 e-ISSN No. 2584-1025

Improving The Effectiveness and Efficiency of Technical Debt Management Automation

Name: Vaishali Jorwekar*, Dr. Rajneeshkaur Sachdeo
Affiliation: MIT School of Computing, MIT School of Computing
Email id: vaishalirjorwekar@gmail.com, rajni.sachdeo@mituniversity.edu.in

Abstract - Efficient Technical Debt Management Automation (TDMA) is crucial in software engineering to mitigate the detrimental effects of technical debt accumulation. This research addresses the challenges of integrated, comprehensive TDMA, aiming to enhance software development processes. The key challenges include the lack of detailed strategy implementation due to interdependencies between TDMA activities, maintenance of TD types in siloes, ambiguity in rules for TD activities, and integration issues among automation artifacts. By proposing a holistic approach, this study seeks to establish a comprehensive strategy with context-aware decision support, leveraging ML/AI for further automation through establishment of digital thread between within the ecosystem. The anticipated outcomes include improved capabilities for identifying, prioritizing, and addressing technical debt, along with establishing an integrated framework for TDMA throughout the software development lifecycle. The implications of this research extend to advancing software engineering practices, fostering innovation, and enhancing the competitiveness of organizations in the dynamic technology landscape.

Keywords — Digital thread, Technical Debt Management, Generative AI, Software Development Lifecycle, Automation Framework

I. INTRODUCTION

In the dynamic landscape of software development, Technical Debt (TD) has emerged as a significant challenge impacting productivity, quality, and ultimately, business outcomes. Stemming from the analogy of financial debt, technical debt refers to the trade-off made during software development where expedient solutions are chosen over optimal ones, leading to accrued inefficiencies and complexities over time. This phenomenon occurs across various dimensions of software development, including code quality, documentation, infrastructure, and versioning.

The ramifications of technical debt are far-reaching, affecting not only development but also maintenance efforts. Studies indicate that developers spend a substantial portion of their time grappling with technical debt, significantly impeding productivity. Moreover, the maintenance phase of software projects becomes increasingly burdensome, with a majority of efforts allocated to addressing issues stemming from technical debt.

Recognizing the pervasive impact of technical debt, effective management strategies are imperative to mitigate its adverse effects. Technical Debt Management (TDM) encompasses a proactive approach to identifying, measuring, prioritizing, and addressing technical debt throughout the Software Development Life Cycle (SDLC). However, despite the critical importance of TDM, several challenges hinder its automation and optimization.

This research aims to address the challenges surrounding the automation of Technical Debt Management (TDM), with a focus on enhancing efficiency and effectiveness in software development practices. By delving into the intricacies of TDM automation, the research endeavors to provide comprehensive solutions that streamline processes, facilitate informed decision-making, and minimize the accumulation of technical debt.

The outcomes of this research are anticipated to represent a significant advancement in the field of software engineering. By offering integrated, context-aware automation solutions for TDM, this study seeks to empower development teams to proactively manage technical debt, thereby enhancing software quality, reducing maintenance overhead, and ultimately improving organizational agility and competitiveness.

II. LITERATURE REVIEW

According to a literature survey, the TD concept 47% of practitioners have some actual experience with TD identification or management, compared to 22% of practitioners who simply have theoretical understanding. Research conducted by authors [5] highlights the varying levels of familiarity and practical experience with technical debt among software development practitioners. While a substantial portion (69%) of participants acknowledged awareness of the TD concept, a significant proportion (31%) had no prior exposure to it. Moreover, within the cohort familiar with technical debt, only 47% possessed practical experience in managing or addressing TD issues, underscoring the need for enhanced education and awareness initiatives in the software engineering community.

Recent research indicates that in software development organizations, TD-caused problems account for an average of 25% of development effort—that is, wasted work. Time pressure and deadlines emerge as primary drivers of technical debt, as indicated by numerous studies [5]. The imperative to deliver within constrained timeframes often compels developers to make trade-offs that prioritize short-term gains over long-term sustainability, thereby accumulating technical debt. Additionally, factors such as non-adoption of good practices, lack of experience, and insufficiently defined processes contribute to the proliferation of technical debt within software projects.

The effects of technical debt reverberate across various dimensions of software development, with delivery delays,

reduced maintainability, and increased rework among the most commonly cited consequences [5]. These adverse effects not only impede project progress but also exacerbate the complexity and cost of software maintenance over time, underscoring the importance of proactive debt management strategies.

A comprehensive understanding of technical debt necessitates delineation across various dimensions, including design, test, code, architecture, documentation, and process debt [5]. Each type of technical debt manifests distinct challenges and implications for software development, ranging from violations of design principles to deficiencies in testing activities and architectural shortcomings. By categorizing technical debt types, researchers aim to provide a nuanced framework for identifying, prioritizing, and addressing debt accumulation throughout the software development lifecycle.

Requirements debt (8.80%) indicate the trade-off between accepted requirements and requirements that are actually executed in a software product. For example, this can be manifested as partially implemented requirements or requirements that are implemented for some cases only [3]..

Architecture debt (10.70%) refers to issues and problems with the architecture of the system that affect the architectural requirements (e.g., performance, maintainability, robustness, etc.) [3]. For instance, breaking the modularity principle or failing to properly segregate concerns might cause serious issues with the system's continued development, upkeep, and expansion.

Design debt (21.99%) describes transgressions of best practices and principles in design. The source code can be examined to find this kind of debt. For example, using code review techniques or static code analysis tools [1]..

Code debt (14.66%) is the term used to describe issues with the source code that make it harder to comprehend and maintain. For instance, too complicated techniques or a lack of code standards. In order to remove code debt, code refactoring is needed [2].

Test debt (19.94%) refers to any problems relating to testing operations (Guo and Seaman, 2011). For instance, low test coverage or scheduled testing that was not carried out.

Process debt (4.11%) refers to inefficient processes. For example, a process changed over time and the existing practices are not efficient any longe [6].

Infrastructure debt (2.93%) refers to issues that if present in an organization significantly reduce the teams' abilities to deliver quality products [6]. For example, using outdated tools or postponing regular software or system updates.

Defect debt (2.79%) refers to known defects that are discovered by testing activities, however the corrective actions are postponed or never implemented. Delays in corrective action decisions can result in a large build-up of TD inside the system [4].

People debt (1.32%) refers to any issues related to people. For instance, deficiency in particular knowledge and abilities that would call for recruiting or further training (Alves et al., 2016). People related issues are much more complex, they involve socio-organizational factors that directly affect productivity and people satisfaction [2].

Other debt types had less than 1% occurrences each: service (0.88%), automated test (0.88%), versioning (0.73%), usability (0.59%), and build (0.59%). Precise definitions for these debt types can be found in (Mandic et al., 2020).

A survey conducted by DXC Technology [8], a Fortune 500 company, underscores the widespread recognition of technical debt among industry practitioners. Drawing insights from 750 CxOs and technology executives, the survey revealed that 99% of respondents acknowledged technical debt as an integral component of their risk management processes (DXC Technology TD Survey). This high level of recognition underscores the significance of technical debt as a pervasive challenge faced by organizations across diverse sectors.

In agile software development, technical debt management automation (TDMA) promises substantial benefits across various roles and stages of the development lifecycle.

- Software developers and engineers can anticipate streamlined workflows and more focused development efforts as TDMA automates the identification and prioritization of technical debt, freeing up time for core tasks.
- Technical leads and architects stand to benefit from improved system reliability and scalability through proactive management of code quality and architectural integrity.
- Project managers will find TDMA invaluable for enhancing project predictability and risk management, providing clear visibility into technical debt resolution progress and its impact on timelines.
- Product managers and owners gain crucial insights into how technical debt affects product features and business goals, enabling them to prioritize development efforts effectively.
- Quality assurance teams can leverage TDMA to bolster product quality by integrating automated testing and bug prevention strategies, ensuring higher reliability and smoother release cycles.
- DevOps teams will appreciate the integration of TDMA into CI/CD pipelines, which streamlines deployment processes and minimizes operational disruptions.
- Executive leadership and stakeholders benefit from TDMA's ability to provide transparent metrics and insights into software quality and development efficiency, fostering confidence in project outcomes and alignment with strategic objectives.
- Finally, clients and end-users experience improved product stability, performance, and user experience as TDMA facilitates proactive maintenance and continuous improvement efforts throughout the development lifecycle.

A. Linkage with Digital Transformation

Furthermore, the survey findings highlight the interplay between technical debt and organizations' pursuit of digital initiatives. Notably, 46% of executives surveyed identified technical debt as a hindrance to their ability to embark on digital transformation journeys (DXC Technology TD Survey). In an era where digitalization is paramount for competitive advantage and business resilience, the presence of technical debt poses a formidable obstacle, impeding organizations' efforts to innovate and adapt to evolving market dynamics.

B. Anticipated Cost Savings and Desired Outcomes

Despite the challenges posed by technical debt, organizations recognize the potential for substantial cost savings through its reduction. According to the DXC Technology survey, respondents anticipated cost savings of up to 39% through effective management and reduction of technical debt (DXC Technology TD Survey). Moreover, the survey identified the mitigation of technical debt as the top desired outcome among executives, highlighting its critical role in enhancing operating margins and fostering organizational efficiency.

C. Analogy with Depreciation

Many industry leaders compare technical debt to depreciation, recognizing it as an inevitable aspect of software development and maintenance. Senior executives perceive technical debt as a challenge that necessitates proactive measures to mitigate and eliminate, akin to managing financial depreciation within their organizations. This acknowledgement underscores the need for strategic approaches to technical debt management that align with broader business objectives & financial considerations.

D. Types of Automation Artifacts for TDM

Managing technical debt involves two fundamental aspects: preventing its accumulation and repaying existing debt through prioritization, incentivization of quality work, and systematic refactoring. The process of technical debt management encompasses identification, measurement, documentation, prioritization, monitoring, repayment, communication, and prevention, with each step contributing to a holistic approach towards debt mitigation.

There are 8 steps in managing the debt and these steps are common irrespective of the TD types or where it happens.

- Identification: Recognize signs of technical debt in the software development process whether it is architecture, design, code or documentation.
- Measurement: Measure the time and resources needed to reduce the technical debt.
- Representation and Documentation: Document and represent the identified technical debt for better understanding and communication.
- Prioritization: involves evaluating and grouping the debt into categories, such as critical, important, low priority, or negligible, to make it easier to prioritize.
- Monitoring: Keep track of the technical debt to increase awareness among team members.
- Repayment: Dedicate time for intentional and systematic refactoring to reduce the debt. Repayment is a business decision that should be made based on the impact and urgency of the technical debt.
- Communication: Educate stakeholders on the true cost of technical debt to gain their support in addressing it.
- Prevention: Make managing technical debt a part of every conversation with developers to ensure it remains a priority.
- Automation artifacts for technical debt management encompass a diverse range of tools and processes tailored to the specific needs and challenges faced by software development teams. These artifacts include:
- Tools: Independent, configurable software solutions providing interfaces such as GUI, command line, or APIs, examples of which include SonarQube and Arcan.
- Plugins: Extensions installed within Integrated Development Environments (IDEs) or other software artifacts to enhance existing capabilities, exemplified by plugins like jDeodorant and Checkstyle.
- Scripts: Context-specific code pieces executing tasks with less configurability, designed to automate specific aspects of technical debt management, such as debt free and piranha.
- Bots: Background runners triggered automatically to automate tasks, offering potential applications in TD documentation and remediation, illustrated by examples like FixMe.

Technical Debt Management (TDM) encompasses a range of activities aimed at identifying, prioritizing, and addressing technical debt throughout the Software Development Life Cycle (SDLC). The process of TDM involves recognizing the implications of expedient decisions made during development, which prioritize speed or release over optimal software quality. Without effective management, technical debt can accumulate, leading to increased costs, delayed time-to-market, and reduced customer satisfaction.

E. Challenges in Automation of TDM

Despite the growing recognition of the importance of automation in TDM, several challenges persist, hindering

comprehensive strategy implementation and interoperability among automation artifacts. These challenges include:

- Interdependencies and Strategy Implementation: The lack of details on the interdependencies between TDM activities poses a significant challenge in devising comprehensive automation strategies. The order of activities is crucial for effective implementation, highlighting the need for a structured approach to managing technical debt throughout the SDLC.

- Fragmentation and Siloed Automation Artifacts: Current automation artifacts often focus on addressing specific types of technical debt in isolation, leading to fragmentation and siloed approaches. This limits the effectiveness of automation in managing the diverse range of technical debt types encountered in software development projects.

- Rule Setting and Quantifiability: Establishing clear and quantifiable rules for prioritizing and executing TDM activities presents another challenge. The lack of standardized rules and metrics complicates decision-making processes, making it difficult to determine when and how to proceed with addressing technical debt effectively.

- Interoperability and Integration: Integrating TDM automation artifacts poses significant challenges due to interoperability issues. Tools often operate in isolation, utilizing different formats and protocols, which hinders seamless integration and data exchange. Overcoming this challenge requires the development of interoperable tools capable of reading from multiple formats and facilitating seamless integration through standardized protocols.

The literature review highlights the critical importance of addressing the challenges in the automation of Technical Debt Management to enhance efficiency, reduce risk, and improve software quality. By understanding the interdependencies between TDM activities, addressing fragmentation in automation artifacts, establishing clear rules and metrics, and promoting interoperability, organizations can unlock the full potential of automation in managing technical debt effectively.

III. OBJECTIVES OF THE STUDY

The research aims to address the lack of integrated, comprehensive software engineering knowledge available for Efficient Technical Debt Management Automation (TDMA). This objective stems from the identified challenges in the State of the Art, which highlighted fragmentation, interoperability issues, and the absence of clear strategies in existing approaches to TDM automation.

A. Significance

Solving the challenge of technical debt management (TDM) in software engineering not only enhances organizational capabilities but also offers significant cost-effectiveness and return on investment (ROI).

Current statistics reveal that developers spend approximately 23% of their time dealing with technical debt, which directly impacts productivity and project timeline. Moreover, maintenance efforts often constitute a substantial portion (50-75%) of overall project efforts, with a focus on functionality enhancements, adaptability improvements, and error corrections being crucial.

- Improved Capabilities: By developing integrated and comprehensive software engineering knowledge for TDMA, the research seeks to enhance the capabilities of organizations in identifying, prioritizing, and addressing technical debt effectively. This would lead to improved software quality, reduced maintenance burden, and enhanced customer satisfaction.

- Establishing Order and Integration: The research aims to establish a structured approach for TDM activities throughout the software development lifecycle (SDLC). By elucidating the interdependencies between TDM activities and promoting integration among automation artifacts, the research seeks to streamline processes and improve overall efficiency in technical debt management.

- Comprehensive Strategy and Decision Support: The development of a comprehensive strategy and context-aware automated decision support for TDMA would empower organizations to make informed decisions regarding technical debt. This would enable stakeholders to prioritize and allocate resources effectively, leading to more efficient debt reduction efforts and optimized project outcomes.

- ML/AI Enablement: Furthermore, the research intends to explore the potential of Machine Learning (ML) and Artificial Intelligence (AI) in further automating TDM processes. By leveraging ML/AI techniques, the research seeks to enhance the automation capabilities of TDM tools, enabling them to adapt to evolving project dynamics and facilitate proactive debt management.

IV. RESEARCH METHODOLOGY

To achieve the research objectives, the following methodological approaches will be employed:

- Literature Review and Synthesis: Building on the State of the Art, a comprehensive review of existing literature in software engineering, technical debt management, and automation will be conducted. This review will identify gaps, challenges, and opportunities in current approaches to TDMA, serving as the foundation for further research.

- Development of Integrated Framework: Based on the insights gained from the literature review, an integrated framework for Efficient Technical Debt Management Automation (TDMA) will be developed. This framework will delineate the interdependencies between TDM activities, propose strategies for integration among automation

artifacts, and outline clear guidelines for effective TDMA implementation.

- Prototype Development: A prototype TDMA system will be developed, incorporating the principles and strategies outlined in the integrated framework. The prototype will leverage existing automation artifacts and ML/AI techniques to demonstrate the feasibility and effectiveness of the proposed approach in addressing the identified challenges in TDM automation.

- Evaluation and Validation: The prototype TDMA system will be evaluated and validated through empirical studies, involving real-world software development projects and practitioners. The evaluation will assess the system's usability, effectiveness, and impact on technical debt reduction efforts, providing empirical evidence of its efficacy and practical utility.

V. FINDINGS AND SUGGESTIONS

Finding 1: Technical debt is not only about code and code quality. If it is not managed from the initial phase, it will continue to accumulate in the later phase and becomes difficult to repay.

For example, if technical debt is not managed in the requirements phase, it can lead to inefficient code, poor configuration control, and neglected documentation. This can further result in postponed bug fixes, failing to address security vulnerabilities, and other issues that can impact the performance, reliability, security, and maintainability of the software system

Findings 2: The literature and industry survey findings underscore the pervasive nature of technical debt within organizations and its profound implications for digital transformation, cost management, and operational efficiency. As organizations grapple with the complexities of modern software development, addressing technical debt emerges as a strategic imperative to foster innovation, mitigate risks, and drive sustainable growth. By understanding the challenges posed by technical debt and leveraging effective management strategies, organizations can navigate the digital landscape with confidence and resilience.

VI. SCOPE OF STUDY

Spotify's [7] development of Backstage represents a significant innovation in managing technical debt within complex microservices architectures. By integrating monitoring, documentation, and deployment tools into a unified digital thread, Spotify created a comprehensive view of their microservices landscape.

This approach not only reduced fragmentation and eliminated silos but also provided a single source of truth for developers, enhancing collaboration and efficiency across teams. Furthermore, leveraging generative AI allowed Spotify to analyze service dependencies, identify outdated components, and recommend proactive updates or refactoring strategies based on real-time data insights. This combination of integrated tools and AI-driven decision-making has enabled Spotify to continuously refine their technical operations and effectively manage ongoing technical debt challenges.

Guided by this case study, this research will address the challenges of managing technical debt (TD) in Agile software development by integrating TDM automation with digital threads and generative AI. It will explore how these technologies can tackle key TDM challenges effectively.

Firstly, the study will investigate how a digital thread can provide a comprehensive view of the software development lifecycle (SDLC), capturing and linking data from various stages and tools to track dependencies between TDM activities. Generative AI will be examined for its ability to analyze the digital thread, identify interdependencies, and suggest optimal sequences for TDM activities.

The research will also address the fragmentation and siloed nature of automation artifacts. It will explore how continuous data flow through digital threads can integrate information from different tools and stages of the SDLC, ensuring a holistic view of technical debt. AI will be used to synthesize data from fragmented sources, providing a unified understanding of TD and proposing unified management strategies.

Additionally, the study will focus on rule setting and quantifiability by examining how digital threads facilitate the collection and analysis of quantitative data across the SDLC, enabling the establishment of TDM metrics. AI will be used to define and refine rules for TDM by analyzing historical data and predicting future impacts.

The research will also investigate interoperability and integration, standardizing data formats and protocols to ensure seamless data exchange. AI will facilitate integration by mapping and translating data formats, enabling effective tool communication and automating data flows.

Furthermore, the research scope includes investigating the ethical implications of AI-driven TDM, ensuring that algorithms are designed and implemented ethically and responsibly. This involves examining issues such as fairness, transparency, and accountability in automated decision-making.

By leveraging digital threads and generative AI, this research aims to enhance visibility, proactive management, and decision-making in TDM, providing Agile teams with more effective tools for managing technical debt and improving software quality and development efficiency.

VII. CONCLUSION

In essence, the research objectives and approach outlined above seek to bridge the gap in software engineering knowledge pertaining to TDMA, offering novel insights, methodologies, and tools to enhance the efficiency and effectiveness of technical debt management in software

development projects. Through a systematic and interdisciplinary approach, the research aims to contribute significantly to the advancement of TDMA practices and ultimately improve the quality and reliability of software systems.

REFERENCES

[1] Li, Y., Soliman, M., & Avgeriou, P. (2022). Identifying self-admitted technical debt in issue tracking systems using machine learning. Empirical software engineering, 27(6), Article 131. https://doi.org/10.1007/s10664- 022-10128-3

[2] Nicolli Rios, Manoel Gomes de Mendonça Neto, Rodrigo Oliveira Spínola, A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners, Information and Software Technology, Volume 102, 2018, Pages 117-145, ISSN 0950-5849,https://doi.org/10.1016/j.infsof.2018.05.010.(https://www.sciencedirect.com/science/article/pii/S0950584918300946)

[3] P. Kruchten, R. L. Nord and I. Ozkaya, "Technical Debt: From Metaphor to Theory and Practice," in IEEE Software, vol. 29, no. 6, pp. 18-21, Nov.-Dec. 2012, doi: 10.1109/MS.2012.167.

[4] W. Snipes, B. Robinson, Y. Guo and C. Seaman, "Defining the decision factors for managing defects: A technical debt perspective," 2012 Third International Workshop on Managing Technical Debt (MTD), Zurich, Switzerland, 2012, pp. 54-60, doi: 10.1109/MTD.2012.6226001.

[5] Robert Ramač, Vladimir Mandić, Nebojša Taušan, Nicolli Rios, Sávio Freire, Boris Pérez, Camilo Castellanos, Darío Correal, Alexia Pacheco, Gustavo Lopez, Clemente Izurieta, Carolyn Seaman, Rodrigo Spinola, Prevalence, common causes and effects of technical debt: Results from a family of surveys with the IT industry, Journal of Systems and Software, Volume 184, 2022, 111114, ISSN 0164-1212, https://doi.org/10.1016/j.jss.2021.111114.

[6] Nicolli S.R. Alves, Thiago S. Mendes, Manoel G. de Mendonça, Rodrigo O. Spínola, Forrest Shull, Carolyn Seaman, Identification and management of technical debt: A systematic mapping study, Information and Software Technology, Volume 70, 2016, Pages 100-121, ISSN 0950-5849, https://doi.org/10.1016/j.infsof.2015.10.008.DXC Technology TD Survey https://dxc.com/content/dam/dxc/projects/dxc-com/au/campaigns/technical-debt/pdfs/dxc-au-tech-debt-tackling-tech-debt.pdf

[7] Spotify Engineering (2020). Backstage: How Spotify Built a Developer Portal. Spotify Engineering Blog. Retrieved from https://engineering.atspotify.com/2020/09/backstage-how-spotify-built-a-developer-portal/