of TEAMSCALE [5]. In addition, plugins in the IDE allow developers to view findings and further information directly in their working environment. Using the pre-commit feature of the IDE plugins, analysis results are provided not only in real-time [7], but even without needing to commit the changes to the version control system. That is, developers get feedback on their code changes in real-time and can address new findings immediately.

* This work was partially funded by the German Federal Ministry of Education and Research (BMBF), grant “SOFIE, 01IS18012A”. The responsibility for this article lies with the authors.

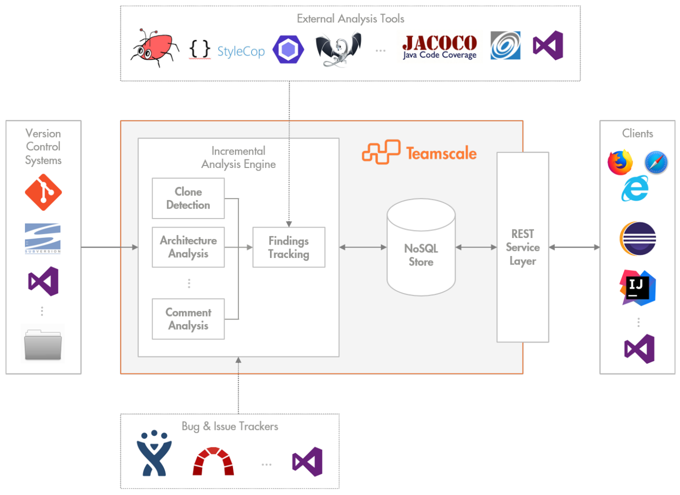

Fig. 1. Architecture of Teamscale

III. QUALITY CONTROL PROCESS

Code quality decays over time if it is not explicitly taken care of [3]. TEAMSCALE supports the quality control process suggested by Steidl et al. [12], which focuses on preserving or improving the code quality of software systems.

Core element of the process are project-specific *Quality Goals* which express how much technical debt is allowed for a specific project:

1) *Indifferent:* No quality monitoring at all, i.e., no technical debt control
2) *Preserving:* No new findings—existing quality findings are tolerated but new ones need to be addressed immediately.
3) *Improving:* No findings in modified code. In the long run, this reduces technical debt in the code base.
4) *Perfective:* No findings at all in the whole code base.

From our experience, depending on the project context (e.g., green-field vs. brown-field engineering, business criticality, expected operation time), a more or less ambitious quality goal should be chosen. For example, in a brown-field engineering project with high business criticality over the next years, we suggest to aim for the *improving* quality goal.

TEAMSCALE is able to differentiate between new findings and findings in modified code. Moreover, using baselines, it is possible to focus on recent changes, e.g., since the last release.

The *Quality Engineer* of the project (which can be a team member, a colleague from another team or an external) regularly writes a report on the current system quality and changes since the last report. As part of the manual analysis of the quality engineer, he also inspects findings that should have been addressed according to the quality goal. For those that need to be addressed, he writes a quality task including a suggestion for resolution. TEAMSCALE keeps track of the findings associated with the task and the resolution status of the task itself.

IV. STATIC AND DYNAMIC ANALYSES

TEAMSCALE provides static and dynamic code analysis for 26 programming languages. In the following, we provide a short overview of the different analysis and how they can be used to reduce technical debt.

A. Code Quality Analyses

a) Code Structure Metrics: Poorly structured code (i.e., long methods, deeply nested code, and long files) is hard to read, understand, and thus, hard to maintain. Findings are created for elements that violate the specified thresholds.

b) Code Duplication: TEAMSCALE uses an incremental, index-based clone detection [8] to reveal code redundancies caused by copy-paste programming. Code duplication is a classic example for technical debt as it is very easy to copy-paste code but increases maintenance efforts in the long run.

c) Comment Completeness: Public interfaces, classes and methods should be documented so that developers quickly get an idea of the implemented functionality, including all pre- and postconditions. Missing or outdated comments cause misunderstandings and wrong uses, which is why TEAMSCALE creates findings for such cases.

d) Architecture Conformance: An up-to-date architecture specification helps developers and managers to keep an overview of their system, its single components and how they are intended to interact with each other. In TEAMSCALE, it is possible to specify architectures as hierarchically nested components which are mapped to files or types [2]. Using policies, architects define which components may interact with each other. New dependencies are displayed in real-time and findings are created for new, unmatched types, so that the architect can easily update the architecture specification.

B. Test Gap Analysis

Non-tested, modified code is much more fault-prone than code that was executed by tests [9]. Test Gap Analysis identifies such test gaps (i.e., added or modified, but intested methods) by combining information from static and dynamic analysis [9]. TEAMSCALE can differentiate between different coverage sources, e.g., unit test coverage from a continuous integration pipeline and coverage from manual tests. Using the results from test gap analysis, testers and managers can decide on a factual basis whether further tests need to be executed.

Test gaps can be identified on ticket basis [11], as well. To do so, TEAMSCALE links issues with code changes and test execution information. This helps developers to see whether their code changes for a specific ticket have been tested.

C. Usage Analysis

An example of a dynamic analysis in TEAMSCALE is usage analysis [4]. TEAMSCALE helps to identify unused code and to prepare its removal. We have seen software systems with millions of lines of code in 28% of the implemented features are not used [10]. Cleanups that remove unused code lower technical debt of the software system as future maintenance efforts will not be spent on unnecessary features anymore.

V. CONCLUSION

TEAMSCALE is a software intelligence platform, which helps developers in their daily life to identify opportunities to reduce existing and avoid additional technical debt. TEAMSCALE creates transparency towards managers on the actual quality of their software systems and indicates what needs to be addressed to improve code and process quality. In this paper, we showed how to apply the quality control process with the help of TEAMSCALE. Addressing findings from the presented analyses will help to reduce technical debt in the long run.

REFERENCES

[1] V. Bauer et al. “A Framework for Incremental Quality Analysis of Large Software Systems”. In: *ICSM*. 2012.

[2] F. Deissenboeck et al. “Flexible Architecture Conformance Assessment with ConQAT”. In: *ICSE*. 2010.

[3] S. G. Eick et al. “Does code decay? Assessing the evidence from change management data”. In: *TSE* 27.1 (2001).

[4] A. Goeb. *How much of your code do you actually use?* Blog. http://cqse.eu/blog-code-usage.

[5] N. Goede. “Quality Control in Action”. In: *Softwaretechnik-Trends* 35.2 (2015).

[6] A. Hassan and T. Xie. “Software Intelligence: The Future of Mining Software Engineering Data”. In: *Workshop on Future of SE Research*. 2010.

[7] L. Heinemann, B. Hummel, and D. Steidl. “Teamscale: Software Quality Control in Real-Time”. In: *ICSE*. 2014.

[8] B. Hummel, E. Juergens, and D. Steidl. “Index-based model clone detection”. In: *International Workshop on Software Clones*. 2011.

[9] E. Juergens and D. Pagano. “Haben wir das Richtige getestet? Erfahrungen mit Test-Gap-Analyse in der Praxis”. In: *Software-QS-Tag*. 2016.

[10] E. Juergens et al. “Feature Profiling for Evolving Systems”. In: *ICPC*. 2011.

[11] J. Rott et al. “Ticket Coverage: Putting Test Coverage into Context”. In: *Workshop on Emerging Trends in Software Metrics*. 2017.

[12] D. Steidl et al. “Continuous Software Quality Control in Practice”. In: *ICSME*. 2014.