DOI: https://doi.org/10.32782/2524-0072/2024-63-3

UDC 004.01:005.94

A TEMPLATE FOR TECHNICAL DESIGN DOCUMENTS TO HELP DISTRIBUTED SOFTWARE DEVELOPMENT TEAMS COLLABORATE BETTER

ШАБЛОН ДИЗАЙНУ ТЕХНІЧНОЇ ДОКУМЕНТАЦІЇ ДЛЯ КРАЩОЇ СПІВПРАЦІ РОЗРОБНИКІВ РОЗПОДІЛЕНИХ КОМАНД

Bogolii Oleksandr
PhD Student,
Higher Education Institution the "KROK" University
ORCID: https://orcid.org/0000-0003-0253-667X

Боголій Олександр
ВНЗ «Університет економіки та права «КРОК»

Technical Design Documents (TDD) are commonly adopted documents in the software development industry that describe the technical solution implementation design. These documents are informal and lack strict requirements regarding their content. The structure often differs from organization to organization. Lack of standardized structure often results in low-quality design documents that can hinder successful technical solution implementation. In this paper, we applied a grounded theory method to analyze various organization-specific TDD structures and proposed a standardized structure for this document. In addition to the plain text structure, we visualized it in the form of a mindmap to facilitate the adoption of this template. Organizations can use the proposed TDD template structure as a basis for the company-specific structure of TDD and may include all or part of the sections proposed.
Keywords: technical design document, template, mindmap, software development, distributed team.

Документи технічного дизайну є загальноприйнятими документи в індустрії розробки програмного забезпечення, що описують дизайн імплементації технічного рішення. Ці документи є неформальними і не мають суворих вимог щодо їхнього змісту. Відсутність стандартизованої структури часто призводить до низької якості проєктних документів, що може стати на заваді успішної реалізації технічного рішення. Тому, корисною практикою є стандартизація структури документації технічного дизайну всередині організації, що сприяє налагодженню процесів розробки програмного забезпечення компанії. В цій статті ми використали метод обґрунтованої теорії, щоб проаналізувати структури документів, що використовуються в різних організаціях для опису технічного рішення. Після цього було застосовано осьове кодування для виявлення взаємозв'язків між категоріями та підкатегоріями в різних структурах документів, а також вибіркове кодування для інтеграції та уточнення визначених категорій. В результаті проведеного дослідження було представлено розроблений стандартизований шаблон документу технічного дизайну. Остаточна структура складається з 11 основних розділів: Вступ, Варіанти рішення, Подальші аспекти, Безпека та конфіденційність, Тестування, Імплементація, Впровадження та розгортання, Інструкції, Оцінка рішення, Відкриті питання та Наступні кроки. Вміст перелічених розділів описаний в основному тексті статті. Окрім звичайної текстової структури, ми візуалізували структуру у вигляді ментальної карти, щоб спростити її сприйняття. Запропонований шаблон був перевірений галузевими експертами та протестований на реальному проєкті. Загалом отримано позитивні відгуки. Організації можуть використовувати запропонований шаблон як основу для створення специфічних для компанії документів технічного дизайну, що можуть включати всі або частину запропонованих розділів.
Ключові слова: документ технічного дизайну, шаблон, ментальна карта, розробка програмного забезпечення, розподілена команда.

1. Introduction

Nowadays, software is often developed in a globally distributed environment. There are many challenges faced by companies that operate in such a setup. The main challenges for efficient software development in a remote distributed

environment are lack of communication, coordination, control, and complicated knowledge sharing (Bogolii, 2023).

Distributed development teams, often working in different timezones, actively use Technical Design Documents (TDD) to share knowledge about technical solution implementation strategies. Sharing knowledge among employees, both within and across teams, is essential for effective knowledge management and fostering innovation (Jackson et al., 2006; Kajko-Mattsson, 2008).

Different development teams have different standards and conventions for technical design templates, depending on their situation. These are informal documents and, thus, do not follow strict guidelines for their content. Most often, TDD defines details of solution architecture, diverse data flow diagrams, data validation rules, high-level documentation of the code, etc.

The heterogeneity of standards for TDD can lead to problems during distributed software development (Henderson, 2024). Without a standard format, each team or individual may create documents in their preferred style, leading to inconsistency across the organization. This can make it difficult for team members to understand each other's documents and collaborate effectively (Aghajani et al., 2019).

In addition, a lack of standardization may result in unclear or incomplete technical design documents with varying quality. This can lead to confusion regarding project requirements, architecture, and implementation details, potentially causing costly errors or rework later in the development process (Kajko-Mattsson, 2008; LinkedIn community, 2023).

Thus, establishing a certain structure for this document inside an organization has proven to be helpful, enabling engineers to enhance their efficiency and improve the quality of their work. A well-organized structure, along with explicit guidelines, forms the foundation for maintaining consistent and clear technical documentation (Aghajani et al., 2019).

Analysis of recent research and publications reveals multiple studies that focus on the standardization of certain aspects of technical documentation processes in organizations (Caponi et al., 2018; Uikey et al., 2011). For example, Alwazae et al. (2015) proposed the Best Practice Document Template (BPDT) to describe a company's best practices in a detailed and systematic way. Another example is the template for requirements documentation (Kalfat et al., 2023) and the template for technical specifications or technical datasheets (Seghiri & Luque Giráldez, 2023). Also worth mentioning is the C4 model for visualizing the architecture of software systems. It helps in creating a standardized approach to software documentation through its structured way of presenting different levels of abstraction (Brown, 2018).

To our knowledge, there were no studies regarding technical design document standardization.

This research aims to overview the existing design document templates used in the software development industry and propose a standardized TDD template incorporating the latest industry trends.

The rest of the paper is structured as follows: Section 2 analyzes the popular TDD templates in software development. Section 3 presents the proposed TDD template and discusses its structure. Section 4 explains how the template was evaluated, and finally, Section 5 concludes the paper.

2. Methods

In this section, we will design the TDD structure that could be useful as a standard template inside a typical software development company. The approach we use in this paper incorporates a grounded theory method (Wolfswinkel et al., 2013; Johannesson & Perjons, 2014). In the following sections, we will describe the steps involved in this process in more detail.

2.1. Define inclusion and exclusion criteria

We start by defining the inclusion and exclusion criteria. Typically, sources other than peer-reviewed journals, conference articles, and book chapters are not considered acceptable data for a scholarly review. Unfortunately, we did not find many research papers presenting real-world technical design documents. That is why we took another approach by examining the technical blogs of known software development companies that share their own TDD structure or existing templates. We accept such sources as valid for our review, as publications in companies' blogs are often reviewed by multiple people, including software architects, product and project managers, the marketing department, etc., so they meet our quality requirements. We restricted our search to big companies with known expertise in software development.

We defined the following search terms: Technical Design Document (TDD), Technical Implementation Document (TID), Technical Specification Document (TSD), Low-Level Design Document (LLD), Detailed Design

Document (DDD) and Requests for Comments (RFC).

All these terms are very often used interchangeably, depending on the organization. Their main goal is to provide detailed technical descriptions of software solutions.

Requests for Comments (RFC) were initially used in the standardization process for Internet protocols, procedures, and technologies by the Internet Engineering Task Force (IETF). Although Technical Design Documents (TDD) and Requests for Comments (RFC) are not the same, many companies borrowed this established publication term for technical design documents.

2.2. Search and Selection

Our research mainly used Google searches to find official technical blogs of companies that share their TDD templates. In addition to this, we also searched Twitter posts for any mentions of our search terms, identified a list of companies, and then proceeded with examining companies' technical blogs for publications about the usage of TDD.

At this stage, we examined the structure of found document templates in detail. Many types of documents are used in the software development industry for different purposes and with varying levels of abstraction. Often, the structure of these documents has similar sections, causing some confusion. For example, several found TDD templates were structurally more similar to Product Requirements Documents (PRD) or High-Level Design Documents (HLD). For this reason, we left only documents whose structure focused on technical aspects of the designed solution, such as process flow, scalability, performance, exception handling, security, etc.

The biggest contribution to our analysis was made by the publications of Golman (2020), Orosz (2018), Stanford University IT (n.d.), Mage (2021), Donovan (2020), Kingson (2023), Slite (2024), and Dupont (2024).

2.3. Analyze

To organize the information, we visualized each selected template as a mindmap, as this format of visualization has proven efficient for studying and organizing ideas (Farrand et al., 2002; Nurlaila, 2013). We added corresponding nodes and subnodes connected with arrows for each section and subsection to illustrate their relationships. At last, we carefully examined the descriptions (if available) of inner subsections to identify the possible list of items they are intended to include. These items were also drowned as subnodes connected to their corresponding subsections.

After this, we applied ‘axial coding’ to identify the interrelationships between categories and their subcategories of different templates and ‘selective coding’ to integrate and refine the identified categories. As part of this activity, we identified sections with similar or overlapping contents, synonyms for the same concepts and activities, etc.

3. Results

We ended up with a template structure, presented below and visualized in Figure 3.

Technical Design Document
1. Introduction
1.1. Front Matter
1.2. Brief Summary
1.3. Related Documents
1.4. Glossary
1.5. Requirements
2. Solutions
2.1. Current Solution
2.1.1. Description
2.1.2. Pros & Cons
2.2. Proposed Solution
2.2.1. UI & UX
2.2.2. System Context Diagram
2.2.3. Process Flow Diagram
2.2.4. API changes
2.2.5. Data storage
2.3. Alternatives
2.3.1. Description
2.3.2. Pros & Cons
3. Further Considerations
3.1. Cost Analysis
3.2. Accessibility
3.3. Regional Considerations
3.4. Third-party Services & Platforms
3.5. Resilience
3.6. Scalability
3.7. Risks
4. Security & Privacy
4.1. Security
4.2. Data Protection & Privacy Compliance
4.3. Data Retention
5. Testing
5.1. Load & Performance Testing
5.2. Testing Plan
5.3. Test Data
5.4. QA
6. Implementation
6.1. Epics & tasks
6.2. Milestones
6.3. Timeline
7. Rollout & Deployment
7.1. Deployment Environments

7.2. Phased Rollout Plan
7.3. Rollback Plan
7.4. Communication Plan
8. Runbook
8.1. Health Metrics
8.2. Troubleshooting
8.3. Alerts
8.4. Monitoring
8.5. Logging
8.6. Operational Procedures
9. Evaluation
9.1. Cost-benefit Analysis
9.2. Performance Metrics
9.3. Impact Assessment
10. Open Questions
11. Next Steps

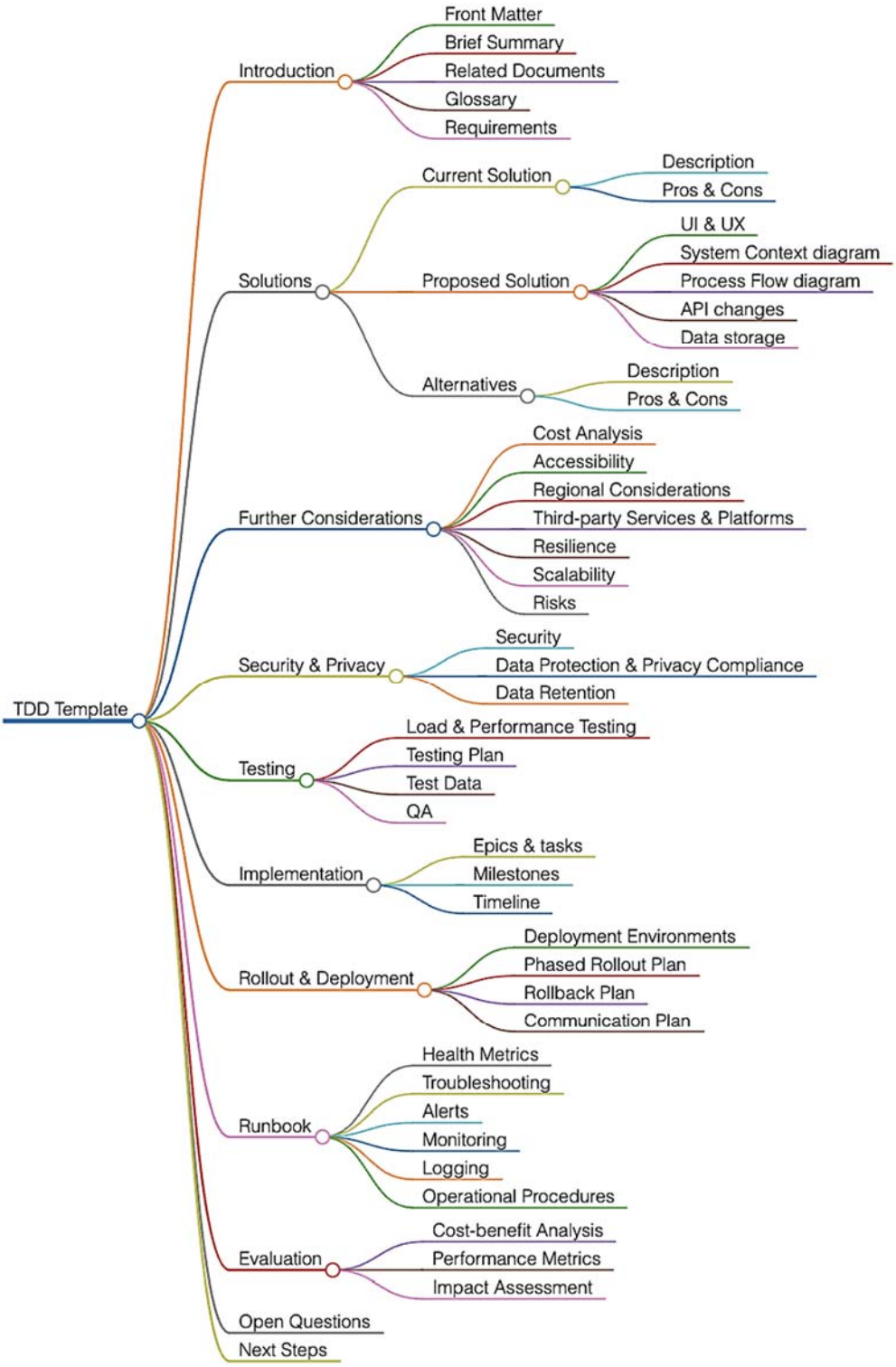

Figure 1. Mindmap for Proposed TDD Template

The proposed TDD template consists of 11 main sections. Below, we describe the contents of these sections.

3.1. Introduction

The design document starts with an ***Introduction*** section, which provides an overview and context for the topic being discussed and the document in general. Here we define four subsections: Front Matter, Brief Summary, Glossary, Requirements, and Related Documents.

Front Matter is the first page of the TDD and is used to define the document's author, which team is working on this solution, who will be the reviewers and approvers, and list the teams (stakeholders) affected by the changes. As the TDD often passes through a few phases of reviews, some significant changes may be made, so adding a change history table listing the most critical updates may be valuable.

A ***Brief Summary*** gives a general overview of the issue (from the user's perspective). Provide an overview of the context, objective background facts, and problems the users are facing. This subsection should also include a brief overview of the solution that is being proposed.

A ***Glossary*** or ***Vocabulary*** lists specific terms and abbreviations used throughout the document. By providing clear and consistent definitions for terms used within the document, a glossary ensures that all readers interpret and understand the terminology in the same way.

The ***Requirements*** section defines the goals we try to achieve with a proposed solution. It usually includes product requirements, user stories or use cases, and requirements. We may define what is outside the scope or not a goal of this document. Technical requirements may include service SLAs, characteristics of components, metrics, and ACID compliance.

Finally, this section should contain links to ***Related Documents*** and supporting materials used when developing the proposed design.

3.2. Solutions

This part of the TDD is focused on illustrating the existing and suggested solutions for the problem and discussing the alternative solutions.

This section starts by illustrating the ***Current Solution*** and its pros and cons.

After a brief overview of the current solution, next is a description of the ***Proposed Solution***. This section is the longest part of the design document and requires the most research, planning, and preparation. It presents an engineering approach to solving the technical problem and often includes the following:

– User Interface & User Experience: This would include how the user interface would be, the features and actions the users would take, and the UI elements.

– System Context Diagram: This demonstrates how the system fits into the broader technical environment and helps readers understand how the new design fits into a familiar setting.

– Process Flow Diagrams: depict the overall process flow so that it is clear to the developer what the outcome is and how to reach it.

– API changes: API endpoints, sample API requests and responses, etc.

– Data storage: Data model and schema changes.

Finally, ***Alternative solution*** designs should be presented. Presenting alternatives with their trade-offs shows explicitly why the selected solution has been chosen.

3.3. Further Considerations

In this part of the document, we will discuss other potential concerns that require consideration. These sections are usually concise and explain how the proposed solution affects the concern and how it will be addressed. Below, we will briefly pass through typical concerns.

Cost analysis presents rough estimates of the cost to run the proposed solution. Typically, this includes the required infrastructure costs (for example, compute instances, databases, storage devices, etc), third-party services, roll-out costs, etc.

Regional considerations describe how the proposed solution will be running in different regions. It includes localization, latency considerations, and various regulatory concerns like customer data storage place, etc.

The ***third-party services***, ***platforms***, and ***software*** section explains the need for those services for the proposed solution, associated costs, security and privacy concerns, limitations, risks, etc.

Resilience characteristics describe the impact on the system in case of various failure types (e.g., single or multiple instance outage, etc).

Scalability reveals if the proposed solution could handle the increased workload or user demands without significant degradation in performance.

This section is not limited to the concerns mentioned above. Other noteworthy concerns are ***Accessibility***, ***Configuration***, ***Idempotency***, ***Risks***, and others.

3.4. Security and Privacy

Due to their importance, security and privacy concerns are put into a dedicated section of TDD. Engaging with privacy and security teams as early as possible is recommended to ensure that designs take them into account from the ground up.

Security concerns should describe how the proposed solution mitigates potential threats and affects the security of other components, services, and systems.

Data Protection and ***Privacy Compliance*** indicate if the solution retains user-related data or transmits such data to (or from) third-party vendors. It should also indicate if the solution needs to comply with local laws and legal policies on data privacy, for example, the General Data Protection Regulation (GDPR).

The ***Data Retention*** section lists retention periods for various types of data, the mechanisms of deleting unneeded data, and data from users requesting to be excluded from processing.

3.5. Testing

This section explains how to ensure the proposed solution works. Typically, this means writing a ***Testing Plan***, highlighting how we would test the changes, and explaining how these tests ensure user requirements are met. Various types of testing may be necessary, such as Unit Testing, Integration Testing, Load, Performance Testing, User Acceptance Testing, and Regression Testing. In the case of dedicated documents for these types of testing, TDD may reference them instead of going into detail.

Test data is crucial in ensuring the software is thoroughly tested and capable of handling various scenarios. This document section should clarify what test data to use and how to create or generate it manually.

Besides mentioned above, this section may also review how ***Quality Assurance*** will be performed during the development process, how the engineering team could prevent defects, and how to improve the overall quality of the developed solution.

3.6. Implementation

This section includes the actionable items (i.e., ***Epics*** and ***Tasks***) required to complete and ship the proposed solution. It defines, as well, a ***Timeline*** and ***Milestones*** to help keep the process organized.

Each milestone should point to the metrics indicating this milestone passed, and tasks should have time estimates for how long it takes to be completed.

3.7. Rollout and Deployment

This section describes the plan for solution ***Deployment*** and the ***Deployment Environments***. It includes a description of various activities, like configuring the software, installing it on the necessary servers or devices, and making it operational for end-users.

To minimize the risks and disruptions that may occur with a full-scale deployment, it is a good practice to write down a ***Rollout Plan***. It involves a phased or controlled release (often via feature flags) of the changes, allowing for testing, feedback gathering, and addressing potential issues before full ***Deployment***.

It is essential to include a ***Rollback Plan*** in case issues occur during or after the deployment of the solution. It includes the necessary steps to restore the system's functionality and stability and avoid data loss.

To minimize surprises to others, a proper ***Communication Plan*** should be presented. This section should describe the impact of the changes on users, stakeholders, and the organization and the most appropriate time to communicate updates.

3.8. Runbook

The purpose of a runbook is to provide step-by-step guidance on how to perform specific tasks, including troubleshooting, maintenance, deployment, or recovery procedures. They are designed to be practical references that enable individuals to execute tasks effectively, even if they are unfamiliar with the specific process or system.

First, this section should define ***Health Metrics*** so it is clear how to ensure the solution works as expected. Once metrics are defined, it is vital to describe how we will perform continuous ***Monitoring*** of these metrics. In addition to monitoring, it is crucial to set up ***Alerts*** that should pop up in case of any issues with the solution. Each alert should list the monitored metric, alert thresholds, possible impact, and ***Troubleshoot*** instructions. To efficiently troubleshoot any issues, describing the ***Logging*** and ***Tracing*** setup may be helpful.

It is the right place to define various ***Operational Procedures***, like data and solution recovery in case of failures, scaling up and down, etc.

3.9. Evaluation

It is crucial to describe how the success of the implementation of proposed solutions can be evaluated. To effectively do this, describe the list of ***Metrics*** to capture and tools used to measure performance against these metrics. Include a ***Cost-benefit Analysis,*** if applicable.

Also, this section may discuss the ***Impact Assessment*** of the proposed solution on users, stakeholders, and the organization.

3.10. Open Questions

In this section, please list any items that require feedback from other engineering teams. These may include problem areas that are unclear on how to resolve and require additional research, investigation, or decisions before implementation.

3.11. Next steps

Outline the next steps to take and assign action items required to move forward with the implementation of the proposed solution.

4. Discussion

In order to evaluate the developed template, in total, four practitioners and academic experts in the area of software development and project management were asked to evaluate and refine the template, and based on their input, attributes were added, deleted, or refined.

Respondents emphasized the completeness and ease of use of the proposed template.

Furthermore, the suggested template underwent real-world evaluation within an IT company specializing in internet security. This template served as the foundational structure for the TDD of a new project undertaken by the company. Overall, it received a good appraisal, and most of the stakeholders indicated its efficacy in guiding the development.

Potential improvements suggested by respondents include providing examples of content for each section (subsection) and providing tooling support for template applications.

5. Conclusion

In this paper, we discussed what functions Technical Design Documents serve inside organizations' software development life cycle and proposed a standardized TDD template structure.

We extensively researched popular TDD templates, visualized corresponding template structures in mindmaps, and applied coding techniques to define a standardized structure for this type of document.

The final structure encounters 11 main sections: Introduction, Solutions, Further Considerations, Security and Privacy, Testing, Implementation, Rollout and Deployment, Runbook, Evaluation, Open Questions, and Next Steps. The majority of them consist of multiple nested subsections. We included a summary of the contents of each section and subsection.

Depending on the organization, this template could be used as is or serve as a basis for the company-specific structure of TDD and may include all or part of the sections proposed.

The proposed template was evaluated by industry experts and tested on a real-world project. Overall, positive feedback was received.

In the future, we plan to gather more feedback from real world applications and enrich the template with examples of content.

REFERENCES:

1. Aghajani, E., Nagy, C., Vega-Marquez, O. L., Linares-Vasquez, M., Moreno, L., Bavota, G., & Lanza, M. (2019). Software Documentation Issues Unveiled. 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), 1199–1210. https://doi.org/10.1109/ICSE.2019.00122

2. Alwazae, M., Perjons, E., & Johannesson, P. (2015). Applying a Template for Best Practice Documentation. *Procedia Computer Science*, 72, 252–260. DOI: https://doi.org/10.1016/j.procs.2015.12.138

3. Bogolii, O. (2023). Agile Software Development in a Remotely Working Geographically Distributed Team: A Systematic Review. *European Project Management Journal*, 13(1): 23–36. DOI: 10.56889/idnv2224

4. Brown, S. (2018). The C4 Model for Software Architecture. InfoQ. https://www.infoq.com/articles/C4-architecture-model/

5. Caponi, A., Di Iorio, A., Vitali, F., Alberti, P., & Scatá, M. (2018). Exploiting patterns and templates for technical documentation (p. 9). DOI: https://doi.org/10.1145/3209280.3209537

6. Donovan, R. (2020, April 6). A practical guide to writing technical specs. Stack Overflow Blog. URL: https://stackoverflow.blog/2020/04/06/a-practical-guide-to-writing-technical-specs/

7. Dupont, E. (2024, January 18). Create a functional and technical design document – Dynamics 365. URL: https://learn.microsoft.com/en-us/dynamics365/guidance/patterns/create-functional-technical-design-document

8. Farrand, P., Hussain, F., & Hennessy, E. (2002). The efficacy of the “mind map” study technique. Medical Education, 36, 426–431. DOI: https://doi.org/10.1046/j.1365-2923.2002.01205.x

9. Golman, J. (2019, September 12). Design Docs at Google. Industrial Empathy. URL: https://www.industrialempathy.com/posts/design-docs-at-google/

10. Henderson, C. (2024, March 21). Top 10 challenges documentation software solves. Paligo. URL: https://paligo.net/blog/software/top-10-challenges-documentation-software-solves/

11.Jackson, S. E., Chuang, C.-H., Harden, E. E., & Jiang, Y. (2006). Toward Developing Human Resource Management Systems for Knowledge-Intensive Teamwork. In Research in Personnel and Human Resources Management (Vol. 25, pp. 27–70). Emerald (MCB UP). DOI: https://doi.org/10.1016/S0742-7301(06)25002-3

12. Johannesson, P., & Perjons, E. (2014). An Introduction to Design Science. Springer International Publishing. DOI: https://doi.org/10.1007/978-3-319-10632-8

13. Kajko-Mattsson, M. (2008). Problems in Agile Trenches. In ESEM'08: Proceedings of the 2008 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (p. 119). DOI: https://doi.org/10.1145/1414004.1414025

14. Kalfat, H., Oussalah, M., & Chikh, A. (2023). ADM: An Agile Template for Requirements Documentation (p. 501). DOI: https://doi.org/10.5220/0012122400003538

15. Kingson, S. (2023). Guide to Create Technical Specification Document with Example. Document 360. https://document360.com/blog/technical-specification-document/

16. LinkedIn community. (2023). What are the most common software project documentation challenges? Linkedin. https://www.linkedin.com/advice/1/what-most-common-software-project-documentation-oyqnc

17. Mage. (2021, September 14). How to Write Technical Design Docs. Dev.to. https://dev.to/mage_ai/how-to-write-technical-design-docs-c02

18. Nurlaila, A. P. (2013). THE USE OF MIND MAPPING TECHNIQUE IN WRITING DESCRIPTIVE TEXT. Journal of English and Education, 1(2), Article 2.

19. Orosz, G. (2018, October 03). Scaling Engineering Teams via RFCs: Writing Things Down. The Pragmatic Engineer. Retrieved July 24, 2023, from https://blog.pragmaticengineer.com/scaling-engineering-teams-via-writing-things-down-rfcs/

20. Seghiri, M., & Luque Giráldez, Á. (2023). Designing a technical specifications template in English a corpus-based approach (EN PRENSA).

21. Stanford University IT. (n.d.). Technical Design. Project Management | University IT. Retrieved June 24, 2023, from https://uit.stanford.edu/pmo/technical-design

22. Uikey, N., Suman, U., & Ramani, A. (2011). A Documented Approach in Agile Software Development. *International Journal of Software Engineering*, 2011–2013.

23. Wolfswinkel, J. F., Furtmueller, E., & Wilderom, C. P. M. (2013). Using grounded theory as a method for rigorously reviewing literature. *European Journal of Information Systems*, 22(1), 45–55. DOI: https://doi.org/10.1057/ejis.2011.51