Topics and decisions in a restructuring project over time

Master's degree

Department of computer science

Free University of Berlin

Vahid Helalat

5157273

vahid.helalat@fu-berlin.de

Born on the 07.12.1991 in Berlin

Advisor Prof. Dr. Lutz Prechelt

Berlin, 9. September 2022

…….

Selbstständigkeitserklärung

Ich erkläre gegenüber der Freien Universität Berlin, dass ich die vorliegende Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe. Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht. Diese Arbeit wurde in gleicher oder ähnlicher Form noch bei keiner anderen Universität als Prüfungsleistung eingereicht.

Berlin, 9. Sep 2022

Vahid Helalat

Abstract

Software development is a complex process with challenges and issues that arise during this process. This thesis explores critical issues and challenges during this process in a restructuring (rewriting) project. Additionally, it aims to help organizations overcome these challenges. A qualitative research method is used to observe the ideas emerging during the software development process. Data is collected through observations and documents. All the meetings are recorded. The data collection process provides helpful information for different perspectives on the issues and decision-making processes. The research presents challenges and issues in four categories feature development, strategy, quality, and software structure. The decisions made in the project are analyzed in the data evaluation. Additionally, the impact of the decisions is examined on the project's success.

Acknowledgment

I would thank my professor for his support and continuous feedback throughout this project. I would thank my product owner for his support during the project and helping me through the topics. I would also thank my team members for responding to questions and participating in the thesis.

TABLE OF CONTENTS

Figures

Abbreviations

No	Short Form	Word
1	MVP	Minimum Viable Product
2	SDLC	Software Development Life Cycle
3	QA	Quality Assurance
4	E2E Testing	End-to-End Testing
5	UI	User Interface
6	DOD	Definition of Done
7	PO	Product Owner
8	POC	Proof Of Concept

1 Introduction

1.1 Motivation

It is common for new companies to fail and must try as quickly as possible to present their product because other competitors may be able to bring their idea to market. Companies must provide products and services as soon as possible to compete with their competitors.

Outdated applications (legacy applications) are hard to maintain. If we wish to meet the future needs of our customers and stay up to date in the market, it is crucial to have a product of the highest quality. To make a product more valuable, it must be modernized.

Some challenges arise during the software development process. During this process, some common issues and challenges arise. Observing the challenges and topics during this process would have been interesting.

The decisions made during the software development process affect the software's success and its market time. It would have also been interesting to observe how these decisions were made during this process. The decisions made during the software development process affect the quality and the time to market.

1.2 Structure of the master thesis

The master's thesis consists of eight chapters, the content and structure of which are explained below.

In this Chapter, the challenges in legacy applications are explained. The importance of modernizing applications is also discussed.

Chapter 2, Research Methodology, explains the aims and objectives of the thesis. The research methodology is also discussed.

Chapter 3, Feature Development, explains the challenges in feature development.

Chapter 4, Discipline/Quality, explains the challenges regarding discipline and quality.

Chapter 5, Strategy, explains methodic challenges.

Chapter 6, Software structure, explains the challenges in software structures.

Chapter 7 Evaluation analyzes the decisions made during the software development process and examines their impact on the project's success.

Chapter 8, Conclusion, summarizes what we have learned during the thesis.

1.3 Background

At a company, an 8-member development team is working on an existing web application (developed by others) in production. The team is relatively young and not very experienced. The application is a job platform that connects candidates and employers. Candidates and employers are connected through the application. A candidate should go through the registration and answer some questions to fill out the profile. According to the information provided in their registration and profile, candidates see the most relevant jobs. The candidate can then browse the positions and apply. Candidate applications can be viewed by employers, who can accept or decline them. Employers can send an invitation to candidates. Employers can see candidates who are matched for their jobs and invite them. An admin dashboard is also needed to manage the application. The application should be multi-language; English and German are enough for now. The company makes money with this application, and the operation must be maintained continuously.

It is becoming difficult to maintain the application as it becomes outdated. More than 250 candidates registered in the application daily, and the application should meet their needs. The number of new candidates does not increase; some leave the program at different stages. We have gotten some feedback from candidates that they have problems with the application.

Candidates mentioned the following problems:

1) They get errors during the registration and cannot finish the registration process.
2) They are not getting the most relevant jobs to their profiles.
3) They are not satisfied with the performance of the application. The application crashes.
4) The application is not user-friendly, and candidates get confused during the application process.

Old applications (legacy applications) can be easily replaced by a competitor with a modern application, even if they serve their intended purpose well.

A legacy application might satisfy customers' current business needs, but they aren't prepared to meet their future needs. A legacy application is not as scalable and flexible as a modern application, and it cannot leverage the latest technologies essential for maintaining an application.

Many legacy applications do not have responsive UI, making working with the latest devices challenging. While a new UI can make an application look more modern, it will not remove the code complexity.

Many developers work on large-scale applications over months and years, resulting in highly complex spaghetti code. Modernizing is the only way to make the product more valuable.

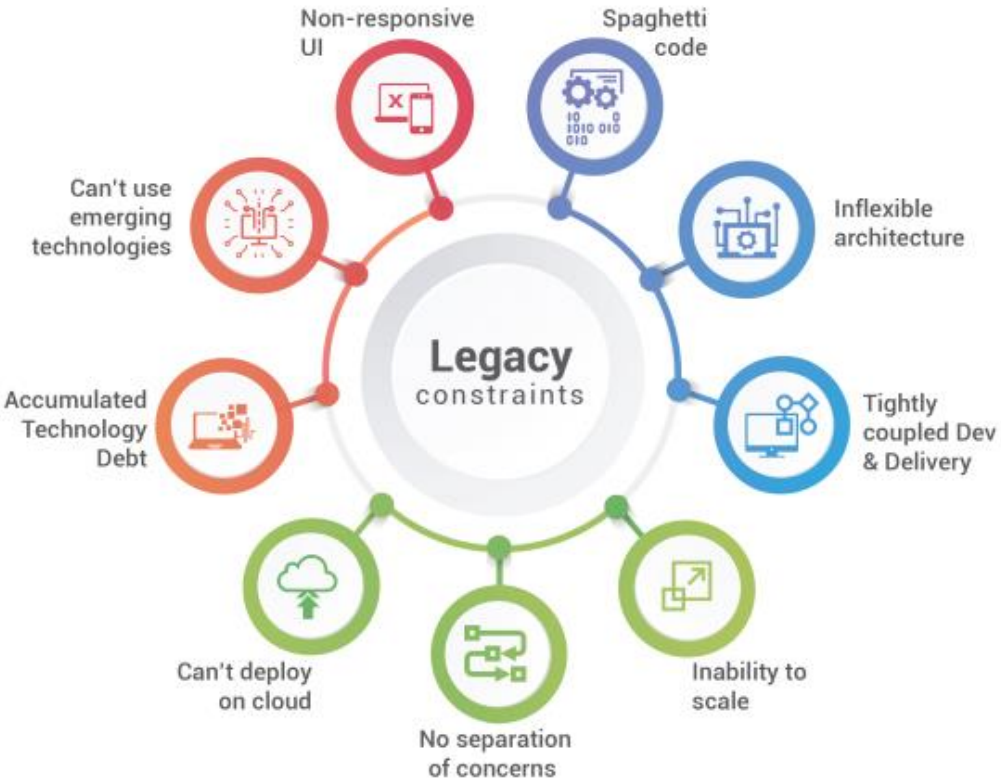


Figure 1: Why should you modernize? [1]

At this time, the topic of Refactoring or Rewrite was discussed.

1.4 Refactoring vs. Rewrite

The purpose of refactoring is to improve code without adding any new functionality. Since basic programming principles were not considered, the code was hard to understand and modify. A goal of the team was to make the code easier to change through refactoring. Troubleshooting the application was also challenging. Sometimes it takes a long time to find an issue.

Our goal was to identify signs of bad code quality and find a way to fix them. After refactoring, the speed of adding new features increased, but problems appeared again after a while, and the pace slowed.

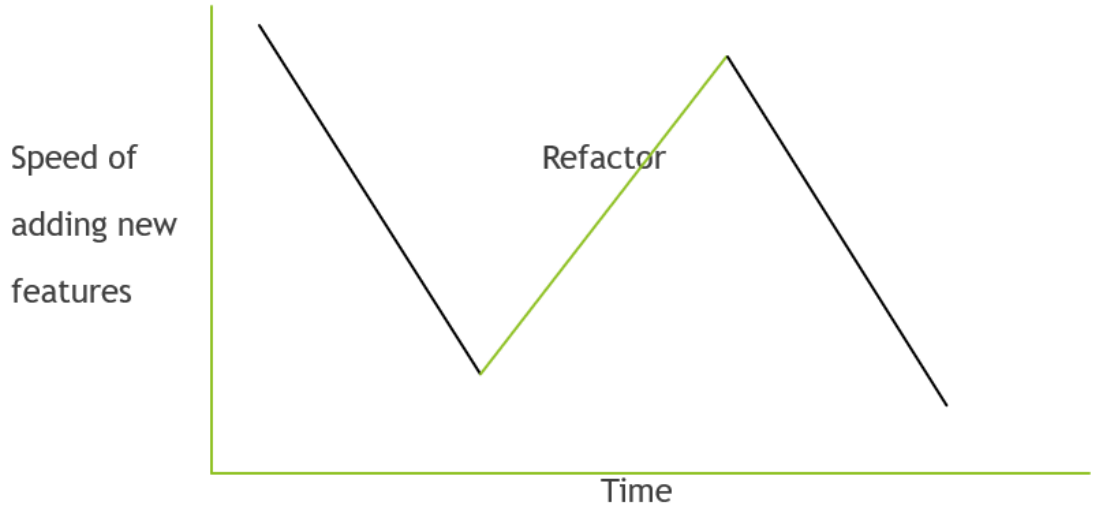


Figure 2: Speed of adding new features after refactoring

Although some complications have been resolved with the refactoring, there are still some significant issues, such as outdated technology Angularjs that Google no longer supports. However, with good architecture and after rewriting, the team is expected to reach an acceptable speed for adding new features.

Defining a clear structure and avoiding strong dependencies are the goals. It is also expected that the complexity of the application will be reduced. As a result, unexpected complexity will become expected complexity.

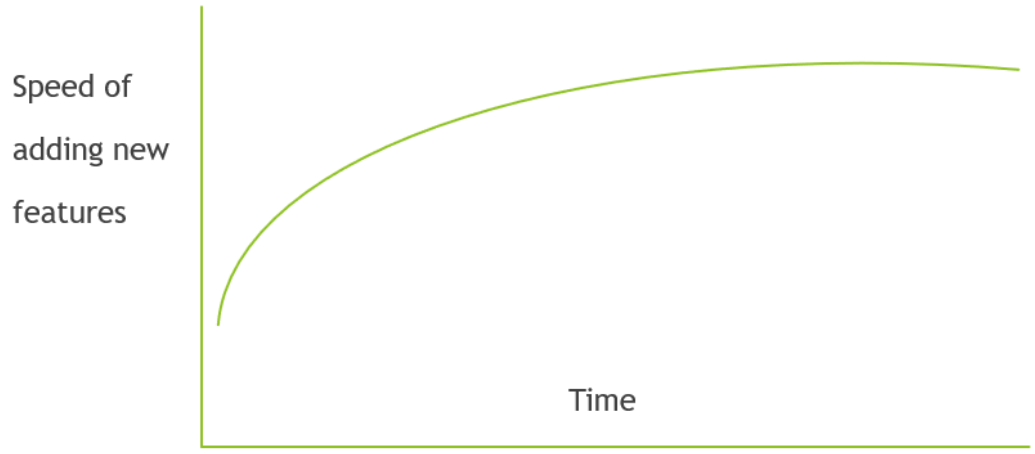


Figure 3: Speed of adding new features with a good architecture [2]

As a result, it was decided to rewrite the application.

1.5 Proof of Concept (POC)

Before starting the process of the development of the application, the topic was brought up for discussion to speed up the development process that was required for creating an application. It can speed up the development process. To start the development phase, it is necessary to identify obstacles. A proof-of-concept application is built to identify these obstacles with the basic functionalities needed for the final application.

During POC, the first topic appeared, Javascript Framework. We need a framework or library with basic written functionality and practical components to create an application.

The first idea is known as Angular, provided by Google. It is not a simple library but a whole framework. Angular offers many functionalities in a framework, what you need for creating a web application. The second idea is known as React, provided by Facebook.

Compared to Angular in react, it is all about Library. React is small and has only basic functionality. According to the team, Angular's clear structure makes code quality easier. React, on the other hand, is easy to learn.

A decision was made to divide team members into two teams, each of which would implement a demo for one week. After a week, both demos were reviewed, and the pros and cons were discussed. After a lengthy discussion on the advantages and disadvantages of both ideas, the decision was made by voting.

The final decision was to use React as a JavaScript library. As a result, the team was satisfied with the decision and rated React as a beneficial tool to use on the project.

1.6 Deadline

Our project started in October and was scheduled to finish by April.

2 Research Methodology

2.1 Aims and Objectives

The primary purpose of this thesis is to examine topics during the development process. Hence, the principal objective of this research is to capture issues and how they change during the development process. The goals can be summarized as follows.

- To observe how the topics change during the development process
- To assess the impact of decisions made on the quality of the project
- To identify the reasons for the emergence of the topics
- To observe the ideas raised in various topics
- To identify which of the proposed ideas are implemented
- To identify the reasons for implementing the proposed ideas
- To identify the reasons for not implementing the proposed ideas
- To examine the advantages and disadvantages of the proposed ideas

2.2 Research Questions

We try to answer the following questions for every topic.

- When and why are they coming up?
- When and why do they fade into the background again?
- Over time, how has the topic changed?
- In each topic, what possible solutions are discussed?
- How and when do the solutions appear? How and when do they disappear?
- How has the developer's opinion changed?

2.3 Expected Outcomes

At the end of the project, it is expected to have many challenges and issues with ideas, but not all of them are needed. This thesis discusses essential and exciting issues during the development process. At the end of the view, we also discuss challenges that have affected the quality of the project. We also see how the decisions made in some cases affected the project.

2.4 Data Collection

This thesis explores critical issues and challenges during the development process. A qualitative research method is used to observe the challenges. The research presents challenges and issues in four categories feature development, strategy, quality, and software structure. Data is collected through observations and documents. All the meetings are recorded to provide different perspectives and opinions on the challenges. It is also essential to illustrate ideas with verbatim quotations.

The software MAXQDA is used to analyze data. The team members are assigned letters from A to J to discuss their ideas and opinions during the thesis. The team members' opinions are collected through an email survey, group channel, or interview questions.

The following scales are used to illustrate team members' opinions on the evaluation of challenges:

- Very Important (4)
- Important (3)
- Less Important (2)
- Not Important (1)
- Do not know (0)

The following scales are used to illustrate team members' opinions on the Expectations or evaluations of ideas:

- Very Helpful (4)
- Helpful (3)
- Neutral (2)
- Not Helpful (1)
- Do not know (0)

3 Feature Development

3.1 Admin Framework

Admin Framework: We need a framework to build an admin panel/dashboard and one that adapts any custom APIs (Application Programming Interface), which can save time in CRUD (Create, Read, Update, Delete) operations, filtering, and creating components for the admin panel. The topic is marked as very important for developers. In the chart below, developer opinions are displayed on how important the issue is for the project.

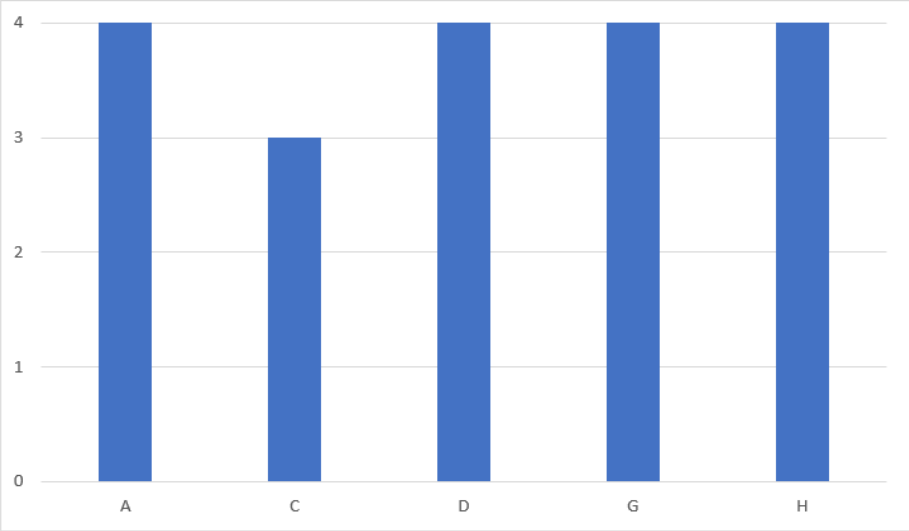


Figure 4: How important is the topic Admin framework?

As the team started to develop the admin panel, the topic was discussed to speed up the development process required for creating an admin panel. An admin framework consists of many written functionality and components that are practical and time-saving in terms of the usual tasks needed to create a dashboard.

At the initiation phase of our project, the team decided on a specific framework known as "Adminbro." As a result, we thought that the topic of framework selection was brought to a conclusion. Unfortunately, after tinkering with the program for a whole week, we realized that "Adminbro" was not the most effective framework for our project.

Consequently, we had to decide to have another framework. After further research, we concluded that "React admin" is our best option for an admin Framework for the moment.

The first idea ("Adminbro") to develop an admin panel was a suggestion from 'G.' Adminbro is a package for creating administration interfaces that can be connected to the application. We only need to provide schemas, and Adminbro generates the user interface.

The second idea, "React admin," came after the first week because our team doubted the decision to select the admin framework. "React admin" is an admin framework that connects to the application programming interface without dealing with entities from our side.

The team's first topic of discussion that team put forward was admin frameworks. The group decided to try Adminbro before selecting it as our main framework.

The team's main problem with "Adminbro" was too much customization. Therefore, the group decided to spend one day developing an example with customization to see if the "React admin" was the right decision.

After using the program for one week, 'A' realized that Adminbro needs much customization. As a result, he changed his opinion of the framework from being very helpful initially to impractical. Contrary to that, he found that react admin was a more valuable and suitable framework for our project.

'G' did not change his opinion and still thinks "Adminbro" is an excellent and helpful framework for simple scenarios. However, as the program is getting complicated, it is hard to use Adminbro because, first, the team does not have enough experience with Adminbro, and second the documentation is not good.

After facing some issues during the development using "Adminbro," 'C' mentioned that the documentation is terrible, and the team is wasting time cause of the documentation. He changed his opinion about "React admin" after using it from neutral to helpful.

The team began using Adminbro at the project's initial phase in collaboration with admin frameworks. After a week, the entire team came down to a unanimous decision that it could slow our work rate and effectiveness due to its high levels of customization and lack of practicality. We also learned that developers do not favor Adminbro for customization since each task would be highly time-consuming in learning how to customize the framework.

The result of the first assessment that the team reached using "React admin" was the library's flexibility. The team figured out that they needed a more flexible library which could also assist them in implementing customization.

The result of the second assessment was that the library needed a good documentation system. As mentioned before, the more complicated a project gets, the more there is a need for customization. Customization is only possible with a well-documented library.

As stated in this report, the end decision was to have React-admin as our main admin framework because it is much more practical than Adminbro.

3.2 State Management

Communication and data sharing between components require a library or simply a way to create contact. It creates a concrete data structure to represent the state of the application. Using state management data flows from application to state and vice versa. State management results in a clear data flow. Based on the developer's ideas, this is one of the most critical parts of the application. In the chart below, developer opinions are displayed on How important is the topic “state management.”

Figure 5: How important is the topic “state management”?

As developer G was looking for an approach to share data between components, the topic of state management came up. He decided to use a process known as "Zustand.” After a couple of days, the issue appeared again as the team doubted the decision made regarding state management. The topic was discussed to discuss all possible libraries that might be useful for implementing state management. The second idea, "Apollo client,” appeared as another solution.

The first library, "Zustand," is easy to use but based on experience, developers do not like the first approach's structure and believe that at a later point in time, other issues could arise from it as a result.

In the chart below, developer opinions are displayed on how useful it could be "Zustand" for the project.

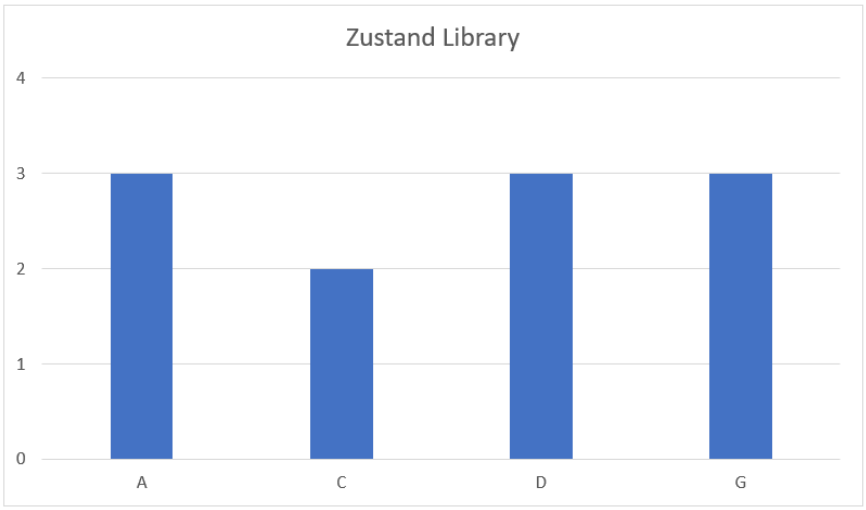


Figure 6: How useful could be the library "Zustand"?

The second approach, "Apollo client," brings more flexibility, and the team likes it more because a similar library is being used in the project. H mentioned that Both approaches could be helpful for the project, but the second approach using the library "Apollo client" is more compatible with the libraries used in the project.

G mentioned that the second approach brings more performance to the application.

In the chart below, developer opinions are displayed on how helpful “Apollo-client” could be for the project.

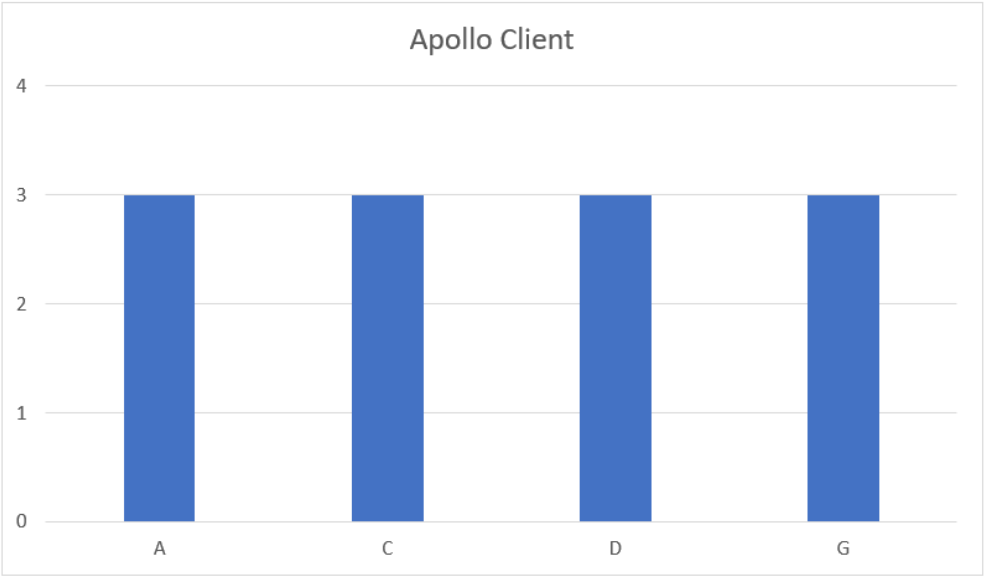


Figure 7: How useful could be the idea of "Apollo client"?

The team decided to use "Apollo client" for state management as it improves performance compared to the other approach. It is also more compatible with the project.

4 Discipline/Quality

4.1 Software Testing

Tests are used to ensure that software products and applications perform as expected. Through testing, bugs are prevented, development costs are reduced, and performance is improved.

The topic appeared at the initiation phase of our project on 21.10.2021 to discuss the team's testing process. The issue was addressed in the application's initiation phase to detect bugs earlier and have a more stable application. Consequently, the team added to the action plan the idea of testing the changes before adding them to the main project.

The topic came up again on 25.11.2022 to see the team's improvement. Unfortunately, this topic did not get any attention. The group puts more afford into developing features, and testing is getting ignored. The team discussed paying more attention to testing and added the same idea of testing the changes before adding them to the main project to the action plan.

The topic appeared again on 27.01.2022 as the team saw many bugs due to ignoring the issue. G mentioned that Tests are not quite in place yet. We see this in this Sprint based on how many bugs we have. We should have appropriately tested in the previous Sprints.

The first idea that the team puts forward for discussion to prevent and catch bugs in the early stages is End-to-End testing (E2E testing).

4.1.1 End-to-End Testing

An E2E test simulates the real user scenario and validates the integration and data integrity of the system under test and its components by testing its flow from start to end.

In the chart below, developer opinions are displayed on how helpful the idea of E2E testing might be for the project.

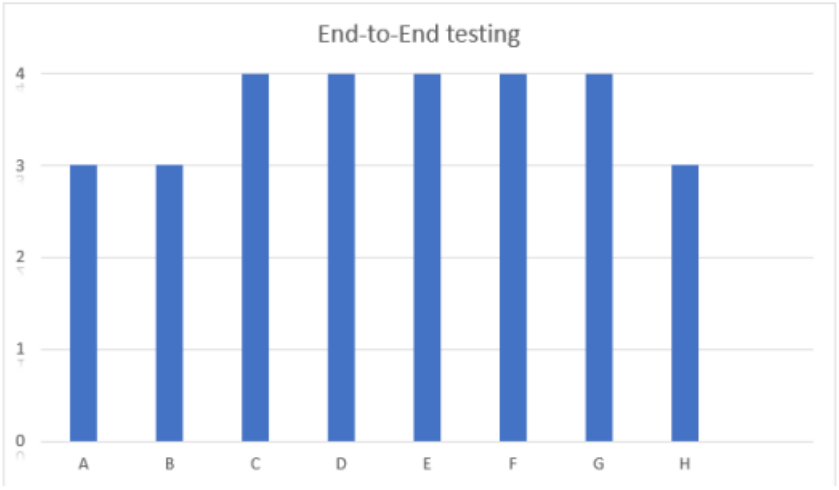


Figure 8: How helpful could end-to-end testing be for the project?

63% of team members (5 members) voted this idea very helpful. The other 37% of team members (3 members) voted this idea as practical.

A discussion of the pros and cons of E2E testing is provided below.

- **Detect Bugs earlier**: In End-to-End testing, the software is tested after changes are made to the project, which means the bugs can be detected at the early stages of the project before going to the final stages.
- **E2E testing Coverage**: The efficiency of E2E testing still depends on the quality of our tests.
- **Useful for Refactoring**: As the Product owner (A) mentioned that refactoring is affecting large parts of the application and can lead to more bugs, E2E testing can be beneficial at this point after refactoring. Automated Test tests the application automatically to ensure that everything is working fine.
- **Spend time on writing tests**: On the other hand, we need to spend time writing End-to-end testing.

As a result, the team decided to focus more on writing E2E tests for the most critical parts of the application. The first task for implementing E2E testing has been added to the board with an estimated eight story points for the registration flow. A setup for E2E testing has been completed, and basic test coverage has been implemented for the registration flow.

Unfortunately, the idea disappeared after that as the team focused on developing new features.

The idea appeared again after finishing the application's development phase on 27.06.2022, during a meeting, as the team completed all features and fixed most of the bugs. It was decided to put more afford before making the application online to ensure that new changes will not cause any more bugs and make changes in technical debt safe. As UI changes often, we need an approach to ensuring application quality. As a result, 29 story points were added to the board regarding test coverage in the most critical parts of the application.

Here is a chart that shows how many story points are invested in E2E testing during development.

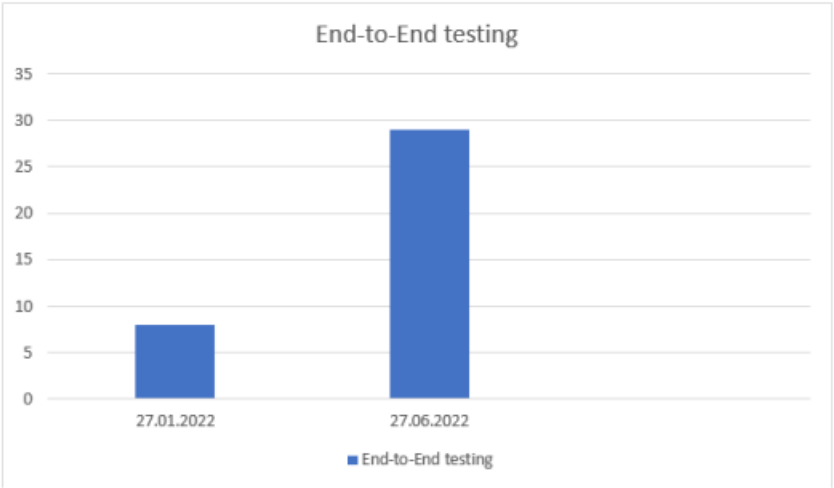


Figure 9: Story points invested in End-to-End testing

The topic returned after launching the product. Some bugs were reported in the application, and the group worked on them. It is more critical this time that the new changes can not cause any more bugs. As we do not have so much time to go through all the changes and it needs much time, ten more story points were added to the board to increase automated testing for the project.

Consequently, the team feels more secure after implementing E2E testing that key functionalities work as they should.

4.1.2 Regression Testing

Regression testing aims to ensure an application's functionality remains the same after any code changes are made. After each update, regression testing is applied whenever new code is added to ensure the system remains sustainable.

Regression testing can be performed only on those parts affected by the changes. To reduce regression testing time and effort, related areas can be selected, and relevant test cases can be applied.

All existing test suites are tested during complete regression testing. This is the most reliable method for detecting and resolving bugs. However, it requires considerable time.

The complete regression approach also makes sense when the application has been adapted to a new platform.

In the chart below, developer opinions are displayed on how helpful the idea of regression testing might be for the project.

Figure 10: How helpful could regression testing be?

The idea appeared on 27.01.2022 in a retrospective meeting as C mentioned that we need to take some time to do regression testing from time to time as we are deploying more new changes. Since the team focused more on developing new features, the idea disappeared then.

The idea appeared again on 29.04.2022 after all features had been almost completed. The team scheduled regression testing for every week until 17.06.2022. Consequently, many bugs were found during the first testing session. It helped a lot in finding bugs and having a stable application.

4.1.3 Unit Testing

A unit test is an independent application part that can be tested. The main objective of unit testing is to isolate written code to test and determine if it works as intended. Unit

tests are performed by the software developers and sometimes by QA specialists during the development process.

The importance of unit testing in the development process lies in its ability to detect flaws in code early before they become more challenging to see later in the testing process.

As a result of finding some bugs in the application, the team came up with the idea on 07.02.2022. G mentioned that unit testing would ensure our services will have tests and prevent us from producing more bugs. Additionally, he suggested that we should not approve changes to the main project without unit testing. For complex parts of the application, testing can be added to the estimation so it will not be postponed as the team develops new features in a rush.

H mentioned that we do not need 100% coverage of tests, but we require 100% coverage when dealing with business logic or adding or updating features.

As a result, the project covers about 55 percent of unit testing in services.

4.2 Acceptance Criteria

A User Story must meet certain conditions to be accepted by a user, customer, or other stakeholders.

Product owners write User Stories and discuss them with the development team during the development process. Generally, Acceptance Criteria are statements or conditions describing an application's characteristics. A state is either satisfied or not satisfied. There is no partial acceptance.

Depending on the product and application under development, Acceptance Criteria can also define boundaries and constraints of the User Story that determine whether the Story is working as expected and ready to be accepted. Acceptance Criteria need to be defined before the iteration and development of the Story starts but can be refined during the Sprint based on the product owner's feedback. [3]

A User Story usually doesn't provide enough details on the functionality and feature to the development team to start designing and coding; developers use the User Story and Acceptance Criteria to start any development work. [3]

Acceptance Criteria need to be written in a language that customers, product owners, and the development team can easily understand. When defining Acceptance Criteria, there should not be any ambiguity regarding the expected outcome and expected behaviors. [3]

The topic appeared at the initiation phase of our project on 21.10.2021 in a retrospective meeting. The issue was discussed because developers need more information about user Stories, which leads to bugs.

C mentioned that we need to have more clear conditions of Acceptance Criteria. G said that the lack of Acceptance Criteria increases the complexity because some states are not considered in estimation meetings. The topic has not been taken into consideration and disappeared.

The topic appeared later on 30.03.2022 after we had some bugs in the application. B mentioned that tasks should have clear Acceptance Criteria to prevent producing more bugs.

In the chart below, developer opinions are displayed on how helpful the idea of improving Acceptance Criteria might be for the project.

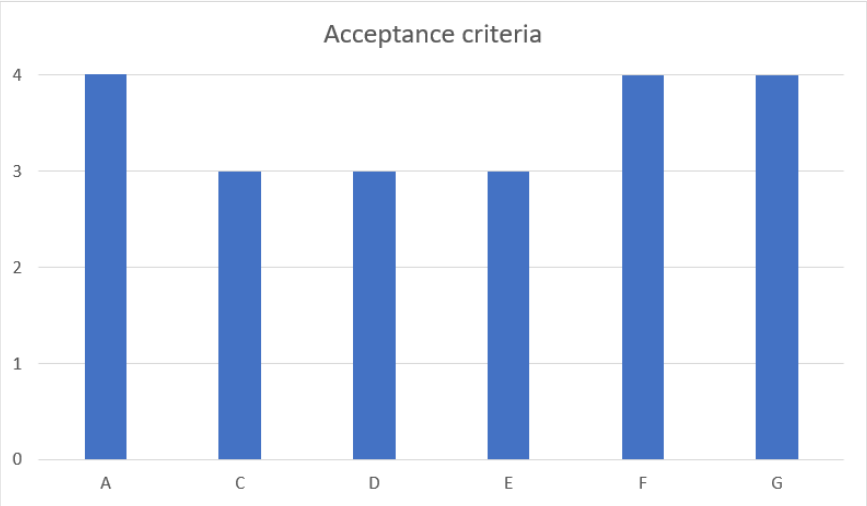


Figure 11: How useful could be the idea of improving Acceptance criteria?

Some bugs were produced during the project due to a lack of clear Acceptance Criteria at the beginning and middle of the project.

4.3 Definition of Done (DOD)

DoD refers to the team's agreement that all tasks or activities have been completed for a Story, Sprint, or Release. A set of conditions applies across all Stories, Sprints, and Releases indicating that no more work is needed. It is a set of standard conditions across all Stories, Sprints, and Releases that no more work is left to be done for a Story, Sprint, or Release. A good DoD helps teams determine when a product increment or a task can be marked as completed.

The DoD ensures that the team adheres to a shared understanding of completion. Unlike Acceptance Criteria, the Definition of Done is not specific to a Story or a feature but is defined in such a way that it applies to all Stories. It is also represented at a Sprint and Release level and applies to all Sprints and Releases on which the team works. [3]

Like the Acceptance Criteria, the DOD needs to be written in simple language that the development team, product owner, and stakeholders can clearly understand. [3]

Before starting the project's development phase, the topic was discussed. During the meeting, we discussed items that should be included in DOD, and the documentation was saved. The idea was to update and adhere to the DOD in every Sprint. After some time, the concept of defining and adhering to DOD vanished utterly.

The topic was returned in a retrospective meeting on 27.01.2022 after some bugs were discovered. G mentioned that I am missing the definition of done. The case has not been taken into consideration and disappeared.

As a result of this topic, E mentioned that we did not use DOD, and we had some problems. It could give us a frame and a guideline. C also said that it should be mandatory to adhere to DOD. The quality of the project depends on that.

In the chart below, developer opinions are displayed on how helpful the idea of updating and adhering to DOD might be for the project.

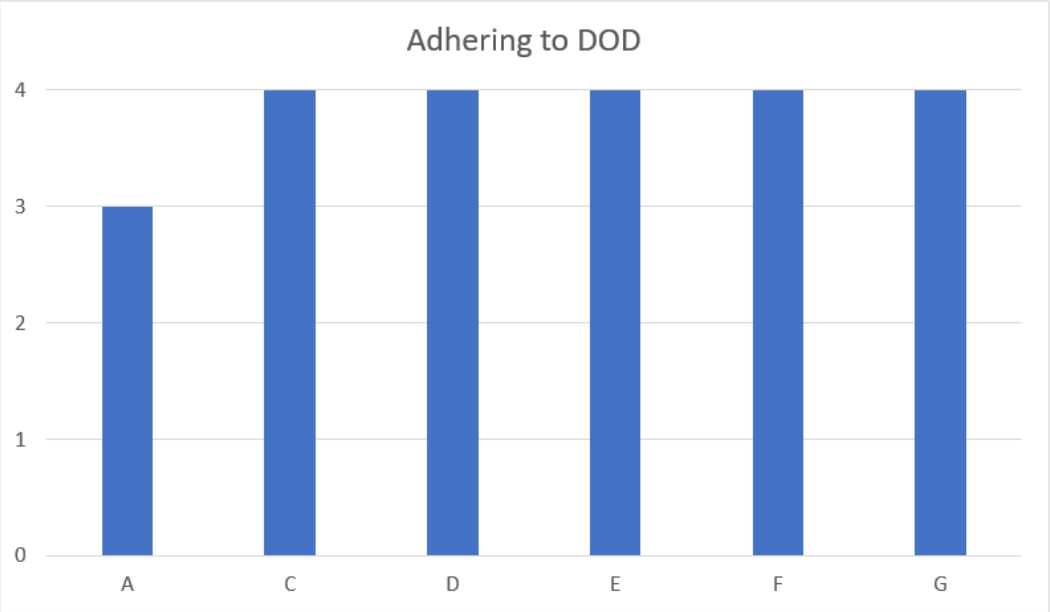


Figure 12: How useful could be the idea of adhering to DOD?

Consequently, we do not know what done means and lose transparency.

4.4 Code Review

Code Review, also known as Peer Code Review, is the act of consciously and systematically convening with one's fellow programmers to check each other's code for mistakes and has been repeatedly shown to accelerate and streamline the process of software development like few other practices can. [4]

In the chart below, developer opinions are displayed on how helpful the idea of improving code review might be for the project.

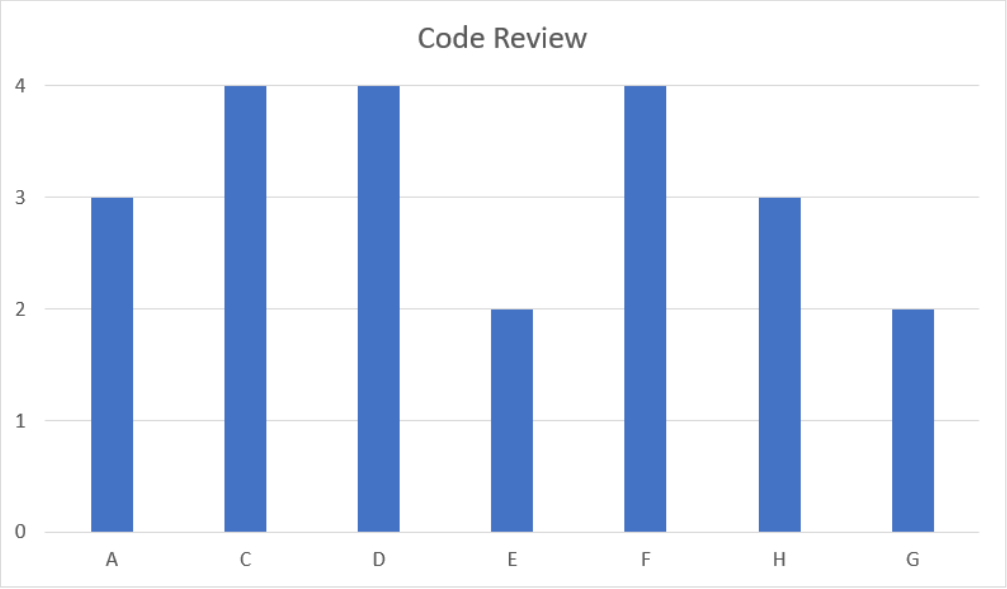


Figure 13: How helpful could the idea of improving code review be?

The topic appeared on 21.10.2021 in a Sprint retrospective meeting to catch bugs before adding to the main project. It is expected that this topic will have a positive impact on code quality. It is also likely to see bugs earlier.

After the discussion, the team decided to add the idea of improving code review to the action plan to monitor this topic as one of the critical topics to reduce bugs produced in the application.

The topic appeared again on 25.11.2021 in a Sprint retrospective meeting to see the team's changes and improvements in the last four weeks. As the developer(C) mentioned, the team thinks it takes time to review, and the topic is getting ignored due to many tasks.

The topic was brought up for discussion again to find a better solution. As a result, H suggested that the problem could be fixed if everyone went through his changes

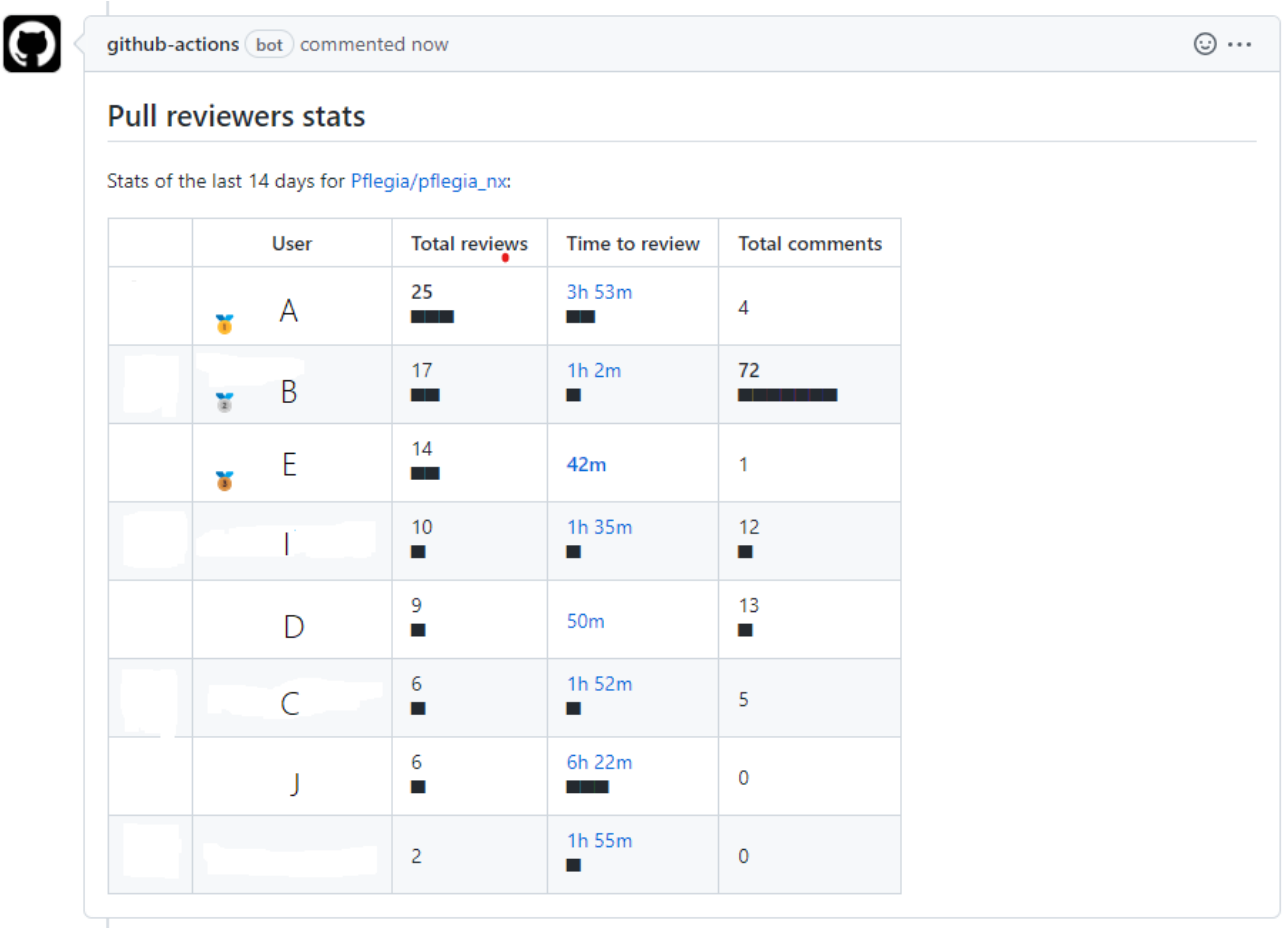
before requesting to review. Lots of issues are avoidable due to reviewing your changes. As a result, the team added the idea of checking your PR and Making a call for complex changes to the action plan.

The topic came up again on 27.01.2022 during a Sprint retrospective to see what the team had accomplished. E mentioned that we are not doing this enough, which is why we have many bugs. The group discussed again to improve; as C said, when reviewing new changes requested, check for the context and where it will be used. E mentioned that one person could focus the week on reviewing the recent changes requested. The topic was added again to the action plan.

On 17.02.2022, the topic was brought up again during a Sprint retrospective. D mentioned that the team is improving on this topic, but we can still improve.

Then B sent an article about the topic and how to fix the problem on 12.03.2022 in the team channel. He started by himself to check all the changes requested to the main project. Mentioned all the people to respect the topic and the ideas that the team put forward for this topic.

The picture below shows the total comments on 29.04.2022 in the last two weeks.



github-actions bot commented now

Pull reviewers stats

Stats of the last 14 days for Pflegia/pflegia_nx:

	User	Total reviews	Time to review	Total comments
	A	25	3h 53m	4
	B	17	1h 2m	72
	E	14	42m	1
	I	10	1h 35m	12
	D	9	50m	13
	C	6	1h 52m	5
	J	6	6h 22m	0
		2	1h 55m	0

Figure 14: Total comments in code review

After Joining B on the team, the number of comments significantly increased.

As a result, the team could catch many bugs before changes were added to the main project. Also, the code quality improved as the team started to improve by reviewing requested changes.

4.5 Technical Debt

The term technical debt refers to what happens when development teams rush through the delivery of a piece of functionality that needs to be refactored later. It is the result of prioritizing speed over quality.

Technical debt happens when shortcuts are taken in writing code so that the goal is achieved faster but at the cost of being more challenging to maintain code. We can accomplish more today than we usually could, but we will pay a higher price later. [5]

There was a discussion on 18.02.2022 about how to deal with technical debt. G mentioned that if you are working on the code, try to improve it. In the case of a hard-to-maintain script, if you are already working on it, why not make it easier for the next developer who will work on it?

Due to time constraints, the team decided to create a file and track all technical debts. They can be fixed later. A few examples of technical debts in the project are naming conventions, removing duplicate code, improving performance, and reducing code complexity.

The topic returned after the development phase on 29.05.2022 to find a solution for technical debts. All technical debts have been cleared, and the developer has requested a review to ensure that the new changes will not cause any more bugs. We ended up with seventy-eight files that were changed.

A mentioned that the application is stable and that the changes might cause more bugs. C said that all developers must go through all the changes before applying these changes. Applying these changes is impossible since the deadline is in two weeks. To ensure that new changes will not cause any more bugs, the team put more effort into writing E2E tests and improving test coverage.

Ultimately, we decided to break down the changes into multiple small parts. The team put priority on naming conventions. Naming conventions are general rules applied when creating text scripts for software programming. They have many different purposes, such as adding clarity and uniformity to scripts, readability for third-party applications, and functionality in specific languages and applications. They range from capitalization and punctuation to adding symbols and identifiers to signify certain functions. [6] As the project matures, it is important to have a naming convention for the whole application.

The next part is improving query performance. This part has not been considered, and before launching, it was returned. As many users will use the application, responding to all requested data as fast as possible without crashing the application is critical. The team used load testing software to simulate the application's demand. Load testing tests the application under lots of conditions and requests and determines if the response rate is satisfied or not. After load testing, we figured out that we needed to put more afford into this part of the application as it crashed in some cases. We were then optimizing queries added to the board to get done. After optimizing the queries, the response rate improved a lot. Consequently, the application performs well in responding the requests.

5 Strategy

5.1 Minimum Viable Product

A minimum viable product (MVP) is a version of a product with just enough features to be usable by early customers who can then provide feedback for future product development. [7]

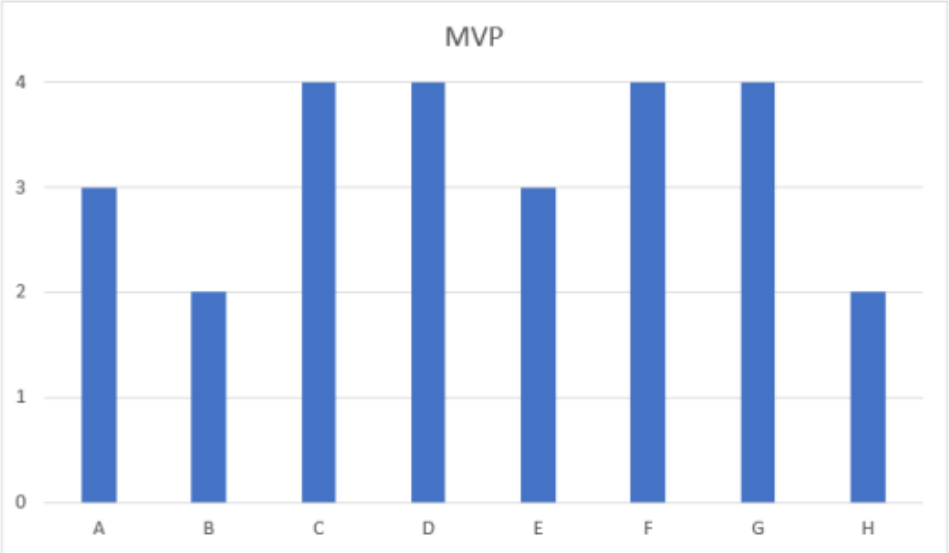


Figure 15: How helpful could MVP be?

The idea appeared after the team looked for a solution for prioritizing bugs. By the end of the Sprint, all tasks should be tested, and a full feature will be available. By the end of the Sprint, all bugs have been fixed and not postponed.

C mentioned that There is a severe gap in testing bugs, the Previous Sprints are getting tested now, and bugs are coming in the next Sprint or later. That is not how Scrum works. Using this approach, the team expects a fully working feature product at the end of every Sprint.

On the other side, other developers believe in a balance here. We need to prioritize bugs so that urgent ones need to be fixed as soon as possible, but minor ones can be fixed later.

In the end, the team decided to have three prioritizations for bugs:

- Major: Breaking bugs that should be fixed as soon as possible.
- Normal: Can be fixed later.
- Minor: Can be fixed later

As a result of this topic, bugs are postponed to the next Sprints, which means features are not entirely done. Also, some codes are written above non-tested code.

5.2 Acceptance by Product Owner (PO)

The team realized some bugs were being produced because of misunderstandings between developers and the product owner. The team changed how new changes are applied to the main project to ensure that all changes are working as expected.

Previously, new changes could be approved and reviewed by other developers, but now they must be approved by PO. As Po is the most knowledgeable about the project, he can catch it before adding it to the main project if something is unclear or missing. Other developers still review new changes, but PO is the only one who approves them.

E mentioned that it has both benefits and downsides. Some bugs can be caught earlier, but we must wait a while. Others believe it should be a mandatory requirement and is necessary for the project.

The chart below displays developer opinions on how useful it might be if only PO could approve new changes.

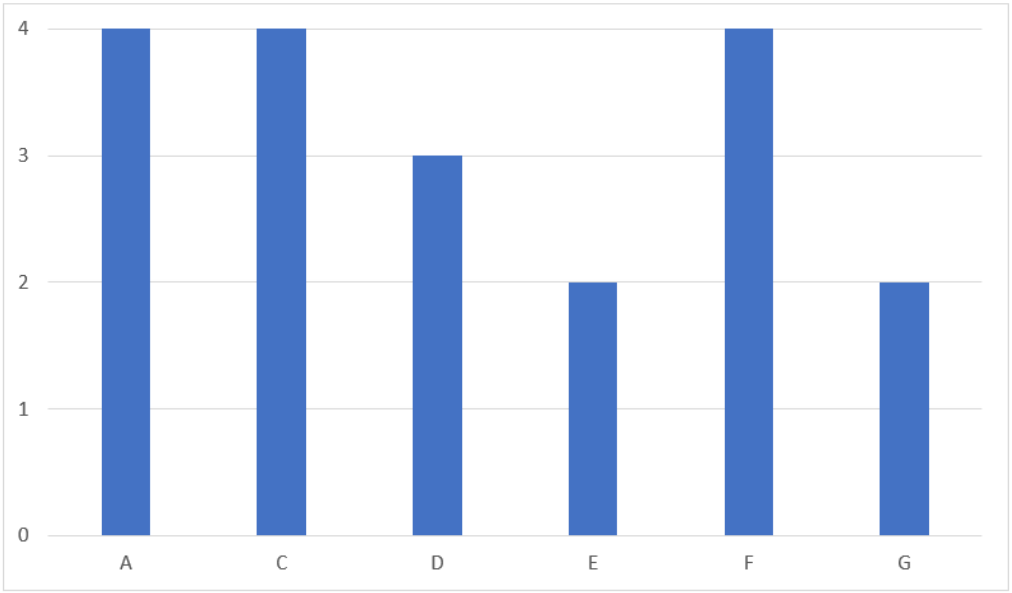


Figure 16: how useful might it be if only PO could approve new changes?

5.3 Feature Demo

As part of the feature demo, the team gathers to review bugs and features completed in the last week and determine whether additional changes are needed. The meeting takes one hour, and the whole team and PO are included.

The topic appeared on 02.03.2022 to assess and review the last changes. Developers can show product owners the changes they have implemented. It is also helpful to give everyone the previous update of the application. Every developer should explain the idea and how they implemented the feature. It also gives everyone a better technical overview and keeps track of ideas in every feature.

After finishing most of the features at the end of April, stakeholders also joined the meeting to see the improvement regularly. As the requirements sometimes change, joining stakeholders in the meeting helped the team to determine which features still need extra work. It also gave the team another view of the application and how the customers work and use it.

After joining stakeholders to the team, the technical part was removed from the meeting because, first, the meeting took long, and stakeholders could not understand the practical aspects and asked to remove this part from the meeting.

In the picture below, team members' opinions are displayed on how helpful the feature demo for the project would be.

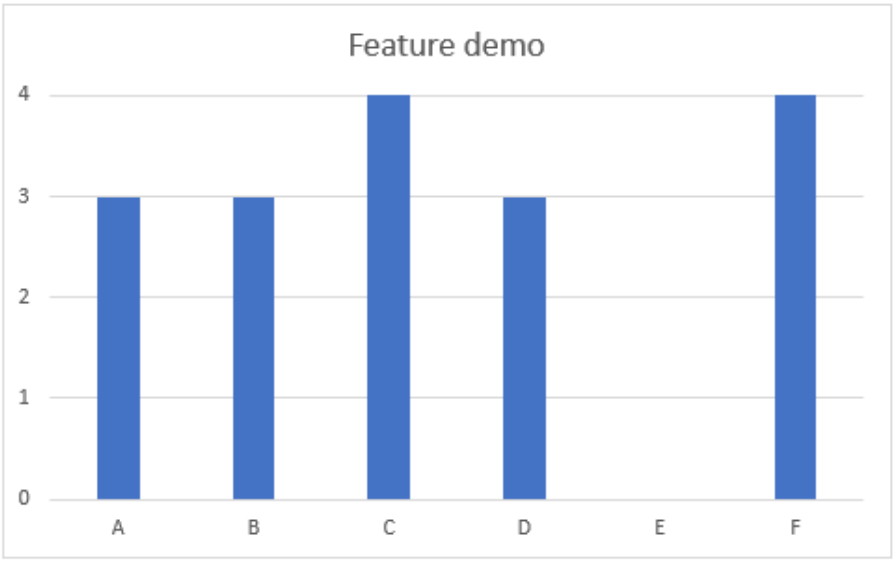


Figure 17: How helpful would be the feature demo?

As an evaluation, C mentioned that we need to have a non-technical meeting to present new features and changes.

Feature demo helped to get continuous feedback from stakeholders and remove the misunderstanding between the team and stakeholders. It could be more helpful if the stakeholders could join the meeting from the beginning of the project.

6 Software Structure

6.1 Refactoring

Refactoring is a systematic process of improving code without creating new functionality that can transform a mess into clean code and simple design. [8]

Clean code is code that is easy to read, understand and maintain. Clean code makes software development predictable and increases the quality of the resulting product. [8]

Dirty code results from inexperience multiplied by tight deadlines, mismanagement, and nasty shortcuts during development. [8]

The chart below shows how important is the topic of refactoring for the project.

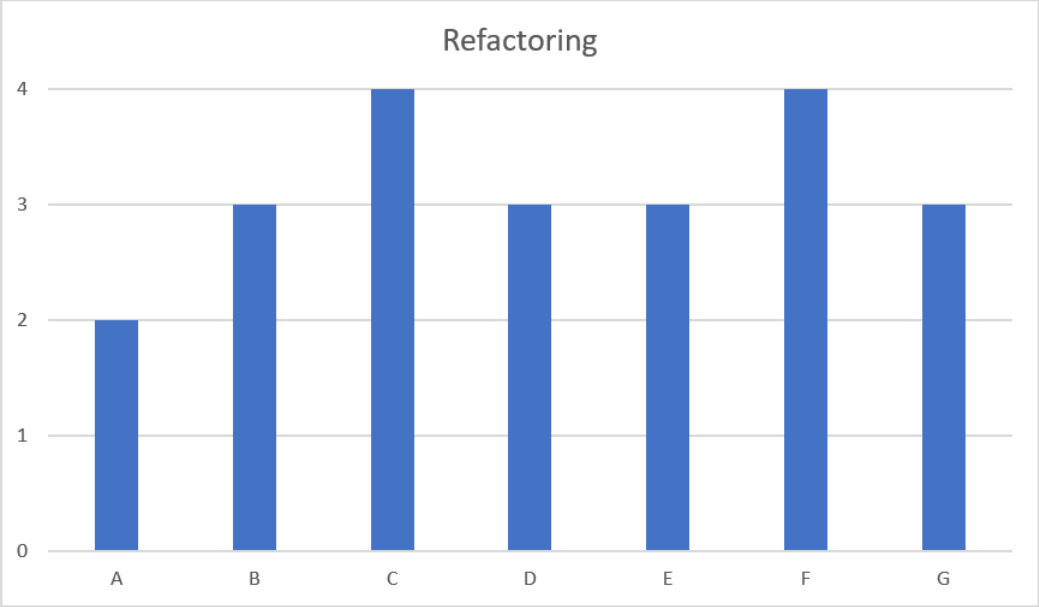


Figure 18: How important is the topic refactoring?

Two developers marked the topic as very important. Four developers marked it as important, and A marked it as less important.

The topic appeared during a meeting on 27.01.2022 as D mentioned that the code starts to be like spaghetti: you do some changes in one place and have side effects in another. That is why we have lots of bugs right now. A mentioned that we have some changes, which works, but three other places relying on the same functionality are broken.

Then G suggested the idea of the "Test first approach." Refactoring affects a large part of the application, leading to more bugs. This idea appeared to make sure after refactoring that everything works as expected.

The chart below shows how helpful this idea could be for the project.

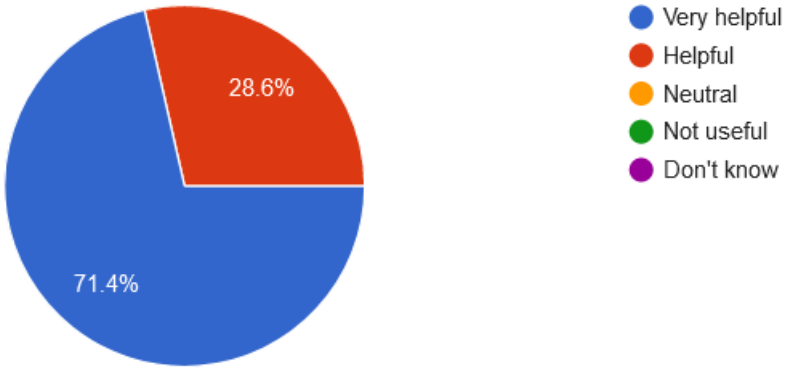


Figure 19: How helpful could be "Test first approach" be for refactoring?

The team started to increase test coverage, but the idea disappeared after one week.

The topic appeared again on 18.02.2022. The team was looking for a unique approach to change the style of SVG s in the whole website. After applying some changes, the team figured out that the change caused some UI problems. Then the idea of breaking down refactoring into smaller pieces appeared. It means we break the changes into small pieces and make them more manageable. Then the team divided the refactoring into two parts.

The chart below shows how helpful this idea could be for the project.

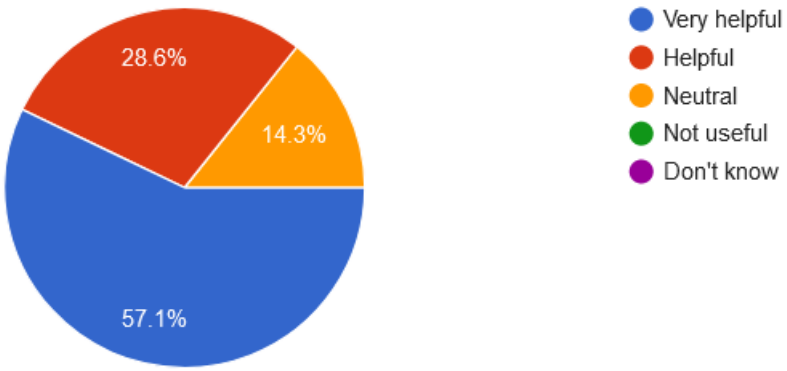


Figure 20: How helpful could the idea of breaking down refactoring into smaller pieces be?

The easy changes are applied in the first part, which does not cause any other bugs. This part was added directly to the board and done in the following days.

There are some complex changes in the second part, and the team needs to check all the changes. This part needs much more time and is added to the project. It appeared again on 29.06.2022 and was added to the board to complete.

6.2 Technical Documentation

Technical documentation is any document that explains a product's functionality or architecture. It is not just about capturing information. It is about presenting it in a way that is easy to read, usable, and helpful for the audience.

The chart below shows how important is the topic documentation for the project. Five developers marked the topic as very important and the other three as important.

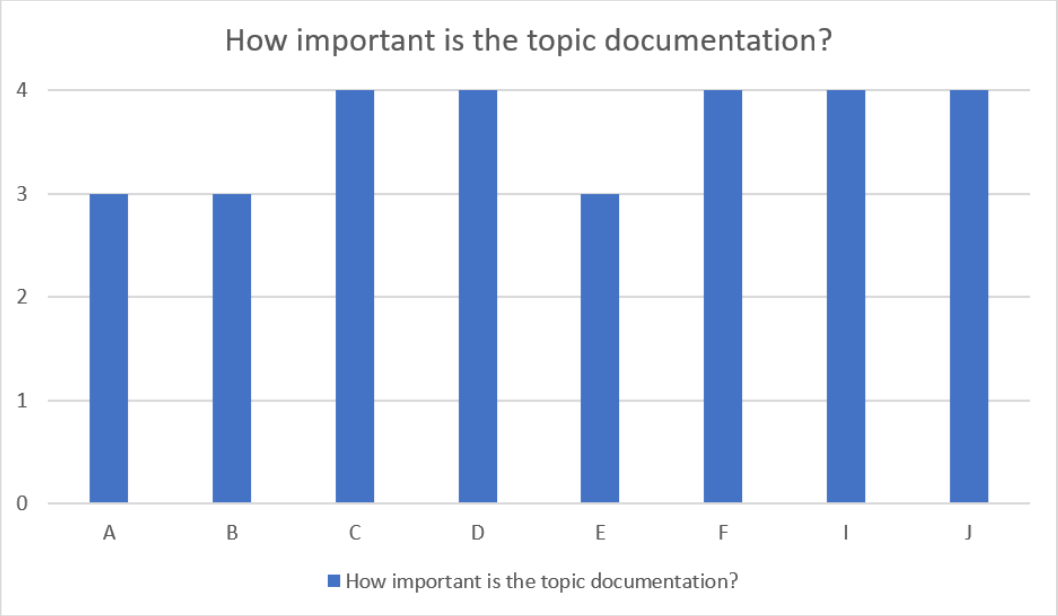


Figure 21: How important is the topic documentation for the project?

The topic was brought up for discussion on 25.11.2021 in a meeting. First, the team discussed why good documentation is necessary. The following points are mentioned.

- There is a need for proper documentation because someone is more focused on certain application parts, and the team loses the concept in those areas.
- Maintaining good software requires documentation.
- The documentation can help us better understand some features of the application.

As a result, to improve documentation, the team added the idea of adding code comments to the action plan. This idea is marked as helpful for developers.

During a meeting on 27.01.2022, the topic was discussed again. Since the team was more focused on developing features, the idea of adding code comments has not been taken into consideration. D mentioned that the lack of documentation would not help maintain the future code.

Additionally, the following ideas are discussed to improve the technical documentation.

1. Write "Readme files": The idea was to keep track of project details to make the code more understandable. This idea is marked as helpful for developers.

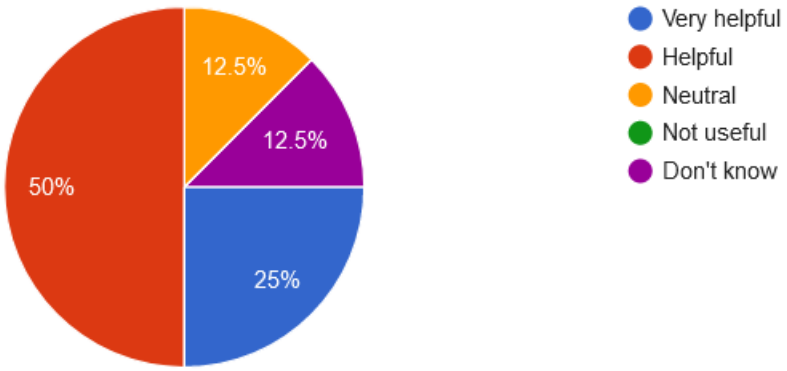


Figure 22: How helpful would writing “Readme files” be?

2. Add "Confluence pages": The project's important information can be saved here. This idea is marked as helpful for developers.

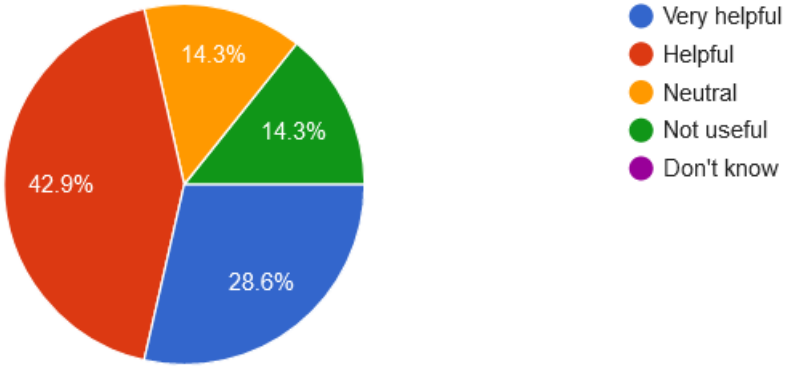


Figure 23: How useful would it be to add "Confluence pages"?

3. Everyone recording one video in a Sprint describing a part of the app: This idea was a suggestion from A. This idea appears to keep track of ideas in every aspect of the application. It is expected that at the end of the project, each feature will have a video that explains how it works. This idea is marked as neutral for developers.

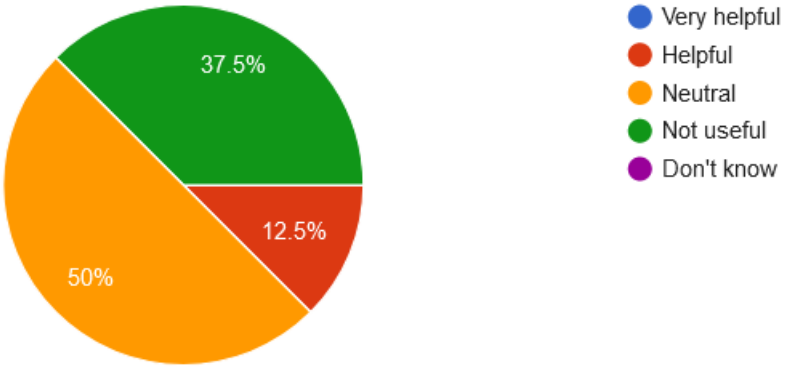


Figure 24: How useful would it be if everyone recorded one video in a Sprint describing a part of the app?

As a result, the idea of recording a video is added to the action plan to improve the documentation.

On 17.02.2022, the topic was again discussed during a meeting to see if the team made any improvements. There was no consideration given to the idea of everyone recording one video in a Sprint because it seemed impractical.

D mentioned that we need to consider documentation time when estimating stories.

The topic appeared again at the end of the project when developer G left the company. C mentioned that he must work on the part of the application that developer G already worked on, but it is hard to understand because no documentation is available.

7 Evaluation

During the evaluation, we will discuss the decisions made in the project and how they affected it.

7.1 Software Development Life Cycle (SDLC)

Software Development Lifecycle refers to the steps involved in developing and modifying software products. SDLC is a process for developing a software project within a software organization. It describes developing, maintaining, replacing, and altering software. Software development processes and software quality are improved using the life cycle methodology. [9]

A typical SDLC is illustrated in the following figure.

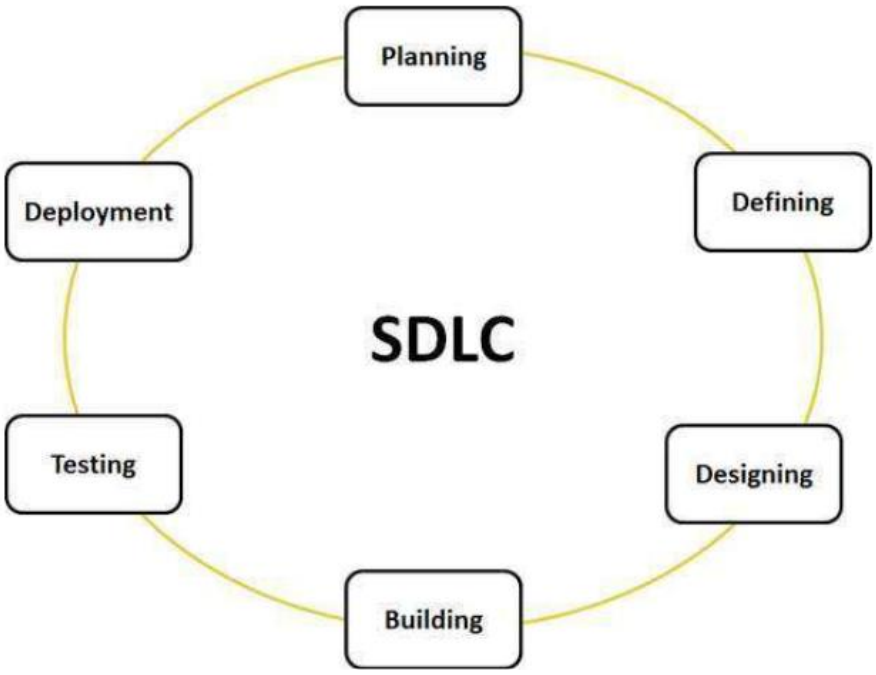


Figure 25: Software Development Life Cycle [9]

7.1.1 Agile Methodology

The term agile stands for moving quickly. Agile methodology is a lightweight methodology for software development. The agile model believes every project needs to be handled differently, and the existing methods must be tailored to best suit the project requirements. In agile development, rather than a single large process model

implemented in conventional SDLC, the development life cycle is divided into smaller parts, called increments or iterations, in which each of these increments touches on each of the conventional phases of development. An iterative approach is taken, and a working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer. [9]

Here is a graphical illustration of the Agile Model:

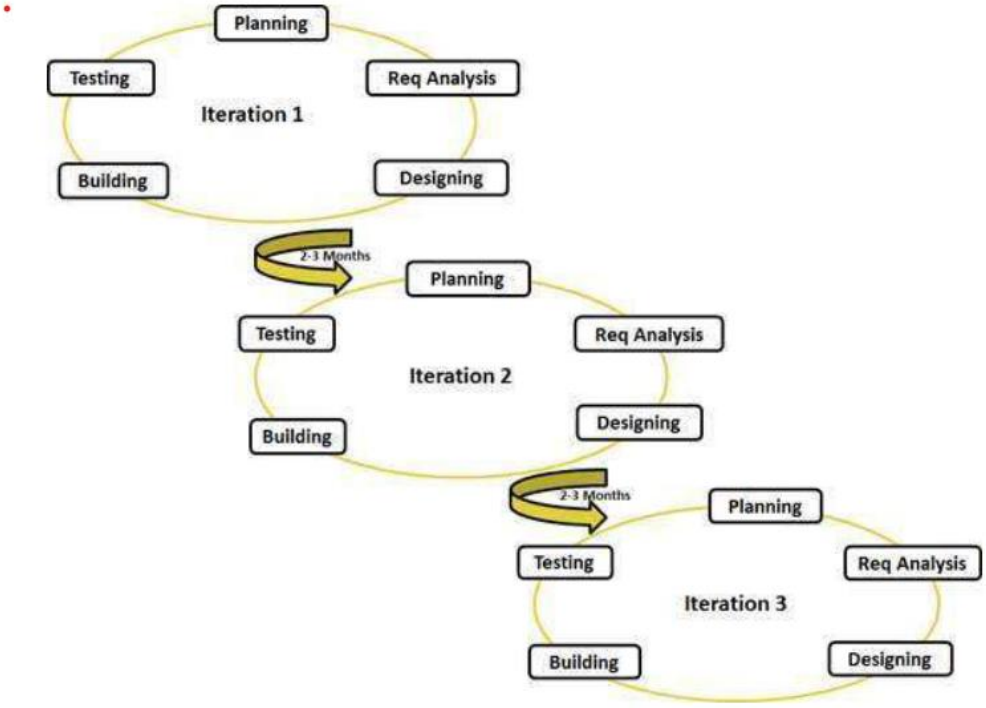


Figure 26: Agile Methodology [9]

7.1.2 Scrum

Scrum is a framework that allows teams to develop software incrementally and iteratively. Within a team-based development environment, Scrum focuses particularly on task management. Following Scrum's basic principles can solve problems and rapidly deliver valuable software. Team members decided to use Scrum as an agile methodology.

7.1.2.1 Testing

Testing is an essential part of the agile software development process. It begins even before development begins in an agile methodology, unlike previous software methodologies, which took place after development was completed. Continual testing is an essential part of agile development, which provides an ongoing feedback loop to the development process.

Every Sprint in Scrum aims to produce a delivery-ready product increment. Testing of the software must be completed before the Sprint ends.

One of the most prevalent anti-patterns identified in a study of 18 teams from 11 companies is the lack of testing and test automation during iteration.

As a result of the MVP topic, the team decided to prioritize bugs into critical, regular, and minor bugs. Major bugs are added directly to the Sprint, but other bugs come later.

C mentioned that there is a serious gap in testing bugs, the Previous Sprints are getting tested now, and bugs are coming in the following Sprints or later. Testing will be postponed to the next Sprints after the development phase, showing the team's "Waterfall-ish" thinking.

Scrum recommends that all software testing is done within the Sprint that creates the software to enable the shipping of a working product increment. [8]

According to [8], new code may be written on top of non-tested code when testing takes place in the next sprints. In addition, the product is not always in a 'potentially shippable' state as it is still non-tested. As test results come in later, bugs must be fixed long after the code is written. A bug might be found after several weeks after the feature was implemented. This leads to a situation where a developer might need some time to recall how the feature was implemented. This increases the cognitive load of developers. Furthermore, hand-offs to testers and back to developers cause wait states where no value is added to the story. Hand-offs also cause disruptions; additional communication is required to explain the defect to a developer.

The project's development phase ended mostly by the end of March 2022. On 06.04.2022, the team started to test the application. We discovered during testing that most features were not working as expected. Most of the features which had been working before were broken.

The chart below shows bugs created and resolved from 25 April to 25 July. The x-axis shows the time, and the y-axis indicates the number of bugs created and resolved in this period. The team has spent ten Sprints fixing bugs.

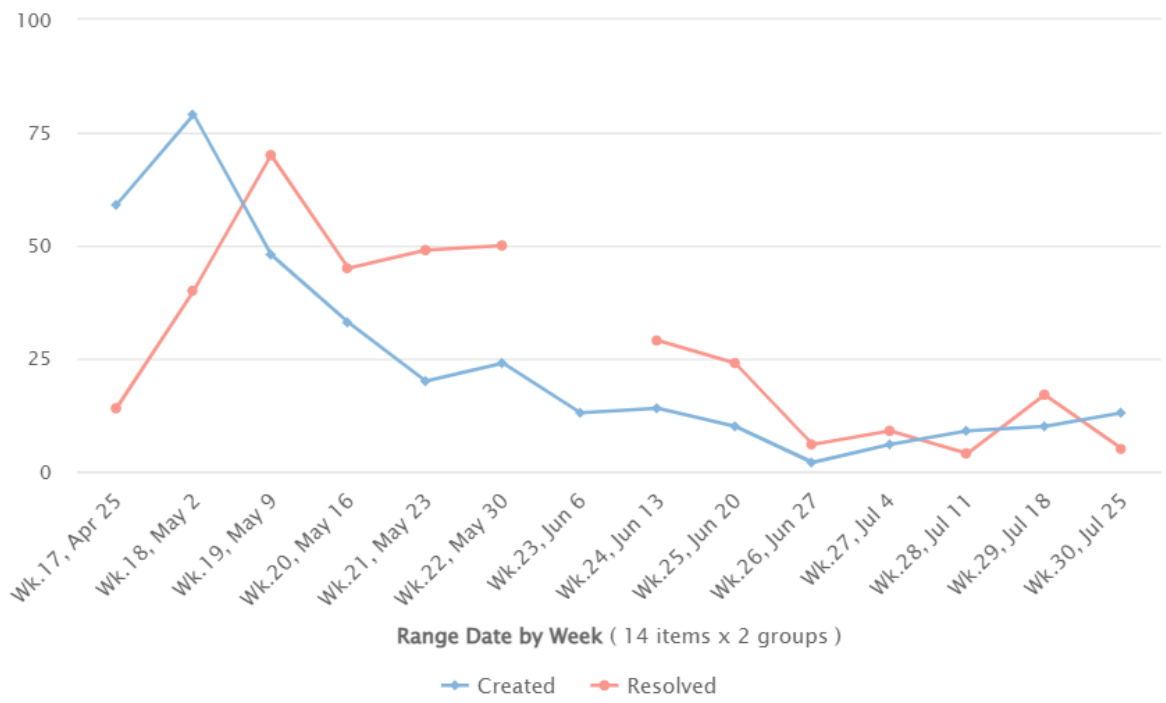


Figure 27: Bugs created and resolved from 25 April to 25 July

The decision was made to assign every part of the application to a developer. Team members are responsible for fixing bugs and being aware of changes affecting their application parts. At the same time, the group changed the strategy, and the PO should approve the changes. It helped to have an extra eye on new changes and catch bugs before adding them to the main project. The decisions here helped to have a stable application.

7.2 Speed vs. quality

The purpose of all software is to solve a problem. It is more valuable to solve a problem today than to solve it tomorrow. It is essential to finish as soon as possible. The time to market is one of the biggest problems for startups. A new product's success or failure can be determined by how quickly it is launched.

Early in the software development process, the team's velocity was much lower than predicted in October and November. We changed our state management and admin framework decisions at the beginning of the application's development phase in October and November. After that, the team set a goal to increase the velocity so that the software would be completed on time. The graph below shows that the team's velocity increased from December and reached its expected level.

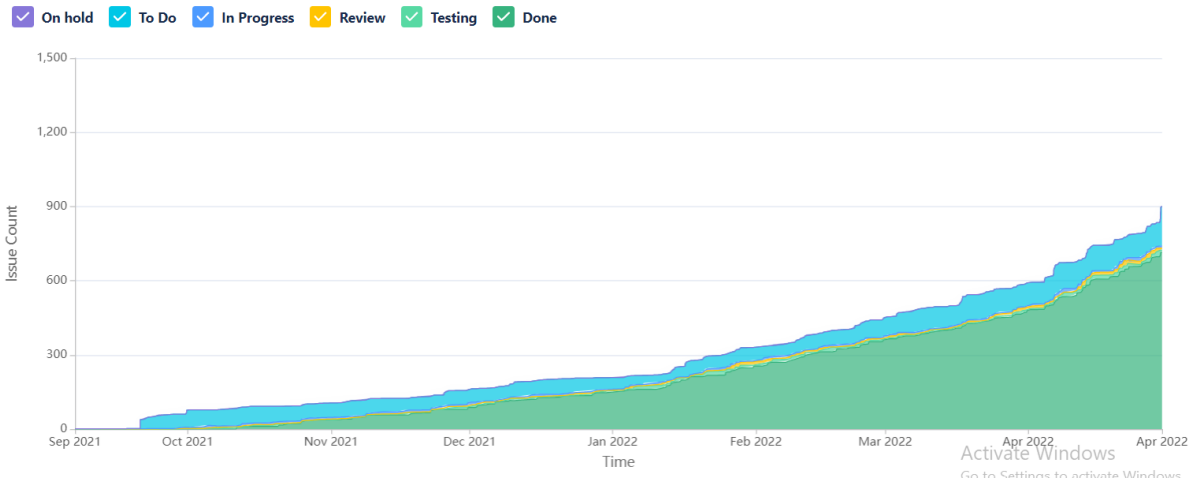


Figure 28: Increase velocity during the time

While the team is improving its velocity, we see the following pattern.

1. A topic appears.
2. Discussion is taking place on the topic.
3. After that, a solution will be found for the topic.
4. The solution is added to the action plan in the next step.
5. In the end, the topic ignores or abandoned in the early stages of development.

This pattern can be seen in testing, code review, technical debt, documentation, and refactoring.

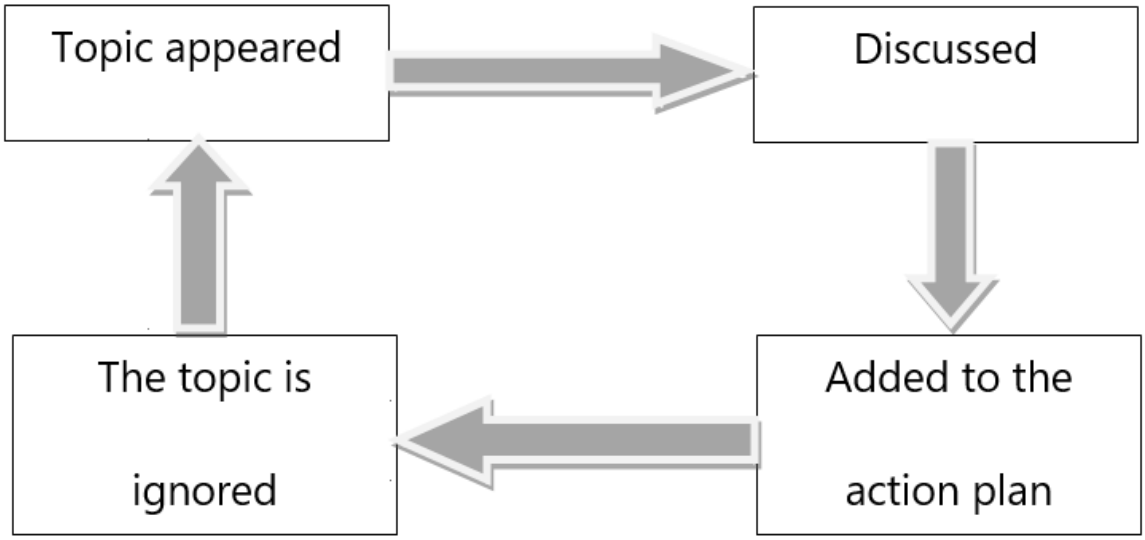


Figure 29: Ignored action plan pattern

Technical debt is an example of taking speed over quality. Technical debt is taking shortcuts in code to gain speed at the cost of bad code quality. As a result, we achieved more than we would typically be able to, but we ended up paying a more significant price in the long run.

In other words, we do not have time now and must deal with the consequences later. As soon as we finished the development phase and were ready to launch the application, we returned to fix technical debts. The topic has been postponed again due to too many changes.

Consequently, the focus on fast delivery of customer value in a short time causes technical debt, which will slow the progress as time progresses.

7.3 Definition of Done

According to the Scrum Guide [10], the Definition of Done (DoD) is defined as follows:

The DOD is a formal description of the state of the Increment when it meets the quality measures required for the product. When a Product Backlog item meets the DOD, an Increment is born. The DOD creates transparency by providing everyone a shared understanding of what work was completed as part of the Increment. If a Product Backlog item does not meet the DOD, it cannot be released or presented at the Sprint Review. Instead, it returns to the Product Backlog for future consideration.
If the DOD for an increment is part of the organization's standards, all Scrum Teams must follow it at a minimum. If it is not an organizational standard, the Scrum Team must create a DOD appropriate for the product. The Developers are required to conform to the DOD.

The DOD in our project was defined at the beginning of the project. Because the first DOD is weak, it needs to be updated as the project matures. Unfortunately, this topic has not been taken into consideration.

A weak DOD can lead to severe problems. Although user stories or releases are reported as completed, their quality and completeness are unknown. The PO might discover after the product is ready for production that some parts are not working.

Undone Work: The difference between the Definition of Done and Potentially Shippable. When the Definition of Done is perfect, there is no Undone Work. If this is not the case, then the organization has to decide (1) How to deal with the Undone Work and (2) How to improve so that there will be less Undone Work in the future. [11]

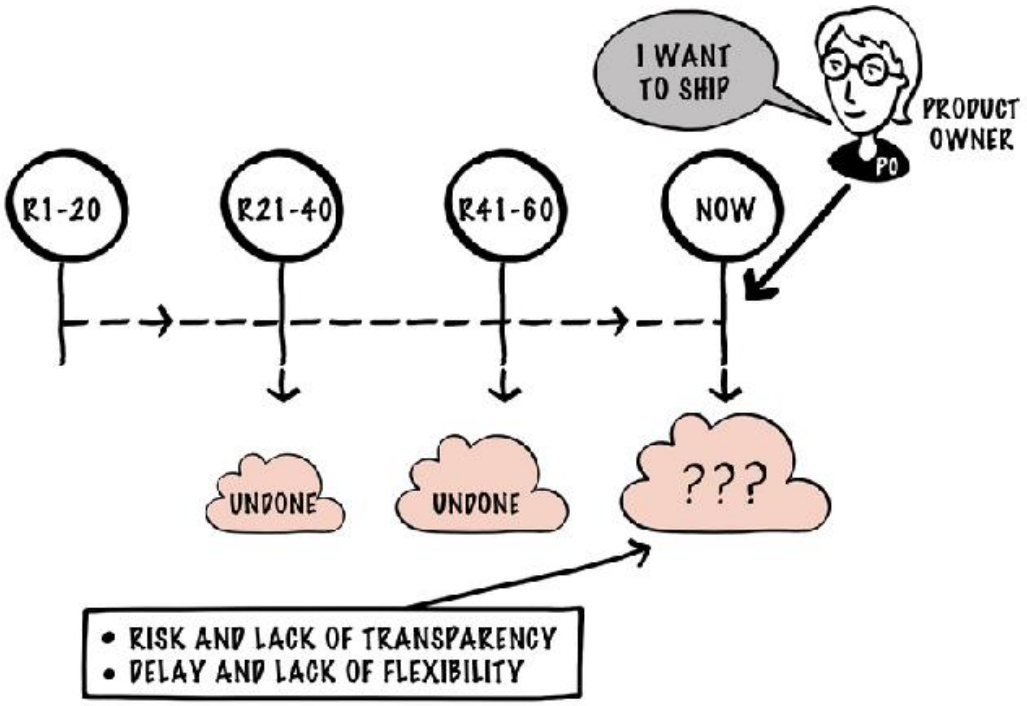


Figure 30: Weak definition of done [11]

The PO expects to deliver the product after the development phase of the project is complete, but because there are undone works, it is not possible.

Delay: To complete the Undone Work, extra effort is required. As a result of the unpredictability of the Undone Work's effort, the pain caused by this delay is aggravated. After finishing most of the features at the end of April, we figured out that most of the features were not working as expected. Some features needed extra work. The following picture shows the delay in the development process that the team spent 12 Sprints from 25 April to 5 Sep to fix bugs.

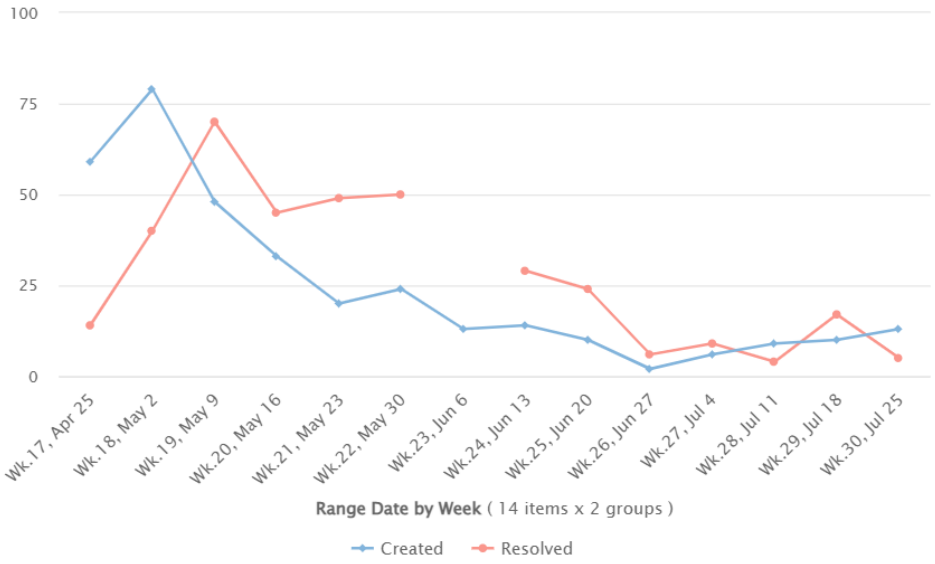


Figure 31: Bugs created and resolved from 25 April to 25 July

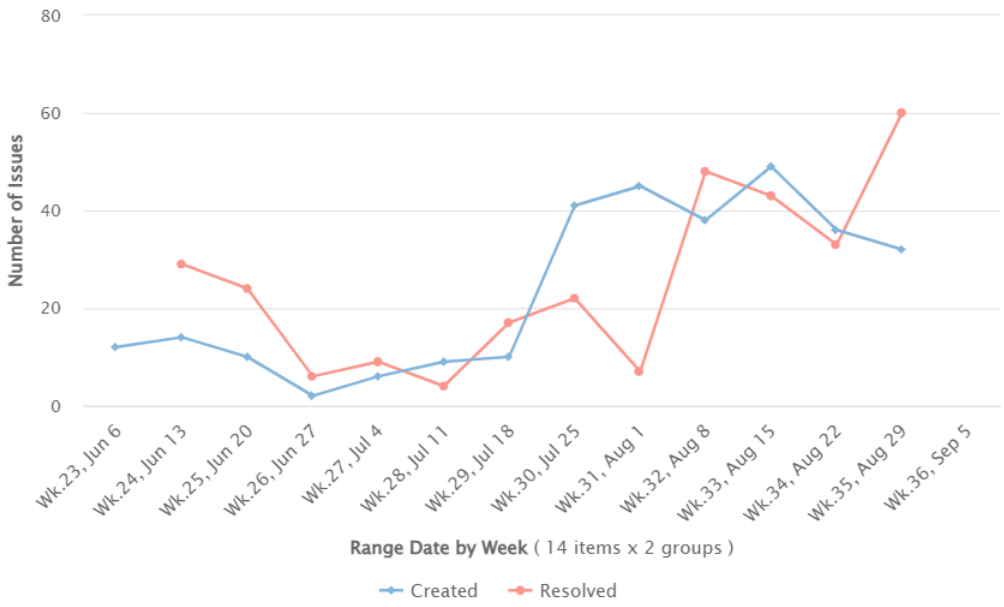


Figure 32: Bugs created and resolved from 6 Jun to 5 Sep

Risk: The work that has not been completed increases the risk of a lack of transparency. A load testing was simulated at the end of the application to test its performance. To ensure the application works as expected, we sent numerous requests to it. As a result of the testing, we need extra work to improve the application's performance.

A good DOD is essential for the project, and a weak DOD can lead to severe risks that we should face later.

8 Conclusion

This research investigates issues and challenges during the software development process in modernizing a legacy application. We examined how the challenges developed during the process and how the decisions affected the project's success.

Lack of Testing and Test Automation: Testing is essential to the agile software development process. When testing occurs in the next Sprint or Sprints, new code may be written on top of non-tested code ("Waterfall-ish" Thinking.) In addition, the product is not always "potentially shippable." Bugs coming in the following sprints also lead to an increased cognitive load of developers. More communication between testers and developers is also needed. Faster delivery of the product could not be achieved by ignoring testing. Instead, we need to pay a more extended price to find and fix bugs.

Automated Testing can help detect bugs earlier before adding them to the main project. It can also be beneficial in Refactoring. During the development process of the application, automated testing became increasingly important.

Speedy delivery: The time to market is one of the biggest problems for startups if they want to compete with the competitors, but the quality also should be considered. Shortcuts in code to gain speed can lead to performance issues and later technical debt the team must face. A higher level of quality does not equate to a slower speed. In contrast, quality makes it possible to progress faster. Activities like test automation, code review, and technical documentation can help teams go faster.

Definition of Done: The DOD creates transparency and determines what Done means in a project. The quality of the project also depends on it. A weak DOD can lead to Risk and Delay. PO wants to deliver the product, but most features need extra work because of the lack of transparency created by a weak DOD. It takes extra effort to finish the undone work. The work that has not been completed increases the risk and causes issues that the team must face later.

8.1 Post Launch

Launching an application does not mean that the work is done. It is also important to maintain the application. The team changed strategies based on what they have learned during the challenges.

Code freeze: It means no changes can be made to the project during this time. The idea was to prevent a buggy release. The team decided to freeze the code two to three days before release and focus on testing during this period.

Backlog Refinement: The goal is to come to a shared understanding of what the Product will accomplish and the order in which it will be achieved. In the early stages of the project, the idea came up for discussion, but it was not considered. Due to the lack of backlog refinement, the team lost sight of the big picture of the application.

9 References

[1] C. Kiran Madhunapantula, „www.comakeit.com," [Online]. Available: www.comakeit.com.

[2] M. Lent, „gotober," [Online]. Available: https://gotober.com/2019/sessions/1119/building-resilient-frontend-architecture.

[3] A. P. Sowmya Purushotham, „Bridging the Gap Between Acceptance Criteria and Definition of Done".

[4] „smartbear," [Online]. Available: https://smartbear.com/learn/code-review/what-is-code-review/.

[5] Gaminer, „hackernoon.com," [Online]. Available: https://hackernoon.com/the-fallacy-of-technical-debt-202f7406337e.

[6] techopedia, „techopedia," [Online]. Available: https://www.techopedia.com/definition/20915/naming-convention-programming.

[7] „productplan," [Online]. Available: https://www.productplan.com/glossary/minimum-viable-product.

[8] „refactoring.guru," [Online]. Available: https://refactoring.guru/refactoring.

[9] B. Vidyapeeth, „A Comparative study of Agile Software Development".

[10] K. S. &. J. Sutherland, „The Scrum Guide," [Online]. Available: https://www.scrum.org/scrum-guide.

[11] „less.works," [Online]. Available: https://less.works/less/framework/definition-of-done.html.

[12] *DIN 1311-2;2002-08, Schwingungen und schwingungsfähige Systeme Teil 2: Lineare, zeitinvariante schwingungsfähige Systeme mit einem Freiheitsgrad.*

[13] „www.ibm.com," [Online]. Available: https://www.ibm.com/topics/software-testing.

[14] „softwaretestinghelp," [Online]. Available: https://www.softwaretestinghelp.com/what-is-end-to-end-testing/.

[15] „katalon.com," [Online]. Available: https://katalon.com/resources-center/blog/regression-testing.

[16] f. may, „https://medium.com," [Online]. Available: https://medium.com/@floyd.may/quality-versus-speed-16df7673202.

[17] V.-P. Eloranta, „Exploring ScrumBut – An Empirical Study of Scrum Anti-Patterns".

[18] M. Perkusich, „A systematic review on the use of Definition of Done on agile software development projects".

[19] „productplan," [Online]. Available: https://www.productplan.com/glossary/technical-debt/.

[20] S. McCormick, „bigeng.io," [Online]. Available: https://www.bigeng.io/why-the-way-we-look-at-technical-debt-is-wrong/.

[21] F. May, „https://medium.com," [Online]. Available: https://medium.com/@floyd.may/quality-versus-speed-16df7673202.