December 04, 2018

DATA-DRIVEN CHARACTERIZATION OF TECHNICAL DEBT IN A COMPLEX INFORMATION SYSTEM

Abhishek Pathak

Hossein Moosavi Kooshki

Mazhar Haque

Girish Babu

DATA-DRIVEN CHARACTERIZATION OF TECHNICAL DEBT IN A COMPLEX INFORMATION SYSTEM

AUTHORS:
Abhishek Pathak
Hossein Moosavi Kooshki
Mazhar Haque
Girish Babu

ABSTRACT

Presented herein are techniques that provide a holistic and integrated abstraction among different categories of technical debt (TD) in a complex software system, as well as among different TD-related data sources such as logs, traces, telemetry, and metrics. The techniques presented herein allow for accelerated, automated, and evolutionary TD management in a complex software development life cycle (SDLC). The techniques learn the context throughout the SDLC pipeline and turn this context into actionable insights for use in repaying the technical debt at the earliest stages of the development process. The techniques presented herein provide an automated and low cost mechanism that may reduce debt within a company.

DETAILED DESCRIPTION

Continuous and fast delivery of software system customer value needs to be supported both in the short and long term. However, technical debt (TD) can severely hamper both the evolution and maintenance of software systems. TD analysis is typically performed in isolation and is limited to developer check-ins/post check-in baseline Static Analysis (SA) runs or regression and/or Unit Test / Smoke Test (UT) scope. Missing pieces in the TD jigsaw puzzle include, for example, lack of soft computing, data-driven analytics, and a connected end-to-end pipeline. These missing pieces typically render TD analysis usable only in certain contexts, and/or limits the scale of TD analysis in software systems. At the telemetry end, conventional TD analysis techniques rely on instrumentation and, at the developer end, conventional TD analysis techniques rely on static analysis. However, in the conventional techniques, the overall system is not

Published by Technical Disclosure Commons, 2018

connected as a data-driven pipeline to provide a meaningful automated machine learning driven pipeline to connect code, static analysis, dynamic analysis, UT / Regressions, Bug Tracking System, Technical Escalations, and End point telemetry data. Therefore, conventional techniques are unable to provide TD in terms of currency (e.g., dollars), effort estimates, Meant Time to Resolve (MTTR), and/or ETA estimates of any changes / fixes to the system, as a connected pipeline. As a result, people in different parts of the SDLC spectrum have different views of software quality and change impact.

According to recent research, the cost of managing code debt in large software enterprises can by as much as 25% of the whole development time. Additional expenses may also be incurred, for example, with complex software and unnecessary code (e.g., requiring extensive testing to eliminate unintended side-effects). For example, assuming a company has roughly 70,000 employees, of which approximately thirty (30) percent (%) are Engineers. Assuming the annual cost of an engineer to be roughly $200,000 USD, this gives an estimate of (70,000*0.3*200,000*0.25) = $ 1.05B USD as the annual cost of managing technical debt within this company. The total addressable market of TD management, even only considering the enterprise software, is significant.

Current TD management is generally an ad-hoc process, where TD is mostly tracked using sprint or product backlogs, common issue trackers or even simple excel spreadsheets. Static analysis tools might be in use, but they do not provide a holistic view of TD. Current efforts in TD management are not systematic and methodical and the traditional X-ray based or drill-down based approaches lack the key features which would be needed to provide a strong data-driven machine learning (ML)-based connected pipeline. For example, conventional techniques lack a needed feedback loop system to make the process adaptive and evolutionary (e.g., current systems require re-engineering and re-tuning every couple of years). Additionally, conventional techniques do not address change estimates at different levels of SDLC, and do not provide data in terms of currency costs (e.g. dollars) of effort, aka Technical Debt. These conventional techniques are usually limited to the developer space and do not venture beyond Bug Tracking System. A lack of a data-driven approach makes most of the present TD estimations inherently static in nature. Lack of soft computing approaches and conventional inclination towards hard-

computing / number crunching base approaches make the resulting systems fragile to the inherent ambiguity that comes with complex system software.

In summary, traditional TD estimations are based on static rule-based methods and/or aggregating different Key Performance Indicators (KPIs) using human-selected weights. The techniques presented herein provide a step towards a paradigm shift to a fully data-driven approach. In particular, the techniques presented herein employ applied mathematics and machine learning to create a solution with the following distinct merits -

-

- Machine Learning (ML)-powered.
- Fully data-driven and based on dynamic weighted sampling.
- Leading indicator to areas prone to TD in future.
- Vertically and horizontally scalable.
- Free of human bias.
- Constantly evolving (through reinforcement learning).
- Supplemented by prescriptive Insights.

Presented herein is data-driven TD analytics platform, referred to as "TeDDy," which connects and integrates multiple tools at different stages of the software development life cycle (SDLC). The proposed solution begins by collecting pre check-in data-driven metrics on code quality. From there, TeDDy goes from to check-in, Static Analysis baseline and UT/ Regression, Bug Tracking System, and beyond to establish a connected data-driven pipeline to benefit users. These users may be, for example, someone invoking TeDDy as a stand-alone runnable application program interface (API) for branch merges (e.g., to observe before and after measures of data-driven ML-based quality indicators), someone writing code, someone debugging on a bug / crash at customer end, *etc*. In its completed form, TeDDy should also be able to provide customers with insights built on top of the inherent measures of Technical Debt (TD), System complexity, and the involved "people effort." As such, TeDDy (i.e., an ML-based and data-driven end-to-end pipeline) provides reliable measures and indicators of Technical, Architectural, Servicability and Security Debts to the development community, while also providing valuable, actionable, and prescription based insights to the other end of the SDLC

spectrum. This includes the Technical Support Engineering, escalations teams, and end-customers.

The techniques presented herein allow an aligned prioritization of TD across the system and enable all of the TD management decisions to be completely data-driven through leveraging the statistics collected on historical data or by benchmarking the system against a collection of baselines. TeDDy also provides an integration abstraction among different categories of TD in a complex information system, as well as among different TD-related data sources like logs, traces, telemetry, and metrics. TeDDy provides a holistic TD estimation metrics system, including trailing and leading indicators. Moreover, TeDDy keeps the connective tissue of the events and affords the operational ability to drill down to raw events. TeDDy also goes beyond descriptive and predictive analytics, by providing actionable prescriptive insights on where to focus the improvement efforts.

Prior to TD analysis, the following data is collected:

- Types and subtypes of TD that needs to be measured/tracked/evaluated.
- Costs of most frequent changes to the system, including direct cost and opportunity cost.
- Owners of TD issues across the system.
- Domains with debt incurred by the time of creation/old architecture/badly instrumented code.

TeDDy provides a standardized, fully automated, and iterative process to identify, estimate, prioritize, and repay TD.

The major steps in the TeDDy's workflow are as follows:

1. Correlate the pre-aggregated data to critical path analysis, which is crucial to determine if components are failing.
2. Identify and categorize TD Key Performance Indicators (KPIs), including:
    - code related features, e.g., code duplication, average/max number of LoC per class/method, the degree of reliance on outdated frameworks/libraries/applications.
    - automated testing related features.
    - security debt.
    - servicability debt.

- establish debuggabilty.
- static analysis related features.
- defect and escalation data related features.
- architecture related features, e.g., using dependency checkers.
- documentation related features.

3. Estimate the size of TD items (in dollars or engineering hours).
    - Identify debt interest thresholds (the amount of interest paid or predicted if the refactoring is not conducted) and provide assessment for rework prioritization.
4. Aggregate TD items in a data-driven pipeline and using a dynamic weighted sampling.
5. Learn from TD trends and predict future prone areas.
6. Give prescriptive and remedial pointers to prioritize.
    - Describe managerial mechanisms to establish a feedback loop between TD analyses and actions that must be performed.
    - Implement plans to track the progress before and after prescriptions

Figure 1, below, illustrates the overall architecture of TeDDy.

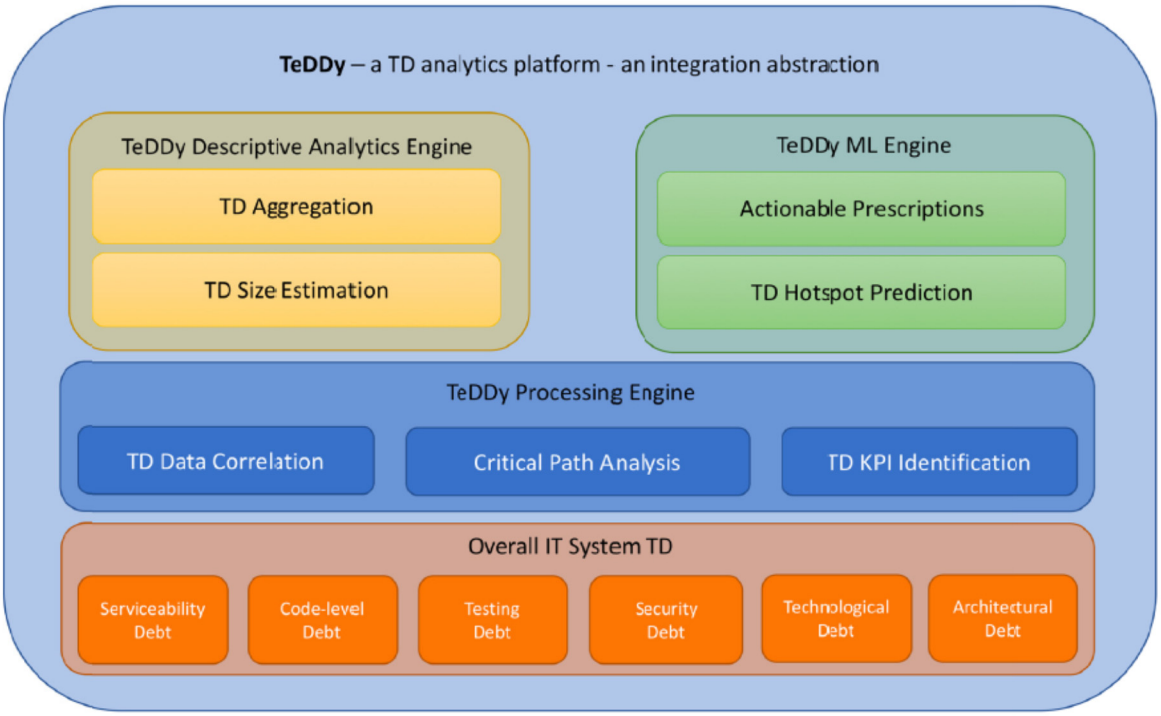

FIG. 1