Prague University of Economics and Business

Faculty of Informatics and Statistics

Ensuring compliance of SDLC documentation within large enterprises

BACHELOR THESIS

Study programme: Applied Informatics

Specialization: Applied Informatics

Author: Matvii Stonaiev

Bachelor Thesis Supervisor: Ing. Richard Antonín Novák, Ph.D.

Prague, May 2024

Acknowledgment

I thank everyone who supported me while creating this Bachelor's thesis. I want to express my immense gratitude to my work supervisor, Ing. Richard Antonín Novák, Ph.D.; without his help, this work would not even exist. Finally, I would like to thank every respondent who agreed to participate in qualitative research.

Abstract

This bachelor thesis focuses on ensuring compliance with Software Development Life Cycle (SDLC) documentation in large enterprises and managing the risks associated with SDLC.

The theoretical part of the project analyses in detail the key concepts associated with the SDLC and highlights the importance of documentation management in the software development process. It also discusses various SDLC methodologies and models that are in use in practice and can be implemented in large enterprises to ensure the quality and efficiency of software development.

The literature research in the theoretical part will discuss the importance and benefits of documentation management within SDLC and provide an overview of relevant standards and norms in this area. It will also explore the obstacles and challenges that large enterprises may face when implementing an SDLC and the strategies that can be used to overcome them.

The practical part will focus on the application of Risk Management methods to name the causes and consequences associated with the different level and quality of documentation produced during the SDLC. Appropriate qualitative research methods (interviewing, observation, experiment and document analysis...) will be used to develop the risk matrix. Emphasis will be placed on interviewing at least ten qualified professionals working in the field of software development and risk management.

The interviews will be conducted to cover key aspects of the SDLC documentation and will focus, in particular, on issues related to risk identification and management. During the interviews, respondents will be asked to evaluate current practices in their companies and to identify areas for improvement.

Based on the qualitative research with interviews and thematic analysis methods conducted, key conclusions will be drawn and key risks associated with ensuring compliance with SDLC documentation will be identified. These findings will be used to develop a Mitigation Plan that will include specific actions and strategies to eliminate or reduce the identified risks.

An appropriate framework (e.g. NIST, COBIT 5, ISO 31 000...) will be selected to address the risks associated with SDLC documentation and the recommendations in the recommended literature and appropriate methods for this area will be followed when conducting qualitative research.

In the final part, the student will carry out an evaluation of the theoretical and practical part, including a discussion of his own conclusions.

Contents

List of Abbreviations

SDLC	Software Development Life Cycle
RAD	Rapid Application Development
XP	Extreme Programming
DSDM	Dynamic Systems Development Method
FDD	Feature-Driven Development
RS	Requirements Specification
SDD	Software Design Document
SA	System Administrators
ISO	International Organization for Standardization
FDA	The U.S. Food and Drug Administration
GDPR	General Data Protection Regulation
CMMI	Capability Maturity Model Integration
SQA	Software Quality Assurance
PCI DSS	Payment Card Industry Data Security Standard
SOX	Sarbanes–Oxley Act
IEEE	Institute of Electrical and Electronics Engineers
GDP	Good Documentation Practices
DMS	Document Management System
GRC	Governance, Risk, and Compliance
API	Application Programming Interface

Introduction

In modern software development, ensuring compliance with various phases and deliverables of the Software Development Life Cycle (SDLC) documentation is critical for large enterprises in managing risks effectively and maintaining high-quality standards. The SDLC methodology, which includes planning, analysis, design, development, testing, and deployment, has become increasingly complex and sophisticated in recent years due to the rapid pace of technological advancements and the growing need for software solutions that led to the diversification of business requirements. As such, organizations must aim to develop a structured and comprehensive approach to SDLC documentation compliance that accounts for various challenges, including regulatory requirements, resource constraints, stakeholder expectations, and project timelines. Moreover, effective risk management practices must be integrated with SDLC documentation compliance to identify potential risks early on and mitigate them proactively.

The author's motivation was driven by a desire to understand how SDLC and documentation work outside his company and provide readers with insights into the most frequent problems and best practices to solve them.

The study will delve into the different types of documentation used throughout the SDLC process and assess their potential associated risks. Ultimately, the findings of this thesis will provide valuable insights for businesses seeking to improve their SDLC documentation processes and reduce the likelihood of risks that could lead to project delays, legal issues, or other adverse outcomes.

Goals

The central objective of this bachelor thesis is to conduct a comprehensive investigation into the various types of risks commonly associated with software development life cycle documentation within larger enterprises and provide a Mitigation Plan with proposed solutions for the risks.

Structure

The theoretical part of the Bachelor's thesis will focus on analyzing the main concepts of SDLC methodologies and phases from a theoretical standpoint. Large companies' documentation, including key deliverables and documentation concepts, will be examined. The norms and standards used to manage SDLC documentation will also be presented. Finally, the last chapter of the theoretical section will address the challenges that might arise in SDLC documentation management and the strategies used to ensure compliance with the process.

The practical part of the research methodology is based on the APO12 (Align, Plan, and Organize) process called Manage Risk from the COBIT 5 framework, which defines objectives, identifies, analyzes, and addresses risk. Qualitative research will be conducted as one-to-one semi-structured interviews to identify and analyze existing risks in large enterprises and to provide readers with the most frequent risks in qualitative research. A risk matrix will be drawn to prioritize identified risks. Solution for these risks will be suggested as a Mitigation Plan to drive the risks effectively.

The thesis concludes by evaluating different work aspects, including the students' conclusions from the study.

Research Question

The main question for the research is:

- What are the potential risks in Software Development Life Cycle Documentation Management, and what strategies could be applied to improve SDLC documentation practices and decrease the possibility of risk appearance?

Methods

Qualitative research methods such as semi-structured interviews, transcription analysis, thematic analysis by Braun and Clarke, concretization and abstraction, induction and deduction, and comparison will be used to identify potential risks

Keywords

Documentation management, Qualitative Research, Quality Assurance (QA), Software Development Life Cycle (SDLC), Risk Management

1 Software Development Life Cycle

SDLC was regarded as a System Development Life Cycle in the 1950s and was used as a model for large enterprises to drive complex business systems that required a lot of calculations, data processing, and modeling. With the development in the information technology sphere, this definition migrated to software development, which focuses on this specific sphere (Shantanu, 2018).

From the start of the Software Development era, people have struggled to find a way to develop software with the most minor budget and time without compromising the quality of the product. When developers avoided using a consolidated framework, this frequently led to implementation and maintenance issues or high coordination costs. This led to attempts to create a unified process for the development of applications (Shantanu, 2018).

Software Development Life Cycle (SDLC) is a framework that defines the process organizations use to develop an application from its origin to the end of its life cycle. The first SDLC model, named waterfall, was invented in 1956 by Herbert D. Benington. In 1970, it was updated and modified by Winston Royce. The waterfall model, also known as the cascade model, created a solid basis for the development of models in the future (Ruparelia, 2010).

Nowadays, the waterfall model is counted as traditional and has variations, such as the Verification and validation model (V-shape Model), Spiral model, Iterative model, Rapid development model (RAD), and Evolutionary model (Ruparelia, 2010).

However, when internet application work began, traditional linear models needed to be impeccable. Thus, there appeared to be a need to create a more collaborative and iterative model. In 2001, a group of senior software developers posted an Agile Manifesto that described the basic principles of how Agile Software Development must work. The main principles that were understated are:

- Individuals and interactions over processes and tools.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan.

That gave a start to Agile SDLC approaches, which have proved influential throughout the years. The most popular Agile SDLC models are Kanban, Extreme Programming (XP), Scrum, Cockburn's Crystal Family, Dynamic System Development Method (DSDM), Feature Driven Development, and Adaptive Software Development (Gelperin, 2008).

The Systems Development Life Cycle (SDLC) was initially designed for business projects. However, with the advancements in Information Technologies, it has been widely adopted in the Software Development sphere. Two major models have been invented to help teams develop and implement software effectively and efficiently: traditional and agile.

1.1 Software Development Life Cycle Models

Even though SDLC models can be divided into different groups and be classified differently depending on criteria, I focused on two main groups of SDLC models:

- Traditional linear models.
- Agile approach methodologies.

1.1.1 Traditional Linear Models

The most known traditional model is the waterfall model. It ensures the completion of the design phase before the development phase.

This model is applicable if all requirements are known and unchangeable in the early stages of development, and all documentation can be prepared before the development phase. This model is frequently unsuitable for real projects because it is hard to change documentation during the project due to the linear approach that leads to high costs, ineffectiveness, and administrative expenses (Andersen, 2023).

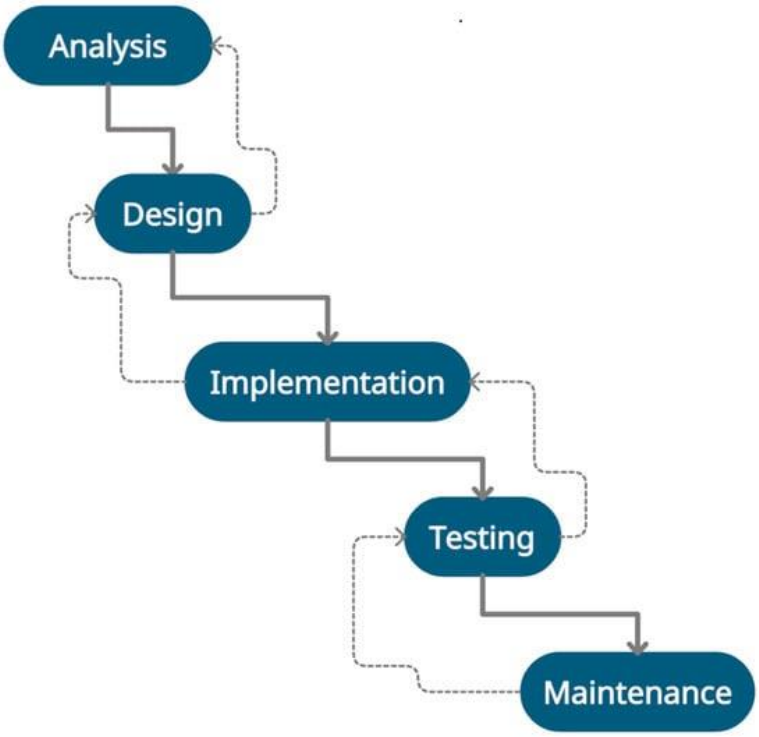

Figure 1.1 Basic Waterfall Model, inspired by Bassil and Sommerville (Saravanos & Curinga, 2023)

The waterfall model has been improved ever since. Royce has created feedback sessions after each phase in this model so the team can always return to the previous stage if needed. In 1988, Birrell and Ould created an extension of the basic waterfall model; they extended the maintenance phase, as shown in Figure 1.2, to ensure constant improvements of the designed product. However, the B-Model was suitable for the same category of products as the waterfall model (Ruparelia, 2010).

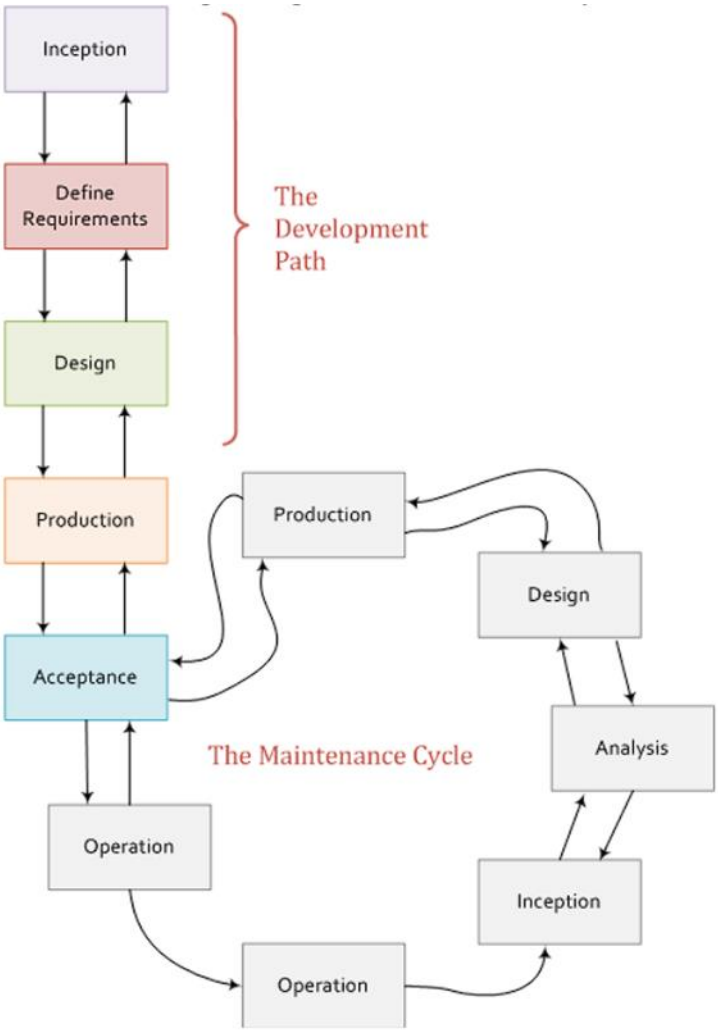

Figure 1.2 The b-model extends the waterfall model (Ruparelia, 2010).

V-Model

NASA invented the vee model, which was first presented in 1991 at a symposium in Tennessee. V-model has two legs: left and right (Ruparelia, 2010).

The V shape model is a well-known representation of the systems development life cycle. It is divided into two legs, with the left leg representing the process of decomposition and definition, where user requirements are broken down into smaller components. The right leg represents the integration and verification of these components into successive levels of implementation and assembly. Every requirement or design stage of the left leg corresponds with the right leg's verification or integration stage (Ruparelia, 2010).

The vee model has been improved into the vee+ and vee++ models for more product categories.

A significant advantage of the V model is its applicability to large-scale projects, where the decomposition, integration, and verification stages can be aligned with all project stakeholders before moving on to the next stage.

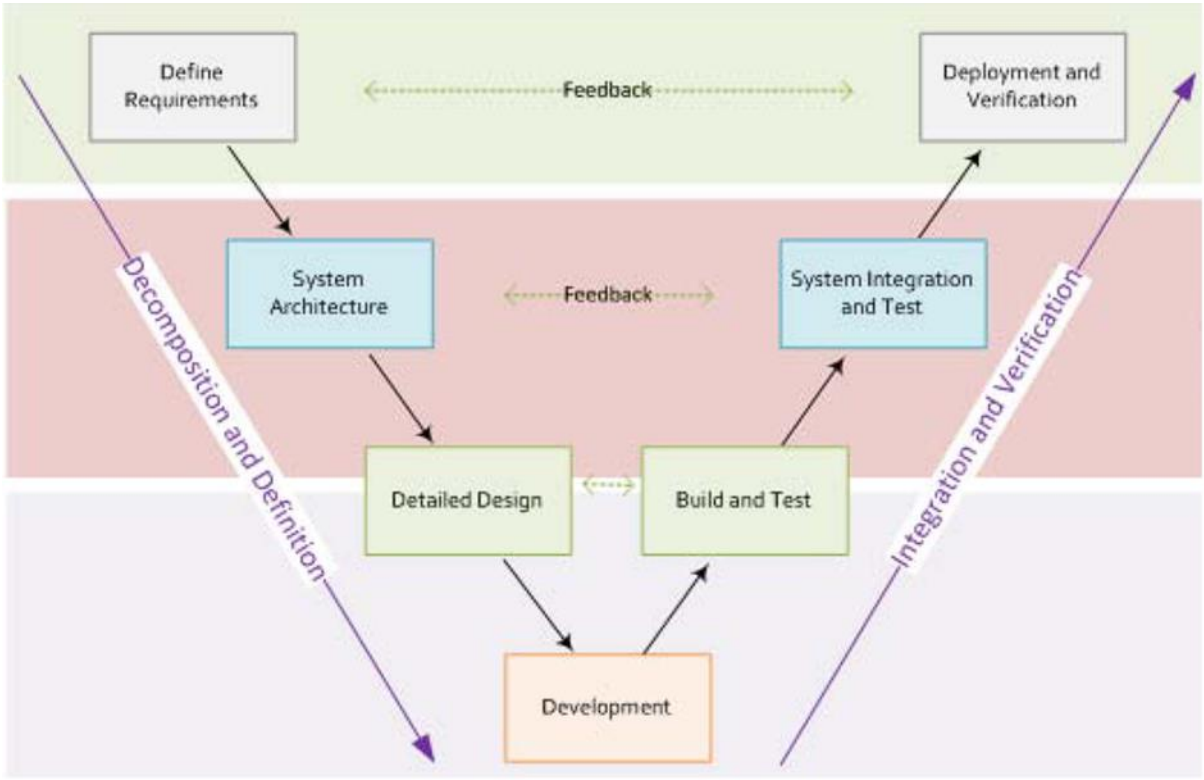

Figure 1.3 V-model with built-in quality assurance (Ruparelia, 2010)

Spiral Model

The waterfall model was modified by Boehm in 1986 to the Spiral Model. The philosophical idiom of the Spiral model is Start small, think big. The main difference from previous models is that the paradigm shifted from a specification-driven approach to a risk-driven one. The spiral model has four quadrants (Ruparelia, 2010):

- Determine objectives.
- Evaluate alternatives and identify and resolve risks.
- Develop and test.
- Plan the next iteration.

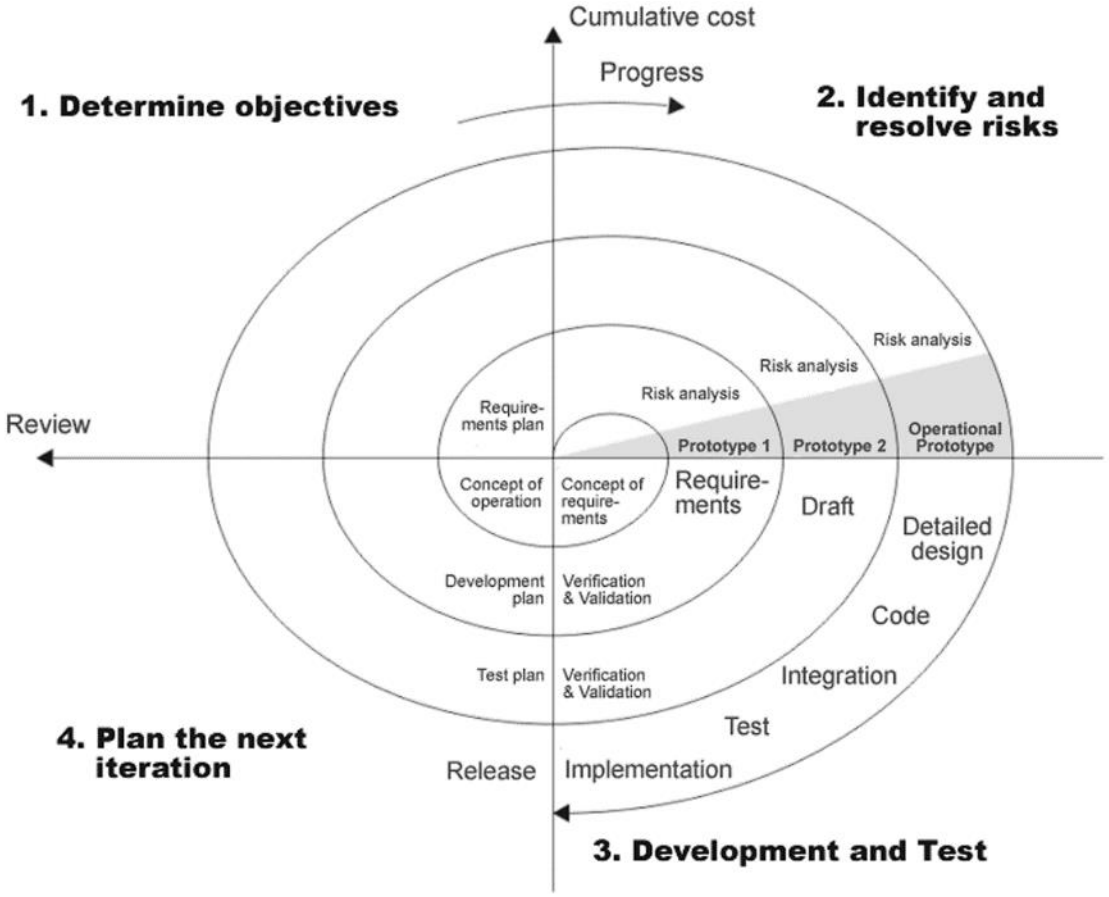

Figure 1.4 Boehm's spiral life-cycle (Ruparelia, 2010)

The essential advantage of the spiral model is a risk-based approach. The main challenge requires adaptive project management with flexible contracts between the stakeholders. Moreover, it requires professional systems designers to identify and eliminate risks connected with software development.

Rapid Application Development

RAD was developed by James Martin in 1991; the approach is based on prototyping in iterative development (Ruparelia, 2010).

RAD requires a collaborative space where stakeholders actively participate in prototyping, suggesting test cases, and working on improvements. Decentralization of decision-making is also applicable to the Rapid Application Development model, so decisions are usually made by the functional team, not by the project manager or lead Developer. The central idiom of the RAD model is to Release early, release often, and listen to your customers. It is assumed that RAD, in its ideas, is a predecessor of Agile models and methodologies (Peyyala, 2022).

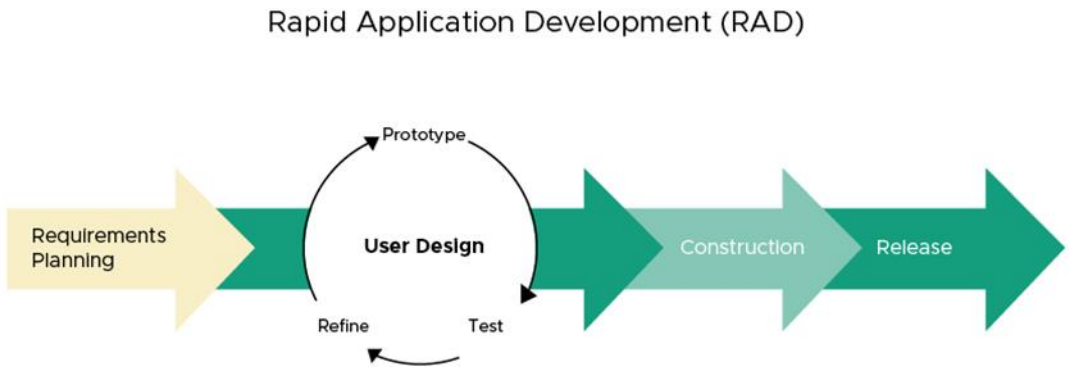

Figure 1.5 Rapid Application Development (Peyyala, 2022)

1.1.2 Agile Approach Methodologies

Agile methodologies were developed as part of the evolution of the Software Development Life Cycle (SDLC) to meet the needs of large enterprises. The issue with the traditional approach of gathering all requirements before development was that it needed to be more efficient. As a result, people started thinking about a model where requirements and design elements could be added at any time during the development stage (Shah et al., 2016).

The agile approach first appeared in 2001 when senior developers decided to base their work on collaboration, fast and simple development with continuous feedback from project parties.

They published 12 principles of the Agile Manifesto:

1. Our highest priority is to satisfy the customer through the early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for a shorter timescale.

4. Business people and developers must work together daily throughout the project.

5. Build projects around motivated individuals. Please give them the environment and support they need and trust them to do the job.

6. Face-to-face conversation is the most efficient and effective method of conveying information to and within a development team.

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity--the art of maximizing the amount of work not done--is essential.

11. The best architectures, requirements, and designs emerge from self-organizing teams.

12. At regular intervals, the team reflects on becoming more effective and then tunes and adjusts its behavior accordingly (Beck et al., 2001).

Agile models are usually drawn as a loop to confirm continuous flow, continuous work, and improvement in the agile methodologies.

The most popular agile methodologies are Kanban, Scrum, Extreme Programming (XP), Feature Driven Development (FDD), and Dynamic Systems Development Method (DSDM) (Shah et al., 2016).

Kanban

Kanban is a system developed by Japanese engineer Taiichi Ohno to meet the needs of Toyota. Ohno's invention transformed Toyota's manufacturing process from a "push" system to a "pull" system. This means that products are no longer produced to fill the market but rather based on the market's demands (Martins, 2024).

During the 2000s, Kanban principles were introduced to software development, explicitly adopting the "pull" system previously used in manufacturing processes. This system allowed developer teams a more streamlined approach to their work, focusing on continuous improvement and operational efficiency (Martins, 2024).

With Kanban, teams can visualize their workflow and identify bottlenecks, allowing for a more proactive approach to addressing issues and improving overall productivity. By adopting this methodology, software development teams have delivered higher-quality products more efficiently.

This methodology allows teams to use a Kanban board to track their work progress. A Kanban board usually comprises columns representing distinct stages of a process, such as To Do, In Progress, and Done. Each column includes cards that represent individual work items or tasks. As a work item moves through various stages, the corresponding card is moved from one column to the next. This enables team members to quickly determine the status of each work item and identify any bottlenecks or areas for improvement (Martins, 2024).

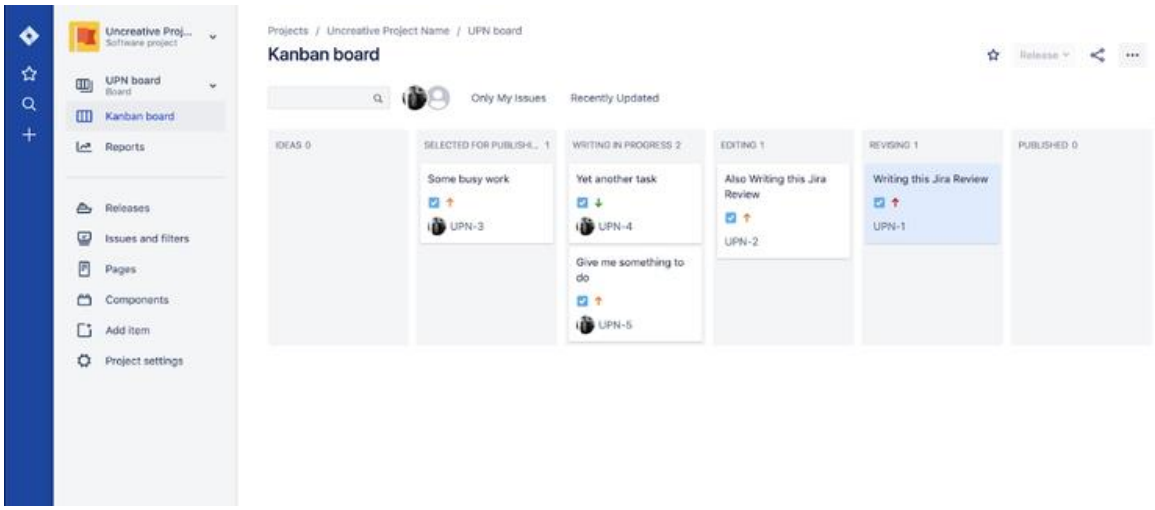

Figure 1.6 JIRA Kanban Board (Morpus, 2024)

Before the advent of Virtual Kanban boards, physical boards were commonly used on-site. However, with the rise of digital technologies, online Kanban boards have become increasingly popular among teams and organizations. Atlassian's JIRA is a trendy online board that offers a range of powerful features and capabilities. With JIRA, teams can easily track and manage their work, visualize their workflow, and collaborate more effectively. The platform provides a centralized hub for project management, enabling teams to stay organized, focused, and productive.

SCRUM

Scrum is a framework introduced by Hirotaka Takeuchi and Ikujiro Nonaka in 1986 in an article they published at Oxford University. This framework is designed to help teams work together to develop complex products. 1995, the framework was further improved and published as the SCRUM Development Process (Martins, 2024).

This process involves breaking down a project into small parts called sprints, each lasting for a short period. During each sprint, the team works to complete specific tasks and goals. The SCRUM Development process also includes regular meetings, known as SCRUM ceremonies, where the team discusses progress, identifies obstacles, and plans for the next sprint (Martins, 2024).

Scrum meetings:

- **Sprint planning**: Define priorities from the backlog for the next sprint.
- **Daily Scrum meeting**: quick discussion on task progress, sharing updates, and triaging new issues.
- **Sprint review**: held at the end of the sprint by developers to provide information on the review of accomplished projects and decide if any other changes are required.

- **Sprint retrospective**: held after the sprint and before the new sprint planning meeting. The main goal is to review what was done wrong or right in this sprint and make necessary changes to the next sprint.

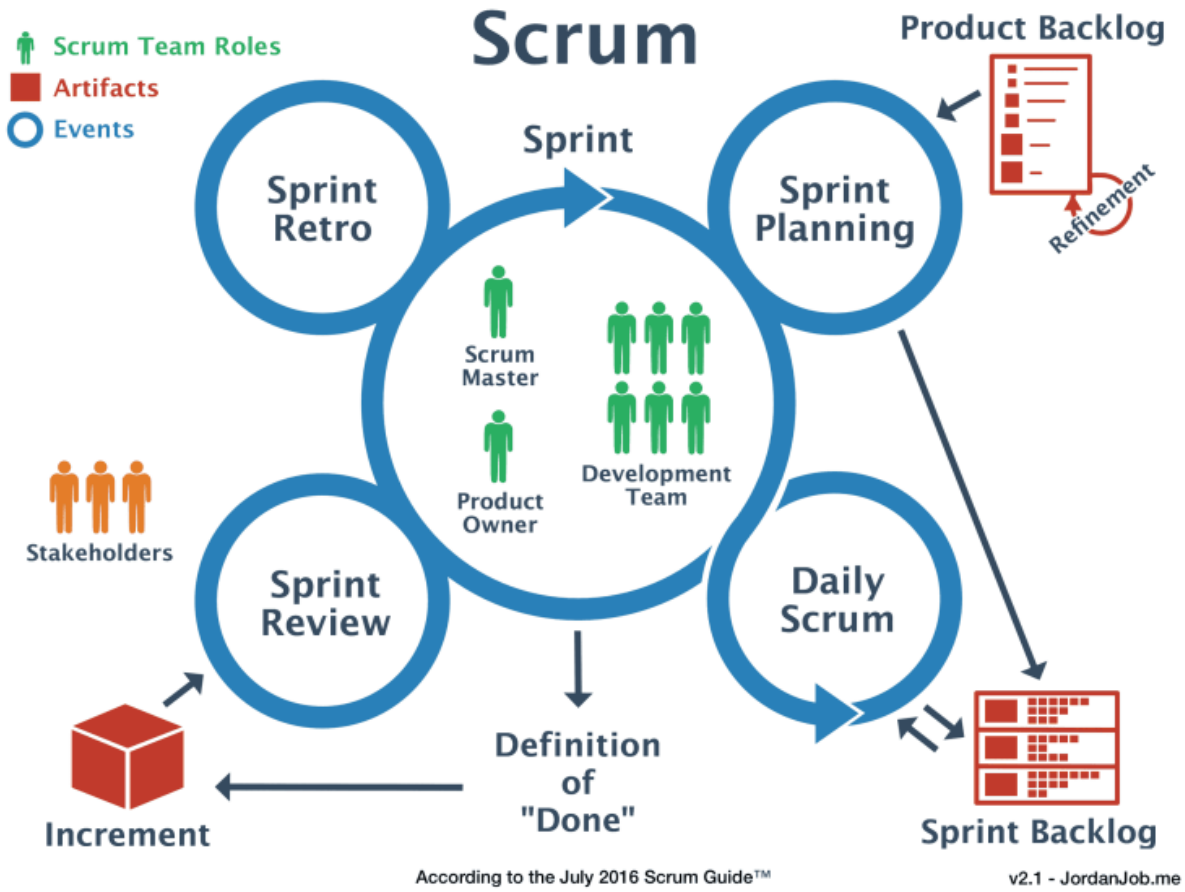

Figure 1.7 SCRUM methodology (Vroon, 2019)

Overall, the SCRUM Development process has proven to effectively manage complex projects and ensure that teams work efficiently towards their goals.

Extreme Programming (XP)

Extreme Programming (XP) is a software development methodology emphasizing teamwork, communication, and customer satisfaction. It has five core values that guide its practices and principles (Ruparelia, 2010):

- **Simplicity**: This value encourages developers to write simple, clean, and transparent code that is easy to understand and maintain. The idea is to keep things simple, which can lead to clarity and errors.
- **Communication**: XP highly values communication between team members, customers, and stakeholders. Frequent and open communication helps ensure everyone is on the same page and reduces misunderstandings and conflicts.
- **Feedback**: XP emphasizes the importance of getting feedback early and often. This helps developers catch errors and improve before they become more significant

problems. Feedback can come from various sources, such as customers, team members, and automated testing.

- **Courage**: This value emphasizes taking calculated risks and making bold decisions. Developers should be willing to challenge assumptions, try new approaches, and speak up when they see issues. Courage also means being willing to admit mistakes and learn from them.
- **Respect**: XP recognizes that every team member brings unique skills, experiences, and perspectives. This value emphasizes treating everyone with respect and dignity, regardless of their role or background. Respect also means valuing diversity and inclusivity and creating a safe and welcoming environment for all team members.

Feature Driven Development

FDD is a software development methodology that combines various elements of Agile SDLC approaches. Its primary objective is to deliver new features to stakeholders quickly. To achieve this, FDD places significant emphasis on obtaining feedback from stakeholders about the features they require, as meeting customer demand is the primary goal of FDD.

This model allows teams to make updates on projects very fast. If any feature has changed or an error appears, it can be quickly changed through iteration and implemented, as all stages are constantly moving.

1.2 Phases of Software Development Life Cycle (SDLC)

The Software Development Life Cycle (SDLC) is a comprehensive set of guidelines that software development teams follow to ensure that software is developed, implemented, and deployed effectively. The SDLC is divided into seven distinct phases that describe the steps required to create successful software solutions. These phases include planning and analysis, requirements analysis, design, development, testing, implementation, and maintenance. Each phase has specific objectives, deliverables, and activities that must be completed before moving on to the next phase. By following the SDLC, development teams can ensure that software is delivered on time, within budget, and to stakeholders' satisfaction.

1.2.1 Planning and Analysis Phase

The first phase of SDLC is where the team defines the scope and purpose of the application. An analysis of possible issues and challenges the team might face during development also occurs during planning. The project manager collects requirements from stakeholders to better understand the app's aim (Clark, 2022). A Project Plan is usually defined in the planning phase.

The best questions to define the software's goals in this phase are:

- What value are we delivering to customers?
- What problem can this software solve?

When all the issues concerning the project's goals are clear, the team is in line with stakeholders, and the team understands what value they bring, the team can move to the next phase.

1.2.2 Requirements Analysis

This phase seeks to identify the final requirements of customers. Interviews and surveys are usually conducted to understand the expectations and demands of end users (Clark, 2022).

The group analyzes the collected data by separating the desirable characteristics from the essential ones. This analysis allows the team to comprehend better the software's functionality, performance, security, and interface requirements.

All this analysis ends in the Requirements Specification (RS) Document. Here, all the requirements are connected and described as a guide for development. The product owner and technical unit usually validate RS to confirm compliance.

1.2.3 Design Phase

The design phase is crucial to establishing a solid foundation for any project. The design team works together to create a comprehensive plan that outlines the development and implementation process. By doing so, the project can be set up for success from the very

beginning and ensure that it meets the expectations of all stakeholders involved (Clark, 2022).

This phase is about making software user-friendly and high-performing.

Key activities for the design phase:

- Build software's structure.
- Design user interface.
- Craft design flow diagrams.
- Identify system dependencies and integration points.
- Set application limitations.

A software design document (SDD) contains all the information processed through the design phase and helps developers during the coding phase.

1.2.4 Development Phase

Developers convert the planning and design phases into tangible code in the development phase.

Developers aim to create user-friendly, functional, and efficient software. Appropriate programming language is used as described in SDD and is driven by coding guidelines.

During this phase, conducting frequent and thorough code reviews is essential. This process involves team members examining and testing each other's code to ensure it meets the project's requirements. This is a critical step as it helps to identify any inconsistencies, bugs, or potential issues that could impact the quality and functionality of the software. It is recommended that sufficient time and resources be allocated for code reviews to ensure the software is of the highest quality.

By the conclusion of this phase, functional software components will be developed and operationalized. It culminates in all preceding stages, such as planning, analysis, and design. Software should strive to be flawless, although it may only sometimes be perfect at this stage of development.

1.2.5 Testing Phase

Once the software development phase is complete, ensuring the product works as intended and meets the user's requirements is crucial. Therefore, the product is moved to the testing environment, where the entire functionality is thoroughly tested. This testing process involves verifying all functionalities and features according to the user's specifications (Martin, 2024). The testing team performs a series of tests to ensure the software is bug-free, stable, and reliable. Any issues or defects during this phase are reported to the development team for fixing. Once all issues are resolved, the product is deemed ready for deployment.

The testing phase is critical to ensure precise work and robustness of applications.

Six key phases of testing:

1. Requirement analysis.
2. Test planning: This stage involves creating a detailed plan for the testing process. The plan should include the testing objectives, scope, and approach. It should also outline the resources required for testing, such as tools, equipment, and personnel (Deveraj, 2023).
3. Test case development: In this stage, test cases are designed and developed based on the testing objectives, scope, and approach defined in the previous stage. Test cases should cover all possible scenarios and edge cases to ensure that the software or system functions as expected (Deveraj, 2023).
4. Test environment setup: This stage involves setting up the testing environment to replicate the production environment as closely as possible. The testing environment should have all the necessary hardware, software, and network configurations to ensure that the testing results are accurate and reliable (Deveraj, 2023).
5. Test execution: The testing is performed based on the test cases developed in the previous stage. The testing should be performed systematically, and the results should be documented for analysis and reporting (Deveraj, 2023).
6. Test closure: This is the final stage of the testing process. It involves analyzing the testing results, documenting the findings, and reporting them to stakeholders. It also involves ensuring that all the testing objectives have been met and that the software or system is ready for deployment. (Deveraj, 2023).

These phases help testers effectively validate software and send bugs back to developers.

1.2.6 Deployment Phase

After the development and testing stages have been completed, the application is ready to be moved to the production environment. This is the final stage before users can access and use the application. The production environment is a live system that supports the application's operational needs. Before deploying the application to the production environment, it needs to undergo a series of checks, including performance testing and security testing, to ensure that it operates correctly and is free from any potential issues. Once these checks have been completed, the application is released to the production environment, where users can access it for their intended purposes.

With the deployment of the application, the team must provide instructions so that the user can efficiently operate the software. It includes System use procedures, manuals, on and off-site training, and continuous support.

1.2.7 Maintenance Phase

The maintenance phase of the Software Development Life Cycle is marked by ongoing support and development, ensuring that the software functions as well as possible for an extended period and that customer expectations are met.

The main goal is to adjust to the software's evolving requirements. This adaptation entails reacting to user input, fixing unforeseen problems, and updating the program to users' changing needs.

Bug fixes, patch implementation, and regular software updates are all included in maintenance tasks. Another essential element is user support, which provides assistance and direction to users with problems with the software.

Long-term planning is also considered during maintenance, such as software replacement or upgrade. The software's lifecycle and technological advancements affect the software's changes and evolution.

2 Documentation in Software Development Life Cycle

As the world of software development has progressed, various SDLC models and methodologies have emerged over the years. This has given rise to different software development characteristics, such as new technologies, business domain advancements, and hardware specifications changes (Berhouma, 2020). These factors have made software development more complex than ever before, and as a result, documentation has become an integral part of the Software Development Life Cycle. By documenting each phase of the SDLC, software developers can ensure that their software is of the highest quality, meets end-users needs, and is updated with the latest industry standards.

Technical documentation carries all the necessary information and data on software. It describes all the processes in SDLC, such as API connections, integrations, requirements, and libraries.

2.1 Types of Documentation in SDLC

Software technical documentation is usually divided into three main categories:

- Product documentation.
- Process documentation.
- Code documentationn

Product and process documents are two essential types of documentation in software development. Product documents provide a detailed description of the product being developed, including its features, functionality, and technical specifications. These documents are critical for ensuring the development team understands what they are building and how it should work.

On the other hand, process documents describe the development and delivery process itself. They outline the steps, which include project planning, design, development, testing, and deployment. Process documents are essential for ensuring that the development team follows a standardized approach to software development, which can help improve the final product's efficiency and quality.

Explaining and defining the functionality, purpose, and usage of software code using annotations, comments, and other forms of documentation is called code documentation. This type of documentation assists developers, users, and maintainers comprehend how the code operates, its intended functionality, and how to interact with it effectively.

2.1.1 Process Documentation

Process documentation covers all activities that happen in the development of the product. The main point of using process documentation is to make the development process more understandable and well-organized.

The primary process documents in the software development life cycle are project/release plans, standard procedures, reports, and metrics.

Project plan: The document should outline a comprehensive plan detailing all the activities that will take place during the release, including the specific tasks that will be accomplished, the associated costs and timeframes for each task, and the deliverables that will be completed during the release. Additionally, it should include the key milestones and resources required to execute the project successfully. The document should be designed to provide stakeholders with a clear understanding of the project's scope, timeline, requirements, expected outcomes, and benefits. The project plan can be divided into a couple of documents, such as the Project resource plan, project deliverables plan, and others.

Standard procedures: This document outlines the guidelines and best practices for coding and UX design that are to be followed during the release phase of the project. It provides a detailed description of the coding standards that should be adhered to, including naming conventions, code formatting, and documentation requirements. The purpose of these standards is to ensure that the final product is of the highest quality, is easy to use and navigate, and meets the needs of its users. Moreover, with coding standards, the product will be easily understood by external developers, which gives flexibility to the product team.

Reports and metrics: At the end of each sprint or iteration, this document serves as a retrospective report that evaluates the efficiency and effectiveness of the allocated time and resources for the entire release process. It provides a comprehensive analysis of the tasks completed during the sprint and the outcomes achieved, highlighting areas that need improvement and areas of success. This report helps the team identify areas for improvement and strategies to optimize the release process in future sprints.

2.1.2 Product Documentation

Product documentation is divided into two big groups:

- System documentation.
- User documentation.

System documentation provides a detailed system description, including its various parts, requirements, design configuration, development process, and user acceptance testing. The documentation is designed to assist developers, testers, and end-users in understanding the software and how it works. It typically includes technical specifications, diagrams, flowcharts, and other relevant information to help stakeholders understand the system's functionality, architecture, and design.

Types of system documentation:

1. **Requirements document:** This comprehensive document provides a detailed account of all the requirements put forward by stakeholders, which must be effectively implemented into the software. It includes a thorough analysis of the various features, functionalities, and performance metrics that are expected, along with any specific technical or business requirements that have been communicated. Additionally, the document outlines the timelines, resources, and deliverables necessary for successfully implementing these requirements, ensuring that all stakeholders are fully satisfied with the final product.
2. **The design document** details the interface design, infrastructural design, and the detailed design of individual components used to build the software. It provides a blueprint for the development team, ensuring the final product aligns with the intended functionality and meets end-users' needs. The interface design outlines the user interface and how it will interact with the software's underlying components. The infrastructural design outlines the overall system architecture and how it will handle data flow and processing. The detailed design of individual components outlines the specifics of each element, including its functionality, inputs, outputs, and interactions with other components.
3. **Quality Assurance documentation:** These documents include a comprehensive set of documents describing the testing processes, methodologies, and strategies used to ensure the quality and reliability of the software. These documents contain detailed information about the tests conducted, the tools and frameworks utilized, and the testing process results. The documentation also includes the test cases and plans, which help identify defects or bugs in the software and ensure that they are fixed before the software is released to the users.
4. **Maintenance guides:** This document is a comprehensive maintenance guide for developers responsible for operating the system. It outlines various processes and procedures that developers should follow to ensure the smooth functioning of the system. In addition to providing step-by-step instructions, this guide includes best practices and troubleshooting tips to help developers address any issues that may arise during maintenance. Developers can refer to this document as a guide whenever they need to perform routine maintenance tasks or resolve technical problems.

User documentation

This type of documentation is created specifically for people who use the application. It is important to note that the documentation may differ depending on the usage angle. For

instance, end-users may require a more fundamental understanding of the system's use. In contrast, system administrators may need to know more in-depth information regarding the technical aspects of the application. Therefore, we can differentiate between two types of documents: end-user documentation (system use), which focuses on how to use the system, and system administrator documentation, which covers more complex technical details and functions.

End-user documents cover the basic functionality of the application and ways in which the user's problem can be solved. The main types of end-user documents are quick start guides, the complete manual, and the troubleshooting manual.

Many users find **quick start guides** the most helpful document as they cover most of the functionality in an easy-to-understand format without diving deep into complete manuals. These guides typically provide clear and concise explanations that make it simpler for users to get started with the product.

The complete manual contains thoroughly described software functionalities the user may need. It includes software and hardware requirements and a detailed description of features, tips, and tricks.

The troubleshooting manual contains the most common issues the user might have with the application. It helps users solve their problems quickly without contacting the application's support team.

System administrators' (SA) documentation contains information on system maintenance after deployment to the production environment. Functional descriptions and system admin guides are the most common SA documents.

The functional description provides information on the product's functionality from the system admin's angle.

The system admin guide explains different behaviors of the product depending on the stage or environment. Moreover, SA documents provide information on how to solve malfunctions.

2.1.3 Code Documentation

Comments: Comments are textual annotations added directly within the code to provide explanations, clarifications, or context about specific code sections. Special symbols or keywords typically precede them to distinguish them from executable code.

Documentation Strings (Docstrings): Docstrings are special comments or strings embedded within functions, classes, or modules to describe their purpose, parameters, return values, and usage. They are often used in languages like Python to generate documentation automatically.

Function and Method Signatures: Clear and descriptive function and method signatures, including parameter names and types, return types, and exceptions raised, help users understand how to use these functions correctly.

Code Structure Documentation: Overview documentation describing the overall architecture, design patterns, and modules used in the codebase can provide valuable context for developers navigating the codebase.

API Documentation: Documentation for application programming interfaces describes how to interact with external libraries, frameworks, or services, including available endpoints, request parameters, response formats, and error handling.

3 Standards/norms for SDLC documentation

Within large enterprises, where complex projects are shared, ensuring compliance with SDLC documentation becomes paramount. Standards and norms help to be effective and deliver high-quality software.

The International Organization Standardization (ISO) and the Institute of Electrical and Electronics Engineers (IEEE) are two of the most known international standards. ISO helps enterprises improve their processes by offering norms and procedures to follow.

They provide guidelines and frameworks for writing, organizing, and maintaining documentation throughout software development and after deployment. These standards, such as ISO/IEC 12207, IEEE 730, and ISO/IEC 15288, define requirements for documentation formats, content, and management practices at each stage of the SDLC. They establish a common language and set expectations for documentation quality, consistency, and compliance across projects and organizations.

Large enterprises often develop internal standards from external international standards and norms but are adapted to specific companies' regulations, organizational structure, processes, and methodologies.

In addition to international standards, businesses that work in regulatory spheres must follow regulatory compliance standards such as the FDA in medical device software or GDPR for data privacy.

The most popular and vital standards are ISO/IEC 12207, IEEE 730, ISO/IEC 15288, Capability Maturity Model Integration (CMMI), and Regulatory Compliance Standards.

3.1 ISO 12207

Firstly, introduced in 1995, ISO 12207 is an international standard for SDLC processes. It aims to be the primary standard for the development and maintenance of software.

The newest version was published in 2017.

ISO 12207 introduces several processes for managing the software development life cycle. This standard does not choose or describe any specific software development model, method, or technique. Instead, it provides processes to be followed for the successful development and deployment of applications.

A series of actions that convert inputs into outputs is known as a process. Different stages can use the same process. The standard defines four main groups:

- Agreement.
- Organizational project-enabling.
- Technical management.
- Technical processes.

Projects use only some processes covered in this standard, so teams usually select and declare suitable processes to base software development.

ISO 15288 is also an applicable standard for SDLC, but it has a broader scope, including hardware components. As my bachelor's thesis is focused on software development, this standard needs to be revised.

3.1.1 Agreement Processes

This text discusses ISO/IEC/IEEE 12207:2017, which includes processes related to acquiring and supplying. These processes involve reaching an agreement between a supplier and an acquirer. The acquisition phase encompasses all activities that initiate a project and can be broken down into various chronologically completed tasks and deliverables. The supply phase involves the development of a project management plan, which details milestones that must be achieved throughout the project.

3.1.2 Organizational Project-enabling Processes

The following processes are described in detail: life cycle model management, infrastructure management, portfolio management, human resource management, quality management, and knowledge management. These processes are intended to enable, manage, and support the system life cycle and related projects for businesses or organizations. Life cycle model management ensures that acquisition and supply efforts are adequately supported, while infrastructure and portfolio management support business and project-specific initiatives throughout the entire system life cycle. The remaining processes ensure the required resources and quality controls are in place to support the organization's project and system endeavors. If an organization lacks an appropriate set of organizational processes, a project executed by the organization may apply those processes directly to the project instead.

3.1.3 Technical Processes

This standard includes 14 different technical processes:

- Business or mission analysis.
- Stakeholder needs and requirements definition.
- Systems/Software requirements definition.
- Architecture definition.

- Design Definition.
- System analysis.
- Implementation.
- Integration.
- Verification.
- Transition.
- Validation.
- Operation.
- Maintenance.
- Disposal.

Various technical activities and personnel (such as information technology professionals, troubleshooters, and software specialists) are involved in the pre-, post-, and during-operation processes. The analysis and definition processes determine how software and projects are implemented. Integration, verification, transition, and validation processes are essential steps that ensure the quality and readiness of the software or project. The operation and maintenance phases occur simultaneously; during the operation phase, users receive assistance while working with the implemented software product, and during the maintenance phase, the product is kept up and running through maintenance tasks. Lastly, the disposal process involves retiring and cleaning up the system or project.

3.1.4 Technical Management Processes

This part describes eight unique processes. Project planning, decision management, configuration management, quality assurance, measurement, information management, risk management, project assessment, and control.

Planning, assessment, control of software, and ensuring quality assurance are the processes that this part provides.

3.2 IEEE 730

This standard establishes requirements for initiating, planning, controlling, and executing a software development or maintenance project's Software Quality Assurance (SQA) processes. (*IEEE Standard for Software Quality Assurance Processes*, 2014).

The activities described in this standard are intended to enable the software project to use the SQA processes to produce and collect evidence that forms the basis for giving a justified statement of confidence that the software product conforms to its established requirements. The purpose of this standard is to provide uniform, minimum acceptable requirements for SQA processes in support of a software project (*IEEE Standard for Software Quality Assurance Processes*, 2014).

This standard is applicable for software products as a part of a system and standalone projects. It covers different activities to ensure the quality of the software. IEEE 730 covers 16 activities divided into three groups: SQA process implementation, product assurance, and process assurance.

Subclause	Title
5.3	SQA process implementation activities
5.3.1	Establish the SQA processes
5.3.2	Coordinate with related software processes
5.3.3	Document SQA planning
5.3.4	Execute the SQA Plan
5.3.5	Manage SQA records
5.3.6	Evaluate organizational independence and objectivity
5.4	Product assurance activities
5.4.2	Evaluate plans for conformance to contracts, standards, and regulations
5.4.3	Evaluate product for conformance to established requirements
5.4.4	Evaluate product for acceptability

Figure 3.1 Activities to ensure the quality of the software (*IEEE Standard for Software Quality Assurance Processes*, 2014)

Subclause	Title
5.4.5	Evaluate product life cycle support for conformance
5.4.6	Measure products
5.5	Process assurance activities
5.5.2	Evaluate life cycle processes and plans for conformance
5.5.3	Evaluate environments for conformance
5.5.4	Evaluate subcontractor processes for conformance
5.5.5	Measure processes
5.5.6	Assess staff skill and knowledge

Figure 3.2 Activities to ensure the quality of the software (continued) (*IEEE Standard for Software Quality Assurance Processes*, 2014)

3.3 ISO 27001

The Worldwide standard for managing the security of information is ISO 27001. This standard has been updated two times after publication. The last update was in the year 2022.

ISO/IEC 27001 places great importance on identifying and evaluating potential information security risks. Risk management practices must be implemented by organizations to identify and assess potential threats, as well as devise appropriate mitigation strategies.

The most recent edition of the ISO/IEC 27001:2022 standard includes a thorough set of security controls divided into four domains. These controls address various facets of information security, such as access control, cryptography, physical security, and incident management.

ISO/IEC 27001 encourages a culture of continuous improvement in information security practices. Regular monitoring, performance evaluation, and periodic reviews help organizations adapt to changing threats and improve the effectiveness of their system.

Companies can obtain the ISO 27001 certification. A company must ensure that the organization meets specific quality management standards. This certification is precious for businesses as it helps them build trust and credibility with their customers and stakeholders. It also ensures that the company complies with regulatory and legal requirements and is better prepared to prevent and handle incidents.

The ISO 27001 certification process thoroughly reviews the company's quality management system, including its policies, procedures, and processes. The certification body will evaluate the organization's ability to consistently provide products and services that meet customer and regulatory requirements and its commitment to continuous improvement.

3.4 ISO 9000 series

ISO 9000s are a set of five global standards for quality management.

ISO 9000 introduces QMS fundamentals, including the seven quality management principles underlying the family of standards.

ISO 9001 outlines the specific requirements that organizations must satisfy to achieve the standard. It is the most used quality management standard all over the world.

ISO 9002 offers a quality assurance model for production and installation.

ISO 9003 details the quality assurance requirements for final inspection and testing.

Lastly, ISO 9004 guides on achieving sustained organizational success.

The series related to ISO 9000 is founded on seven Quality Management Principles (QMP):

1. Customer focus: All enterprises must understand actual and future needs, understand customer requirements, and fulfill expectations as a business depends on customers.
2. Leadership: Leaders establish the purpose and direction of the enterprise for all the workers. They should create an effective environment where people can fully commit to achieving the company's objectives and fulfilling their ambitions.
3. Engagement of people: Leaders must find a way to use people effectively to achieve the organization's prosperity.
4. Process approach: Managing activities and related resources as a process can lead to better results.
5. Improvement: Improvement of the performance of the whole organization must be a permanent aim to reach.
6. Evidence-based decision making: First, analyze data and information and make effective decisions from received data.
7. Relationship management: The success of an organization and its external providers are interconnected, and a mutually advantageous relationship boosts the potential of both to generate value.

3.5 Regulatory Compliance Standards

Regulatory compliance standards in the Software Development Life Cycle (SDLC) are governed by various standards and regulations depending on the industry, geographic location, and nature of the software application being developed. Essential standards and regulations related to regulatory activities in SDLC:

The U.S. Food and Drug Administration (FDA) oversees the regulation of software used in medical devices. To comply with FDA regulations, software developers must adhere to 21 CFR Part 820 (Quality System Regulation) and 21 CFR Part 11 (Electronic Records; Electronic Signatures). These regulations mandate that the software development process, including validation, verification, and risk management activities, must be documented. The standard requiring documentation of the software development process and deployment activities is included in the mentioned regulations.

The General Data Protection Regulation (GDPR) is a set of rules in the European Union (EU) that safeguard personal data. Companies that develop software applications that handle the personal data of EU citizens must adhere to GDPR, which entails designing and defaulting to data protection measures, minimizing data, and ensuring the security and privacy of personal data.

The Payment Card Industry Data Security Standard (PCI DSS) is a worldwide regulation that oversees the safety of payment card information. To adhere to PCI DSS obligations, software application developers handling payment card data must implement secure coding techniques, conduct routine security evaluations, and maintain records of security controls.

SOX, or the Sarbanes-Oxley Act, is a United States law governing financial reporting and corporate governance. To comply with SOX requirements, software developers working on financial systems are required to maintain financial reporting systems that are accurate and reliable, implement internal controls, and ensure the security and integrity of economic data.

Organizations are required to meet specific standards to ensure compliance. Meeting these standards involves implementing strong security measures, conducting risk assessments, documenting security policies and procedures, and demonstrating adherence to regulatory requirements. If an organization fails to comply with these standards, it could face legal penalties, financial losses, damage to its reputation, and loss of customer trust.

3.6 Documentation support following ISO and IEEE standards

SDLC documentation is essential to follow ISO and IEEE standards effectively.

Adherence to standards: Enterprises can ensure compliance with ISO and IEEE norms and standards by maintaining quality documentation. This documentation provides a clear framework for implementing required processes and activities. Teams can understand and follow the prescribed standards throughout the SDLC using well-documented procedures, templates, and guidelines.

Compliance verification: Documentation provides proof of adherence to ISO and IEEE standards during audits and assessments. Auditors can validate that processes, activities, and outputs align with the standards' requirements by examining the documentation during reviews. An enterprise's compliance efforts are bolstered by extensive and well-maintained documentation, which improves its credibility.

Standardization: Quality documentation promotes consistency and standardization in SDLC practices across the enterprise. Standardized templates, formats, and naming conventions for documents ensure uniformity and clarity, making it easier to interpret and apply ISO and IEEE standards consistently across projects and teams.

Risk management: Documenting risk assessments, mitigation strategies, and contingency plans is crucial in effective risk management. By doing so, enterprises can identify, evaluate, and address risks associated with non-compliance with ISO and IEEE standards. This approach helps mitigate potential risks and ensures compliance objectives are achieved.

Continuous improvement: By documenting processes for performance measurement, feedback collection, and process improvement, enterprises can continuously evaluate and

enhance their SDLC practices to align with evolving standards and industry best practices. This ongoing evaluation and enhancement facilitate the improvement of SDLC practices.

In summary, documentation is vital in following different norms and standards in the Software Development Life Cycle. Documentation will provide the enterprise with a firm basis to comply with and follow standards.

4 Challenges in documentation SDLC management

Integrating Software Development Life Cycle (SDLC) documentation in large enterprises can pose challenges in ensuring proper adoption and smooth flow of documents across the organization. This is mainly because large enterprises typically have complex and multi-layered systems, processes, and hierarchies, making managing and distributing SDLC documents difficult. Adopting SDLC documentation is crucial for ensuring that software development projects are executed efficiently, meeting all relevant regulatory and compliance requirements, and achieving the desired outcomes. However, achieving this requires a well-planned and coordinated approach, including appropriate tools, processes, and training to ensure all stakeholders are aligned and working towards a common goal.

4.1 Identification of typical challenges faced by large enterprises in managing SDLC documentation

These challenges contribute to documentation-related risks and compliance issues within organizations. After the literature review, here are the main challenges that organizations face during SDLC documentation management:

Consistency and Standardization: Maintaining consistency and standardization in document formats, templates, and content can be difficult when different teams use varying tools and practices, resulting in consistency across various projects and misunderstandings (Gupta et al., 2015).

Documentation Quality and Accuracy: Maintaining accurate documentation that reflects the software architecture, requirements, design, and test plans is challenging due to the scale and complexity of projects (Theunissen et al., 2022).

Training and Awareness: Ensuring all parties know documentation procedures and their significance can be challenging. Provide sufficient training or awareness to ensure smooth documentation practices among different teams.

Maintaining Documentation Relevance: Ensuring that SDLC documentation remains current and accurate throughout the project lifecycle is essential. However, this can be difficult in fast-paced environments where changes occur frequently and must be reflected in the documentation (Theunissen et al., 2022).

Security and Confidentiality: It is of the utmost importance to guarantee the safety and privacy of SDLC documentation, particularly in sectors such as healthcare and finance. Extensive corporations must establish strong access controls and encryption mechanisms to safeguard sensitive information (Fung et al., 2003).

Compliance and Regulatory Requirements: Operating in regulated industries with strict compliance requirements, such as healthcare and finance, is typical for large enterprises. One major challenge is ensuring that their SDLC documentation meets these standards and is ready for audit (Gupta et al., 2015).

Documentation Overload: Large corporations handle many projects concurrently, resulting in significant paperwork. This quantity can be daunting, making it challenging to manage, revise, and sustain all SDLC documentation efficiently (Imoh, 2023).

Documents Digitalization: Many companies still rely on paper documents for their day-to-day operations. However, this can pose a significant challenge when maintaining accurate records for audit purposes. Transferring paper documentation to digital form can be time-consuming and have errors. It requires specialized equipment, software, and trained personnel to ensure the process is carried out correctly.

5 Strategies for ensuring compliance with documentation

Documentation ensures a smooth and efficient development process in the Software Development Life Cycle (SDLC). However, creating and maintaining documentation can be challenging, especially when ensuring consistency and quality across different phases of the SDLC. To tackle these challenges, organizations can adopt various strategies and best practices. These include establishing clear documentation standards, using templates and tools to streamline documentation processes, involving key stakeholders in the documentation process, and conducting regular reviews and updates to ensure that documentation remains relevant and current. By implementing these strategies, organizations can improve the quality and accuracy of their SDLC documentation, minimize errors and issues, and ultimately deliver high-quality software products.

5.1 Best practices for effective documentation management processes

The most crucial thing is to establish documentation standards. Define clear and comprehensive documentation standards, including templates, formats, and guidelines for different types of documents. Ensure alignment with industry best practices, regulatory requirements, and organizational needs. For example, in alignment with FDA regulations, if the enterprise is a part of the pharmaceutical industry, make documentation follow Good Documentation Practices (GDP). Follow ISO 9000 series and ISO 12207 for every sector or project.

Hereafter we extract best practices from the above methodological standards for SDLC documentation:

Identify Document Owners and Establish Document Review: While developing the product**,** identify the document owners responsible for writing, maintaining, and updating the document. Attach Product Owner and Technical Unit for Quality control of written information and approval in the official repository if applicable.

Use Version Control: Implementing version control systems such as Confluence or Git can be incredibly useful for managing software development projects. These systems allow you to keep track of past versions of your document and make it easy to roll back to a previous version in case of errors or changes. With version control, you can easily see who made what changes, when, and why, making it much easier to collaborate with other colleagues on a project. By doing so, you can ensure that your document is up to date and has correct information.

Ensure Accessibility and Security: It must be kept in secure repositories with appropriate access controls when storing necessary documentation. Access controls should

be based on roles and responsibilities to ensure that only authorized personnel can access the information. Additionally, implementing encryption and backups is crucial to protect documentation from unauthorized access, loss, or corruption. Encryption can help to prevent data breaches and protect sensitive information. In contrast, backups can help to ensure that data is not lost in the event of a system failure or other unexpected event. It is essential to regularly review and update security measures to ensure they remain practical and current.

Training and Support: Team members should be provided with training sessions and resources to educate them on documentation standards, tools, and best practices. Ongoing support and guidance should also be provided to ensure adherence to documentation management processes.

Regular Audit Documentation Processes: Regular audits of documentation processes and practices are recommended to identify areas for improvement. This can help ensure that documentation is accurate, up-to-date, and consistent. The audits can also help identify gaps or inefficiencies in the documentation process that may impact productivity or quality. It is also essential to gather feedback from stakeholders and users to assess the effectiveness and usability of documentation. This can help ensure that the documentation meets its intended audience's needs and can be used to improve future documentation efforts.

Document Retention and Disposal Policies: Create guidelines for retaining documents that comply with regulatory requirements and the organization's needs.

Develop protocols for securely archiving and disposing of outdated documents while complying with relevant regulations.

By implementing these best practices, organizations can improve documentation management processes, enhance collaboration among project stakeholders, and ensure the delivery of high-quality software products that meet regulatory standards and customer expectations. Consistent documentation practices are foundational to successful project outcomes and organizational efficiency.

5.2 Tools for ensuring documentation accuracy and consistency

1. Document Management Systems (DMS):

Microsoft SharePoint: Offers robust document management capabilities with version control, access controls, and workflow automation.

Documentum: A popular enterprise content management system that supports document lifecycle management, compliance, and records retention.

2. Version Control Systems:

Git: A distributed version control system widely used for tracking changes in source code and documentation. Git repositories can be used to manage document versions.

3. Requirements Management Tools:

IBM Engineering Requirements Management DOORS: A requirements management tool that helps define, capture, and manage requirements with traceability and compliance features.

Jama Connect: Enables teams to author, review, and manage requirements in a collaborative environment while ensuring compliance with standards.

4. Quality Management Tools:

HP ALM (Application Lifecycle Management): Provides capabilities for test management, defect tracking, and quality assurance with compliance reporting features.

JIRA Software: Offers agile project management and issue-tracking capabilities that can be customized to support compliance requirements.

5. Compliance and Audit Tools:

MetricStream: Provides governance, risk, and compliance (GRC) solutions with audit management features for assessing and ensuring compliance with SDLC processes and documentation.

Archer: Offers a comprehensive platform for managing enterprise governance, risk management, and compliance activities.

6. Electronic Document Signing and Approval Tools:

DocuSign: Facilitates electronic document signing and approvals with audit trails to ensure compliance with approval processes.

Adobe Sign: Integrates with document management systems to enable secure electronic signatures and approvals for compliance-related documents.

7. Workflow Automation Tools:

Nintex: Enables organizations to automate document approval workflows and compliance processes, ensuring consistency and adherence to regulatory requirements.

K2: Provides process automation and workflow management capabilities to streamline document management processes while ensuring compliance and traceability.

A few basic programs are commonly used to effectively manage software development life cycle (SDLC) documentation. However, it is worth noting that larger enterprises often develop software solutions that integrate seamlessly with all other internal applications. This customized software typically has security features that meet the company's standards. By using these proprietary tools, organizations can streamline their SDLC processes and ensure that all documentation is professionally managed and secured.

6 Qualitative Research

Effective risk management is essential for project success and compliance in software development. Documentation is critical in capturing requirements, design decisions, and testing procedures. However, inadequate documentation can pose significant risks to project outcomes. The aim is to identify and address risks associated with varying levels and qualities of documentation throughout the SDLC process.

The following sections will detail the introduction to the aim of the thesis, information on the research methodology, type of qualitative research, respondents, the process of coding and categorizing data, and interviews. Results based on qualitative research are presented, and a risk matrix is identified. At the end of the practical part, a Mitigation Plan with the most significant risks and their solutions is offered, and a conclusion is drawn.

6.1 Aim of the Qualitative Research

The central objective is to comprehensively investigate the risks associated with software development life cycle documentation within larger enterprises and provide a Mitigation Plan with proposed solutions.

Research methodology involves a thorough assessment and analysis of the identified risks, including their potential impact and likelihood of occurrence. A plan will be developed to address these identified risks and enhance compliance with SDLC documentation standards.

Ultimately, the goal of this study is to provide actionable recommendations that can be utilized to improve SDLC documentation practices. By doing so, it might be possible to minimize the risks and negative consequences associated with inadequate documentation management during the SDLC process.

6.2 Research Methodology

The **COBIT 5** framework will be applied for research. From this framework, the most suitable processes have been chosen. **APO12 (Align, Plan, and Organize)** process called **Manage Risk** will be used to define objectives and identify, analyze, and address risks. It is designed to establish a systematic approach for identifying, assessing, prioritizing, and managing risks that could impact an organization's ability to achieve its objectives.

Risk identification and data collection: conducted through qualitative research. The primary method of data collection is interviews. One-to-one semi-structured interviews with ten professionals working in different parts of the Software Development Life Cycle Chain. All interviews were conducted via the Microsoft Teams platform with interview transcription.

Data analysis: After collecting data, the transcript analysis method was used to understand better and create keywords and patterns for analysis. It uses the tool MAXQDA. In this software, codes are assigned to text during its analysis. Following the analysis method of the occurrence of codes on their basis, the risks are defined after analysis in MAXQDA software pattern interpretation and extraction of risks.

Risk Assessment: A risk matrix is developed based on the results of the transcript analysis method, which identifies risks according to their potential impact and likelihood.

Risk Response Planning: Based on this matrix, a Mitigation Plan contains specific actions and strategies to address and reduce the identified risks related to SDLC documentation.

Conclusions and discussions will be drawn at the end of the practical part.

6.2.1 Respondents

Qualitative research was used for the bachelor thesis. The answers to questions based on the aim of the thesis were conducted through one-to-one semi-structured interviews with respondents.

The bachelor thesis author in the role of interviewer and respondents in the role of interviewee conducted all interviews. All interviews were completed via the Microsoft Teams platform to get a transcription of every interview.

A total of 10 interviews were conducted. Respondents have been chosen following certain conditions for the position and qualification of the person. Age and gender were not of any importance when selecting respondents.

The main conditions are:

- The person works in a large enterprise (at least 250 persons employed).
- The person represents one of the positions working directly with the Software Development Lifecycle.
- The person has at least five years of experience.

All interviews were conducted in agreement with both sides based on anonymity. Due to the non-disclosure agreement, the names of the respondents and company names are not mentioned in this thesis. Also, transcripts are not published due to the existence of proprietary information. At the beginning of the interview, respondents are asked which part of SDLC they are representing (project driving, development, testing, support) and the sphere of their company (banking, governance solutions, pharmaceutical industry) to understand if any regulation laws exist.

During the interview, the interviewee could stop the interview, and this transcript would never be used if the respondent did not want to answer any of the questions for any reason; this right was reserved for him.

A table presenting basic information on respondents is provided to aid in understanding their field of work and enterprise sphere.

Table 6.1. Basic information on interview respondents (Source: Author)

Interview	Position	Company Sphere
1	Project Manager	Pharmaceutical
2	Senior Automation Quality Assurance Engineer	Software Solutions
3	Back-End Developer	Banking
4	Business Analyst	Audit
5	Front-End Developer	Banking
6	Full Stack Developer	Banking
7	Head of IT and Compliance	AI Solutions
8	Angular Developer	Banking
9	Full Stack Developer	Solutions Development
10	Software Development Engineer	Payroll Development and HR Solutions

6.2.2 Interview Questions

Questions for the interview were thoroughly chosen based on the theoretical part and aim of the qualitative research. As a semi-structured interview, some questions might not appear in this chapter if the author wants a deeper understanding of some parts from particular respondents. All of the questions were set out to correlate with the aim of the research and to be comparable with the answers of different respondents.

Questions have been divided into three main categories: Software Development Lifecycle, documentation management, and risks associated with SDLC Documentation. These categories represent broad topics into which the interviewer and interviewee will get a deep dive.

Sub-questions are prepared to cover the main points of every category.

Here is the list of the questions used in the interview:

Basic info:

- What part of SDLC are you working in? Your position and sphere of company.

C1: Software Development Life Cycle

- Which model of software development are you using? Waterfall or Agile development?
- Can you describe the typical software development life cycle (SDLC) followed in your organization?

C2: Documentation Management

- Describe the situation of your questions regarding SDLC and documentation management; what will you do?
- Who are the key stakeholders involved in SDLC documentation, and what are their roles and responsibilities? Do you have a Product Owner, Technical Unit, Scrum Master, and Technical Writer?
- Imagine you need to change RS; how will you do that?
- Have you ever had a problem with finding old requirements?
- What challenges or issues have you encountered during requirements gathering, and how are these addressed?
- When doing the application release, do you have a specific plan for release where the scope, aim, and basic description of deliverables are documented?
- What processes or practices does your organization follow to ensure code quality and readability?
- Have you ever found information that is not up to date in the documentation? How did you deal with it?
- Do you have any System Admin documentation and system use (guides, videos of how to use, manuals)? What will you do if you need to understand how the software works?

C3: Risks associated with SDLC Documentation

- In your experience, what are the common risks associated with SDLC documentation?
- Have you ever been asked how to make some things in applications that you have developed?
- Has it ever happened that when one developer left the company, there were problems understanding his code?
- Has it ever happened that you could not find the document you are looking for?

6.3 MAXQDA Analysis

Upon concluding the interview process, all transcribed conversations were reviewed, edited, and formatted for analysis. These transcriptions were then imported into the MAXQDA software tool, a specialized platform that facilitates qualitative data processing.

Figure 6.1 List of transcribed interviews for qualitative research (Source: Author)

The primary objective of this exercise was to thoroughly evaluate all of the conversations to identify any potential risks or concerns raised during the interviews. This enabled me to learn the data in depth and to extract any noteworthy insights that could be beneficial in formulating informed decisions and strategies.

6.3.1 Coding of Data

Data was analyzed after carefully reviewing the interviews, and patterns indicating potential risks were identified. This information was used to create a set of codes that would allow us to detect and mitigate these risks more effectively. Following these codes and patterns, we can detect the most common interview risks.

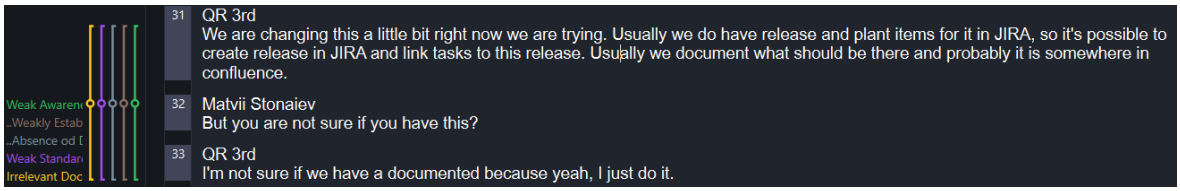

Figure 6.2 Example of codes used in the transcribed interviews (Source: Author)

Based on the theoretical part of the thesis, the following risk patterns have been identified and require attention:

- Awareness of SDLC Documentation Processes: Ensuring all stakeholders know the Software Development Life Cycle (SDLC) documentation processes is essential. This includes creating, reviewing, and approving documents at each phase of the SDLC.
- Problems with Documentation: Ensuring the documentation created is accurate, complete, and up-to-date is essential. Problems with documentation can lead to confusion, delays, and errors.

- Security of Documentation: The security of documentation is critical, as sensitive information may be contained within. It is essential to ensure that access to documentation is restricted to authorized personnel and that proper security measures are in place to protect against unauthorized access or theft.
- Standardization: Standardization of documentation is essential to ensure consistency and clarity across all documents. This includes the use of standard templates, formats, and terminology.
- Documents Overload: It is essential to avoid document overload, as it can lead to confusion and delays. It is vital to ensure that only necessary documentation is created and organized in a manner that is easy to navigate and understand.

From these patterns, codes, and sub-codes have been identified:

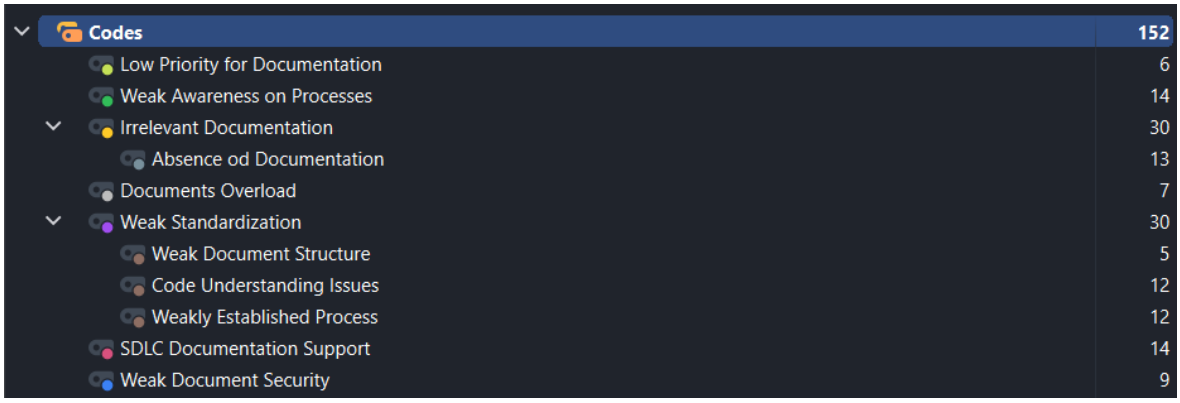

Figure 6.3 Codes identified in MAXQDA tool (Source: Author)

Overall, seven main codes and four sub-codes were identified using the MAXQDA tool. Every code lights up the risks that might appear during SDLC. Code was used when specific patterns appeared in the transcribed interviews.

A low Priority for Documentation code was used when respondents highlighted in their answers that SDLC documentation is not the priority for their team or, for example, they do not find high importance in maintaining quality documentation.

Weak Awareness of Processes highlights low knowledge of the company's SDLC or documentation management processes. This code is applied when a person or company's personnel are unaware of some basic information concerning software development life cycle and documentation or are unfamiliar with documentation locations throughout the company.

Irrelevant Documentation code is one of the most used during transcription analysis. It shows the problem with not up-to-date documentation, irrelevant information, or insufficient information in the specific document. Sub-code Absence of Documentation described when the interviewees' company abandoned using the release plan, requirements specification, etc. documentation.

Documents Overload highlighted the problem of a documentation management structure that is too complicated, where respondents cannot find the specific document they are looking for.

A Weak Standardization code was used if the SDLC process in the company was not established well enough. This code is applied if the interviewee lacks documentation, standards are blurred, documents do not have specific templates, and are poorly structured. It also has three sub-codes: Weak Document Structure, Code Understanding Issues, and Weakly Established Process. Each of them stands for problems with standards and norms in the company.

SDLC Documentation Support code draws attention to weak support for workers if they have any questions regarding documentation management or SDLC. For example, Suppose there is no special consultation team throughout the company. In that case, teams use the SCRUM methodology without having a SCRUM master, and technical writer assistance is unavailable during the documentation writing process, so this code is applicable.

Weak Document Security code highlights the problem of documentation security throughout the large enterprise. For instance, this code was used when the interviewee's company did not have any Database management system/ official repository to store the documentation securely.

7 Results

The research results can be presented after extracting the codes from the transcription interviews using the MAXQDA tool. MAXQDA is equipped with features that allow for the creation of various statistical data, such as the frequency of code usage and their distribution. This data will help identify and define the most common risks associated with software development life cycle (SDLC) and documentation management. To present the data in a more accessible format, Figure 7.1 has been created, which represents the percentage and quantity of usage of each code.

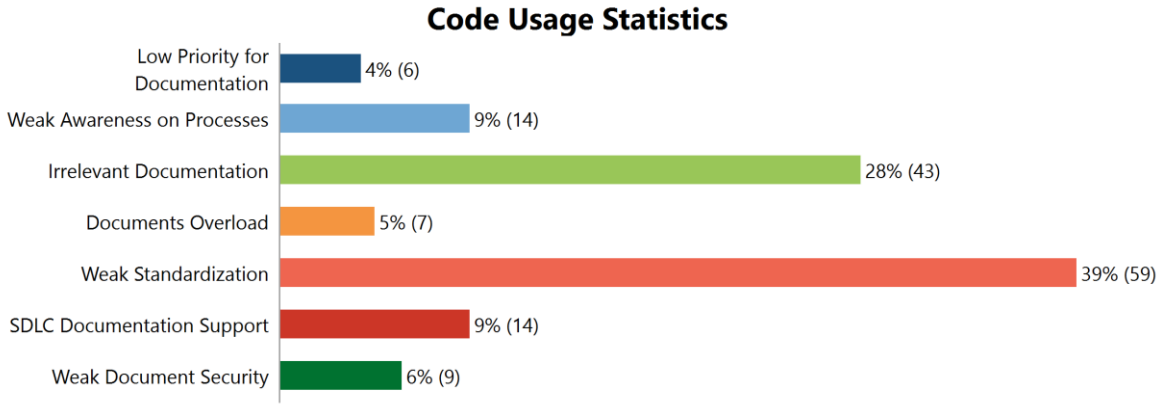

Figure 7.1 Code usage statistics extracted from MAXQDA tool (Source: Author)

The most common codes that appear are the most common risks associated with SDLC and document management. The five top-used codes are:

- Weak Standardization: This code appeared at least once in every transcript and was used 59 times. Three sub codes were identified:
    - Code Understanding Issues.
    - Weakly Established Processes.
    - Weak Document Structure.
- Irrelevant Documentation: This code was used 43 times. One sub code was identified:
    - Absence of Documentation.
- Weak Awareness of Processes: This code was used 14 times.
- SDLC Documentation Support: This code was used precisely 14 times.
- Weak Document Security: This code was used 9 times.

Sub-codes are the codes that have been used under the main codes. Sub-code for Irrelevant Documentation is Absence of Documentation. This code was used only if document was absent in the organization. In all other cases just code Irrelevant Documentation was used. That's why sub code Absence of Documentation has only 13 codes but the primar code has 43 uses.

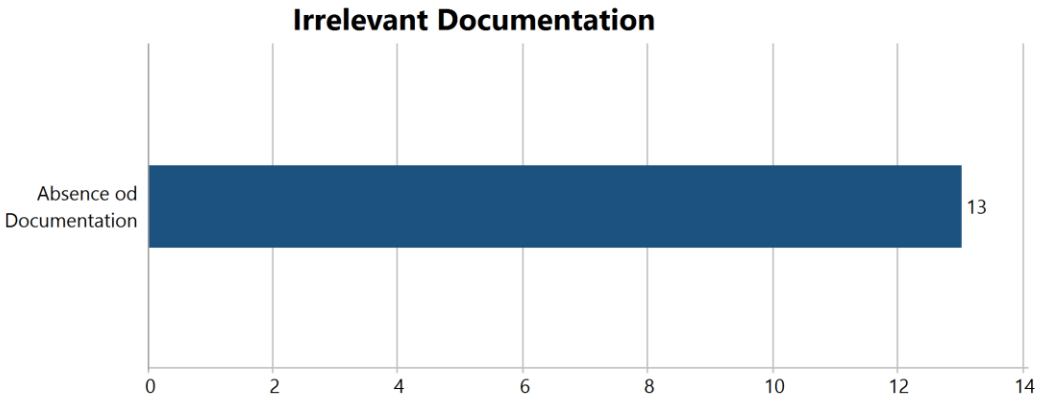

Figure 7.2 Sub-codes usage for Irrelevant Documentation code (Source: Author)

For Weak Standardization, three sub-codes were identified, as Weak Standardization is an extensive code. The most frequent risks in standardization are Code Understanding Issues and Weakly Established Processes; both these sub-codes were used 12 times.

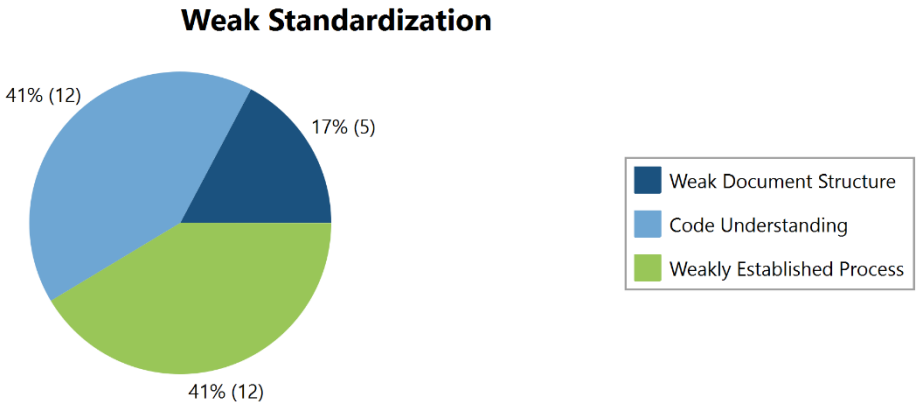

Figure 7.3 Sub-codes usage for Weak Standardization code (Source: Author)

7.1 Risk Matrix

A risk matrix enables the systematic assessment and prioritization of risks based on their likelihood and potential impact. It provides a visual representation that allows stakeholders to understand risks and allocate resources accordingly quickly. Essentially, a risk matrix allows organizations to make informed decisions about which risks to address first and how to mitigate them effectively. After building a risk matrix, a Mitigation Plan will be completed based on it.

The risk Matrix is built on the data analyzed in the qualitative research. The author received the information from one-to-one semi-structured interviews and analyzed it in the MAXQDA tool. Each risk will be given its probability and impact on a scale from one to five.

Probability is divided into five categories (Sutherland et al., 2021):

- Rare – can happen on a rare occasion. **Value: 1.**
- Unlikely – quite possible to happen. **Value: 2.**

- Moderate – possible to happen. **Value: 3.**
- Likely – very likely to happen. **Value: 4.**
- Almost Certain – sure to happen. **Value: 5.**

Impact is also divided into five categories (Sutherland et al., 2021):

- Limited – will not cause serious issues. **Value: 1.**
- Minor – might cause a little trouble. **Value: 2.**
- Moderate – can cause moderate issues. **Value: 3.**
- Significant – can cause major issues in the system functions. **Value: 4.**
- Catastrophic – can result in fatality. **Value: 5.**

The risk level can be counted after defining each risk's probability and impact. Risk level counts as the multiplication of probability and impact. For example, if the probability is four and the impact is three, the risk level will be 12.

Risk level has four categories depending on the value of the risk. Digits and names for the risk level are taken from proposed risk matrix by the article of Guevara, Patricia:

- **1-4**: Acceptable – maintenance measures, no actions need to be taken (Guevara, 2022).
- **5-9**: Adequate – risk has to be analyzed deeply (Guevara, 2022).
- **10-16**: Tolerable – must be analyzed and improvements suggested (Guevara, 2022).
- **17-25**: Unacceptable – must be treated as an urgent risk to the organization's process flow (Guevara, 2022).

The first risk identified with the MAXQDA tool is **Low Priority for Documentation.** This code appeared six times and is used in four percent of all codes. So, the probability level is stated as Unlikely, and the value assigned is 1. Documentation is a source of information on software and strategies used in a team workflow. The impact of this risk is stated as Moderate with a value of 3 because it may cause a weak understanding of the product for external people and even the internal teams.

Weak Awareness of Processes code appeared 14 times during transcription analysis; nine percent of code usage statistics allow assigning a moderate probability and value of 3 for this risk. Lack of understanding of internal SDLC and document management processes might lead to disruption of processes inside the team and incompetent usage of senior employees' teams. So, the impact value is set at three and marked as moderate.

Irrelevant Documentation code is one of the most used during analysis. This code was used 43 times, which makes up twenty-eight percent of code usage. This risk appeared in nine out of ten interviews, so the level of likely with the value of 4 can be assigned. Irrelevant Documentation of its absence might lead to a misunderstanding of the goals and requirements of the project and might lead to consequences. So, the level of moderate and value at three is assigned.

Documents Overload was used seven times in 6 different transcriptions, which makes up four percent of all the codes. Probability is assigned as moderate with a value of 3. The

impact is set as minor with a value of 2 because documentation overload might lead to different kinds of confusion and delays in releases.

Weak Standardization is the most used code, and it has been assigned 59 times in all interviews. So, all interviewees had problems with code documentation, weak SDLC standards, weak template structure for documentation, or all of the above. So, the probability level is almost inevitable with the value of 5. The impact is also high as well because of standardization issues. There are code understanding issues, hard-to-get and hard-to-read documentation, and the SDLC process is a mess that leads to poor product delivery results. The impact is significant, with a value of 4.

SDLC Documentation support defines mainly the risks connected with the absence of trained professional staff, which can help to write, check compliance, consult, and deliver documentation effectively. This code was used only 14 times, with nine percent of all code usage times, but in all interviews, the probability is defined as likely with a value of 4. The impact is set to moderate with a value of 3 because weak SDLC doc support might lead to non-compliant documentation and a bad understanding of processes, leading to poor software delivery results.

Weak document security code was used nine times in seven different interviews. The main problem was the absence of a Database Management System that could store SDLC documentation securely. Probability is set at moderate with a value of 3. The impact is very high as not secure documentation could lead to misuse of documents, leaks of internal proprietary or confidential data that leads to data compromise, or even blackmail. The impact level is set to catastrophic with a value of 5.

A comprehensive risk matrix was created using the Excel tool to evaluate and categorize the risk associated with each identified code.

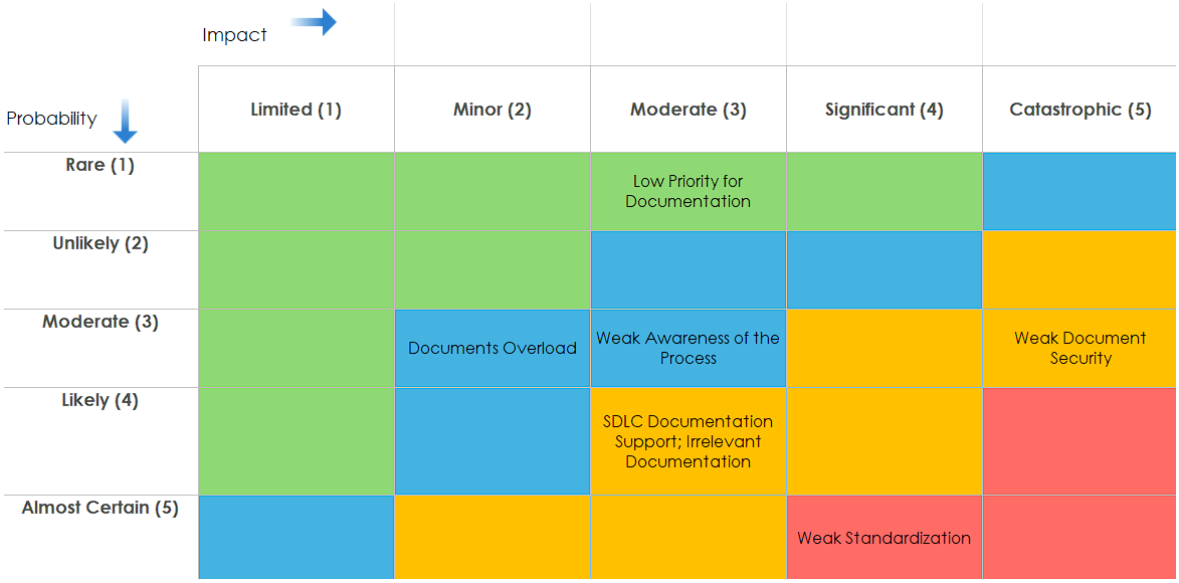

Figure 7.4 Risk Matrix for identified risks (Source: Author)

The risks in green fields are identified as Acceptable. There is only one risk in this category, and it is a **Low priority for Documentation**.

Blue fields apply to an Adequate level of risk. **Document Overload** and **Weak Awareness of the process** fall under this category.

Yellow fields represent a Tolerable level of risk. Most risks fall under this category: **SDLC Documentation Support**, **Irrelevant Documentation**, and **Weak Document Security**.

The most risky category is the Unacceptable level of risk. It is marked as red fields, and **Weak Standardization** risk is situated here.

Table 7.1 Risks probability, impact, and risk level (Source: Author)

Risk	Probability	Impact	Risk Level
Weak Standardization	5	4	**20**
Weak Document Security	3	5	**15**
Irrelevant Documentation	4	3	**12**
SDLC Documentation Support	4	3	**12**
Weak Awareness of the Process	3	3	**9**
Documents Overload	3	2	**6**
Low Priority for Documentation	1	3	**3**

7.2 Risk Mitigation Plan

This chapter introduces the Mitigation Plan, a critical component of the bachelor's thesis aimed at addressing the risks identified through qualitative research and analyzed through MAXQDA and the risk matrix.

The Mitigation Plan serves as a framework designed to navigate the complexities of SDLC documentation management by offering practical strategies to overcome potential risks. It contains industry best practices, standards and norms practices, and our qualitative research findings.

It outlines the plan's key components, including mitigation strategies, responsible parties, and success criteria. This Mitigation Plan does not describe any timelines because predicting real time-consuming for every bullet point is challenging.

This chapter endeavors to equip enterprises with actionable measures to improve their SDLC documentation practices and predict potential risks. By proactively identifying and addressing risks, organizations can improve the quality, reliability, and compliance of their software development life cycle processes.

The Mitigation Plan for **five potential risks with the highest level of risk** will be presented.

7.2.1 Weak Standardization

Weak Standardization can pose several risks that may affect a product's quality and timely delivery. One of the primary risks is the weak establishment of processes, which can lead to inconsistency and confusion in the team's work. This can result in product delivery delays due to a lack of clarity regarding the project scope and requirements.

Another risk is code understanding issues, which can arise when team members are unfamiliar with the code standards used in the project. This can cause misunderstandings and errors in the code, leading to functional issues and further delays in product delivery.

The weak document structure is also a significant risk that can cause problems in product development. When documents are poorly structured, reading the documentation and understanding the proper scope and software functionalities can be challenging.

Overall, Weak Standardization can lead to a poor understanding of the product functions, making it difficult to identify opportunities for improvement. It can also create issues with the product's future maintenance and updates.

Mitigation Strategy for **Weakly Established Process** and **Weak Document Structure**:

- **Analyze**: Conduct a comprehensive analysis of existing processes throughout the organization and compare them to certified ISO standards. Analyse documentation.
- **Identify** areas lacking standardization and identify problems with document structure.
- **Develop**: Develop new or improve existing SDLC policies based on ISO 12207. Develop or improve templates for the documentation
- **Distribute**: Make these policies and documentation templates mandatory for the product development of the whole enterprise.
- **Raise Awareness**: Provide training for employees to raise awareness of the standards and templates applied in the organization.

Responsible Parties:

- SDLC process team and head of IT and Compliance: The main aim is to analyze, develop, and establish a standardized process throughout the company. Additionally, create a standardized template for every document.
- Training Department: Provide quality training for a better understanding of employees.
- Project Managers/ Product Owners: Control compliance with standards and templates in the project team.
- Quality Assurance team: Conduct regular reviews and audits on project teams' SDLC process and documentation management.

Success Criteria:

- Positive feedback from stakeholders on the clarity of documentation and structure of processes.
- New policy for standardization is easy to find, and 100% of employees have gone through specific training and are aware of the current SDLC process and documentation management standards.
- Every template is available and can be downloaded from the repository.
- 100% of teams follow the same global policy.
- 100% of teams use standardized templates for the documentation.

Mitigation Strategy for **Code Understanding Issues:**

- **Analyze**: Analyze the most common issues in understanding code.
- **Create**: Create more strict code reviews. Introduce code documentation to describe functions, best practices, issues, and coding standards.
- **Distribute**: Make these code standards and code documentation mandatory for the product development of the whole enterprise.
- **Raise Awareness**: Provide training for employees to raise awareness of the code standards and code documentation applied in the organization.

Responsible Parties:

- SDLC process team and senior developers: The main aim is to analyze, develop, and establish code standards and documentation.
- Training Department: Provide quality training for a better understanding of developers.
- Technical Unit/ Senior Developer: Control of compliance with standards in the development team.
- Quality Assurance team: Conduct regular reviews and audits on the code quality and compliance with code standards.

Success Criteria:

- 100% of developers document code.
- Best practices and code standards are applied by the developers when writing the code.
- 100% of developers know the code standards and best practices.
- Code standards can be found easily, and code documentation templates are prepared for download by every developer on the website/ repository.

7.2.2 Weak Document Security

Insufficient document security measures can seriously threaten a company's critical aspects, such as sensitive data, confidential information, and intellectual property. Weak

document security may result in unauthorized access, data breaches, theft, or loss of crucial information, leading to negative consequences such as financial losses, reputational damage, legal liabilities, and compliance violations. Therefore, organizations must implement robust document security protocols and procedures to safeguard their digital and physical assets and protect themselves from potential risks and threats.

Mitigation Strategy:

- **Analyze**: Analyze the most common issues connected with document security.
- **Implement** a Document Management System to provide additional security for all types of documentation. Introduce restricted access to SDLC documentation in the official repository based on their position and role. Implement encryption for sensitive, confidential, or proprietary documentation when communicating with stakeholders or other parties.
- **Distribute** DMS software to employees and provide training on how to use the software.
- **Raise Awareness** of common security threats such as phishing or social engineering across the employees.

Responsible Parties:

- SDLC process team and security team: The main aim is to analyze, develop, and establish DMS, restricted access, and encryption of documentation.
- Training Department: Provide quality training for a better understanding of developers.
- Quality Assurance team: Conduct regular reviews and audits on the level of security provided for documentation.

Success Criteria:

- Every employee can only access the documentation the access level gives him.
- Achievement of 100% compliance with encryption standards and access control policies within the organization.
- 100% of employees know the document management system and how to use it properly.

7.2.3 Irrelevant Documentation

The risk of Irrelevant Documentation within the SDLC process is a vital concern for large enterprises. This risk arises due to several causes, including a lack of clarity on documentation requirements and objectives, failure to review and update documentation regularly, and inadequate communication and collaboration between stakeholders.

Having Irrelevant Documentation can waste time and resources on producing and maintaining unnecessary documentation. Failure to review and update documentation regularly can result in outdated or irrelevant information that may lead to confusion or misalignment with project requirements. Finding relevant information in outdated documentation can be difficult, leading to decreased productivity and project delays.

The consequences of Irrelevant Documentation within the SDLC process can be severe. Wasted time and resources on producing and maintaining unnecessary documentation can hurt the project's budget and timeline.

Mitigation Strategy:

- **Establish** templates, Good Documentation Practices, and standards for documentation
- **Conduct regular checks and audits** of the documentation to meet the standards of compliant documentation.

Responsible Parties:

- SDLC process team and head of IT and Compliance: Create a standardized template for every document. Introduce GDP across the organization.
- Technical Writer: Follow the company's standards and good documentation practices.
- Quality Assurance team: Conduct regular reviews and audits of the documentation in project teams.

Success Criteria:

- Majority of documentation is compliant after regular audits.
- Improvement in stakeholder satisfaction and productivity, as evidenced by feedback surveys and project performance metrics.

7.2.4 SDLC Documentation Support

The lack of proper software development life cycle (SDLC) documentation support can significantly impact the overall quality of the documentation. When there is a lack of support, the documentation might be inconsistent, non-compliant, and confusing in scope. This can lead to unclear requirements, inadequate testing, and difficulty maintaining the documentation. As a result, it can compromise the quality, reliability, and credibility of the software product in question. Therefore, it is crucial to ensure adequate SDLC documentation support is in place to ensure the documentation is comprehensive, accurate, and aligned with the project objectives.

Mitigation Strategy:

- **Hire personnel** to help the development team with the documentation. For example, an IT compliance consultant or technical writer. This will give more free time to developers and business analytics while increasing the effectiveness and clarity of written documentation.
- **Create** a consultation space for employees to receive professional support from experts on questions they are interested in.

Responsible Parties:

- Head of IT and Compliance: Hire and create an IT compliance Consultation team or hire technical writers.
- IT compliance Consultation team: Provide support in documentation issues.
- Technical Writer: Provide support in documentation writing for the teams.

Success Criteria:

- The SDLC documentation support team provides effective consultations for employees.
- 100% of teams have a technical writer assigned to their team.
- Documents are compliant with company standards and GDP.

7.2.5 Weak Awareness of the Process

When team members have a weak awareness of the processes involved in software development, it can result in unstructured processes, Irrelevant Documentation, and poor results in delivering software on time and within budget. This lack of understanding can cause confusion and miscommunication, leading to delays and higher costs. All team members must clearly understand the processes involved in software development to ensure that the project is completed efficiently and effectively. This includes understanding the purpose of each step in the process, the roles and responsibilities of team members, and the tools and techniques used to manage the project.

Mitigation Strategy:

- **Simplify and continuously improve** process policy for better understanding.
- **Provide training** on SDLC process and documentation management for product teams
- **Create space** for policy and process description pages to increase accessibility to this information.

Responsible Parties:

- Training Department: Provide quality training to understand developers better and create space for all process documentation, for example, on Confluence.

Success Criteria:

- Information on processes and documentation management is accessible, user-friendly, and thorough.
- Product teams attended necessary training for a better understanding of the process.

8 Discussion

The primary objective of this research study is to comprehensively examine and analyze the different types of risks typically associated with software development life cycle documentation within larger enterprises. By leveraging qualitative research techniques and conducting a comprehensive MAXQDA analysis, the thesis author aimed to identify and understand the most commonly occurring risks in this context. Furthermore, a risk matrix was used to prioritize these risks based on their probability and potential impact on the project outcomes. Finally, a Mitigation Plan outlining strategies and approaches for effectively managing and mitigating these risks was suggested. Overall, this research study seeks to provide valuable insights and guidance to organizations looking to undertake software development projects more efficiently and risk-awarely.

8.1 Key Findings

Key findings of the bachelor thesis are risks identified during the qualitative research, their analysis, and proposed solutions. The author of the thesis works in a large enterprise in the compliance consultation sphere, so it was exciting to find out about issues with SDLC documentation management in other companies.

A total of 7 potential risks have been identified, and statistics on code usage are available in Chapter 7. The most used codes were Weak Standardization, Irrelevant Documentation, and Weak Awareness of Processes.

Risks identified with their risk levels and the following risk matrix can be found in Chapter 7.1 Risk Matrix.

The Weak Standardization issue was chosen as the highest level of risk because all interviewees had issues with the established standardized process, which, from the author's point of view, is the basis for delivering effective solutions. Most teams in these large enterprises follow their own process, leading to weak communication with other teams and messes in the documentation.

For almost every interviewee appeared code understanding issues, as stated in Chapter 7.2.1 Weak Standardization could be solved by documenting code functionalities. That will take more time but will provide the team with a much better understanding of code written by their colleagues. Furthermore, with AI's progressing use, it can also be used for code documentation.

Every interviewee issued Irrelevant Documentation in his projects. Moreover, no one has a technical writer position in their team. As proposed by the author in the Mitigation Plan, Irrelevant Documentation issues can be solved by hiring a technical writer for documentation purposes. These things are very tightly connected. Of course, it will not

completely eliminate non-compliant documentation, but the number of issues will decrease instantly.

The author's experience of weak awareness in his company became a theory confirmed by qualitative research. Eight out of ten interviewees had a weak awareness of some processes. As stated in the Mitigation Plan, this could be solved by simplifying existing policies and providing quality training to raise awareness.

One of the key findings of the thesis is a Mitigation Plan with proposed strategies described in Chapter 7.2. This plan described mitigation strategies, responsible parties, and success criteria for 5 top risks.

Compared with challenges in SDLC documentation from Chapter 4 of the literature review, qualitative research defined similar potential risks that might appear in large enterprises. This proved the relevance of the qualitative research and alignment with literature sources related to SDLC documentation management.

8.2 Limitations

The most significant limitation of the qualitative research was ethical considerations. The basics of confidentiality did not cover some aspects the author was interested in.

Respondents bias is another limitation that could lead to unreliable or incomplete answers because of reluctance to disclose sensitive information.

Since it is a bachelor's thesis, a quality research sample was ten employees of large enterprises, so the information and results might not be referred to as relevant in comparison, for example, with the screening of 100 respondents. So, risk matrix probability and impact were based on quality research conducted, and some data might be changed with more extensive samples.

In Chapter 7.2, Mitigation Plan, although proposing solutions without specific cases, timelines, and information on allocated resources was challenging, the author suggested strategies based on the global case, infinite timeline, and resources. Success criteria were specified as optimistically perfect results, even though every team could apply their definition of success for the short-term or long-term.

8.3 Contribution

This bachelor's thesis is a comprehensive resource for large teams or managers seeking to optimize their Software Development Life Cycle Documentation Management practices globally. By examining prevalent risks inherent in documentation processes, it offers detailed strategies and maintenance solutions tailored to address these challenges effectively. Through rigorous analysis, the thesis provides actionable insights to enhance documentation management's overall efficiency and reliability throughout the SDLC phases.

In future research, the author would like to stay with the same research field but improve the size of the research sample and maybe follow quantitative research methods to receive more insights. In this bachelor's thesis, the author analyzed data manually, even though some raw AI language models for research are already available on the market. Patterns of risks were similar, but manual analysis was preferred so that AI analysis might be chosen for the following research. For the following diploma thesis, the author would like to choose a specific enterprise to analyze the company's specific risks and provide solutions based on timelines and allocated resources.

Conclusion

The central objective of this bachelor thesis was to conduct a comprehensive investigation into the various types of risks commonly associated with software development life cycle documentation within larger enterprises and provide a Mitigation Plan with proposed solutions for the risks.

The investigation was conducted based on the APO12 (Align, Plan, and Organize) process called Manage Risk from the COBIT 5 framework, which defines objectives, identifies, analyzes, and addresses risks. The identification of risks was conducted using qualitative research methods: semi-structured interviews, data analysis with MAXQDA, and comparison.

The results are described in Chapter 7. Seven risks were identified from the research with the help of the MAXQDA tool for qualitative analysis. A risk Matrix for prioritizing risks was created, and a visual representation can be found in Chapter 7.1. The results show the most significant potential risks associated with the Software Development Life Cycle in Large Enterprises: Weak Standardization, Weak Document Security, Irrelevant Documentation, and Weak Awareness of Processes.

A comprehensive Mitigation Plan was developed to address the top 5 risks identified in the project. The plan includes a detailed mitigation strategy outlining the steps to mitigate each potential risk, including the responsible parties needed for implementation. Additionally, the plan includes success criteria that might be used to evaluate the effectiveness of the mitigation strategies.

The goal set out in this thesis has been successfully achieved based on the information provided above. All the established objectives and criteria have been met, and the results are consistent with the expected outcomes.

This bachelor's thesis is an extensive guide for teams or managers looking to enhance their Software Development Life Cycle Documentation Management practices globally. It provides an in-depth analysis of the common risks associated with documentation processes. It presents detailed strategies and maintenance solutions specifically designed to address these challenges effectively and provide comprehensive guidance to organizations seeking to optimize their documentation management practices to maximize efficiency, minimize risks, and improve overall productivity.

List of references

AltexSoft Inc. (2018, January 16). *Software Documentation Types and Best Practices*. Medium. https://blog.prototypr.io/software-documentation-types-and-best-practices-1726ca595c7f

Andersen, N. B. (2023, March 6). *What Is the Waterfall Methodology*? Built In National. https://builtin.com/software-engineering-perspectives/waterfall-methodology

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., & Grenning, J. (2001). *Principles behind the Agile Manifesto*. Agile Manifesto. https://agilemanifesto.org/principles.html

Berhouma, H. (2020). A Generic Model for Software Documentation and its Application in Embedded Systems Developed with Scrum. *Proceedings of the 2020 9th International Conference on Software and Information Engineering (ICSIE)*, 33–36. https://doi.org/10.1145/3436829.3436858

Braun, V., & Clarke, V. (2021). *Thematic Analysis: A Practical Guide*. SAGE.

Clark, H. (2022, May 20). *The Software Development Life Cycle (SDLC): 7 Phases and 5 Models*. The Product Manager. https://theproductmanager.com/topics/software-development-life-cycle/

Cohn, M. (2005). *Agile Estimating and Planning*. Pearson Education.

Deveraj, K. (2023, May 31). *6 Key Phases of Testing in Software Testing*. Testsigma Blog. https://testsigma.com/blog/phases-of-testing/

de Vicente Mohino, J., Bermejo Higuera, J., Bermejo Higuera, J. R., & Sicilia Montalvo, J. A. (2019). The Application of a New Secure Software Development Life Cycle (S-SDLC) with Agile Methodologies. *Electronics*, *8*(11), Article 11. https://doi.org/10.3390/electronics8111218

Duijm, N. J. (2015). Recommendations on the use and design of risk matrices. *Safety Science*, *76*, 21–31. https://doi.org/10.1016/j.ssci.2015.02.014

Fung, P., Kwok, L., & Longley, D. (2003). Electronic information security documentation. *Proceedings of the Australasian Information Security Workshop Conference on ACSW Frontiers 2003 - Volume 21*, *21*, 25–31. https://dl-acm-org.zdroje.vse.cz/doi/10.5555/827987.827991

Geperin, D. (2008). Exploring agile. *Proceedings of the 2008 International Workshop on Scrtinizing Agile Practices or Shoot-out at the Agile Corral*, 1–3. https://doi.org/10.1145/1370143.1370144

Gill, T. A. (2023, December 13). *20 Common Challenges In Document Management Today*. Document Management System Folderit. https://www.folderit.com/blog/20-common-challenges-in-document-management-today/

Guevara, P. (2022, April 27). *What is a 5x5 Risk Matrix & How to Use it?* SafetyCulture. https://safetyculture.com/topics/risk-assessment/5x5-risk-matrix/

Gupta, M., Sureka, A., Padmanabhuni, S., & Asadullah, A. M. (2015). Identifying Software Process Management Challenges: Survey of Practitioners in a Large Global IT Company. *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 346–356. https://doi.org/10.1109/MSR.2015.39

IEEE. (2014). *IEEE Standard for Software Quality Assurance Processes*. https://doi.org/10.1109/IEEESTD.2014.6835311

Imoh, W. (2023, October 9). *Documentation Overload: Can Comprehensive Documentation Hinder Product Adoption?* Hackmamba. https://hackmamba.io/blog/2023/10/documentation-overload-can-comprehensive-documentation-hinder-product-adoption/

Junker, T. L., Bakker, A. B., Gorgievski, M. J., & Derks, D. (2022). Agile work practices and employee proactivity: A multilevel study. *Human Relations*, *75*(12), 2189–2217. https://doi.org/10.1177/00187267211030101

Martins, J. (2024, January 19). *What Is Kanban? A Beginner's Guide for Agile Teams*. Asana. https://asana.com/resources/what-is-kanban

Martins, J. (2024, February 1). *Scrum: The Most Popular Agile Framework*. Asana. https://asana.com/resources/what-is-scrum

Martin, M. (2024, February 24). *Software Development Life Cycle (SDLC) Phases & Models*. Guru99. https://www.guru99.com/software-development-life-cycle-tutorial.html

Peyyala, D. (2022, January 21). *Understanding Rapid Application Development*. Tynybay. https://www.tynybay.com/our-thinking/understanding-rapid-application-development

Pukdesree, S. (2017). The comparative study of collaborative learning and SDLC model to develop IT group projects. *TEM Journal*, *6*, 800–809. https://doi.org/10.18421/TEM64-20

Roman, Š., & Šeďová, K. (2007). *Kvalitativní výzkum v pedagogických vědách*. PORTÁL.

Ruparelia, N. B. (2010). Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes*, *35*(3), 8–13. https://doi.org/10.1145/1764810.1764814

Sacolick, I. (2022, April 8). *A brief history of the agile methodology*. InfoWorld. https://www.infoworld.com/article/3655646/a-brief-history-of-the-agile-methodology.html

Saravanos, A., & Curinga, M. X. (2023). Simulating the Software Development Lifecycle: The Waterfall Model. *Applied System Innovation*, *6*(6), Article 6. https://doi.org/10.3390/asi6060108

Shah, U. S., Jinwala, D. C., & Patel, S. J. (2016). An Excursion to Software Development Life Cycle Models: An Old to Ever-growing Models. *ACM SIGSOFT Software Engineering Notes*, *41*(1), 1–6. https://doi.org/10.1145/2853073.2853080

Shantanu, C. (2018, November 26). *Evolution of System Development Life Cycle (SDLC)*. LinkedIn. https://www.linkedin.com/pulse/evolution-system-development-life-cycle-sdlc-shantanu-choudhary/

Simhadri, R. S., & Shameem, M. (2023). Challenges in Requirements Gathering for Agile Software Development. *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, 406–413. https://doi.org/10.1145/3593434.3594237

Suryn, W. (2014). *Software Quality Engineering: A Practitioner's Approach*. John Wiley & Sons.

Sutherland, H., Recchia, G., Dryhurst, S., & Freeman, A. L. J. (2021). How People Understand Risk Matrices, and How Matrix Design Can Improve their Use: Findings from Randomized Controlled Studies. *Risk Analysis*, *42*(5), 1023–1041. https://doi.org/10.1111/risa.13822

Theunissen, T., van Heesch, U., & Avgeriou, P. (2022). A mapping study on documentation in Continuous Software Development. *Information and Software Technology*, *142*, 1–29. https://doi.org/10.1016/j.infsof.2021.106733

Varol, D. (2023, September 20). *7 Best Practices for Good Documentation Management*. Scilife. https://www.scilife.io/blog/documentation-management-best-practices

Vroon, M. (2019, March 20). *Agile Scrum*. MARCEL VROON. https://marcelvroon.nl/agile-scrum/