A Proposed Technical Debt Management Approach Applied on Software Projects in Egypt

Abeer Salah Eldeen Hamed[1*], Hazem M. Elbakry[2], Alaa Eldin Riad[3] and Ramadan Moawad[4]

[1*] Faculty of Computer Information Sciences, Mansoura University, Egypt.
abeer.salah.hamed@gmail.com , Orcid: https://orcid.org/0000-0001-7595-0922

[2] Faculty of Computer Information Sciences, Mansoura University, Egypt.
elbakry@mans.edu.eg, Orcid: https://orcid.org/0000-0002-4798-0427

[3] Faculty of Computer Information Sciences, Mansoura University, Egypt.
amriad2000@mans.edu.eg, Orcid: http://orcid.org/0000-0002-0091-3083

[4] Faculty of Computing &Information Technology, Future University in Egypt, Cairo, Egypt.
Ramdan.mowad@fue.edu.eg, Orcid: https://orcid.org/0000-0003-2349-8203

Received: June 19, 2023; Accepted: August 20, 2023; Published: August 30, 2023

Abstract

Technical Debt (TD) is a metaphor that can be described as the technical issues that are hidden from end users and customers, but in fact hinder the development efforts during system evolution and future enhancements. Due to tight budgets and timelines, TD is frequently incurred, which may lead to technical, financial, and quality issues that make future maintenance more costly or impossible. While business professionals concentrate on external issues related to customer satisfaction, in fact they rarely pay attention to internal software quality defects and maintenance, which would rather cause future interest payments. In this research study, we propose a TD management (TDM) approach with best practices developed using Design Science Research (DSR) and conducted with multiple case studies for the software development team in Small and Medium Enterprises (SMEs). The study demonstrates that the proposed approach for measuring the impact of internal / external software quality leads to increased awareness of TD's occurrence and thereby provides processes for preventing, identifying, prioritizing, monitoring, and repaying TD to satisfy both customer value and technical requirements, along with halting project failures and cost overruns. In actuality, applying our proposed TDM approach leads to a deeper comprehension of TD contraction in selected software companies, improved team morale and motivation, as well as an enhancement in its maintainability.

Keywords: Design Science Research (DSR), Small and Medium Enterprises (SMEs), Software Quality, Technical Debt (TD), Technical Debt Management (TDM).

1 Introduction

As a matter of fact, the software systems frequently require changes related to new or changing requirements. Development teams are frequently required to prioritize new changes over internal product quality; in this context, the Technical Debt metaphor has an impact on the financial world that

*Corresponding author: Faculty of Computer Information Sciences, Mansoura University, Egypt.

shall represent a key challenge in the software development industry(Pavlič et al., 2022; Wiese et al., 2022). The term "Technical Debt" was first used by Ward Cunningham in 1992 as an incremental measure of long-term problems, and project shortcuts during software evolution with aim of ensuring speedy releases that could deliver a benefit in the near term, but which on the other hand could adjust more expensive in the medium to long term, the thing that could evidently affect the software maintenance and evolution. The presence of TD is essential under some circumstances due to unstable business or environmental forces that are either internal or external to the organization. (Coelho et al., 2021; Lenarduzzi, Besker, et al., 2019) TD issues affect several software development process artifacts that are related to the limited project timeline, for example, incomplete tests, pending refactored code, as well as outdated documentation, which can make future changes more costly or impossible(Rios, Mendonça Neto, et al., 2018; Wiese et al., 2021).In addition to the above, TD has a negative impact on the morale of developers, which prevents them from accomplishing their duties and objectives.(Besker et al., 2020; Ghanbari et al., 2017) Undoubtedly, the existence of TD increases risks for a software project and makes it challenging to control. Thus, Technical Debt Management (TDM) is essential to maintaining control over the growth of TD and recognizing the causes that may compel development teams to accrue various types of debt, in addition to defining the consequences of their presence on software projects(Coelho et al., 2021; Rios, Spínola, et al., 2018).

In the future of software engineering, the TD metaphor will encompass internal software quality, maintenance, reengineering, and economics, whose TD interests build up over time. The longer it takes to pay, the more it costs. (Alfayez & Boehm, 2019; Avgeriou et al., 2015) From theory to applications, managing TD as a basic software engineering practice is an investment activity that applies economic theories and offers specific procedures and tools that apply data science and analytics to help decision-making, so that a TD item can be tackled at the most appropriate time. (Coelho et al., 2021; Ramasubbu & Kemerer, 2019) The TDM goal concentrates on identifying the best practices that researchers and practitioners might further assess to extend their understanding of the technical debt metaphor so they will be able to achieve a balance between the advantages and the drawbacks of TD presence(Avgeriou et al., 2015; Berenguer et al., 2021). It is critical for researchers and practitioners to understand the proposed strategies, initiatives, and tools to support the management of TD. (Rios, Mendonça Neto, et al., 2018)

Our research study focuses on applying the proposed approach to SMEs because TD has significant impacts on SMEs development and business processes due to several challenges such as resource limitations, time to market, lack of awareness, and tight budgets.(Lenarduzzi, Orava, et al., 2019) Throughout this paper, our goal shall be to study and investigate TDM with a best practice for each phase in the proposed TD management approach (Prevention, Identification, Monitoring, Prioritization, Repayment) provided with three case studies for three different SMEs software companies. Section 2 highlights related literature review and background. Section 3 focuses on a proposed TDM approach. Section 4 presents the case studies and the research results. Section 5 presents the conclusion and future work.

2 Literature Review

This section discusses TDM in several studies and documents along with the best practices from practitioner sources. TDM has been studied by researchers and practitioners; Li et al. have collected comprehensive TDM research, offering a classification of TD concepts and presenting the current state of TDM research based on 94 studies. The results reveal an uncertain definition of the TD term as well as the need for additional empirical studies with high-quality information on the TDM process, TDM techniques in industrial contexts, and tools for managing the several classifications of TD. (Li et al., 2015)

Alves et al. have identified artifacts that have been frequently utilized in software projects based on publications from 2010 to 2014. They have selected 100 studies to discover debt items. These studies have shown increased attention on methods for finding debt items in the source code, but other artifacts, including requirements specifications, documentation, and test reports, have only been mentioned occasionally. The authors have introduced a preliminary taxonomy of TD classification and provided a list of criteria to identify TD as well as management strategies. Moreover, they have proposed that the intense attention on source code may be related to the availability of numerous tools that do static code analysis, and hence facilitating the detection of debt items. (Alves et al., 2016; Berenguer et al., 2021)

In like manner, Coelho, R., et al., have investigated the current state of technical debt tools for activities, functionalities, and kinds of technical debt that are handled by existing tools which support the TDM in software projects. They have also defined the recent tools that concentrate on TD prevention, replacement, and prioritization activities which represent development research trends(Coelho et al., 2021).

On the other hand, Lenarduzzi et al. have conducted a systematic literature review (SLR) on technical debt prioritization to represent the state of the art regarding strategies, techniques, factors, measures, and tools used in practice or research in order to prioritize technical debt. They have likewise discovered that there is no agreement on the prioritization and assessment of the TD items critical elements(Freitas et al., 2022; Lenarduzzi et al., 2021; Lenarduzzi, Besker, et al., 2019).

Alfayez et al. have conducted a study on technical debt prioritization to identify current TD prioritization methodologies that consider cost, value, and resource restrictions, as well as a lack of industry review(Alfayez et al., 2020; Pina et al., 2021). Codabux, Z., and B. Williams have identified the best practices for characterizing, prioritizing, and managing technical debt so that practitioners can evaluate and improve their knowledge about the TD metaphor. (Codabux & Williams, 2013)

Most previous studies have focused on identifying best practices for specific processes without focusing on gathering all processes and defining best practices for each process. To lessen the literature gap, we present the TD management approach in a view to handling TD issues in software enterprises and facilitating decision-making regarding the necessity of paying off a technical debt item also determining the best time to do it.

3 A Proposed Managing Technical Debt Approach

TDM is a crucial part of the software engineering curriculum that helps the stakeholders to determine the importance of eliminating a technical debt item and then decide the most appropriate moment to do this. In case TD is managed properly, it can help the project accomplish its obejectives faster and at lower costs. Therefore, TD management concentrates on minimizing its negative impact, which is deemed a crucial aspect for the success of a software project and enhance developers' morale (Freire et al., 2020; Ghanbari et al., 2017; Ramasubbu & Kemerer, 2019) . This section elaborates the proposed TDM approach using Design science research model as displayed in Fig [1]; in addition, it illustrates the goal and practices of the implementation details for each phase.

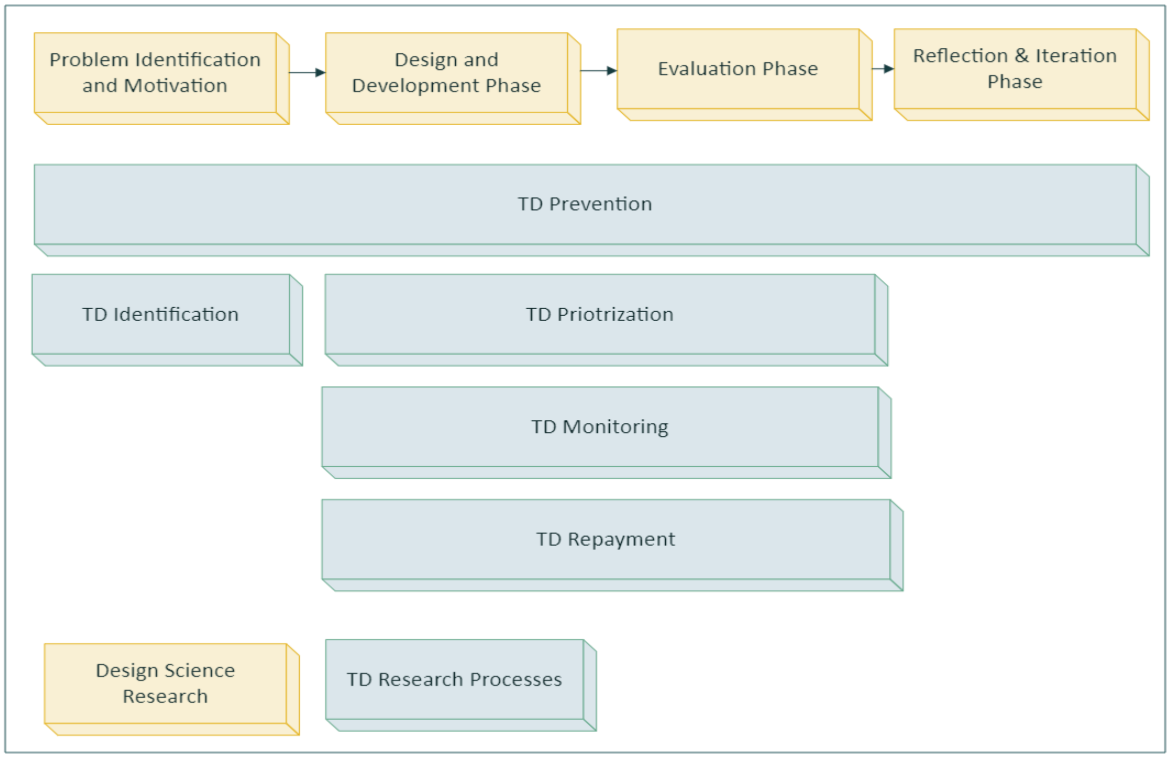


Figure 1: Technical Debt Management Approach

Phase 1: Technical Debt Prevention

Technical debt prevention is conducted effectively among Design Science research methodology phases. The goal of this phase is to prevent the occurrence of TD items(Alfayez & Boehm, 2019; Rios, Mendonça Neto, et al., 2018). Actually, it is a preferred phase rather than the TD repayment because the avoidance of TD costs might be less than paying it off due to the fact that re-engineering and refactoring shall be more expensive than developing the right software from the beginning of implementation(Freire et al., 2020; Wiese et al., 2021).

The TDM research study concentrates on understanding the causes and effects of TD items, as well as the causes that force the development team to incur TD items during implementation and the impact of these defects on the project (Berenguer et al., 2021; Rios et al., 2020). Understanding the causes of TD can allow development teams to take preventative actions that help reduce debt items during software development. Furthermore, it can be considered a lesson learned that can be applied to preventing TD items in the next sprint or an early phase of project implementation. The knowledge of TD effects can help prioritize TD items to pay off by supporting a more comprehensive impact analysis and the identification of corrective actions in order to mitigate any negative effects on the project. Fig [2] displays a list of the main causes and effects(Berenguer et al., 2021). Despite TD prevention is the most current best practice adopted by software companies when the implications are understood and modifying or rewriting code and other artifacts to eliminate the accumulated defects and their negative impact, still TD Prevention is not a common practice in TD management. (Coelho et al., 2021; Lenarduzzi, Besker, et al., 2019).

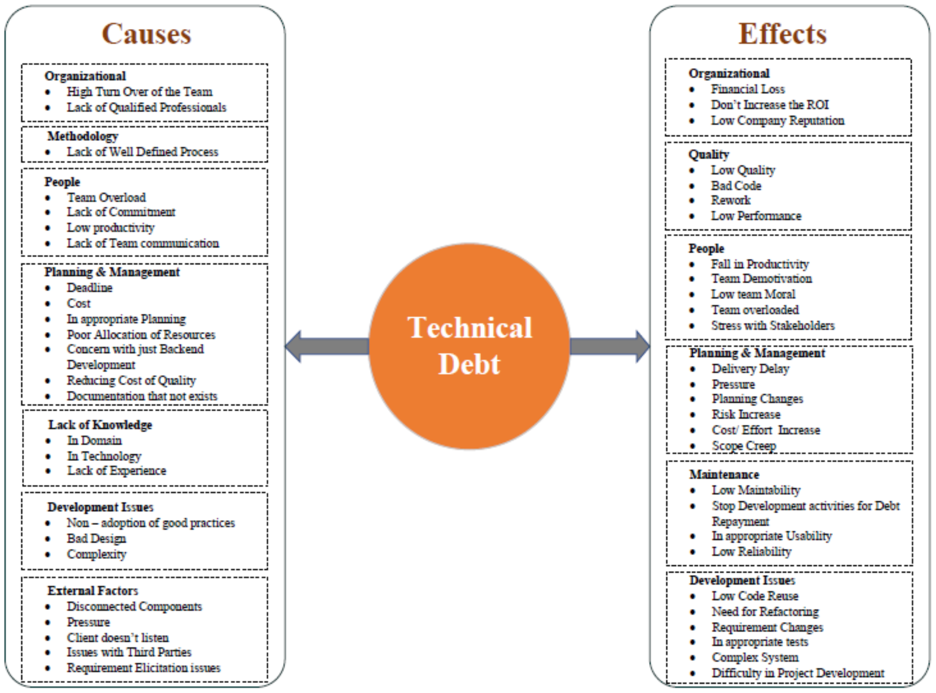


Figure 2: Technical Debt Causes and Effects

Phase 2: Identify and Classify Technical Debt

This phase comprises the procedures intended to identify and realize the different TD items, including other software development process artifacts, which allows to develop an effective and efficient management strategy(Rios, Mendonça Neto, et al., 2018). However, the expansion of studies in this area has raised attention only on facilitating the search for debt items in technical artifacts (source code) for identifying and evaluating debt items in the source code. This represents but a small percentage of the overall view of TD issues. In consequence, this study propounds to expand the identification technique with aim of assisting teams in collecting accurate information and thus estimating the economic effects of their technical debt decisions(Berenguer et al., 2021; Ramasubbu & Kemerer, 2019; Rios, Mendonça Neto, et al., 2018).

C. Seaman classifies TD items of the traditional software development life cycle into design debt, documentation debt, coding debt, testing debt, and defect debt produced during software implementation(Codabux & Williams, 2013). However, different types of data sources can provide relevant details that facilitate the discovery of TD. Systems for defect management and version control are examples of potential data sources that can be evaluated to identify the amount of debt in a project(Alves et al., 2016).

The implementation of an appropriate management approach for the TD items in a project is just as crucial as their identification. For this reason, techniques like modern portfolio theory and cost-benefit analysis have been proposed. Table [1] presents a definition for each TD type and points out some incidents that can be found in software projects. This relationship between different types of debt and their corresponding indicators is significant because it enables informed choices regarding the indicators that might be applied when deciding how to manage a specific type of debt. (Alves et al., 2016; Farias et al., 2020; Lenarduzzi, Besker, et al., 2019).

Table 1: Technical Debt Types Definition

TD Types	**Definition**
Requirements Debt	Refers to the difference, under domain assumptions and limitations, between the ideal requirements specification and the actual implementation of the system. For example, partial fulfillment of requirements, not covering all cases, and not fully satisfying non-functional requirements.
Architectural Debt	Refers to issues with product architecture, such as the violation of modularity, complex architectural behavioral dependencies, architectural compliance issues, system-level structure quality issues, non-uniform usage of architectural policies and patterns, a lack of ability to handle interdependent resources, a lack of ability to address non-functional requirements, and the implementation of immature architecture techniques. This kind of debt cannot be settled with straightforward code changes, it rather needs large-scale development efforts.
Design Debt	Refers to debt that may be found by inspecting the source code and finding instances where the rules of good object-oriented design have been broken (e.g., very large or tightly coupled classes, excessive design complexity, certain types of code smells).
Code Debt	Refers to issues within the source code that can make maintenance more difficult since they can make it less legible. Typically, it is improperly written code that breaks coding conventions or best practices. For example, code duplication, bad style that decreases the readability of code and excessively complex code.
Test Debt	Refers to issues discovered during the testing phase that may have an impact on quality. Examples of a lack of testing (unit tests, integration tests, and acceptance tests), insufficient test coverage, and a lack of test case planning.
Build Debt	Refers to defects in a software system, it's built system, or it's built procedure that result in a build that is extremely complex and difficult. Throughout the build procedure, code that doesn't add value for the client may be utilized. Additionally, the build process will become needlessly slow if it must run ill-defined dependencies and the manual build process as well.
Documentation Debt	Refers to missing, insufficient, incomplete, or outdated documentation in any aspect of software development. For example, there is outdated architecture documentation and a shortage of code comments.
Infrastructure Debt	Refers to an inadequate setup of technology, development-related procedures, and supporting tools that, if they exist in the software organizations, may cause some development operation to be delayed, hindered, etc. Such an inefficient setup limits the team's ability to produce high-quality products.
Versioning Debt	Refers to issues with source code versioning, such as unnecessary code forks.
Usability Debt	Refers to decisions made on usability that should be modified later. Examples of this debt include the software's inconsistent navigational features and the absence of usability standards.
Service Debt	Refers to the poor selection and modification of web services that result in a discrepancy between the features of the services and the requirements of the applications. This type of debt is significant for systems with service-oriented architectures.
Defect Debt	Refers to known flaws, errors, or bugs that are typically found during testing software systems or reported by end users on bug tracking systems. Decisions to postpone or delay flaw corrections can result in considerable TD for a product, which makes it more difficult to solve it later.
People Debt	Refers to human problems that, if they exist in the software organization, may cause some development operations to be delayed or hindered. This sort of debt includes expertise concentrated in too few people as a result of delayed recruiting and / or training.
Process Debt	Refers to ineffective procedures, such as when the process can no longer be used for its intended purpose.

Phase 3: Technical Debt Monitoring / Cost Estimation

The primary goal of the project management tool is to track and monitor technical debt artifacts that have an impact on the project backlog (Coelho et al., 2021). Estimating the costs and benefits of refactoring activities is essential for managing TD as it helps with work-process planning and prevents potential cost and schedule overruns(Besker et al., 2019). TD Monitoring phase is conducted by a guide of design and development phase and evaluation phase of Design Science research (Yudistira, B.G.K., 2022). This phase keeps track of TD item activities and observes the cost and benefit value of them throughout the evolution of the project over time (Alfayez & Boehm, 2019; Li et al., 2015; Rios, Mendonça Neto, et al., 2018).

Ward Cunningham utilizes the concept of principal and interest probability to clarify the financial metaphor of technical debt. TD can cause significant cost overruns if it remains untreated in the software; besides, the internal software quality defects can result in high maintenance costs. Therefore, prioritizing TD items in the backlog is the initial phase of TDM to determine which factors influence these decisions(Besker et al., 2019; Codabux & Williams, 2013; Codabux et al., 2017). AnaconDebt and JCaliper are two examples of cost-benefit analysis tools that introduce estimation approaches aims to support the effort and resource Prioritization for refactoring strategies(Coelho et al., 2021; Stettina et al., 2023).

Phase 4: Technical Debt Prioritization

Technical Debt issues need to be managed and refactored by development teams. Therefore, it is essential to understand when prioritizing the payment of technical debt interest over developing features or addressing faults to reduce the negative consequences of the most critical debts first (Codabux & Williams, 2013; Lenarduzzi, Besker, et al., 2019). The prioritization phase is conducted by a guide of design and development phase and evaluation phase of Design Science research. Fig [3] presents a conceptual TD prioritization model display five different aspects identify how practitioners carry out the TD prioritization process and what external influences may affect the process (Besker et al., 2019).

Making a prioritized list of TD items is an important phase in the TDM decision-making process to enable technical and business stakeholders' expectations to be aligned. Moreover, it is crucial to understand which factors affect these decisions (Besker et al., 2019; De Almeida et al., 2021). This phase involves ordering the identified TD items in the backlog based on precise and predefined rules to select a set f TD items that should be repaired first in order to maximize the benefits of repayment activity while reducing its cost and deciding which items can be tolerated until a later release. (Alfayez & Boehm, 2019; Pina et al., 2021) Snipes et al. identify a list of factors that influence the decision maker in determining whether to fix or postpone the defect, as follows:

- Defect Severity.
- Classification (Long/Short) Term.
- Customer Impact.
- Existence of a workaround.
- The effort to implement the fix.
- Dedicate a team to the fix's implementation.
- Fix Impact.
- The potential fix's risk.
- The scope of required testing.
- Bug fixing time.

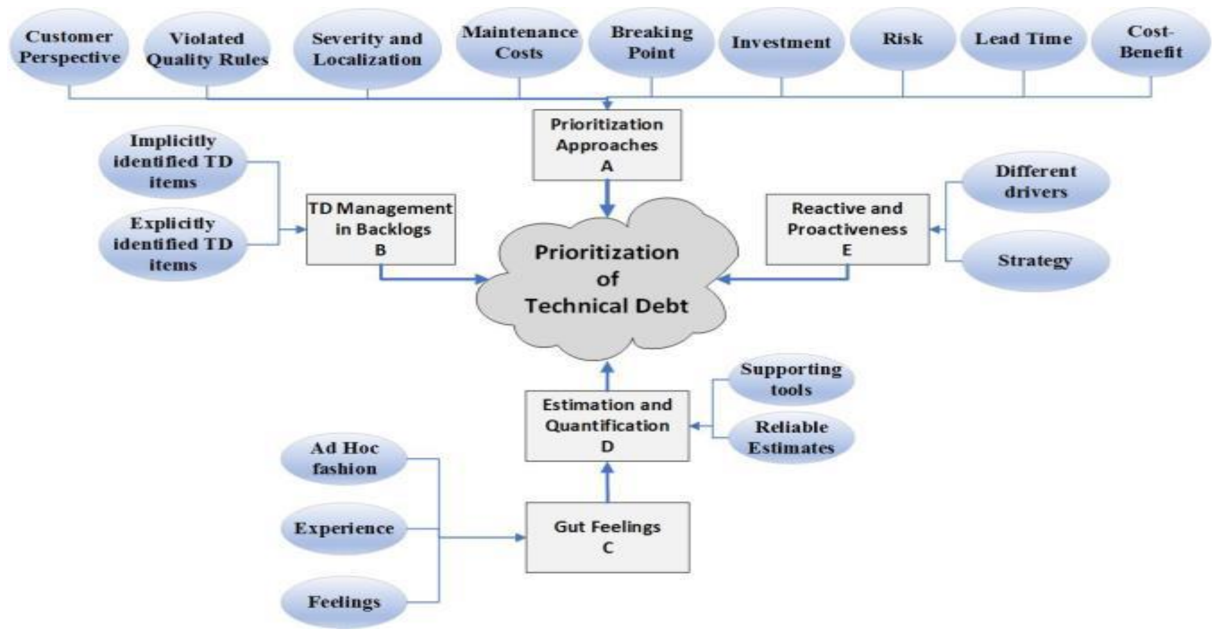

Customer Perspective
Violated Quality Rules
Severity and Localization
Maintenance Costs
Breaking Point
Investment
Risk
Lead Time
Cost-Benefit
Prioritization Approaches A
Implicitly identified TD items
Explicitly identified TD items
TD Management in Backlogs B
Prioritization of Technical Debt
Reactive and Proactiveness E
Different drivers
Strategy
Supporting tools
Reliable Estimates
Estimation and Quantification D
Ad Hoc fashion
Experience
Feelings
Gut Feelings C

Figure 3: Conceptual Model for TD Prioritization

The factors are enumerated in descending order of importance, with the most prioritized impact approach based solely on the severity and customer impact (Alfayez & Boehm, 2019; Codabux & Williams, 2013). The prioritization methods categorized in three solutions: Boolean, indicating whether technical debt items should be paid or not; Categories, showing the payment severity (high, medium, and low risk); Ordered List, sorting which of the items on the list shall be paid off first. (Pina et al., 2021)The prioritization process should be conducted carefully to decrease the amount of TD. Each organization or project must determine the prioritization criteria by designating a priority for each TD item according to business needs and technical perspective. (De Almeida et al., 2021)

We can notice the wide support for the TD prioritization process to make a successful investment in the evolution of software, but it still meets challenges. Furthermore, most TD Prioritization tools allow the users to assign priority to a specific TD item without providing clarity on which TD items should be prioritized. (Coelho et al., 2021; Lenarduzzi, Besker, et al., 2019; Pina et al., 2021)

Phase 5: Repayment of Technical Debt

It is a matter of importance to pay TD gradually and strategically. Therefore, TD issues need to be managed and refactored by software companies to compromise between allocating developer time, effort, and resources to creating new features and allocating them to refactoring TD items. It may reach a crisis point where it’s necessary to conduct a significant refactoring or complete software replacement (Besker et al., 2019; Freire et al., 2020; M. Bomfim & A. Santos, 2017). TDM shall decide the most proper time to apply for tackling and refactoring TD items. This phase concentrates on repayment and mitigating or resolving TD items in a software system (Alfayez & Boehm, 2019; Berenguer et al., 2021; Rosser & Norton, 2021). Fig [4] shows a sample of the decision-making process that is used to carry out the refactoring process (M. Bomfim & A. Santos, 2017).

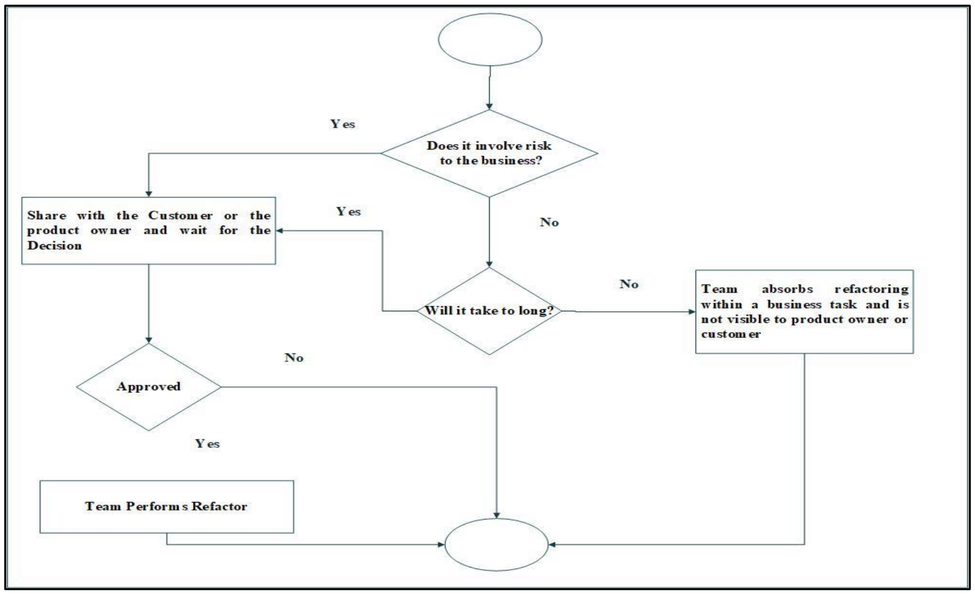

Does it involve risk to the business?
Yes
No
Share with the Customer or the product owner and wait for the Decision
Yes
Will it take to long?
No
Team absorbs refactoring within a business task and is not visible to product owner or customer
Approved
No
Yes
Team Performs Refactor

Figure 4: Refactoring Standard Decision Flow

Refactoring involves "tidying up" the code to make it more maintainable without changing its functionality; it commonly competes with adding new functionality and correcting bugs. The repayment activity refers to actions performed to assist in decision-making around the suitable timing to pay off debt items (Codabux & Williams, 2013; Rios, Mendonça Neto, et al., 2018). However, it often involves resolving technical debt using reengineering or refactoring techniques. Using refactoring techniques to repay technical debt in the software industry is considered challenging. For example, it can be difficult to guarantee that during refactoring, software behavior will stay stable (Coelho et al., 2021; Lenarduzzi, Besker, et al., 2019), in addition to the cost constraint that must be fulfilled which typically applies to the repayment actions (Alfayez & Boehm, 2019; Freire et al., 2023).

The number of available supporting software tools for addressing repayment activity is low. As an example, JCaliper evaluates TD and suggests a set of refactoring to deal with it using search-based software engineering techniques. It uses algorithms for local search to find a near optimal situation and to suggest TD repayment actions, automatically determining the quantity, kind, and order of reaching activities required to produce the design without TD (Besker et al., 2019; Coelho et al., 2021; Freitas et al., 2022).

4 Case Study

This investigation aims to comprehend how software companies manage TD. The main research question is, "*Do the TD management elements enhance the quality of software implementation issues?"* To answer this question, we apply the proposed TDM approach to the real world. In doing so, we have chosen three different SMEs to investigate TDM in software projects and evaluate the results of applying the processes prescribed by the proposed approach in their commercial software development, especially as SMEs by nature require to manage their budget wisely due to the limited timeline and budget TD generated with the intention to repay when the company receives sufficient funding.

In the same manner that has been used in earlier software studies utilizing action research and contextual approaches, we have likewise worked closely with the selected three SMEs software companies with various business development teams to gather a set of process approaches practiced in real-world software development activities for the three projects. Fig [5] display Technical debt approach landscape provided with best practices to be applied in real world case studies. (Harun & Lichter, 2015)

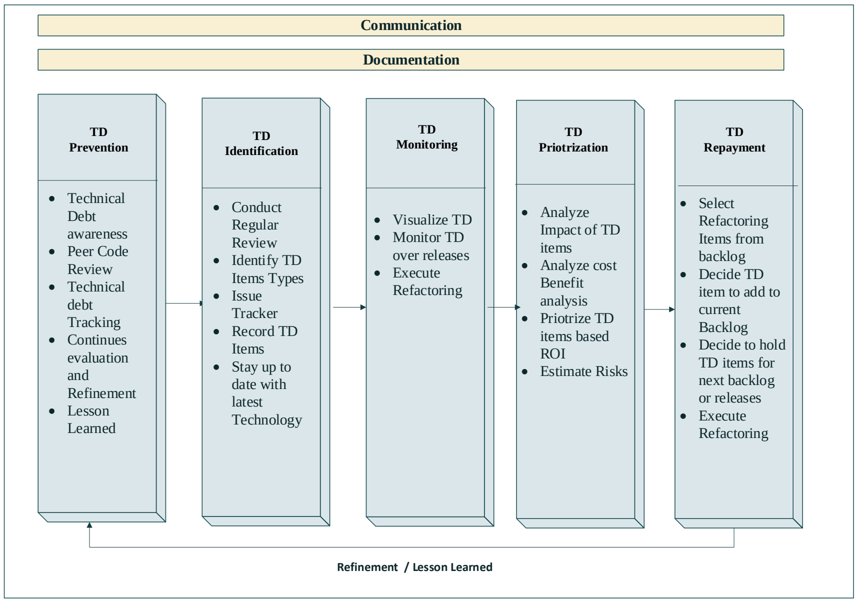


Figure 5: Technical Debt Management Landscape

The names of the selected projects and the software companies are confidential in their requests; therefore, we shall refer to the individual projects in this research by case numbers. An outline of the involved companies is in Table [2]. (Pavlič et al., 2022; Ramasubbu & Kemerer, 2019) Projects were chosen under specific criteria at the selected software companies: 1. Ongoing projects with plenty of requests for changes; 2. Projects with signs that TD has not been handled (Oliveira et al., 2015).

We introduce three projects one of them has been fully published and in public production for several years, and the other two projects are not fully finalized at the moment of performing the research. The proposed TDM approach is implemented in both cases No. 2 and No. 3. For case No. 1, we have found it better to select a project in the company that has not applied processes to manage TD so we can compare the companies that have applied to those who have not applied the approach proposed in our research study. We have urged the development teams to agree to devote their valuable time to the self-admitting process in order to produce production-ready software. In addition to developers' insights, we use in our research source code analysis tools for the technical debt measurement, which focus on quality control of source code and related services, including quantifying code metrics, design and architecture artifacts, code smell, and validating code standards that can be used to display TD signs (Coelho et al., 2021; Pavlič et al., 2022).

Table 2: Field Research Sites

	Case 1	**Case 2**	**Case 3**
Business Domain	• Portals • Systems • Mobile Apps	• Digital • Technology • Business Solutions	Telecommunication
Company Size	50 - 100	>100	>1000
Company Age	Last 5 years	Last 8 years	Last 14 years
Process Maturity	Not Assessed	ISO 9001	ISO 9001, ISO 50001

In our research study, an open-source tool for quality management is chosen as a source code analysis tool called SonarQube; one of the best-rated tools for TD measurement activities. The tool uses the SQALE method to calculate technical debt. The tool measures and identifies TD under a predetermined set of rules to run continuous automated tests that are eligible to identify bugs, highlight performance bottlenecks, detect security vulnerabilities, and detect improper programming practices, such as code smells, in over 20 programming languages. Furthermore, it is not associated with a specific TD type (Alfayez & Boehm, 2019; Coelho et al., 2021; Pavlič et al., 2022).

SonarQube version 9.8.0 is used in our case studies to analyze the source code of our project in order to extract each TD item and its information, which includes the rule that this TD item has violated, a message suggesting how to repay this TD item, the file in which this TD item is identified, the start and the end line, the estimated time in minutes to fix this TD item, along with the indicated issue's criticality by its severity which includes (blocker, critical, major, minor). (Alfayez & Boehm, 2019; Ramasubbu & Kemerer, 2019) Technical debt remediation is made easier with the support of a detailed view of each debt, which also shows important files and makes suggestions for improvement. In all three cases, we have begun with gathering project details like project size (LOC), business domain, technology, team members, and product development platforms used by the software companies.

In Table [3], we have gathered the required details for each project, such as the solution domain, programming languages, project duration, etc. presented the three-field test research. In later stage, we have interviewed the engineers who are assigned to the system implementation and listened to their feedback regarding the reasons behind technical debt issues in the system.

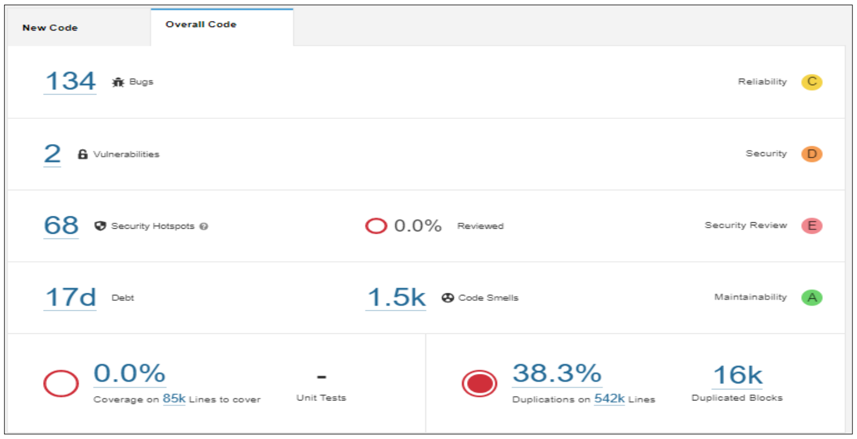
In our research study, we conducted a face-to-face interview with software practitioners from three selected projects to investigate the relationship between TD impacts and developers' morale. Interviews were conducted based on two factors: affective antecedents and future/goal antecedents. Affective antecedents refer to mood, sensation, emotion, and attitude-related factors. Future / goal antecedents consist of variables related to how individuals perceive the difference between the future and the present, as well as their future goals. (Ghanbari et al., 2017) We gathered information from software practitioners (Scrum Master, Developer, Tester, and Architecture) for each project to consider the consequences of TD on the morale and other psychological aspects of developers.

Table 3: Projects Details

Case No.	Release Date	Technologies	Project Domain	Team No.	Size
# 1	Jan 2020	• Net V. 5.0 • SQL Server	Attendance Portal	4	16 KLOC
# 2	Sep 2022	• Java • Springboot • RabbitMQ • postgresDB	Chatbot	7	43.5 KLOC
# 3	Under Development	• java • angular • springboot • RabbitMQ • postgresDB	Digital Channel	4	10.5 KLOC

Case 1

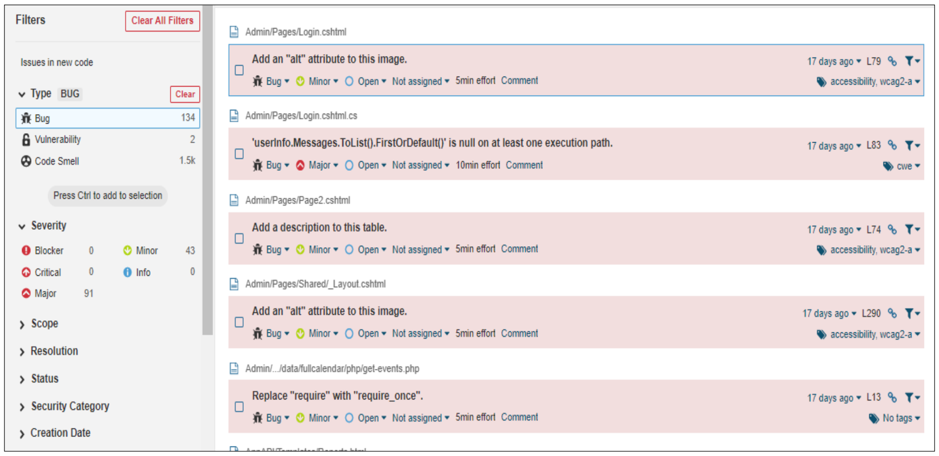
This is a startup software company. The selected project is an attendance web portal (AP) so employees can submit and monitor their vacation balance and get vacation approval cycles. In addition, the portal is integrated with the fingerprint machine so the employees can monitor their attendance. The project is implemented on a very tight timeline for public release, focusing on adding new features without taking into consideration fixing technical defects during project implementation.



134 Bugs — Reliability C
2 Vulnerabilities — Security D
68 Security Hotspots — 0.0% Reviewed — Security Review E
17d Debt — 1.5k Code Smells — Maintainability A
0.0% Coverage on 85k Lines to cover — - Unit Tests
38.3% Duplications on 542k Lines — 16k Duplicated Blocks

Figure 6: AP Code Analysis Result

The exercise is started after 3 years of keeping changes on the project, ignoring technical debt items, and relying on quick and dirty fixes during the project journey. We have interviewed the developers who have created and updated the source code for the selected project to get full information about its status. In our exercise, we have used the SonarQube tool on selected project source code with the aim of measuring the size of the project, identifying maintainability issues, together with determining the amount of bug-fixing and rework that will be needed for each sort of software defect that has been registered on the project. Fig [6] shows the outcome of the code analysis result and the fixing duration of the current release.



Admin/Pages/Login.cshtml
Add an "alt" attribute to this image. — Bug · Minor · Open · Not assigned · 5min effort · Comment — 17 days ago · L79 — accessibility, wcag2-a

Admin/Pages/Login.cshtml.cs
'userInfo.Messages.ToList().FirstOrDefault()' is null on at least one execution path. — Bug · Major · Open · Not assigned · 10min effort · Comment — 17 days ago · L83 — cwe

Admin/Pages/Page2.cshtml
Add a description to this table. — Bug · Minor · Open · Not assigned · 5min effort · Comment — 17 days ago · L74 — accessibility, wcag2-a

Admin/Pages/Shared/_Layout.cshtml
Add an "alt" attribute to this image. — Bug · Minor · Open · Not assigned · 5min effort · Comment — 17 days ago · L290 — accessibility, wcag2-a

Admin/.../data/fullcalendar/php/get-events.php
Replace "require" with "require_once". — Bug · Minor · Open · Not assigned · 5min effort · Comment — 17 days ago · L13 — No tags

Figure 7: AP List of SonarQube Issues

Fig [7] represents a detailed part for a code defect with a detailed description for each defect. We can observe that most of the TD items that have been reported are associated with "code smell" or "poor quality code" (B Perez, Series 2020).

Moreover, the SonarQube tool provides a severity rating and a remediation effort. As mentioned in Fig [8], the internal defects are distributed according to their severity level. One overarching issue raised by the interviewees has been their lack of knowledge of how the software would develop moving forward, which has made it challenging for them to rank the required and urgent TD items for refactoring, the benefits of restructuring TD, and the result of reducing its negative effects. However, inadequate information on the TD items to support such activities and a lack of delegated duty for delivering such information, combined with higher management's lack of a strategy and pressure on the team members to meet deadlines and deliver customer value, push them to continuously lower the priority of TD refactoring in favor of introducing new features quickly.

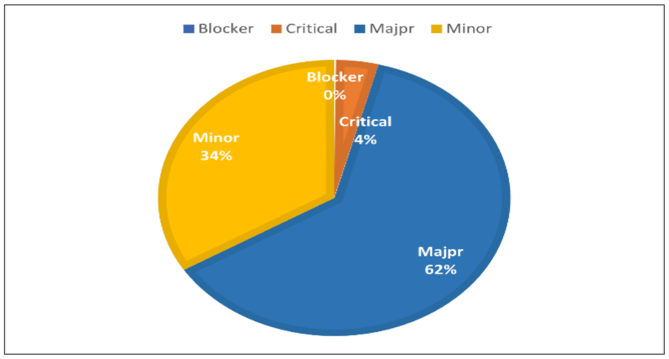


Figure 8: AP - Internal Defect Severity

We have noted from the discussion that their company lacks a clear plan of action for dealing with refactoring efforts. Their approach is built on a clustering strategy, where the necessary refactoring tasks are connected to or necessary for certain prioritized new features or customer issues. Explanations for these outcomes include inadequate information on TD items to support such activities, a lack of delegated duty for delivering such information, combined with higher management's lack of a strategy, and an explicit focus on the software's TD repair.

Nevertheless, this company approach not only reflects code debt; it also affects other artifacts during project implementation, such as incomplete tests or outdated documentation, which are examples of TD items for the selected project. Moreover, during interviews with the project dedicated team for TD, it was noted that there was a significant communication gap in the TD conversations between technical and business players. Interviewees mentioned that TD had a negative influence over their development and upcoming activities; also, working on legacy code that conatins a significant amount of TD is not simple to comprehend. They were so frustrated, especially since one of the developers claimed that the selected company does not appropriately reward TD management.

With the assistance of the developers, we have pointed out the causes of these results. Due to the tight timeline and fast delivery, developers incur technical debt every time they skip a step during implementation, which has long-term effects on TD and project output. Having TD in the software has reduced project quality, decreased productivity, lowered developer morale, increased maintenance costs, and caused project delays, to name but a few examples. Due to the project challenges, the company has decided to revamp the project from scratch and take these causes as preventive action for the new project. The lesson learned from this case is that due to time to market, delivering customer value, and limited resources, the company shall frequently make trade-offs between the software development process's expenses in terms of crucial time and resources and the overall quality of the project and balance between the existence of TD and the effect of its presence.

Case 2

Selected software enterprises have a wide range of businesses. The company is in the process of implementing the ISO 9001 process for improving their quality and meeting customer expectations, and we have suggested applying the proposed TDM approach we have created with the newly implemented processes. The selected project is a chatbot, a piece of software used to carry on text-based or text-to-speech online chat conversations. During our study, the company has been conducting a process revamp at the time we have started cooperating with them in a chatbot project. As we move through the selected project, we have invited the whole assigned team onto the project (the product owner, scrum master, and team members). The chatbot team is classified as (1 scrum master, 4 developers, and 2 testing engineers). The team working on product development uses an agile methodology which leads to extremely fast TD occurrences due to a focus on quick delivery to satisfy the requirements of many stakeholders.

However, this emphasis on delivering software rapidly frequently drives the development team to accumulate technical debt, which refers to design or implementation structures that are practical in the near term but create a technical context that can make future updating improvements more expensive or impossible. During interviews, we have asked participants to share their thoughts and raise concerns about the software's quality and their need to pay TD in a proactive and preventative manner. The project's product owner requests that we provide our approach for the development team to manage technical debt. We have prepared brief sessions with the team to introduce the technical debt metaphor, collaborate closely, and elaborate on each step for our approach to be applied in a real software production project. During our exercise, we have not concentrated on categories of technical debt unrelated to source code (e.g., requirements debt, documentation debt, architecture debt, etc.).

We have relied on the research results that show a link between the identified technical debt issues and source code ownership. We have interviewed the developers who have written and updated the source code for the selected project to get full information about its status. We have focused on our exercise on code debt. We have asked the team to prepare a backlog to report TD items on it. We have respectfully requested the developers to evaluate their code thoroughly and report technical debt items. We have proposed the SonarQube code analysis and TD recognition tool on Chatbot source code to identify maintainability issues and know the amount of bug-fixing and rework that would be needed for each defect type that have been registered on the project. The chatbot consists of 14 modules, and they run the tool on each module. Fig [9] shows the summary of the outcome of running the SonarQube code analysis results for all modules. The developers describe each technical debt item's title, description, type on the product backlog and the effort estimated for each TD item.

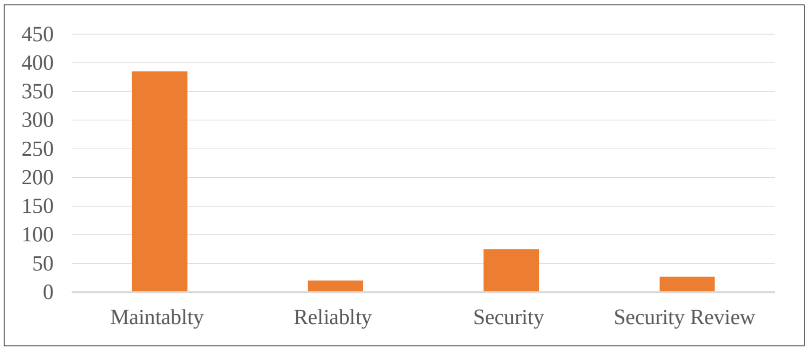


Figure 9: Chatbot Code Analysis Result

The technical team manages the debt items in the primary backlog, so they can give it a higher priority to organize the weekly activities on the Kanban map. Therefore, TD items should be prioritized according to their severity. Fig [10] shows a summary of the code internal defects for the 14 modules, distributed according to their severity level. Refactoring TD items for repayment, we suggest that developers balance available time in an iteration or release to give clients value and reduce TD. However, we recommend that the team focus its efforts on lowering TD for roughly 20% of the time after one iteration or release. The development team start to apply our approach from Release 2, and gradually, our approach allows the development team to control their debt and simply repay it during the project journey in every release.

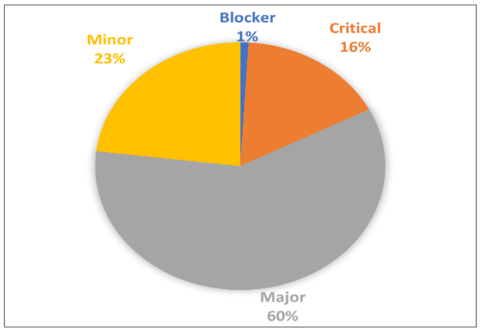


Figure 10: Chatbot Code Defect Severity

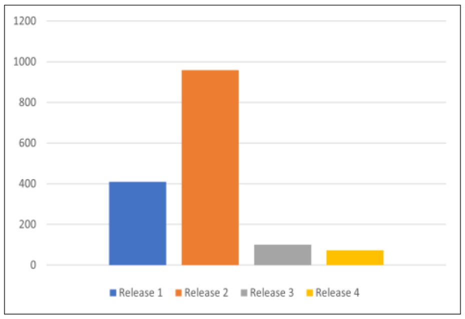


Figure 11: Chatbot Total Defects per Release on Project Backlog

Fig [11] shows the payment of TD defects in addition to Testing Team defects as each release includes fixing internal TD defects in parallel with adding new features on several releases prioritized according to defect severity that is reflected on project functionalities. As observed from Fig [11], defects have increased due to the start of fixing TD defects and then decreased gradually in the incoming releases. It is crucial to pay TD strategically and gradually.

Most interviewees believe that identifying and communicating TD allows them to improve their skills by gaining a comprehension of their errors and learning new concepts for managing TD. In addition, it appears that managing TD gives developers a sense of accomplishment and success. Several interviewees stated that refactoring and re-paying TD is an engaging activity that motivates them and enables them to continuously improve the quality of their work. As a result of this exercise, we conclude that putting our TDM proposed approach into effect is evidently advantageous for software quality level, improvement in tam morale, and the increase in project maintainability.

Case 3

The selected Company has different offices worldwide and provides solutions for telecommunications firms. The main objective of this exercise is to enhance the business unit's technical debt management capabilities. The selected company is operating at ISO process maturity and has an internal process drive to create corporate wide guidelines for handling TD. They aim to continue actively developing the aging product for at least another five years and anticipate that market conditions will be beneficial for it. Therefore, the members of the product development team, Scrum masters, and product owners support the implementation of an effective technical debt management approach to facilitate the management of the product's ongoing evolution.

The selected project is a (Digital Channel) web application with a front-end and back-end module designed for production use and created by the development team in the field that serve as our exercise. The company applies an agile software development methodology, which is utilized for organizing and setting priorities for the development tasks carried out throughout each iteration. For process governance, they use Scrum masters, while product owners are in charge of managing releases, customer engagement, and the requirement backlog. They are using a range of tools, such as Confluence to keep track of requirements and Jira for defect reporting. In addition, the project backlog is divided into new features and TD defects. These tools have allowed us to collect the required metrics we need for our research investigation. During the workshop held with the project team classified as (1 Scrum Master, 2 developers, and 1 Testing Engineer), we have discussed our findings from an earlier technical debt study as well as the TDM approach that are proposed. We have suggested integrating the proposed approach we have created with our suggested processes with the currently implemented processes, activities, and tools for TDM support, which can be a platform for defining how the company will handle TD in its projects.

Lack of knowledge about the software's development in the future is one of the practitioner's top concerns, and as a result, it has been difficult for them to prioritize necessary and urgent TD items for refactoring. Because of the emphasis on new features, some debt items "remain in the queue forever" if not given high priority. The company is currently identifying which debt items are the most important, based on whether the issue has affected customer satisfaction or if it inhibits the completion of other tasks.

However, most TD choices are made based on the manager's "gut feelings" rather than precise evidence collected through suitable assessment, as mentioned in the prioritization phase of our approach. The "gut feeling" has a significant impact on the outcome of TD prioritization. We have prepared brief sessions with the team to introduce the TDM processes. One of the main topics that come up is how to decide which activities should be prioritized and which should be put off or postponed, as previously mentioned in the prioritization phase of the present proposed approach.

The project team has decided to use the suggested approach and agreed that the discussion of areas for improvement and handling software TD within the context of the development team is considered to take place at daily and weekly meetings, planning meetings, reviews, and retrospectives. The development team starts using the proposed approach on the project release cycle in August 2022. Therefore, we can use information from release cycles that lasted until the production date for our analysis. For TD identification and code analysis, the development team is using SonarQube on Digital Channel source code to know the amount of bug-fixing and rework that would be needed for each defect type that has been registered on the project.

The Digital Channel consists of 4 modules, and they run the tool on the 4 modules. The product's source code accurately depicts its current state which satisfies all initial requirements and is prepared for initial deployment in production to achieve software continuous improvement, TD defects and new tasks shall be added to the project backlog, and prioritized refactoring shall be prioritized. The development team has direct input into how TD is prioritized to be refactored in their project backlogs. The team have informed the product owner about the refactoring time and discussed the involved risks and the product owner's chosen business requirement priority while organizing the sprint's activities. The definition of done is likewise thoroughly defined by the team, and tasks are only completed if the code is clear. To evaluate the performance of our process system, we have collected data from weekly release cycles over the project's journey. Figs [12–17] show the number of defects fixed per sprint, prioritized according to their severity during the project's journey.

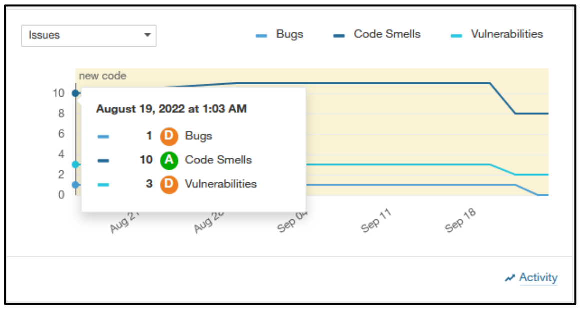


Figure 12: Digital Channel Release1

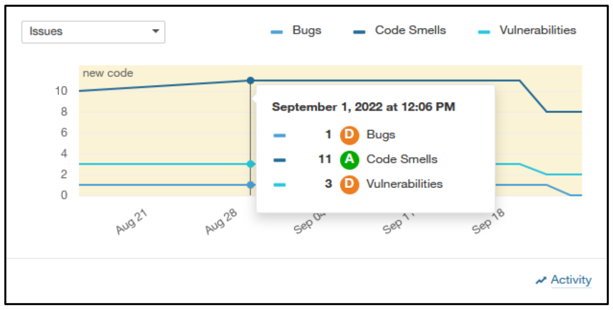


Figure 13: Digital Channel Release2

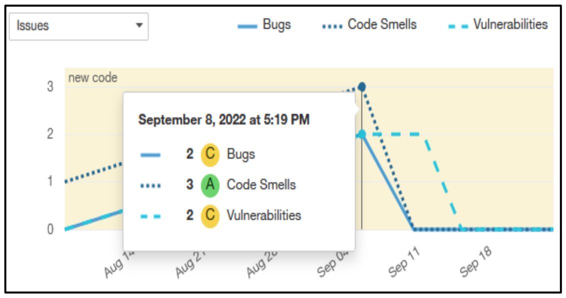


Figure 14: Digital Channel Release3

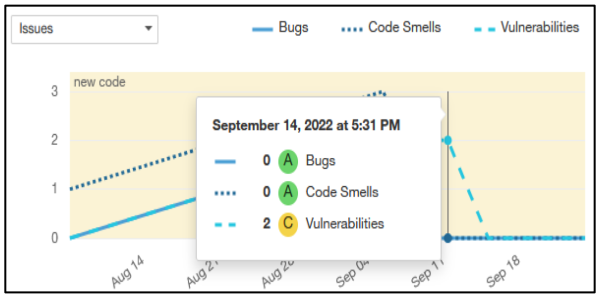


Figure 15: Digital Channel Release4

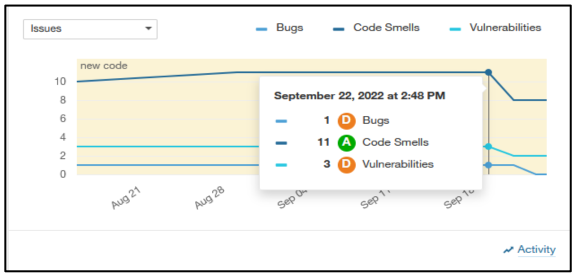


Figure 16: Digital Channel Release5

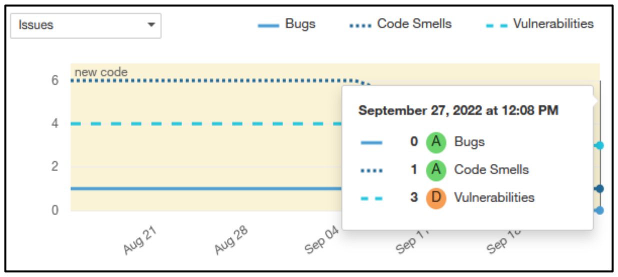
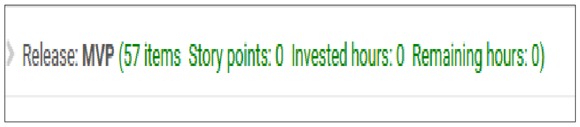
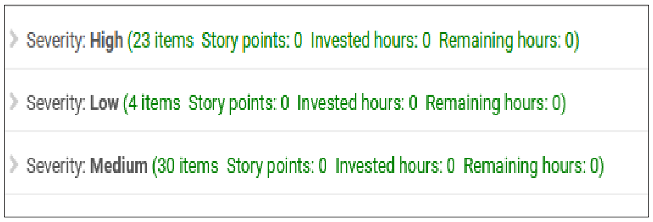


Figure 17: Digital Channel Release6



Release: MVP (57 items Story points: 0 Invested hours: 0 Remaining hours: 0)

Figure 18: Total Number of Defects for all Releases on Project Backlog

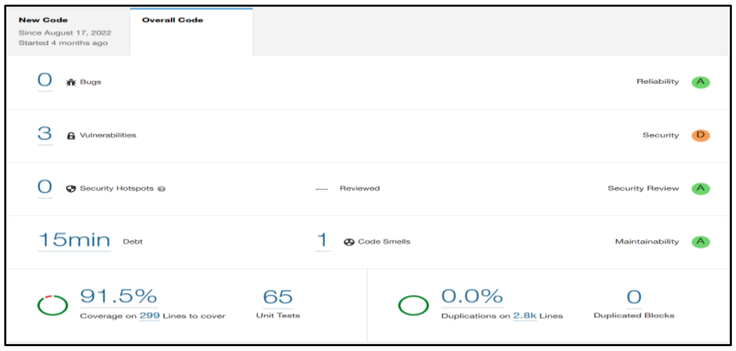


Severity: High (23 items Story points: 0 Invested hours: 0 Remaining hours: 0)

Severity: Low (4 items Story points: 0 Invested hours: 0 Remaining hours: 0)

Severity: Medium (30 items Story points: 0 Invested hours: 0 Remaining hours: 0)

Figure 19: Total Defects Classified by Severity on Project Backlog



New Code
Since August 17, 2022
Started 4 months ago

Overall Code

0 Bugs — Reliability A

3 Vulnerabilities — Security D

0 Security Hotspots — Reviewed — Security Review A

15min Debt — 1 Code Smells — Maintainability A

91.5% Coverage on 299 Lines to cover — 65 Unit Tests

0.0% Duplications on 2.8k Lines — 0 Duplicated Blocks

Figure 20: SonarQube Results at Final Release

We have recommended that the team reserve a full or partial sprint to work exclusively on backlog improvements for refactoring items where developers are pushed to refactor things from their backlogs. Figs [14, 15] are a full sprint for fixing TD code defects. Figs [18, 19] show the total number of defects on the project backlog for all sprints classified according to their severity in the project backlog. Fig [20] shows the final run of SonarQube after fixing code defects. In contrast, most interviewees emphasized the significance of TD management, which, in their view, makes the future superior to the present.

As a result, the software project may benefit from focusing on the software's correctness, utilizing a cost-benefit analysis to compare various TD items in terms of their low cost and high payoff, improving team motivation and morale, and increasing its maintainability. So, despite the relatively substantial investment in evaluations, the gains obtained in terms of contraction in defects and failures which in turn causes a reduction of defects costs.

5 Limitation and Threats to Validity

In this section, we examine potential threats to the accuracy of our work. While carrying out the present study, we have been encountered by certain limitations and validity threats for the research and, hence influencing its findings. We thereby advise researchers to take them into account before applying and expanding our findings.

Our study of putting the proposed approach into practice in the actual software development processes at three software companies enhances the study's external validity. Nevertheless, we have been only permitted to thoroughly research a small number of projects. We acknowledge that the study's limited sample size restricts its generality, and results from our small sample of the three cases may not be representative of all software companies. It is necessary to conduct additional research on the acceptance and application of the approach in a range of commercial and industrial contexts.

The research study exclusively relies solely on SonarQube tool for code analysis, including metrics and rules to assess code quality, extract each technical debt item, determine defect severity, and calculate estimation time for fixing technical debt items.

One of impediments faced by the researcher during conducting such studies is the limited access to the project documentation of the relevant software companies. As a result, the sample studies concentrate on code debt, which is frequently connected to the items under investigation, but they only represent a small part of the overall picture. We intend to consider different TD types in our research study to reach more reliable and experimentally supported findings.

6 Conclusion and Future Work

Software organizations shall frequently balance the expense of the software development process in terms of the required time and resources against the software's overall quality. The presence of TD affects several artifacts during the software development process; the impact of these risks shall be known to all project stakeholders. Our study contributes to TD research studies. In this study, we propose an approach to managing technical debt by providing processes for preventing, identifying, prioritizing, monitoring, and repaying TD to satisfy both software technical requirements and add value to the customer in order to put an end to project overruns and failures while cautiously balancing the trade-offs between quick delivery of required functionalities and high-quality product.

The goal is to comprehend how technical debt is defined, dealt with, monitored, and prioritized, as well as how each component influences technical debt decision making. We have introduced detailed descriptions, the goal, and practices for implementation details for each TDM approach phase. To evaluate the result of our proposed TDM approach, we have carried out an evaluation by applying TDM to 3 different SME software companies. The size of the selected companies has varied from relatively small companies (Case 1) to relatively mid-sized companies (Cases 2 and 3). The selected projects to examine the TDM approach are actual software projects, developed for actual clients, by actual professional developers, to give the research a significant advantage.

During applying our proposed approach for both Cases 2 and 3, we faced some challenges. The most vital aspect was managers lack of awareness of the necessity for refactoring, as they were not always aware of the risks and benefits of conducting a refactoring. In addition, all participants cited the difficulty in prioritizing and recouping the refactoring effort when multiple items and teams were involved. It required "double" the effort to rank the item using distinct criteria. Another challenge is ensuring effective and clear communication between team members and project stakeholders, an important aspect for successful implementation.

The analysis shows that employing our proposed TDM approach leads to a deeper comprehension of TD contraction in selected software companies Case 1: The company has decided to revamp the project from scratch due to the accumulated debt and take these causes as a preventive action for the new project by applying the proposed approach. Case 2: We investigated and analyzed what had occurred in the past to avoid repeating past mistakes. The project stakeholders' feedback after applying our approach is that putting the TDM approach in place is advantageous for software quality level, improved team morale and motivation, as well as an enhancement in its maintainability. Case 3: When ranking necessary and urgent TD items for refactoring, choices are made based on the manager’s "gut feelings" rather than precise evidence collected through suitable assessment, as mentioned in the prioritization phase of our approach. The "gut feeling" has a significant impact on the outcome of TD prioritization.

As discussed during interviews with three companies, we believe that TD lowers the morale of developers because it prevents them from completing their development duties, making progress, and achieving their objectives. Whereas TD management can boost their morale. The interview data indicate that TD can have a negative impact on affective and future/goal antecedents of morale. Alternatively, TD management has a positive impact on both factors’ morale, the future/goal dimension particularly.

As a result of applying our TDM approach, the software project may benefit from focusing on the software's correctness, utilizing a cost-benefit analysis to compare various TD items in terms of low cost and high payoff, showing considerable improvement in the team morale and motivation, in addition to achieving an increase in its maintainability.

Further studies are necessary to fully support and generalize the findings of this study; large-sample field testing utilizing the TDM proposed approach is essentially required. It will be beneficial to conduct more testing of the TDM approach at other software companies to broaden the processes that are considered for the five phases of technical debt management recommended by the approach. It can also be improved in the future through considering how the TDM approach can be successfully implemented in businesses that use various software development processes and quality management standards. With the participation of a broad range of business processes, the ecological validity of our findings may thus be confirmed in further study. On the whole, it is hereby incontestably evident that the proposed approach to technical debt management presented in this study offers a solid theoretical and empirical base for future researches undertaking the establishment of effective and efficient procedures of software development.

References

[1] Alfayez, R., & Boehm, B. (2019). Technical debt prioritization: A search-based approach. *In IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, 434-445.

[2] Alfayez, R., Alwehaibi, W., Winn, R., Venson, E., & Boehm, B. (2020, June). A systematic literature review of technical debt prioritization. In *Proceedings of the 3rd international conference on technical debt*, 1-10.

[3] Alves, N.S.R., Mendes, T.S., de Mendonça, M.G., Spínola, R.O., Shull, F., & Seaman, C. (2016). Identification and management of technical debt: A systematic mapping study. *Information and Software Technology*, *70*, 100-121.

[4] Avgeriou, P., Kruchten, P., Nord, R.L., Ozkaya, I., & Seaman, C. (2015). Reducing friction in software development. *IEEE Software*, *33*(1), 66-73.

[5] Berenguer, C., Borges, A., Freire, S., Rios, N., Tausan, N., Ramac, R., & Spínola, R. (2021, November). Technical debt is not only about code and we need to be aware about it. *In Proceedings of the XX Brazilian Symposium on Software Quality*, 1-12.

[6] Besker, T., Ghanbari, H., Martini, A., & Bosch, J. (2020). The influence of Technical Debt on software developer morale. *Journal of Systems and Software*, *167*.

[7] Besker, T., Martini, A., & Bosch, J. (2019). Technical debt triage in backlog management. *In IEEE/ACM International Conference on Technical Debt (TechDebt)*, 13-22.

[8] Codabux, Z., & Williams, B. (2013). Managing technical debt: An industrial case study. *In IEEE 4th International Workshop on Managing Technical Debt (MTD)*, 8-15.

[9] Codabux, Z., Williams, B.J., Bradshaw, G.L., & Cantor, M. (2017). An empirical assessment of technical debt practices in industry. *Journal of Software: Evolution and Process, 29*(10), 1-32.

[10] De Almeida, R.R., do Nascimento Ribeiro, R., Treude, C., & Kulesza, U. (2021). Business-driven technical debt prioritization: An industrial case study. *In IEEE/ACM International Conference on Technical Debt (TechDebt)*, 74-83.

[11] de Freitas Farias, M.A., de Mendonça Neto, M.G., Kalinowski, M., & Spínola, R.O. (2020). Identifying self-admitted technical debt through code comment analysis with a contextualized vocabulary. *Information and Software Technology*, *121*.

[12] Freire, S., Rios, N., Mendonça, M., Falessi, D., Seaman, C., Izurieta, C., & Spínola, R.O. (2020). Actions and impediments for technical debt prevention: results from a global family of industrial surveys. *In Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 1548-1555.

[13] Freire, S., Rios, N., Pérez, B., Castellanos, C., Correal, D., Ramač, R., Mandić, V., Taušan, N., López, G., & Pacheco, A. (2023). Software practitioners' point of view on technical debt payment. *Journal of Systems and Software*, *196*.

[14] Freitas, G., Reboucas, R., & Coelho, R. (2022). Exploring Technical Debt Tools: A Systematic Mapping Study. *In Enterprise Information Systems: 23rd International Conference, ICEIS 2021, Virtual Event, Revised Selected Papers*, *455*. Springer Nature.

[15] Ghanbari, H., Besker, T., Martini, A., & Bosch, J. (2017). Looking for peace of mind? manage your (technical) debt: An exploratory field study. *In ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 384-393.

[16] Harun, M.F., & Lichter, H. (2015). Towards a Technical Debt Management Framework based on Cost-Benefit Analysis. In *Proceedings of the 10 th International Conference on Software Engineering Advances*.

[17] Lenarduzzi, V., Besker, T., Taibi, D., Martini, A., & Arcelli Fontana, F. (2021). A systematic literature review on Technical Debt prioritization: Strategies, processes, factors, and tools. *Journal of Systems and Software*, *171*.

[18] Lenarduzzi, V., Besker, T., Taibi, D., Martini, A., & Fontana, F.A. (2019). Technical debt prioritization: State of the art. A systematic literature review.

[19] Lenarduzzi, V., Orava, T., Saarimäki, N., Systa, K., & Taibi, D. (2019). An empirical study on technical debt in a finnish SME. *In ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 1-6.

[20] Li, Z., Avgeriou, P., & Liang, P. (2015). A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, *101*, 193-220.

[21] M. Bomfim, M., & A. Santos, V. (2017). Strategies for reducing technical debt in agile teams. In *Agile Methods: 7th Brazilian Workshop, WBMA 2016, Curitiba, Brazil, Revised Selected Papers 7*, 60-71. Springer International Publishing.

[22] Oliveira, F., Goldman, A., & Santos, V. (2015). Managing technical debt in software projects using scrum: An action research. *In IEEE Agile Conference*, 50-59.

[23] Pavlič, L., Hliš, T., Heričko, M., & Beranič, T. (2022). The Gap between the Admitted and the Measured Technical Debt: An Empirical Study. *Applied Sciences*, *12*(15), 1-21.

[24] Perez, B., Castellanos, C., & Correal, D. (2020). Developing a theory based on the causes of technical debt injection into software projects in Colombia. In *Journal of Physics: Conference Series*, *1587*(1), 1-7. IOP Publishing.

[25] Pina, D., Goldman, A., & Tonin, G. (2021). Technical debt prioritization: Taxonomy, methods results, and practical characteristics. *In IEEE 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 206-213.

[26] Ramasubbu, N., & Kemerer, C.F. (2019). Integrating Technical Debt Management and Software Quality Management Processes: A Normative Framework and Field Tests. *IEEE Transactions on Software Engineering*, *45*(3), 285-300.

[27] Rios, N., Mendonça Neto, M.G.d., & Spínola, R.O. (2018). A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners. *Information and Software Technology*, *102*, 117-145.

[28] Rios, N., Spínola, R.O., Mendonça, M., & Seaman, C. (2018). The most common causes and effects of technical debt: first results from a global family of industrial surveys. *In Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 1-10.

[29] Rios, N., Spínola, R.O., Mendonça, M., & Seaman, C. (2020). The practitioners' point of view on the concept of technical debt and its causes and consequences: a design for a global family of industrial surveys and its first results from Brazil. *Empirical Software Engineering*, *25*, 3216-3287.

[30] Rosser, L.A., & Norton, J.H. (2021). A systems perspective on technical debt. *In IEEE Aerospace Conference (50100)*, 1-10.

[31] Saraiva, D., Neto, J.G., Kulesza, U., Freitas, G., Reboucas, R., & Coelho, R. (2021). Technical Debt Tools: A Systematic Mapping Study. *ICEIS (2)*, 88-98.

[32] Stettina, C.J., Garbajosa, J., & Kruchten, P. (2023). *Agile Processes in Software Engineering and Extreme Programming: 24th International Conference on Agile Software Development, XP 2023, Amsterdam, The Netherlands, Proceedings*, 193. Springer Nature.

[33] Wiese, M., Rachow, P., Riebisch, M., & Schwarze, J. (2022). Preventing technical debt with the TAP framework for Technical Debt Aware Management. *Information and Software Technology*, *148*.

[34] Wiese, M., Riebisch, M., & Schwarze, J. (2021). Preventing Technical Debt by Technical Debt Aware Project Management. *In IEEE/ACM International Conference on Technical Debt (TechDebt),* 84-93.

[35] Yudistira, B.G.K., Anggoro, R., & Shiddiqi, A.M. (2022). Addition of Neighbors in the Number of Vanets Node Factors: DSR-PNT Performance Study. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, *13*(4), 196-210.

Authors Biography

A. Hamed received the B.Sc. degree in computer and information system from Sadat Academy for Management sciences and the M.Sc. degree in Information system from Arab Academy for science, Technology and Maritime, currently is a PhD student at Faculty of Computer and Information, Mansoura University, Egypt. Her research interests include Software Quality Assurance and Testing, Technical Debt, and software Process Improvement. Her email abeer.salah.hamed@gmail.com , ORCID https://orcid.org/0000-0001-7595-0922

Prof. Dr EL-Bakry Hazem received the B.Sc. degree in electronics engineering and the M.Sc. degree in electrical commination engineering from the Faculty of Engineering, Mansoura University, Egypt, in 1992 and 1995, respectively, and the PHD degree from the university of Aizu, Japan in 2007. He is currently a full professor with the Faculty of Computer Science and Information Systems, Mansoura University, where he is also the Head of Information System Department. His research interests include Soft Computing, Image Processing, Neural Network, Biometrics and Pattern Recognition. His email elbakry@mans.edu.eg, ORCID https://orcid.org/0000-0002-4798-0427

Prof. Dr Alaa Eldin M. Riad is a professor at Faculty of Computers and Information, Mansoura University. He has received BS, MS, and PhD. in Electrical Engineering in 1982, 1988, and 1992 respectively, from Mansoura University, Faculty of Engineering, Egypt. Dr. Riad worked as the dean of the faculty of Computers and Information, Mansoura University. In addition, he worked as the chairman of information systems department, Faculty of Computers and Information. His research interests include intelligent information systems, mobile agents and networks, E-learning, and big data. His email amriad2000@mans.edu.eg,
ORCID: http://orcid.org/0000-0002-0091-3083

Prof. Dr Ramdan Moawad received the B.Sc. degree in Electric Engineering and the M.Sc. degree in computer engineering from Military Technical College, and the Ph.D. degree in Software Engineering from the ENSAE College, France. He taught several courses in CS and CE in several institutions including the American University in Cairo, the Military Technical College, Cairo University and the Arab Academy for Science and Technology. He joined Future University in 2011 and currently working as Vice-Dean of FCIT. His research interests include software engineering and software quality assurance. His email Ramdan.mowad@fue.edu.eg, ORCID https://orcid.org/0000-0003-2349-8203