Skuld: A self-learning tool for impact-driven technical debt management

Josep Burgaya Pujols
Skuld Technologies
Barcelona, Spain
josep.burgaya@skuld.ai

Pieter Bas
Skuld Technologies
Barcelona, Spain
pieter.bas@skuld.ai

Silverio Martínez-Fernández
Fraunhofer IESE and UPC
Kaiserslautern, Germany
silverio.martinez@upc.edu

Antonio Martini
Skuld and University of Oslo
Oslo, Norway
antonio.martini@ifi.uio.no

Adam Trendowicz
Fraunhofer IESE
Kaiserslautern, Germany
adam.trendowicz@iese.fraunhofer.de

ABSTRACT

As the development progresses, software projects tend to accumulate Technical Debt and become harder to maintain. Multiple tools exist with the mission to help practitioners to better manage Technical Debt. Despite this progress, there is a lack of tools providing actionable and self-learned suggestions to practitioners aimed at mitigating the impact of Technical Debt in real projects. We aim to create a data-driven, lightweight, and self-learning tool positioning highly impactful refactoring proposals on a Jira backlog. Bearing this goal in mind, the first two authors have founded a startup, called Skuld.ai, with the vision of becoming the go-to software renovation company. In this tool paper, we present the software architecture and demonstrate the main functionalities of our tool. It has been showcased to practitioners, receiving positive feedback. Currently, its release to the market is underway thanks to an industry-research institute collaboration with Fraunhofer IESE to incorporate self-learning technical debt capabilities.

CCS CONCEPTS

- Software and its engineering → Maintaining software
- Information systems → Data analytics; Data mining

KEYWORDS

Technical Debt, Project Management, Tool, Data-Driven Development

ACM Reference format:

Josep Burgaya Pujols, Pieter Bas et al. 2020. Skuld: A self-learning tool for impact-driven technical debt management. In *TechDebt 2020*. Seoul, South Korea, May 25-26, 2020, 2 pages. <https://doi.org/1>

1 Introduction

Technical Debt (TD) is regarded as sub-optimal solutions taken in the course of software development, that even though it might have been beneficial in the short term, cause an additional negative impact in the long term (e.g., to software maintainability).

The long-term negative impact is referred to as the interest paid on the debt [1]. As long as the debt is not repaid the organization pays interest, which is likely to grow in the future. Indeed, in a recent study, companies reported that on average 33% of the developer workweek is spent addressing TD, which in absolute terms amounts to approximately \$300 billion worldwide [2]. In this context, the research community has been gaining interest in TD in recent years and new tools are now emerging [3, 4].

Current market solutions present challenges such as: not identify and prioritize smells by their specific negative impact, and only focusing on a subset of aspect or types of TD (e.g., Code TD, Architecture TD, Documentation TD) [5]; and more importantly, tools are rule-based and lack self-learning capabilities. Additionally, to identify gaps in the current market, we conducted twenty-five interviews with a diverse (in terms of geography and team size) sample of engineers and technical managers. We found out the following problems in state-of-the-art tools' recommendations to mitigate TD: (a) current tools generally provide a plethora of individual TD items with little to no associated negative impact; (b) they are not positioned where decisions take place, therefore they do not take part in the sprint/iteration objective shaping process; and (c) the value of carrying them out (future interest decrease) is not made apparent, therefore stakeholders with decision powers are inclined to prioritize objectives that they feel are more relevant business-wise.

In this context, Skuld was founded with the purpose of fulfilling these gaps with the vision to become the go-to company for software renovation. In this tool paper, we present a data-driven, lightweight and self-learning tool to proactively manage TD for software development teams. This tool paper presents the software architecture and main functionalities in Sections 2 and 3.

2 Tool architecture

The tool is, built in the form of a classic Lambda architecture [6] as depicted in Figure 1, is divided in four main layers.

Figure 1: Architecture overview and data flow of the presented tool.

Connectors and analyzers. The first step is to gather data from its configured data sources including Jira and Git. The data is then processed using specialized analyzers such as Sonarqube for Code TD, Skuld Artio for Architectural TD or other analyzers that specialize in one dimension of TD. This is one of the main characteristics of the tool's architecture as it is designed to be as lightweight as possible. To set up the tool, it is merely required to configure a connection to the most essential data sources. We have found that a major hurdle in the adoption of tools that use several sources of information is the fact that it typically requires several days of work in order to configure them properly [7]. Furthermore, the tool is a Jira plugin making the process of purchasing, downloading and installing as simple as possible, maximizing user addressability.

Application. Using the metrics and smells obtained by the analyzers, the tool makes use of its quality *prediction models* to identify relevant quality deficits in the code at a multi-dimensional level and to estimate their negative impact on project performance. As a result, it is able to assess the project performance health status.

Learning. This is an innovative part of the tool as cross-project data will be collected and leveraged to create and continuously evolve quality *prediction models* that will enable TD estimation including its long-term impact. These self-learning capabilities will constantly improve the tool's estimations. In order to accurately detect the existing TD in projects, it will be context-aware and will learn from evolutionary bugs.

Presentation and Interaction. Inside Jira, the tool will present a dashboard with the most relevant project and business metrics including an estimation of its future evolution. The tool will also support root-cause analysis, in which users can trace project performance deficits back to associated quality deficits in software code. Jira is a widely used tool, and therefore Skuld is compatible with the most used technologies in software teams, which attracts users not willing to deploy yet another tool. Moreover, it will provide actionable recommendations for engineering teams to address TD on time, with the confidence that these are grounded on the available project data and motivated in specific predicted improvements of the metrics.

3 Tool demo

The main view of the tool is a dashboard located at the Jira project with metrics relevant to both technical and business decisions makers (see Figure 2). Refactoring Proposals (RP) to solve TD are automatically positioned in the Jira project issues backlog in the form of issues. RPs are characteristically actionable and provide information of the expected benefit of carrying them out. These are created in a completely autonomous way thanks to the built-in self-learning capabilities provided by Fraunhofer IESE.

Figure 2: Demo of the tool. Dashboard of metrics.

Currently the team is working on creating a TD index so companies can freely benchmark their software TD score against their peers. Further features are available on <https://www.skuld.ai/>

REFERENCES

- [1] Martini, A, Bosch, J. 2017. On the interest of architectural technical debt: Uncovering the contagious debt phenomenon. J Softw Evol Proc.
- [2] Stripe. 2018. Software Engineering Efficiency and Its \$3 Trillion Impact on Global GDP. The developer coefficient. Retrieved February 19th, 2020 from <https://stripe.com/files/reports/the-developer-coefficient.pdf>
- [3] 2018. Proceedings of the 2018 International Conference on Technical Debt. SESSION: Tools track. ACM.
- [4] 2019. Proceedings of the Second International Conference on Technical Debt. SESSION: Tool demos at viger. IEEE Press.
- [5] Zengyang Li, Paris Avgeriou, and Peng Liang. 2015. A systematic mapping study on technical debt and its management. J. Syst. Softw.
- [6] Marz, Nathan; Warren, James. *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications, 2013.
- [7] S. Martínez-Fernández et al. Continuously Assessing and Improving Software Quality With Software Analytics Tools: A Case Study. 2019. IEEE Access