Scholastic Seed Inc.

www.gjeis.com

global Journal of ENTERPRISE INFORMATION SYSTEM

Framework and Techniques for Managing Technical Debt in Software Development Lifecycle

– Sameer S Paradkar*

Enterprise Architect, Architecture Group, Business & Platforms Solutions, Global Delivery Centre - India

sameer.paradkar@atos.net *https://orcid.org/0000-0001-7517-1574*

ARTICLE HISTORY

Paper Nomenclature: View Point

Paper Code: GJEISV13I1JM2021VP2

Submission at Portal (www.gjeis.com): 01-Feb-2021

Manuscript Acknowledged: 06-Feb-2021

Originality Check: 06-Feb-2021

Originality Test (Plag) Ratio (Plagiarism Checker-X): 03%

Author Revert with Rectified Copy: 19-Feb-2021

Peer Reviewers Comment (Open): 20-Feb-2021

Single Blind Reviewers Explanation: 01-March 2021

Double Blind Reviewers Interpretation: 02-March-2021

Triple Blind Reviewers Annotations: 11- March-2021

Authour Update (w.r.t. correction, suggestion & observation): 18-March-2021

Camera-Ready-Copy: 20-March-2021

Editorial Board Excerpt & Citation: 24-March-2021

Published Online First: 31-March-2021

ABSTRACT

Purpose: Technical Debt and the subsequent problems are discussed often in industry and scientic journals. In this paper, we introduce a framework for managing technical Debt. The solution consists of methods to manage and repay technical debt. The goal of the framework is to provide a better overview of the Technical Debt items for the IT Development team and SMEs. Technical Debt calculation is the cost of fixing structural quality issues in an application that, if left unfixed, puts the business at critical risk. Technical Debt typically includes issues that are highly likely to cause severe business disruption; and may not include all problems, just the most critically serious ones.Technical Debt describe problems that ariseduring development when workarounds aremade due to tight project deadlines or technical improvements are neglected over a longer time. In atechnical metaphor to financial debt, the workaroundis interpreted as the debt and the resulting problemsas interest rates. Additionally, the paper also catagorizes Technical Debtand provides a view on the metrics and tools.

KEYWORDS Technical Debt | Code Analyzers | Frameworks | KPI | Metrices

**Corresponding Author (Sameer)*

- Present Volume & Issue (Cycle): Volume 13 | Issue 1 | Jan-Mar 2021
- International Standard Serial Number: Online ISSN: 0975-1432 | Print ISSN: 0975-153X
- DOI (Crossref, USA) https://doi.org/10.18311/gjeis/2021
- Bibliographic database: OCLC Number (WorldCat): 988732114
- Impact Factor: 2.69 (GIF, Citescore, SIF), CiteFactor: 1.0 (2020-21)
- Editor-in-Chief: Dr. Subodh Kesharwani
- Frequency: Quarterly
- Published Since: 2009
- Research database: EBSCO https://www.ebsco.com
- Review Pedagogy: Single Blind Review/ Double Blind Review/ Triple Blind Review/ Open Review

- Publisher: Scholastic Seed Inc. and KARAM Society
- Place: New Delhi, India.
- Repository (figshare): 704442/13

Introduction

Technical Debt represents the effort required to fix issues that are embedded in the software implementation when an application is released to production. Technical Debt across different technology stacks, are estimated based on the number of engineering flaws and violations of good architectural and coding practices in the software implementation. This data-driven approach to analyze and articulate the hotspots and violations of engineering practices provides an objective and actionable estimate of Technical Debt.

The estimation of the Technical Debt of an average-sized application of 300,000 lines of code (LOC) is around 3.6$'s per LOC.Minimizing defects is one of the effective ways to keep development costs down and manage technical Debt, which is a priority for many organizations. As the cost of fixing defects increases exponentially as software progresses through the development lifecycle, it's critical to catch defects as early as possible. The costs of discovering defects after release are significant: up to 30 times more than if you catch them in the architectural and design phases.

Modern day enterprises are critically dependent on business applications. These applications are a collection of data and business logic encapsulated in programming constructs and plethora of platform components, such as operating systems, databases, hardware and network infrastructure. These components are mutable and each one of them would be slowly but inevitably diverging from its ideal state to a suboptimal level, which potentially leads towards obsolescence or failure. Through judicious investment, IT teams and executives can fight off the ravages of time and reverse the aging process to reduce the technical debt.

Technical Debt–Problem Context

Over time, the number of platforms and applications delivering enterprise capabilities grows significantly, leading to duplication of solutions, overlap of capabilities across multiple platforms, and layers of customization. These all contribute to technical debt, which will negatively impact business agility, modernization, digital transformation, and the ability to be innovative. The majority of the IT budget goes onrun, instead of on developing new capabilities.

There are a number of digital forces from AI, IoT, Blockchain to Cloud and SaaS. Organizations across all industries are constantly responding to these digital forces, whether it is by transitioning to the cloud, adopting a new SaaS application, replacing a legacy system, or integrating the data behind all these different applications and systems. These emerging digital forces are creating a proliferation of projects for IT teams. And as more projects arise, IT is taking shortcuts to complete them on time and meet the needs of the business.

These shortcuts become IT's only choice because they are extremely constrained—both in terms of the number of resources available and time. Typically IT teamsare not adequately staffed to meet their business needs. As a result of these shortcuts, IT creates more technical debt than ever before, further draining their teams' resources.It is critical that we address technical debt, because legacy systems prevent teams from moving quickly, innovating, modernizing, and delivering new capabilities that are aligned with Intel's digital transformation.Catagories of technical debt:

Type of Technical Debt	Desription
Planned Debt	This is introduced when quick changes are done to reduce time to market
Accidental Debt	This is a result of systems evolving over time. When new capabilities are introduced, it takes more time to implement them because the design may not scale—thus requiring significant refactoring.
Unavoidable Debt	This is the result of complexity introduced over time with many incremental changes and deviations from the original design. This type of debt is difficult to fix and therefore, we must attempt to prevent this type of debt from occurring in the first place.

Table 1: Types of Technical Debt

The accumulation of technical debt impacts both the cost to deliver solutions and the ability to respond to customers' needs. High technical debt leads to lower productivity, reduced quality, and a need for constant design and code refactoring. Accruing technical debt results in higher operational costs, employee in-efficiency, and slower time to market. However, more importantly, makeshift solutions stackingat top legacy systems ultimately take more time and money to revise, leaving fewer resources for innovation and growth. One of the key pillars of digital transformation is technical debt reduction. Reducing technical debt and modernizing legacy systems by applying the technical debt framework will enable to invest in new capabilities and digital transformation initiatives for future success and reduce cybersecurity risk.

Framework for Addressing Technical Debt

Sporadically pursuing technical debt is not effective. Instead, we propose a framework to guide the technical debt efforts. This framework is holistic, in that it encompasses the full scope of technical debt to drive prioritization, aid in decision making, and fuel digital transformation. Our unique framework spans the entire business and application domain.

The framework leverages the following vectors to measure technical debt across the application stack:

Framework Vector	Description	Assessment Attributes
Business Value	Business benefits enabled by the technology	Business application compatibility Data and information quality/timeliness Facility for ease of use and change
Technical Value	Adherence of the technology to standards and practices	Architectural alignment Continuity and resilience Data protection and privacy Scalability & performance
Cost	Total cost of ownership (TCO) of the technology	Hardware Licensing Maintenance Support
Risk	Day-to-day burden imposed by the technology	Skills Competency Compliance Maintainability Reputational risk Supportability

Table 2: Technical Debt Management Framework

Using automated tools to measure and publish the technical debt metrics helps raise visibility and make technical debt reduction an enterprise-wide priority.As part of application governance, it is important to introduce defined criteria to measure and score technical debt. This helpsto quantify risk and technical debt creation. The computed score will help governance bodies to approve or reject a project before it gets too far along. Adopting this model brings techical debt to the surface, making it much more visible and forcing correct decisions. The paper proposes a method to compute the cost of a technical debt item and its impact to the enterprise to prioritize technical debt issues and focus on the ones that will most benefit. The cost includes the Principal which is the effort to address the technical debt item and the Interest which is the maintenance costand the risk that the debt might get out of control.

It is important to note that the interest can continue to increase based on time and other events that is, the cost of technical debt continues to rise if not addressed early. This model enables us to characterize every technical debt opportunitywith verify that it aligns to the target technology roadmap. It also helps us identify the items that carry the most debt to determinetechnical debt reduction initiatives that will bring significant immediate and long-term value. Using this approach will allows to establish the core foundation required to perform the assessment to reduce technical debt systematically at an enterprise level. Each application is assessed and tagged appropriately based on the framework.

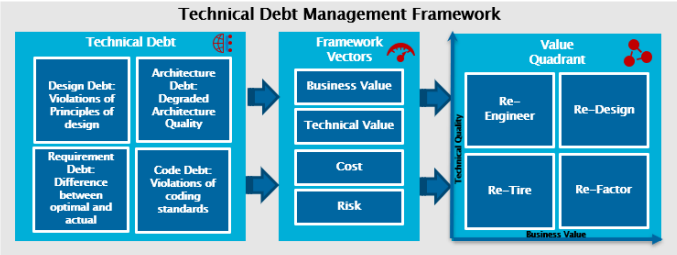


Figure 1: Technical Debt Management Framework

Identify technical debt at the code and design levels by leveraging open source platform for functional, structural and vulnerability code analysis. These tools continuously and automatically inspects code quality, finds bugs, security vulnerabilities, and source code characteristics that may indicate a deeper problem and issues.Next, establish a method for calculating the TCO for each opportunity. This method enables to assess costs consistently. Make sure to include all aspects of the system, including costs, licenses, hardware, support, and headcount. The assessment will result in identifying potential business benefits and a reduction in the number of platforms and services, which are better aligned to the enterprise strategy, resulting inreductions in the enterprise landscape.

Addressing Technical Debt

Poor software quality leads to huge technical debt and are common challenges in real-life software projects. Carrying out a software structural, functional and vulnerability assessment effectively and adopting the recommendations from it improves the design and implementation quality thus addressing the technical debt. A comprehensive assessment requires us to know the requirements in detail and weight different design aspects in accordance with the requirements. The amount of effort and time required to carry out a comprehensive design assessment can be quite high. The key is to pro actively leveraging software quality assessment tools for structural and functional analysis thereby addressing technical debt.

These structural and functional quality assessment tools analyses code, and identifies software quality issues. The tools analyse the health, complexity & cost of the applicationportfolio, technical debt, architecture, and system & code-level analytics. These tools provide insights into the complex software structures to make informed decisions, communicate about software health, measure efficiency and prevent software catastrophes.The tools help you **reduce technical debt and improve the maintainability** of business-critical applications.

Studies have shown that a large percentage of software budgets are spent on identifying and correcting software defects. This makes sense to invest in technologies that can help cut these costs. Automated products can drasticlly reduce the amount of time spent on software reviews, reducing development costs while improving time to market and efficiency. These products can also easily identify issues that can be missed during a manual inspection, increasing overall code quality and therefore customer satisfaction. A well-rounded approach to software quality assessment includes automated products throughout the software development lifecycle—during the stages when the code is being developed and after it is complete—to improve the quality. Leveraging both static and dynamic analysis tools, organizations can improve quality throughout the software development lifecycle. Static analysis products examine the software without executing the program. They apply a set of rules to the software code that help identify structural, functional and security quality issues early in the development lifecycle. Static analysis can help you identify and eradicate flaws before your applications are deployedduring the implementation phase, which usually results in a less costly remediation process. Dynamic analysis products monitor programs while they're running, enabling you to identify run-time issues that can't be detected by examining the developed code.

By combining the static and dynamic analysis products, you can improve your code quality, regardless of the individual skill level or whether the code was produced in-house or offsite. Together, these analysers can automateand help development teams identify quality and compliance issues throughout the software development cycle. As a result, one can reduce the defects and technical Debtin the applications, making them easier to maintain; decreasing development costs; and accelerating your time to market.

Static Analysis - Automated Measured of Technical Debt

The complexity of today's business applications has exceeded the capacity of individuals or teams to articulate the end-2-end picture. Software programmers may be experts in one or two technologies and languages, but none will have expertise and knowledge in all the languages and technologies leveraged to build modern day applications. This is where the automated analysers play a vital role as part of the engagement SDLC. There are three types of analysers that can be leveraged for application quality analysis and assessment which are explained in the following sections.

Functional Analysis

Functional code analyser's assesses quality,interms, degree of compliance with the coding practices of software engineering that promote security, extensibility, reliability, and maintainability. Functional analysers find weaknesses in program code that might lead to vulnerabilities. Functional analysers analysethe source code for specific defects as well as for compliance with various coding standards and coding guidelines. A fewtools alsoclub the feature to identifiessecurity vulnerabilities and hotspots during development and catch these critical issues. Fixing these flaws during implementationphase can reduce the number of builds necessary to produce an optimum and secured product and educate the development team about coding practices and guidelines.Functional analysers review the source code to detect common bad practices, catch bugs, and make sure the development adheres to standards and guidelines. Most code analysis tools define a series of rulesets (100+ rules) that identify different categories of issues in the code, for example: programming errors, coding standards violations, and security vulnerabilities.

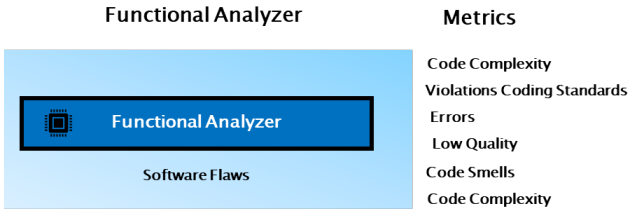


Figure 2: Functional Analysis

Structural Analysis

The challenges of modern software systems converge ultimately to their architecture. As systems become more complex and huger, their architectures assume ever greater importance in managing their growing coherence, reliability, and integrity. When architectural integrity is compromised, the probability for a serious operational bottleneck increases dramatically. Interactions among layers and subsystems will become increasingly more complex to articulate. Software Composition Analyserslook inside to identify architecture quality issues.The analyser's read, analyse and semantically understand all major kinds of source code, across all layers of an application (GUI, logic and data). By analysing all tiers of complex software, critical application health metrics like robustness, maintainability, transferability, flexibility, performance, or security can be measured and compliance to best practices can be assessed. Theanalyser'slookattheapplicationfromastatic viewpoint butareabletosimulatehowtheapplication will run, connecting all pieces of the puzzle, looking across different languages anddatabases. Hence, analysers are able to perform analysis of the entire application or system and its structural health.

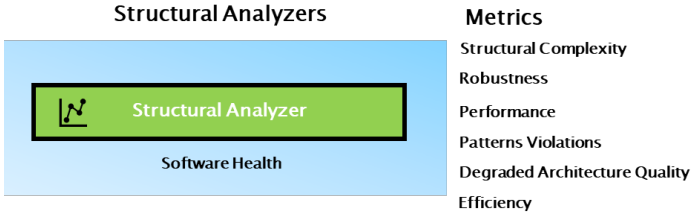


Figure 3: Structural Analysis

Security VulnerabiltyAnalysis

Static Application Security (SAST) Toolsare designed to analyze source code or compiled versions of code to help find security flaws. Some tools are integrated with the IDE. For the types of problems that can be detected during the software development phase itself, this is a powerful phase within the development life cycle to employ automated tools, as it provides immediate feedback on the issues they might be introducing into the code during software development. This immediate feedback is very useful, especially when compared to finding vulnerabilities much later in the development cycle. Analyzer toolsperform both dynamic (automated penetration test) and static (automated code review) analysis and finds security vulnerabilities that include malicious code as well as the absence of functionality that may lead to security breaches. Analyzers can determine whether sufficient encryption is employed and whether a piece of software contains any application backdoors through hard-coded user names or passwords. These tools employ a binary scanning approach that produces more accurate testing results, using methodologies developed and continually refined by world-classexpert. The tool may returns fewer false positives, developers can spend more time remediating problems and less time sifting through non-threats.Analyzersscanning the binary level, reviewing the compiled or "byte" code rather than source code, one gets the most accurate and comprehensive analysis. All applications, regardless of their origin, can be scanned and reviewed by such analyzers. Analyzers can even assess third-party software at the binary level, without requiring access to the source code. Security Analyzersare simply the most effective solution for source code security analysis.

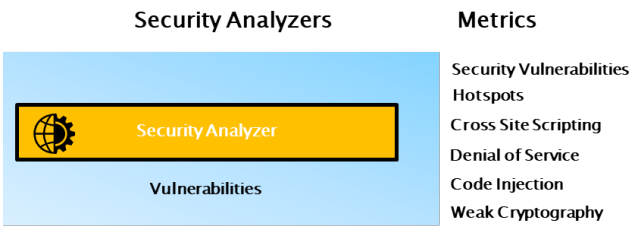


Figure 4: Security Vulnerability Analysis

Dynamic Analysis – Run-Time Analysis

Dynamic Analysis are a set ofprocesses and tools to ensure that application remains highly available and responds to user requests within an acceptable time limit. Monitoring tools help to achieve these goals by monitoring metrics such as response time, memory, network bandwidth, IOPS,and CPU time. The next generation tools that are based on cutting edge technologies like machine learning and AI provide ways to diagnose, triage, and resolve issues and bottlenecks in applications and infrastructure. Theaspects that are critical interms of application monitoring or APM aremonitoring metrics, tracing and logging.

Application monitoring provides detailed visibility into the performance, availability, response times, and user experience of application and itsunderlying infrastructure. The application monitoring helpsnot only to monitor but rapidlytriage, diagnose, and resolve issues leveraging cutting edge tools and technologies. Application monitoring tools collect, stores, and analyse the necessary data and metadata for troubleshooting, optimizing performance, root cause analysis, and final resolution. They typically rely on different types of instrumentation and profiling processes to provide real-time insights into the application health and its status. When performance exceeds automatically defined thresholds, application teams are notified and then can drill down contextually to trace transaction and performance issues across the distributed infrastructure for triage and resolution.

Key Characteristics of the Modern Monitoring Systems:

- **Anomaly Detection**. Anomaly detection capabilities in monitoring tools can automatically alert users when metrics deviate from the thresholds, assess their impact, all without human intervention.
- **Correlations**. Correlation engines go a level deeper and compare allparameters that contribute to metric outcomes. They analyse and measure subtle changes in one or more business metrics.
- **Root Cause Analysis**. Root cause analysis engines go even further and suggest possible causes of a deviation from the normal benchmark of a business metric or a group of metrics. These engines articulate the cause from historical correlations, or they provide IT-users with tools to assist them localize a cause by comparing multiple correlations.
- **Automation**: After detecting an anomaly, the product will determine its root cause, suggest a remediation, and predict the futureevents. They may even suggest ways to optimize the business process to eliminate future incidents.

Conclusion and Further Work

Integrate technical debt management into DevOps model to make technical debt visible. This will avoid irresponsible technical debt, and capture any deliberately or prudent debt as part of the product backlog. Dedicated a certain percentage of team effort (the exact percentage will vary depending on the enterprise) to work on technical debt items with small refactoring installments in each iteration. Adopting such a model keeps everyone informed and drives the right

prioritization of new functionality. Effect culture change by educating organization about the importance of managing technical debt. The required new mindset embraces technical debt management as a key component of good software development practice and a key enabler of continued digital transformation and success.

The resulting budget can now be applied to modernization, innovation, and digital transformation initiatives. There is a significant success in changing IT culture, making technicaldebt management part of our everyday thinking. This culture change is paramount in sustaining technical debt management over the long term.The enterprise benefits of technical debt reduction are:

- **Efficiency**:Less required TEAMS, lower support costs, and less suppliers to manage.
- **Stability**: Fewer changes to the core platform means fewer bugs to fix.
- **Agility**: Faster pace of change (faster validation).
- **Reusability**: Business units know what capabilities are available, and developers know how to introduce new functionality.

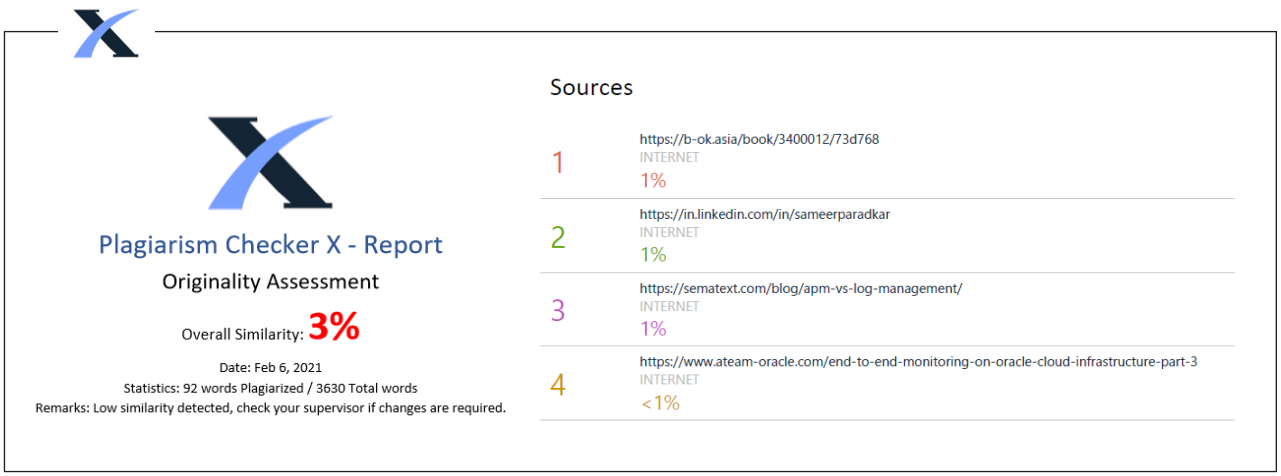
As next major step modernize legacy applications and systems that are critical for the businesses. It is important to provide support and secure legacy applications that are critical to business. Modernization can help identify legacy applications that are candidates for leveraging containers, microservices, cloud, and other initiatives aligned to target business goals. Modernization is a strategic investment that allows innovation while enabling technical debt reduction. Establish technology standards, CoE, and governance frameworks, integrate them into the software delivery of new capabilities across the enterprise. Update standards and guidelines to keep pace with technology trends and the business strategy.The success hinges on our ability to optimize at every layer, then focus on what makes a difference for the business.

References

- [1], Jean-Louis Letouzey, Michel Ilkiewicz, "Managing Technical Debt with the SQALE Method", IEEE Software, Vol.29, Issue.6, 2012
- [2], Frederico Oliveira, Alfredo Goldman, Viviane Santos, "Managing Technical Debt in Software ProjectsUsing Scrum: An Action Research", Agile Conference, 2015
- [3], Muhammad Firdaus Harun, Horst Lichter. "Towards a Technical Debt Management Framework based on Cost-Benefit Analysis", The Tenth International Conference on Software Engineering Advances, 2015
- [4], Marion Wiese, "Introducing a Framework for Managing Technical Debt Developed by Practitioners"
- [5], Ilya Khomyakov, Zufar Makhmutov, Ruzilya Mirgalimova and Alberto Sillitti, "Automated Measurement of Technical Debt: A Systematic Literature Review", International Conference on Enterprise Information Systems, 2019
- [6], Carlos Fernández-Sánchez, Juan Garbajosa, Carlos Vidal, Agustín Yagüe, "An Analysis of Techniques and Methods for Technical Debt Management: a Reflection from the Architecture Perspective", International Workshop on Software Architecture and Metrics, 2015
- [7], Carlos Fern´andez-S´anchez, Juan Garbajosa, Juan Garbajosa, "A Framework to Aid in Decision Making for Technical Debt Management", International Workshop on Managing Technical Debt (MTD), 2015

Annexure 1

Submission Date	Submission Id	Word Count	Character Count
06-Feb-2021	1512941507 (Checker X)	3409	24040



Reviewers Memorandum

External Critic (National): The paper is well structured and planned. Usage of tables and figures have made paper more understandable and interesting to read.

Outer Reviewer's (Global) Observation: Even though the article is crisp, then also it has tried to cover various significant aspects. It has introduced a framework for managing technical Debt. The solution consists of methods to manage and repay technical debt.

Citation

Sameer S Paradkar
"Framework and Techniques for Managing Technical Debt in Software Development Lifecycle"
Volume-13, Issue-1, Jan-Mar 2021. (www.gjeis.com)

https://doi.org/10.18311/gjeis/2021
Volume-13, Issue-1, Jan-Mar 2021
Online iSSN : 0975-1432, **Print iSSN** : 0975-153X
Frequency : Quarterly, Published Since : 2009

Google Citations: Since 2009
H-Index = 96
i10-Index: 964

Source: https://scholar.google.co.in/citations?user=S47TtNkAAAAJ&hl=en

Conflict of Interest: Author of a Paper had no conflict neither financially nor academically.

Editorial Excerpt

The article has 03% of plagiarism which is the accepted percentage as per the norms and standards of the journal for the publication. As per the editorial board's observations and blind reviewers' remarks the paper had some minor revisions which were communicated on a timely basis to the authors (Sameer) and accordingly all the corrections had been incorporated as and when directed and required to do so. The comments related to this manuscript are noticeably related to the theme "**Managing Technical Debt in Software Development Lifecycle**" both subject-wise and research-wise. The paper proposes a framework for managing technical Debt with an objective to provide a better overview of the Technical Debt items for the IT Development team and SMEs. The paper also catagorizes Technical Debt and provides a view on the metrics and tools. The solution consists of methods to manage and repay technical debt. Overall, the paper promises to provide a strong base for the further studies in the area. After comprehensive reviews and editorial board's remarks the manuscript has been categorised and decided to publish under "**View Point**" category.

Acknowledgement

The acknowledgment section is an essential part of all academic research papers. It provides appropriate recognition to all contributors for their hard work and effort taken while writing a paper. The data presented and analyzed in this paper by (Sameer) were collected first handily and wherever it has been taken the proper acknowledgment and endorsement depicts. The author is highly indebted to others who had facilitated in accomplishing the research. Last but not least endorse all reviewers and editors of GJEIS in publishing in a present issue.

Disclaimer

All views expressed in this paper are my/our own. Some of the content is taken from open source websites & some are copyright free for the purpose of disseminating knowledge. Those some We/I had mentioned above in the references section and acknowledged/cited as when and where required. The author/s has cited their joint own work mostly, Tables/Data from other referenced sources in this particular paper with the narrative & endorsement has been presented within quotes and reference at the bottom of the article accordingly & appropriately. Finally, some of the contents which are taken or overlapped from open source websites for the knowledge purpose. Those some of i/we had mentioned above in the references section. On the other hand opinions expressed in this paper are those of the author and do not reflect the views of the GJEIS. The author has made every effort to ensure that the information in this paper is correct, any remaining errors and deficiencies is solely the responsibility of the author.

Scholastic Seed Inc.
e-Publishing Aggregator & Periodical Mentor
www.scholasticseed.in