Metrology Product Development for Industry and Science

Software Project Management Best Practices From a Multiple Case-Study Analysis

Catarina Raquel Jesus Lopes

Master's Dissertation

Advisor at FEUP: Prof. Xavier Andrade
Work Advisor: Eng. Pedro Besteiro from LSI

Master in Engineering and Industrial Management

2023-06-16

Abstract

This dissertation was developed within the context of the completion of the master's degree in Engineering and Industrial Management at the Faculty of Engineering of the University of Porto. The main goal of this thesis is to present several case studies of projects developed in a metrology product development company, analyze them from a software project management view, and reach conclusions about the biggest conflicts and impediments that can influence the final outcome.

The host company is interested in learning about the unique difficulties that project management and software development face, their causes, and solutions. These are the goals of this dissertation—to arrive at findings and give a best practices manual to aid in the battle against the most common problems found in complex software development projects that delay or impact the final outcome.

The methodology used in developing this dissertation comprises understanding how project management functions in software development projects, outlining each project's flow, coming to individual conclusions about each one, and finally comparing all of them to reach a final conclusion and synthesize the best practices manual.

It was concluded that the main problems with project management and software development are related to a lack of defined leadership, communication issues between different teams inside the project, and inaccurate requirements gathering. Other disputes are caused by a shortage of project goals, a dearth of market research and proofs of concept, and expectation management problems.

Some of these conflicts proved to be so critical that caused projects to fail and whole applications to be put on hold.

As a result, suggestions on how to prevent and stop these disputes from harming the project were developed. The most important lessons learned were the importance of a defined leadership with a strong product owner, capable to communicate and lead the project, the significance of a thorough communication strategy that encourages and engages employees to form connections with one another, the necessity of regular meetings to foster the co-creation of goals, requirements, and application progress, the importance of using SMART goals, and the matter of taking time before the execution to conduct thorough market research.

Resumo

Esta dissertação foi desenvolvida no âmbito da conclusão do mestrado em Engenharia e Gestão Industrial na Faculdade de Engenharia da Universidade do Porto. O principal objetivo desta dissertação é apresentar vários casos de estudo de projectos desenvolvidos numa empresa de desenvolvimento de produtos de metrologia, analisá-los do ponto de vista da gestão de projectos de software e chegar a conclusões sobre os maiores conflitos e impedimentos que podem influenciar o resultado final.

A empresa está interessada em descobrir os problemas mais comuns em gestão de projetos e desenvolvimento de software, bem como as causas e soluções desses problemas. Os objetivos desta dissertação são chegar a conclusões e fornecer um manual de melhores práticas para ajudar na luta contra estes dilemas encontrados em projetos complexos de desenvolvimento de software, que atrasam ou influenciam o resultado final.

Os principais componentes desta dissertação incluíram a compreensão do funcionamento da gestão de projectos em desenvolvimento de software, a descrição extensiva de cada projeto, a obtenção de conclusões individuais sobre cada um deles e, finalmente, a comparação entre todos para chegar a uma remate final e avançar com o manual de melhores práticas.

Concluiu-se que os principais obstáculos estão relacionados com problemas de liderança, comunicação entre as diferentes equipas do projeto e com uma definição imprecisa de requisitos. Outros são causados por excesso ou escassez de objectivos, falta de estudos de mercado e provas de conceito, ou mesmo problemas de gestão de expectativas.

Alguns destes conflitos foram tão graves que alguns projetos falharam e aplicações foram canceladas.

Como resultado, foram desenvolvidas sugestões sobre como melhorar e impedir que estas adversidades prejudicassem o projeto. As lições mais importantes foram a importância de haver um *product owner* capaz de realmente liderar a equipa e ser transparente quanto a objetivos, a relevância de uma estratégia de comunicação que encoraje e envolva os funcionários a estabelecerem ligações entre si, a necessidade de reuniões regulares para promover a co-criação de objectivos, requisitos e progresso da aplicação, o valor de utilizar objectivos SMART e a importância de reservar algum tempo antes da execução para realizar um estudo de mercado o mais completo possível.

Acknowledgements

First of all, I would like to express my sincere gratitude to Eng. Pedro Besteiro, my work advisor, without whom this dissertation wouldn't be possible. Thank you for all the guidance, expert advice, and continuous support. To Prof. Xavier Andrade, my advisor at FEUP, a special thank you for the incredible support and for always being available to guide, help, and for making this dissertation what it is today.

A big thank you to the institution, FEUP, whose commitment to research and education has given me access to a challenging academic setting. My intellectual growth and development have been greatly influenced by their willingness to share information and resources.

To my mother, Florbela, who made me the woman I am today, always believed in me and never let me quit. Thank you mom, I hope to make you proud, this is for you.

To my sister, Cátia, my role model, the one I can always look up to and find inspiration. Your support is forever priceless, thank you.

To my whole family, Carla, and António, thank you for always giving me the love and support I needed and for accompanying me on this journey.

To Daniel, for always being on my side and listening to me, for always treating me with care in the most stressful phases, and for loving me every day. And lastly, to all my colleagues and friends, for the countless work projects, exams and years spent supporting each other on this ride. I cherish all of you with all my heart.

"Limitations live only in our minds. But if we use our imagination, our possibilities become limitless"

Jamie Paolinetti

Contents

Acronyms and Symbols

CI	Continuous Integration
CPM	Critical Path Method
IT	Information Technology
OM	Optical Microscope
PERT	Program Evaluation and Review Method
PO	Product Owner
PTC	PEER Tool Composer
PTO	PEER Tool Orchestrator
SDLC	Software Development Life Cycle
SEM	Scanning Electron Microscope
SRS	Software Requirement Specification
TPS	Toyota Production System
UI	User Interface
XP	Extreme Programming

List of Figures

List of Tables

Chapter 1

Introduction

This dissertation was conducted within the Master in Engineering and Industrial Management at the Faculty of Engineering of Porto University in the specific context of Project Management in Software Development projects. This chapter aims to present the main motivation and goals of the dissertation, present the company where it was developed, explain in detail how the project was structured, and give a general overview of this document structure.

1.1 Project Framework and Motivation

Now more than ever, due to the size, complexity, and diversity of software systems, there is a constant need to interact across organizational, geographical, cultural, and socioeconomic boundaries (Mens et al., 2019). Developing software implies having a team capable of communicating constantly and cooperating closely in all phases of the process. Beyond that, the developers should also have a line of communication with the stakeholders and business partners to meet exacting requirements, support different interaction styles, and meet growing demands for functionality, quality, flexibility, and cost-effectiveness.

Good management is essential for the effectiveness and success of any project because it keeps the project moving and the channels of communication and knowledge open. This is especially true for projects like software development where the requirements are constantly changing, the deadlines are challenging, and the entire process is intricate.

The main causes why software development projects fail are mainly related to management problems, communication inefficiency, changes in requirements, and lack of global vision of goals across the team (PMI, 2017). Figure 1.1 shows exactly that.

The rate of project failure in the information technology (IT) industry is still very high (Keil et al., 2013). A "marked decrease in project success rates" was noted in 2009, with 44% of all IT projects being delivered late, over budget, or failing to satisfy specifications, and another 24% being canceled before completion or delivered but never utilized.

According to (Lopez-Martinez et al., 2016), the biggest problems in software development, specifically in agile methodologies, can be of four scopes: project, process, organization, or people. The conclusion was that the block with the biggest impact was People. This occurs due to the difficulty of the management of the whole process, lack of communication, lack of training, misunderstanding of client's requirements, and poor vision of the project scope.

For this dissertation, the management in software development will be deconstructed and analyzed to gain knowledge on the main phases that can make or break a project. For this, four projects in place at the company where this work was based will be described and a step-by-step of requirements and conclusions will be produced. Among these, there are examples of good and bad management and project development, and by analyzing the outcomes and comparing them, we can understand and learn how to manage projects with these dimensions and overcome the main difficulties in agile methodologies, namely Scrum.

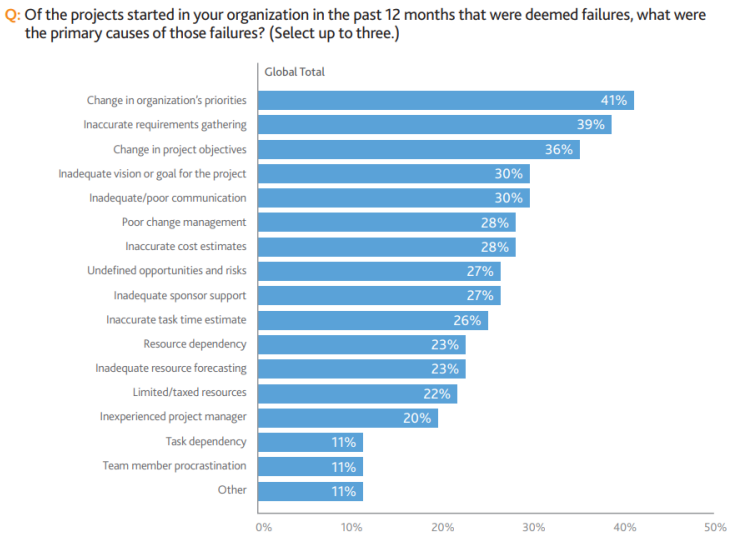


Figure 1.1: Main reasons why software development projects fail (PMI, 2017)

1.2 The Project at LSI, Integração de Serviços e Informática, LDA

In 2001, four engineering students founded LSI. The business develops hardware and software products for both businesses and consumers, as well as engineering services for a variety of sectors. The goal of LSI is to offer specialized solutions that span a variety of disciplines, including software development, microelectronics development, and world-class development skills, as well as cutting-edge approaches and procedures. For this, they customize Agile approaches, namely Scrum, with an emphasis on feature delivery and quality maximization.

In this report, I analyze LSI's four projects on how different management and ways of organizing process development can impact the outcome. In fact, it's from them that the company wants to learn and get information for future projects. With this dissertation, they can assess the

necessity for improvements when working with different entities and clients and from that, be able to improve projects outcome, transparency, and communication.

All the projects come from major German companies, LSI being the main development team. Some of them are already finished and others are still in progress. Their investigation will reveal details on how LSI manages its team and interacts with the requirements team, the scientists, and the entire project management team.

1.3 Goals

The main goal of this dissertation is to extract knowledge from four case studies by understanding the critical points when looking at project and product management in software development. By contrasting them, we will be able to identify the present issues for the company to address.

The report aims to answer the following questions:

1. What specific issues and bottlenecks are affecting the project management and software development processes at the company?

2. What has been determined to be the true causes of the issues?

3. How can the company overcome the identified problems?

The ultimate objective is to improve the future decision of priorities, requirements, and system design when confronted with stakeholders and use cases with very different profiles i.e., laboratory teams, production teams, and factory teams.

Additionally, this dissertation has significant importance for the whole metrology development sector. Since these projects and scenarios accurately reflect other projects in the industry, we may extrapolate conclusions from them and offer recommendations in the best practices guide that may be more beneficial for the market.

1.4 Methodology

The methodology of the project shows how the entire project was organized and designed to respond to the stated study questions.

By gathering data, hosting informal workshops, observing the process, and taking part in two of the study's projects, the first phase of the project sought to understand how the entire company operated in terms of structure, processes, its surrounding context, and evolution over the previous few years.

This made it possible to gather qualitative data, specifically, project descriptions, useful for the practical chapters of the dissertation, and other information for the theoretical part. Above that, entering the environment of the projects helped in understanding what is Agile, particularly Scrum, and how it works in practice.

For the second stage of the project, after the research and understanding of the values of the company and the industry, the questions above mentioned start to be addressed and answered.

A thorough grasp of projects and their description is essential to address the first and second questions. Being a member of the team in two of the projects gave access to intimate knowledge and firsthand experience with the projects and pressing issues. For the other two projects, it was easy to identify problems and bottlenecks impacting project management choices with training, research, and interviews in the company.

For the third question, it is necessary to list potential solutions by comparing qualitative and quantitative data and making suggestions for decision-making processes. To enhance future projects and avoid problems like those found, a best practices guide was created.

1.5 Work Plan

To achieve the goals and follow the methodology of the dissertation, a Gantt Chart is presented as a work plan for the flow of the dissertation. It is divided into three main parts: the research work, the practical work, and the writing of the dissertation. Figure 1.2 is a representation of the tasks that will help in the success of the report.

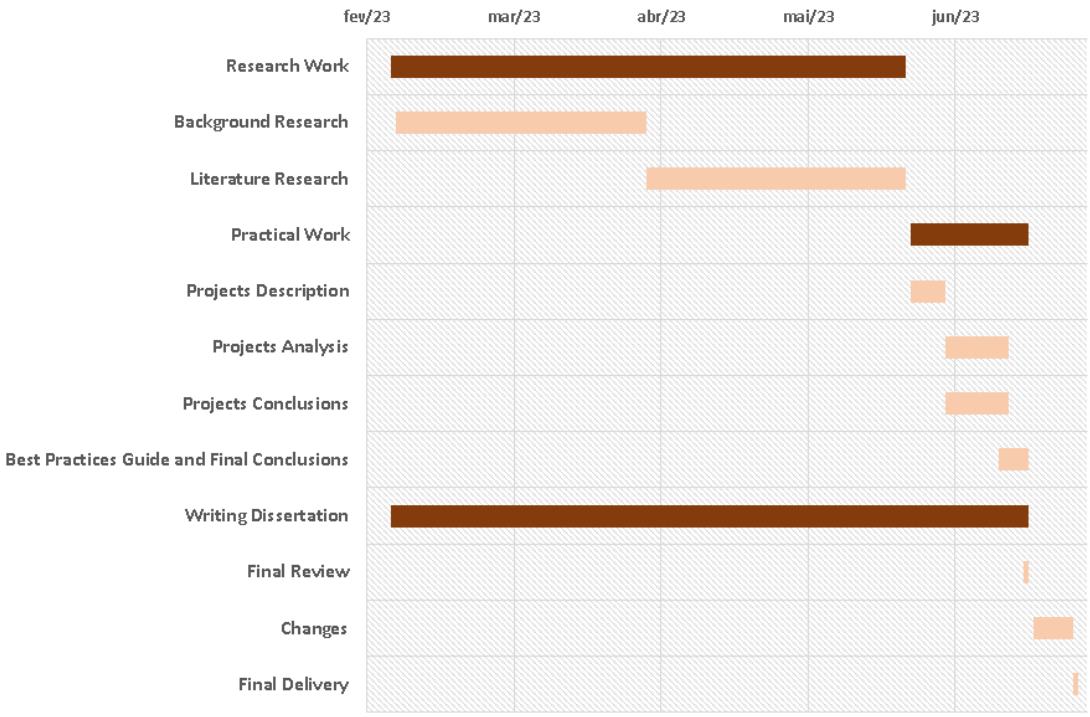


Figure 1.2: Project Gantt chart

Due to the intricacy of concepts and procedures that must be completely understood in order to provide accurate analysis and suggestions on the practical side, the research work makes up the

majority of the plan. The dissertation is being written as the theoretical portion is being completed to record all the thoughts and ideas that are being looked up and found.

1.6 Dissertation Structure

Along with the introduction, this dissertation is composed of seven other chapters.

Chapter 2, *Background - Options placed in the Development of Software Automatism's*, presents a theoretical overview of the most important concepts and hypotheses in software development automatism. These will be incorporated into the report as important notions to take into consideration.

Moving to Chapter 3, *Literature Review in Software Development*, this presents the state of the art of the dissertation. Key concepts like the software development process, project management, and existing standards for process automation are presented and described extensively in order to provide the needed basic knowledge for the whole report.

Chapters 4, 5, 6, and 7 provide the practical information about each of the analyzed case studies of the dissertation. Each has a full description of the project as well as its main problems or conflicts in product/software development management. Each of these chapters ends with a full conclusion and summary of possible ideas that could improve the outcome of the project.

This report ends with Chapter 8, *Conclusions*, which delivers an overall comparative analysis of all case studies presented and provides a best-practices guide for future projects developed at the targetted company.

Chapter 2

Background - Options placed in the Development of Software Automatism's

Beginning with some theoretical information on factors to consider while creating software automatism, this chapter sets the stage for the dissertation. It then continues by listing the benefits and drawbacks of each of those choices in terms of the project's overall success. Each of these options can impact the development along the way and so, it's important, according to the requirements and goals of each project and company, to choose with caution and consideration.

2.1 In-House Development versus Third-Party Solutions

When discussing in-house development versus third-party solutions, the controversy is about the decision to use one or the other. In fact, each comes with a pack of advantages and disadvantages that can influence decision-making depending on several factors.

In-house development refers to building a product from the ground up with the use of only companies' resources and developers. Third-party solutions offer licensed ready-made applications, developed by external vendors or service providers, with multiple integrations and feedback already in place. Using third-party solutions may appear to be the greatest option at first glance since it suggests less work for a firm, and the use of up-to-date apps may provide a much faster process to a company that is just driven by the speed of processes. When a corporation properly knows the goal and value they want to get from automation, this perspective shifts, and it's now vital to weigh the pros and cons of each choice and comprehend the long-term ramifications of each.

Upon a decision, the main areas to consider are costs, time, functionality, and resources. One can be a much better option than the other depending on the needs and constraints of the business and the main goal is to optimize the value of the final product.

Table 2.2: Comparison between In-House Development and Third-Party Solutions (adapted from Aitzaz et al., 2016)

Area	In-House Development	Third-Party Solutions
Costs	• Large initial investment and expenses with building, maintaining, and upgrading custom software	• Small investment with low short-term expenses • In the long term, license fees can be higher than maintenance costs from in-house development
Time	• Takes time to build	• Fast implementation
Functionality	• Allows flexibility and customization according to needs and requirements • Can increase visibility • Innovative application with new capabilities that can bring a competitive edge to the company by leveraging the most efficient, connected, and scalable technology	• Limited item functionality with fixed capabilities • Hard to customize according to unique needs and requirements and difficult to optimize • Slows down growth and innovation and can put a company at a competitive disadvantage
Resources	• Requires experienced developers that have a high acquisition cost • Finding and hiring skilled developers can be challenging and time-consuming	• Requires at least one experienced developer to manage the tool

Taking a closer look at Table 2.2, in terms of costs, while in-house development implies a bigger investment in training employees, and building and maintaining the custom software, third-party solutions come with a lower investment in the short term since the company does not need to spend employees' time and use its assets. The only things needed are license fees to use the pre-build application and training personnel. Although this is true for the short term, these license-fee costs in the long term can be higher than in-house maintenance costs.

Regarding time, in-house development takes much longer to implement since the company needs to develop from the ground up a customized and tested application with all the necessary features. A whole software development process needs to be created and managed inside the

company and a team needs to be allocated to that task. In a third-party solution, the most time-consuming task is choosing the right pre-made application for the organization.

Concerning functionality, the first option allows flexibility and customization according to the organization's needs by providing an innovative application. The company can get a competitive edge, surpassing the competition. On the contrary, the second option brings limited functionality that may not check every requirement box needed and slows the growth of any company.

Lastly, resources needed in in-house development include a team of experienced and skilled developers while third-party solutions require only at least one experienced developer to manage the tool.

Let's imagine that a software development company XPTO wants a new data viewer software to be able, at any time, to observe the analytics of the business with graphs and predictions for the future. Now, XPTO needs to decide whether to develop the application with its developers or acquire a well-known software already used in the market by many competitors. We have two possible extreme scenarios:

1. If the company has experienced developers, its values incorporate being an innovative organization, on the edge of competition, and has enough capital and time to invest, then XPTO would prefer to develop in-house, due to the opportunity to customize according to needs and reaffirm themselves as one of the top companies in the market, able to produce more advanced technologies and leveraging that push to surpass the competition.

2. If the company cares about delivering value through fast time-to-market, and is still in the start-up phase, so the capital is low, then XPTO would prefer to acquire a third-party solution that can provide results quicker and cheaper and then in the future can be upgraded to an owned application.

While the decision is evident for these extreme scenarios, it becomes nuanced when the factors considered are leveled and combined differently.

2.2 High-Level versus Low-Level Development Languages

High-level programming languages and low-level programming languages are the two broad categories of programming languages. High-level languages are intended to be easily readable and closer to human language. In contrast, low-level languages are closer to machine language and more difficult for humans to read and write (Grant, 2020).

High-level programming languages, such as Python, C# and Java, are intended to abstract away the underlying hardware and provide developers with powerful tools for quickly and efficiently building complex applications. These languages are well-known for their ease of use, as they are frequently accompanied by large libraries and frameworks that make the development process easier. One big advantage of this type of language is that it does not deal with memory management. Although being the most used languages in the development processes, they have poor performance compared to low-level languages.

Low-level programming languages, such as machine code and assembly language, on the other hand, are considerably more closely related to the hardware and need programmers to interact directly with the architecture of the computer. Due to their direct access to the machine's hardware resources, these languages are renowned for their quickness and effectiveness. In terms of downsides, the bigger disadvantage of this type of language comes from the difficulty of learning and the poor support that exists around this topic.

Low-level languages provide greater control and flexibility whereas high-level languages increase productivity and ease of use. For example, developers frequently opt for low-level languages when creating embedded systems or operating systems since speed and memory efficiency are key considerations. For complex applications, high-level languages make the abstraction and development processes easier.

Figure 2.1: High and low-level language examples (ProgramizPRO, 2022)

When analyzing Figure 2.1, it is easier to understand the practical difference between these two types of languages. Both codes are doing the same, providing a function to add two numbers, still, while the high-level language is readable and understandable by any human, the second, for those who know nothing about low-level languages, is unreadable. This proves how high-level can turn out to be so easy to use.

Another visible difference is the number of lines of code. High-level is more succinct, with functions that can be written in one simple line. Low-level needs more lines of code to do one simple task like adding two numbers.

2.3 Development Strategies: Continuous Integration

Continuous Integration (CI) is a best practice that should always be taken into consideration and implemented in these types of projects.

In software development, the paradigm between quality, costs, and time-to-market is crucial to every project. A company always wants to achieve the best quality, with the lowest costs and as fast as possible, but sometimes, choices need to be made to reach an optimal point between these three variables.

A software project is deemed successful if it delivers the product with an agreed-upon quality level on time without going over-budget (Paulk et al., 1993).

For that, the concept of CI is presented. This is a software development technique focused on code quality in all phases of development. In CI, developers upload new code to the code base at least once per day, integrating it more frequently during the development cycle, and providing 100% test coverage at both the high and low levels, that is, in the system as a whole and on each class/unit of the project.

The technique provides full documentation of the interface written by developers and core components involved in the project as well as a document with all the style rules and code architecture to have more transparent scalability maintenance. This documentation is something that can be built on top of what already exists.

To find integration problems early, when they are simpler to fix, automated testing is performed against each builded iteration. This helps to prevent errors at the final merging for the release (IBM, 2020). The build process is generally streamlined by CI, which leads to better software (quality) and more predictable delivery timelines (time-to-market).

In synthesis, the biggest benefits achieved using continuous integration are (adapted from Professional, 2019):

- **Reduction of risks** - since CI integrates tests and inspections several times a day, there is a greater chance that errors can be discovered in the early stages of any project. Bugs are still present, but CI makes detecting and fixing them much easier.

- **Generation of deployable software at all times and places** - CI enables us to frequently make minor modifications to the source code and include those modifications with the rest of the code. Those then go to a reviewer on the team that gives feedback and makes sure that there aren't any problems with the new integration. If there are, they can identify and correct it at the moment.

  The modifications only go to the main branch when they are bug-free and ready for deployment. This makes sure that the main source is always ready to be reproduced without problems and bugs.

- **Increase in project transparency and product confidence** - provides real-time information analytics of build status and quality metrics of the code. With every integration, the team can see how their changes impact the outcome and understand trends in building successes or failures.

Figure 2.2 helps to understand continuous integration as a full process. The developers create integrations of code for a project and make a pull request to the central code repository. This triggers a CI process where that parcel of code goes to a server. There, the application build is performed and numerous unit tests are run through the whole code. In the end, the outcome (success or failure), is reported to the developer and the project manager.

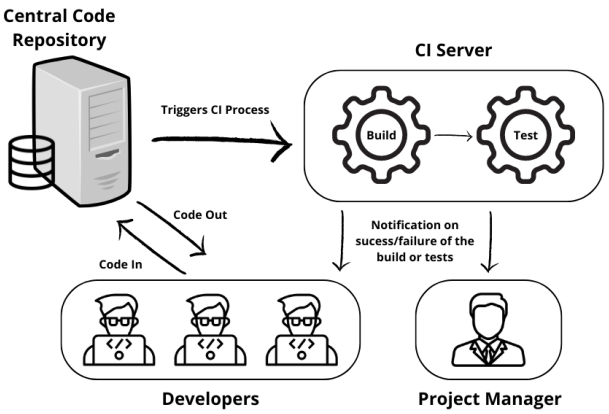


Figure 2.2: Continuous Integration process

In case of failure, the report provides details of why the build or test incurred a failure. Then, the developer needs to fix the broken builds/tests until they pass. Working with CI is all about continuously developing on a known stable foundation. All software developers have continuous integration as a fundamental for every project since it enables delivering high-quality results. Although, this comes at a cost, which includes time and money.

When in need of a faster process, a company may decide to skip steps, and application teams can test in the laboratory software components that respond to specific and urgent needs without taking into account the process of continuous integration/quality. It's up to the organization to decide which mechanisms they want to involve in the software development process to minimize risk without creating blocks along the way.

Chapter 3

Literature Review in Software Development

In this chapter, the literature review for the dissertation starts with a presentation of the software development process and existing methodologies, giving an extensive description of the steps included in a process and the differences between traditional and agile methods. A definition and comparison of the most used methodologies is presented.

It then continues by presenting project management and important concepts, finalizing by explaining existing standards and products.

3.1 Software Development Process

The software industry receives high-quality, dependable, cost-effective, and on-time products thanks to the software development process (Shylesh, 2017). This process can also be designated as the software development life cycle (SDLC), which contains a complete plan for describing how to design, develop, maintain, and increase the efficiency of a product. The SDLC process highlights the approaches that increase overall software quality and development process and is usually composed of 6 steps: planning, analysis, design, implementation, testing, and maintenance, which can vary according to the methodology and type of process the company is looking to achieve.

Figure 3.1 represents the main phases that make a software development process (Saeed et al., 2019):

1) **Planning** - *What do we want?*

   This phase includes developing project goals and objectives, determining development terms and expenses, and analyzing and developing client needs. The planning phase also includes standards for validity assurance and identifying project hazards. The team must determine the project's viability and how to properly implement the project while minimizing risk.

   This is the most vital and basic phase of every life cycle and it's completed by the senior members after the first meeting with the owner of the software system.

2) **Analysis** - *Let's define requirements for what we want.*

In this phase, the team defines the product requirements to achieve for the project and documents every detail. Then, this goes to the customer for validation. A business analyst, who is in charge of researching, structuring, and documenting previously established requirements, as well as a development specialist and a software testing specialist, examine the approved requirements documentation for the quality of the provided characteristics and information of the future product, together with the possibility of its implementation.

This stage provides the software requirement specification (SRS) document, which covers all of the requirements for the product to be designed and created.

3) **Design** - *How will we get what we want?*

The software requirement specification is used as input to define the architecture of the software product that is being created and several architectures are developed, for later approval and delimitation by the stakeholders.

While in the design phase, all stakeholders are involved in the review process of the potential architectures based on several factors like risk assessment, product reliability, budget, and time-to-market. The option with the best prospects is chosen and the development stage can begin.

Failure at this point of the life cycle will very probably lead to cost overruns at best and to the project's complete collapse at worst.

4) **Implementation** - *Let's create what we want.*

This is the longest phase of the software development life cycle. The implementation refers to the creation of the product, according to the designed requirements and architecture. Tasks are broken down into pieces or modules and given to different developers during the coding phase and several tools are used to create code, including compilers, interpreters, and debuggers.

5) **Testing** - *Did we get what we want?*

In this stage, developers test the generated product to make sure the entire program functions in line with the client's requirements that are listed in the SRS document. Software flaws are reported, tracked, corrected, and retested to ensure high-quality production.

This procedure is repeated until the program is free of bugs, stable, and meets the system's business requirements.

6) **Maintenance** - *Let's start using what we built.*

When a product has undergone testing and is ready for use, it is formally introduced to the market. In accordance with an organization's business strategy, a product rollout may

occasionally be phased in. After that, the product can either be released as is or with proposed changes for the target market niche depending on the feedback. After the product is introduced to the market, it is supported by the developers for the customer.

Figure 3.1: Main stages of the software development life cycle (Shylesh, 2017)

A software development methodology is a form of project management for software development. Selecting a suitable management structure may make a significant difference in producing a successful result whether cost, meeting deadlines, customer satisfaction, software robustness, or avoiding expenditures on failed projects are considered (Rozhnova et al., 2022).

It's possible to divide the software development methodologies into two categories: the traditional and the agile methodologies. Inside each one, multiple methods can be applied to projects, and the choice for one on top of another has to do with project requirements, company culture, and the goals of the organization.

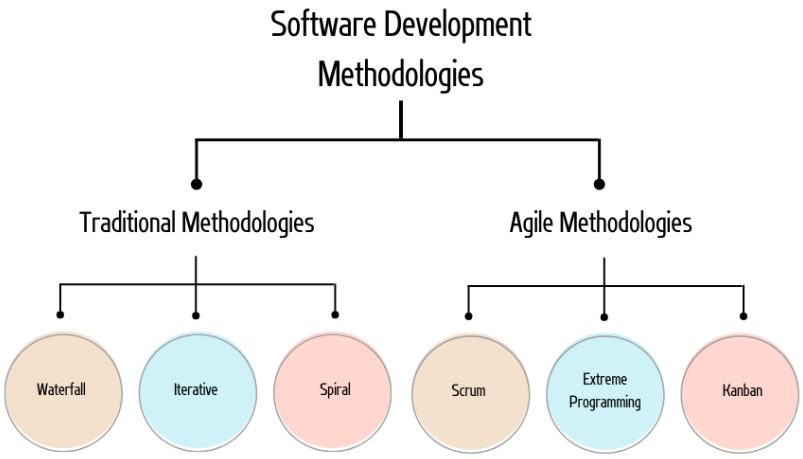
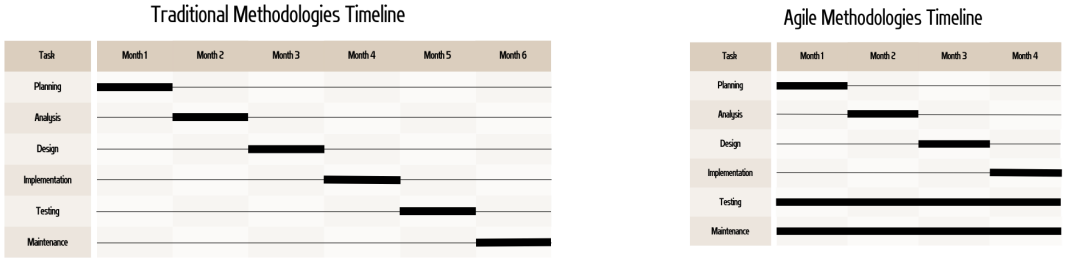


Figure 3.2: Main software development methodologies

Nowadays, the most common Traditional methods are Waterfall, Iterative, and Spiral. For Agile, the more used ones are Scrum, Extreme Programming, and Kanban. Taking a closer look at the strategies that are mostly used nowadays, the following Figure 3.2 can be reproduced.

The biggest difference between traditional methodologies and agile is the timeline of each of the steps already defined (Figure 3.3).



(a) Traditional Methodologies Timeline

(b) Agile Methodologies Timeline

Figure 3.3: Comparison between Traditional and Agile Methodologies Timeline

In traditional methodologies, a very strict timeline is followed, where each phase occurs individually, only one phase at a time. These can occur only once or multiple times in different integrations. On the contrary, in agile methodologies, while the first four phases are strict and occur individually, testing and maintenance occur at all times throughout the whole process of development. This gives a better understanding to the company of what is the present state of the project and provides information on bugs and problems in the development, at all times.

3.1.1 Traditional Methodologies

- **Waterfall**

This was the first software development model created. Waterfall is a sequential process model that does not overlap. Each process phase is performed sequentially in cascade and one phase cannot start until the previous one is completed. It's the most simple of methods and easy to implement (Shylesh, 2017).

Waterfall is best suited for small projects where requirements are more constant throughout the duration of the development. The strategy is strict and plan-driven, with limited room for modifications, and incurs a lot of risks. Progress is hard to measure in this type of method (Saeed et al., 2019).

- **Iterative**

In the iterative model, one-step iterations are built to move along in the development. Each one starts with the definition of requirements, to add some new functionality, moving to the analysis of feasibility (Saeed et al., 2019). If everything is approved, the design and development can begin.

In the end, a software product increment is assembled. These iterations occur until there are no more requirements needed from the stakeholder and the product is labeled as ready for market.

This type of model fits best when a project has the complete set of requirements clear and defined since the beginning in order to separate each of the requirements in iterations and there is a need for a fast time-to-market of the product. Since it's an iterative process with strict goals and definitions, it produces results quicker than other models.

- **Spiral**

Spiral is a combination of Waterfall and Iterative methods (Mathur and Acharya, 2015). This is a risk-driven strategy, which means the emphasis is on managing risk through numerous software development rounds. In its visual form, it appears like a spiral with multiple loops. The precise number of loops of the spiral might vary from project to project. The project manager might alter the number of phases required to build the product depending on the project's risks.

This methodology is suited for large projects and companies that have a worried mind about risk. It's flexible in its requirements, by enabling changes through the phases. Despite that, it's a very complex model that can be expensive and time-consuming since from the start, project managers don't have a clue about how many phases are going to exist, so it can continue indefinitely (Mirza and Datta, 2019).

3.1.2 Agile Methodologies

- **Scrum**

Being the methodology in use at the company where the current dissertation was developed, Scrum is the most important model to dig deeper into and fully understand because it will be the main target of study in all the cases in chapters 4, 5, 6, and 7.

Scrum focuses primarily on agile management and efficient development team organization. It is the most well-known technique for teaching agility since it is adaptable and straightforward (Altameem, 2015). A lot of concepts come from Scrum and to better understand how it works, an extensive description of each is needed.

In terms of roles, there are 3 main roles inside this model. The first is the Product Owner, who represents the customer's wants and needs, manages the product backlog, prioritizes requirements, and is responsible for fulfilling deadlines and keeping costs inside the budget (Mahalakshmi and Sundararajan, 2013). He's also the main accountable for the final approval of the code.

The second role is the Scrum Master. This member helps the team in any way needed and is the leader of the development process. The Scrum Master is also responsible for managing the daily scrum and sprint planning meetings.

Finally, the development team is responsible for the creation of the application and is self-organizing and self-aware. Each member is liable for the failure or success of the tasks they are assigned to.

Sprint-based iterations are what drive the Scrum process. The team decides which backlog items will be built and tested throughout each sprint as well as how the sprint will be planned (Salameh, 2014). Each sprint usually lasts for 2/3 weeks, ending with a review meeting with the whole team, to discuss the tasks that were implemented and their status as well as test functionality if necessary.

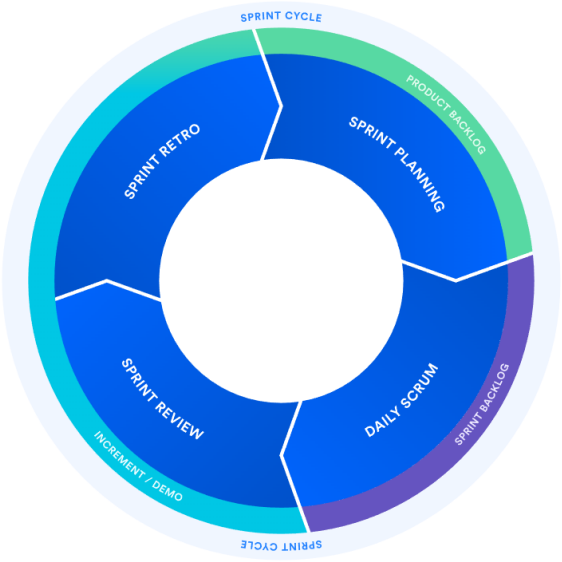


Figure 3.4: Sprint Cycle in Scrum Methodology (Rehkopf, 2023b)

Taking a closer look at Figure 3.4, the sprint cycle is composed of sprint planning, a meeting that marks the beginning of a new sprint. Sprint planning's goal is to specify what can be completed in a sprint and how it will be done. The entire scrum team collaborates on sprint planning.

Every planning starts with the definition of the sprint goals and how the team is going to reach those goals. After that, the scrum master goes to the product backlog, a list of tasks for the development team that is prioritized according to the requirements needs, and chooses the ones that fit the goals and are required to accomplish them (Schwaber, 2004). The team discusses how they can perform the planned tasks, assign them to each member and give a rough estimation of time for each portion.

Another important portion of the sprint cycle is the daily scrum, daily meetings with the purpose of keeping track of the team's work and identifying any obstacles and problems that could hinder the team's ability to accomplish the sprint target. Here, the sprint backlog, where the current tasks for the current sprint are grouped, is updated.

Finally, the sprint review meeting ends the sprint. The scrum master invites the development team to present what they accomplished in the iteration and show the progress made. Final tasks are closed if finished, and the ones that could not be performed are moved to the next sprint. A final assessment of the success or failure of the sprint is given, by the amount of work done (Schwaber,

2004). It's also important to see the burndown of the capacity versus estimated work to understand if the team is being overworked, to prevent burnout.

The biggest strengths of scrum are the effective and efficient communication among team members, the continuous feedback from customers, the top-quality development provides, and the facility to measure growth and productivity with the daily meetings. Despite this, scrum needs a team of experienced developers using and knowing Scrum (Mirza and Datta, 2019).

- **Extreme Programming**

Extreme Programming (XP) helps software be more responsive to consumer needs and of higher quality by providing continuous integration with the client. To manage them back into the actual process, this method divides the software development process into smaller pieces and focuses on good code-writing principles. Extreme programming places a greater emphasis on the development process than on the management of software projects (Altameem, 2015).

The success of XP is a result of its emphasis on client satisfaction. This technique gives the software the client requires when the client requires it, as opposed to offering everything you could want at some unspecified future period. Even late in the development cycle, Extreme Programming gives developers the confidence to adapt to changing customer requirements (Wells, 2013). Uppermost, XP is a methodology denoted by fast prototyping and pair programming (Mirza and Datta, 2019).

Separating specifications saves money by modifying the program to perform all tasks incrementally as the development process progresses rather than planning, designing, and building the whole software. XP is based on the frequent iteration through which the developers implement User Stories. Metaphors are suggested by the project team based on user stories (GeeksforGeeks, 2023).

Metaphors are a popular way to imagine how the system would operate. For some features, the development team could choose to create a Spike. A Spike is a relatively straightforward software designed to investigate the viability of a proposed solution. It may be compared to a prototype.

This methodology fits best inside small and simple projects that involve new technologies or for research purposes.

- **Kanban**

In a 'pull' manufacturing system, as in the Toyota Production System (TPS), Kanban acts as a signaling tool that directs the movement of parts. The idea behind it is to visualize the current process as a series of steps written on a whiteboard (Kirovska and Koceski, 2015).

The main goal of this model is to reduce work-in-progress and ensure work items move quickly to the next steps and increase business value. Usually, electronic boards are used, like Jira (Jira, 2023) or SwiftKanban (Nimble, 2023), to organize these tickets, so the whole team has access to the progress of the project.

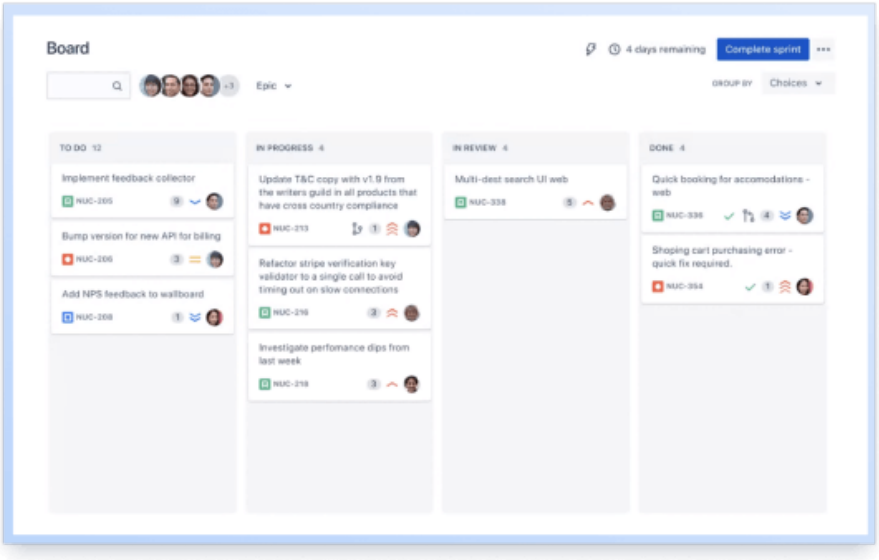


Figure 3.5: Jira Board - Kanban application (Jira, 2023)

A software development Kanban board usually has four types of columns, "To Do", "In Progress", "In Review", and finally "Done" tickets, that make the state of the task visible.

The best scenario to apply this methodology is when requirements are constantly changing and need a fast release of functionality. Also, when requirements can be separated into tickets for small improvements or functions for the application.

Each ticket has an estimation of time for development and after the developer logs the time spent on the task, the project manager can create analytics charts to see which tasks were underestimated or overestimated. This improves the flow of the project and can give an idea of how long the project will take.

The biggest difference between Kanban and Scrum is that they are both project management frameworks, but the first relies on visual tasks to manage workflows, while the second helps teams structure and manage their work through a set of values, principles, and practices (Rehkopf, 2023a).

Kanban helps in managing production and increasing communication between the team and the stakeholders but lacks of details about its implementation (Mirza and Datta, 2019).

3.1.3 Comparison Traditional versus Agile

Taking a look at Table 3.1, agile software development is a method for developing software incrementally. Contrarily, traditional software development approaches or plan-driven software might be characterized as a more formal method of software development.

Table 3.1: Comparison between Traditional and Agile Methodologies (adapted from Mahalakshmi and Sundararajan, 2013 and Islam and Ferworn, 2020)

Feature	Traditional Methodologies	Agile Methodologies
Definition	• Sequential and plan-driven approach	• Iterative and incremental approach
Development Process	• Emphasizes upfront planning, documentation, and requirements	• Emphasizes adaptative planning, collaboration, and customer involvement
Advantages	• Formal documentation • Predictable timeline • Clear milestones	• Flexibility and adaptability • Customer satisfaction • Frequent feedback
Disadvantages	• Rigid approach (less responsive to changes) • Delayed customer feedback	• May require an experienced team • Less control and potential for scope creep
Suitable Projects	• Large-scale projects with stable requirements	• Projects with evolving requirements and dynamic environments usually in small teams or big teams divided into sub-teams
Communication	• Formal and structured communication channels	• Informal and frequent communication plus face-to-face interactions
Customer Involvement	• Limited involvement during the development	• Continuous involvement and feedback from stakeholders
Risk Management	• Risk analysis and mitigation during the planning phase	• Ongoing risk identification and mitigation • Early adaptation

Project Documentation	• Extensive documentation throughout the project	• Minimal documentation for focus on working software
Time and Cost estimation	• Detailed planning and estimation upfront	• Progressive estimation and frequent re-evaluation

While traditional methods emphasize upfront planning, with low to no communication with clients in the development phase, agile brings total customer involvement in the development, with constant adaptations according to needs and changes in the requirements.

In traditional approaches, the project scope, time, and cost are determined and closely monitored for any alterations. Traditional project management emphasizes requirements collecting, analysis, and design up front to produce higher-quality products, to minimize modifications during the duration of the project (Reiff and Schlegel, 2022). Agile project management makes an effort to adapt project execution to changes in the surrounding environment and service scope. Agile requirements are frequently functional, informal, and subject to change.

Agile techniques are built on a few essential software development procedures, as opposed to traditional approaches to the field (Akbar, 2019). The success of projects depends on these agile methodologies' fundamental procedures, especially when offshore development is involved.

Although the majority of agile approaches are thought to be simple and straightforward to grasp, adoption can occasionally be challenging. It is challenging to integrate agile approaches into a company's culture since they are not immediately apparent (Lopez-Martinez et al. (2016)).

It has been demonstrated that despite the significant expenditure made, the majority of software development projects have failed because they are unable to adapt to the changing demands of users (Altameem, 2015). This has prompted software engineers to suggest adaptable and practical methods, such as agile development approaches, to produce high-quality software.

As the process produces high-quality products, it influences software development. It has a significant impact on the developers, helping them to dedicate their efforts to attaining the project's goals. The team members are inspired by the managers, which improves the creativity and innovation that are essential to obtain success. The approach also uses effective communication techniques to help teams and stakeholders produce high-quality software. The greater level of stakeholder involvement aids in identifying and fixing project flaws early on, which lowers costs.

With 71% of firms currently stating they employ agile techniques in their projects more frequently than in the past, agile is a topic that is becoming more and more important in project management (PMI, 2017).

LSI, the company in analysis, incorporates Scrum in almost every project. Analyzing each case study in its context can help us understand how this model impacts the development and how it can perform with success or failure, according to the way it is implemented and managed. This

study can help the company improve its software development quality and time-to-market and by doing so, decrease expenditures.

3.2 Project Management

As software development generally is divided into projects, the topic of project management is extremely important for the study presented. The concepts and methodologies brought by project management are interconnected with software development management as well.

Project management is the application of certain knowledge, skills, tools, and processes to create something of value for people (PMI (2023)). This concept comes along with the term project, which is defined as a transient attempt to add value through distinctive goods, services, and procedures. Some projects are created to solve issues rapidly and others need time to fully grow into something valuable. There can be projects for almost everything, and each needs to be managed according to the requirements and goals of the company behind it.

A procedure for scheduling a group of project activities is known as the critical path method (CPM) (Khan and Mir, 2021). It frequently works in tandem with the program evaluation and review method (PERT). Finding the longest stretch of dependent tasks, and calculating the time needed to accomplish each one from beginning to end, can reveal a key path in project management.

This technique is used by project managers to create a project schedule and estimate the total duration of a project. It helps by visually creating a diagram that represents the sequence of tasks needed to complete the project. Once these paths are determined, the critical path is calculated, which determines the duration of the project (Aliyu, 2013).

CPM can be beneficial for a project by identifying task dependencies and risks, accurately estimating durations, ability to monitor progress, and transparency with the team.

Another powerful tool in project management is Gantt Charts. These illustrate work completed over some time about the time planned for the work. A Gantt Chart can include the start and end dates of tasks, milestones, dependencies between tasks, assignees, and more. It's the perfect tool when you want to display and monitor the whole progress of a project, by task or iteration, for a visually comprehensive result. In agile methodologies, Gantt Charts are created for each sprint.

Above all, one of the principles of project management is the Team. The development team is the one who can make or break a project, so every project manager needs to take into consideration their motivation and spirit. Managing people can be one of the hardest tasks to perform.

Greater motivation can be fostered via an empowered team atmosphere within the project group, department, and organization. A fundamental component of creating a self-directed work team or a high-performing unit is empowerment. Four essential elements make up empowerment: team member authority, competent resources, accurate information, and accountability for the finished task (Peterson, 2007). A happy unit brings faster results with better quality.

3.3 Existing Standards and Products for Process Automation

Industrial automation is the use of control devices, to operate industrial machinery and processes with the least amount of human intervention and the automated replacement of hazardous assembly operations.

In the industrial automation market, more precisely the semiconductor industry, where LSI operates, fabrication can be difficult. Precision has emerged as a crucial issue in semiconductor production as chip size reduces and semiconductor demand rises. One response the market was able to generate with diverse specifications, were standards to ensure high-quality manufacturing. Standards ensure the compatibility and interoperability of goods and services, improving product quality and reliability.

The SEMI Standards Program rose to the interchange of data between users and suppliers leading to the creation of precise and timely specifications and other industry standards that are economically significant (SEMI, 2023). Guidelines for fundamental semiconductor design and production processes are established by SEMI standards. Nowadays most clients demand to include SEMI Standards in projects related to this industry.

By using standards, businesses can save manufacturing costs, improve dependability and efficiency, and gain access to international markets. Individual participation facilitates personal growth, gives access to developing technologies, and lessens the amount of design labor.

There are already some automation tools in the market that use SEMI standards for specific operational scenarios like the PEER® Tool Orchestrator (PTO®) (PeerGroup, 2023a). This PeerGroup tool automates any type of semiconductor tool to improve productivity throughout the entire equipment delivery chain and increase the speed of development. Above that, it also increases production reliability and has a control framework with a GUI-based tool designer, PEER Tool Composer (PTC).

Chapter 4

Automator Project

The Automator Project was one of the projects LSI took part in. It consisted of doing a software application for a company that produces optical metrology devices that emit light (LEDs, displays TFT, cameras, sensors, and others).

4.1 Case Study Description

The client organization had a dominant position in the market, but its devices were integrated into complex systems developed by third-party companies. They sold their cameras or sensors to other companies that developed the final application for the client. In the typical application, their device is integrated into a machine for quality inspection and control in an assembly line.

Although the company's analysis algorithms and features fit the client's final criteria, it only sold to third parties, who then put it into their own products (cameras, sensors, or other items) and sold to the consumer.

Because the integrated equipment had a higher value in industrial applications than their gadget did, the company was losing extra value with this type of business.

These elements generated a drive within the company to design goods that can be offered directly to customers and to enter that market. This would imply developing:

- Needed software for fast customization/implementation of clients' requirements. For example, valve or sensor control automation, automation of owned equipment, or even data acquisition;
- Software for fast implementation of acquisition processes and laboratory analysis to validate clients' solutions/requirements.

This motivation led to the Automator Project, where the main goal was to develop a product to automate numerous sequences of tasks into multiple single commands. For example, if we want to analyze a piece, we first need to align it, take an image, and then analyze the image. Each of these steps, when joined together, corresponds to a sequence. The project aimed to create an application to generate multiple sequences. A good example of a similar product or something

close to what was envisioned is PeerGroup's Tool Orchestrator (PTO), an equipment automation and control framework with a GUI-based tool designer (PeerGroup, 2023b).

Although seeming a very simple and easy-to-understand concept, it proved to be very hard to convince everyone inside the organization about the applications for the product.

A part of the German team wanted to sell the product, others to utilize it in the laboratory, others to use it as an internal tool, and there were even those who believed that the company should not enter this type of business. Different mindsets and opinions, each with its own advantages and disadvantages.

With all these different views, the target of the project was not defined from the start because it was impossible to reach a consensus.

The product owner (PO) wanted something that would allow the integrators to develop the customer's solutions. The developers would create the first sequences, and then the customer could create their own or change the ones already implemented.

There were product management problems. One team wanted the development of analysis algorithms to validate what they already had and not industrial automation. Another wanted something very complex: having a full drag-and-drop feature inside the application. In fact, some of the requirements were so intricate, they almost looked like a programming language for itself and nearly as convoluted as one.

The software team knew that not every requirement could be met. For example, not everything could be drag-and-drop. Even inside the application team, which often is composed of physics engineers, who do not have a software development background, some can program something, and others, nothing at all. Everyone is a stakeholder, but no one is able to understand the other.

At a certain point in the development process, users could program in C# (programming language) directly into stages of sequences. These steps could be defined by code or drag-and-drop. This pointed to an increase in high-complexity features.

Additional conflicts and dilemmas arose when discussing User Interface (UI) requirements and runtime dashboards.

When attempting to satisfy the standards for laboratory cases of use, where studies have a declared beginning and conclusion, it is difficult to explain that there are industrial applications, such as system, line, or environmental monitoring, that function continuously and do not have a beginning and end.

Small functionalities/features continued to grow too much, even before the first real use case.

The project had two very distinct phases:

1. **Fast implementation of mandatory features and demonstration of proof of concept** - no problem in this phase, since every feature was needed for the application. Output was demonstrated with success.

2. **Definition of advanced requirements** - instead of defining the more general aspects of the application, the backlog was built out of details of how to configure sequences. There was a

total dispersion into small features and impossible to explain to the stakeholders. With each demonstration, stakeholders would be able to understand less of what the application was.

The project reached a point where things were so mixed up, so complex, and so full of little details that those involved in it could not see the global vision anymore.

The outcome of the project turned out to be a very good application but with no particular practical use, inside or outside the company. The Automator Project was then terminated and the application was put on hold for an undetermined time.

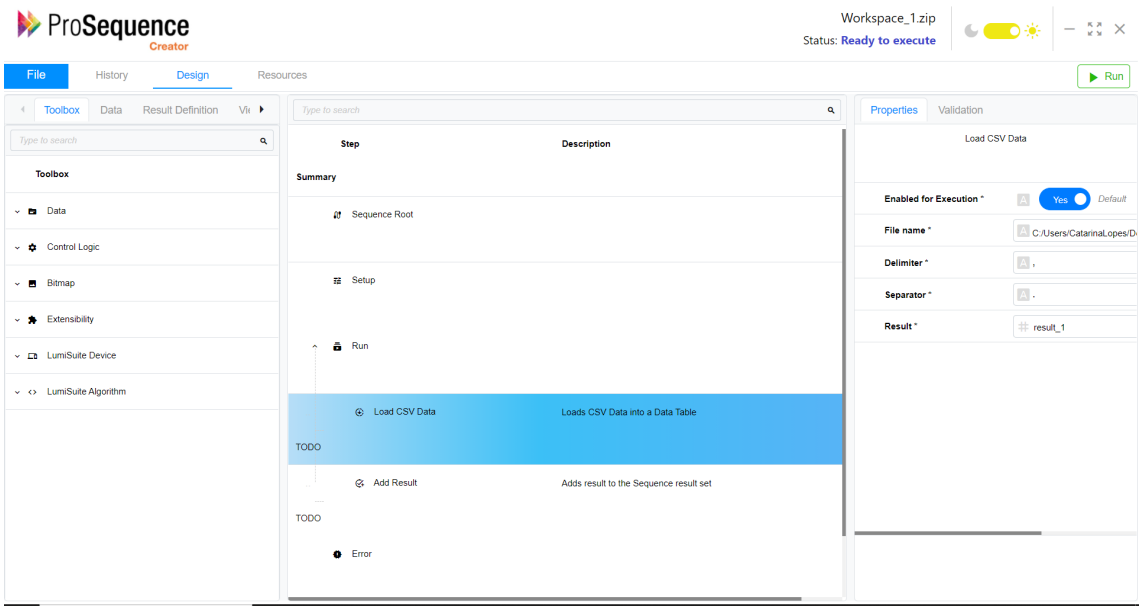


Figure 4.1: Automator Interface: Design View

Figure 4.1, shows the design view of the final delivered Automator application. In this view, sequences could be defined and constructed with the features already discussed. The user may drag and drop potential pre-defined stages from the toolbox onto their sequence on the left side. The primary sequence is specified and built in the middle, and then each step's attributes may be changed on the right, where the user can even program certain particular features.

4.2 Case Study Analysis

After over two years of execution, the Automator Project was declared a failure.

This is an excellent example of how product management decisions can make or break a project. In this case, since the beginning, the main target was not defined, and with no clear vision of what needed to be done and for what purpose, the different teams involved had divergent ideas and that was where the real problems started to surface.

To try to fight the misunderstanding between teams and their wishes for product requirements, the PO's team strategy was to try to satisfy everyone's demands. This led to high-complexity features, that grew until the main goal was completely lost in micro features. The requirements did

not fit together, and the application seemed somewhat useless for much of the initial requirements planned.

With each iteration, it seemed that the complexity increased and the notion of the application decreased. Due to the stakeholders' lack of understanding of the use cases for which the features were built, it was becoming more difficult to explain to them why the backlog was so full of complex features.

Without a clear purpose in sight, the software team simply followed the PO directives while being free to interpret the goal as they saw fit.

The lack of market research on current applications or comparable solutions was a major issue with the Automator Project. Without market research, the company was unable to have a correct perception of the product and customers' needs.

With market research, teams may develop attainable company objectives, rather than relying on vague ideals of quick success. It is easy to want to be different when developing new software for consumers, and to get lured off track by new technical features. However, market research has proven that what clients value most is their issues solved. Market research assists in generating, defining, and prioritizing these problems.

In this project, there was also a lack of management of stakeholders' expectations from the PO.

Assuming that producing in-house was a good decision, that may not be the case, the company should have started with extensive market research and then created a pilot application, with a finished backlog, or at least a very completed one, to have clear goals and requirements.

With the pilot application, a stakeholders meeting would have been the ideal choice to determine whether the pilot met the client's expectations and then proceed with the project based on that initial impression.

4.3 Key Takeaways

The key premise of the project is that the PO must recognize that it is impossible to please every stakeholder involved in the project. The client's wants and needs should be the priority and communication needs to be fostered between teams in order to have clear requirements at all times and work as a whole.

In brief, the biggest problems found in this project are:

- Lack of defined leadership;
- Misunderstanding between teams involved in the project - communication issues;
- Inaccurate requirements gathering;
- Lack of project goals;
- Lack of market research;

- Issues with expectations management inside the company.

What should have been done?

1. Market evaluation and analysis - compare to existing applications and tools;
2. Clear vision and goal from the start of the project;
3. Management of stakeholders' expectations;
4. Reduced set of primary stakeholders;
5. Prioritize communication between different teams working on the same project.

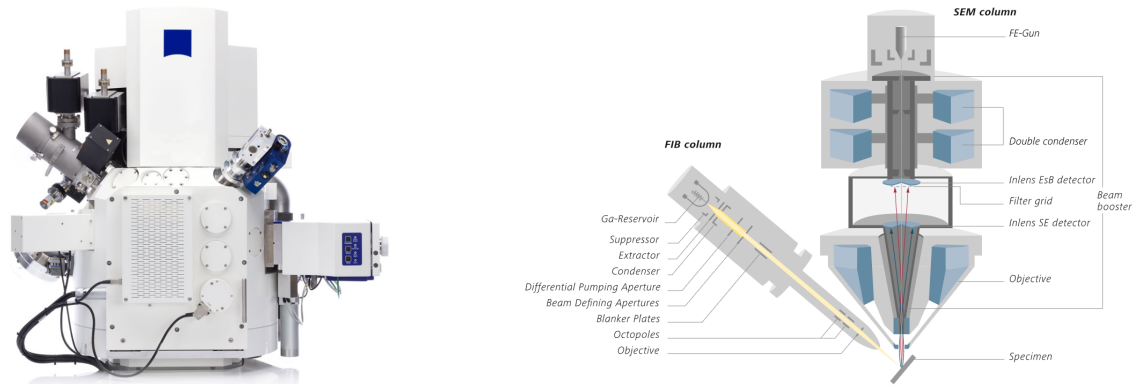
Chapter 5

Tomorrowland Project

The Tomorrowland Project is one of the ongoing initiatives at LSI. The primary objective is to create metrology equipment for the semiconductor industry, utilizing the existing capabilities of a standard Cross Beam microscope.

5.1 Case Study Description

A Cross beam consists of a Scanning Electron Microscope (SEM) and a FIB (focused ion beam) plus one or more GIS (gas injection systems). Picture 5.1 corresponds to the real representation of the microscope and its constituents.



(a) Cross Beam Microscope

(b) Cross Beam Details

Figure 5.1: Cross Beam Microscope and it's Constitution

The software team's goal in controlling the microscope is to design an application that would allow scientists to control all activities from the interface. The program includes functionality such as the ability to load, unload, vent, and even operate the cameras used to photograph the wafers.

The device is responsible for handling wafers and giving the user the ability to load wafers and navigate on them.

Since it's a device with lots of different features and components, there are multiple teams working at the same time on the project, with different requirements and goals in sight.

The software engineers aim to create a foolproof application that can be marketed to clients while simultaneously being utilized in the lab for proofs of concept and tests. By inventing new features and parts, scientists working on microscope components want to increase the capacity and performance of each component. Above all, application engineers aim to develop a marketable product that will be successful in the semiconductor industry.

From the start, the management team assumed that a functional base with a working platform already existed in the microscope. They thought that the application already had proof of concept, and skipped that step.

Another element that was absent at the start was a specification of the features needed for the final application. Knowing the features that the client needs for the application and for what purpose allows developers to design in accordance with those objectives, making development faster, more focused, and with a clear set of goals in mind.

Without a proof of concept and a final user profile, the application engineers were struggling to create a tool that had any real use. Different teams started to disperse in requirements, and software was drawn with too many detailed features, without ever proving the real value of the application.

Requirements started to be harder to reach and with all the teams working on the same code base, with different styles, issues started to appear. Teams started to disagree with each other, and none respected the segregation of responsibilities.

Development slowed, and objectives and goals from several teams became lost in the details. After two years, there was still no proof of concept, and the product was not yet ready for the market.

A decision was made by the management team. After evaluating different options, a German third-party solution was integrated into the microscope for automation control to substitute a part of the application. This allowed teams to focus on proof of concept and gave more security to the whole operation.

With a diminished workload, features could be defined more clearly and requirements aligned between teams.

5.2 Case Study Analysis

The Tomorrowland Project is now growing at a slow rate, but things are getting done with more drive and attention than previously. The management team's fundamental failing in the early phases was thinking that the application already had proof of concept and skipping that step. Proof of concept ensures the product brings value to end users, while it helps developers understand the limitations of their idea.

If management had begun by focusing all efforts on a proof of concept rather than rushing into production, a prototype with clear ideas for cures to the key problem that the program is

attempting to ease would have been especially valuable. The prototype might be given to the ultimate user to see whether the initial concept meets expectations and gather feedback to proceed with development.

This would also help in terms of not having too many project goals at the same time from different teams.

Management should also evaluate how many teams are operating at the same time with the same instrument, the microscope. Each has its own set of criteria and objectives. This might become too complicated to handle since everyone wants to strive towards their goal without considering the motivation of the other teams. Work becomes slower and some implementations might affect another's work without sufficient meetings and communication channels to discuss what is occurring and who is working on what. For example, if the product development team upgrades one of the microscope's components, the tool becomes unavailable, and the software team loses productivity since they lose test time on the tool to create their application.

It is critical for each team to be aware of what is going on around the product since features from one team may correlate to the goals of another. When this occurs, people can collaborate and increase performance by working together.

5.3 Key Takeaways

The project's core premise is that management should delay production and development until a very thorough proof of concept has been established. Having specific objectives and specifications that correspond to the requirements of the customer is essential.

In brief, the biggest problems found in this project are:

- Lack of defined leadership;
- Misunderstanding between teams involved in the project - communication issues;
- Inaccurate requirements gathering;
- No proof of concept before development;
- Focus on the main project goal;
- Poor milestone definition.

What should have been done?

1. Foster co-creation of requirements to balance different teams' goals;
2. Focus initial effort on proof of concept;
3. Prioritize communication between different teams working on the same project;
4. Create a prioritization chart and concentrate goals into small phases of the project.

Chapter 6

Alpha: Scanning Electron Microscope Project

Intending to develop an electronic microscope of the finest quality with innovative and complex technology, a German company decided to implement the Alpha Project and integrate LSI as the main development team.

6.1 Case Study Description

The Alpha microscope is characterized by an electron beam corrected by a magnetic mirror. Figure 6.1, shows the real tool.

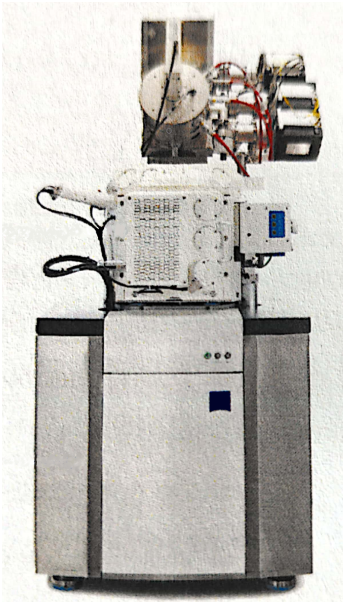


Figure 6.1: Alpha Microscope

There are three main teams involved in this project:

- **Product Development Team focused on performance (scientists)** - responsible for putting the machine together and adjusting it for optimized results. They are one of the main stakeholders at this stage of the project and their requirements have top priority. The software features requested will be focused on the tools needed for the development of the equipment itself, not for the final application
- **Software Development Team** - responsible for the development of the software application
- **Production Requirements Team** - composed by designers and business supporters. Responsible for designing the product and its specifications and helping the PO maximize product value

Before any software development, the first stage of the project was the proof of concept (creating evidence and documentation about the feasibility of the idea), lasting more than 10 years and applied by a team of scientists/physicists. Here, only the adaptation of old components was needed to create test applications to prove the value of the project.

Once they knew the product worked and had value, then the team moved its efforts into constructing a prototype, and the software development could begin in parallel.

Since the product development team focuses on building the best microscope possible, they tend to take less into consideration the application and software for the product. The software team focuses on creating the application for controlling the microscope and exploring the potential for mass production of the application. Developers want to generate high-quality products as quickly and affordably as feasible, whereas scientists typically take longer and incur more production expenses to ensure the quality of their contribution.

The product development team is the one that draws the machine and defines the requirements at this stage of the project. The software team can speed up development by taking those requirements and implementing them precisely as the lab gives them. Still, after the first iteration, there is no usable product to provide to customers.

The biggest challenge is trying to make the product development team and software developers work together and fulfill both wants and needs, towards the best outcome possible.

The continuation of the project came from the decision to implement a new software, that could replace the one already in use, with new features and better functionality. For that, a deal was made with the laboratory to create an independent interface on the outside to control the microscope. This interface would integrate the possibility to write MatLab scripts to control the microscope for laboratory investigations, tests, and proof of concepts. The end goal was to migrate those scripts into the application since they could not go to the client.

The result? MatLab scripts ended on the client side including everything the scientists did with the application. There was not enough time to migrate everything from the scripts and even tools for testing product assembly, calibration, and others, not requested by the client, came along.

The project was rushed and since the application had external MatLab scripts, it was not thread-safe (code will work even if many environments are executing it simultaneously) and was not tested to work in parallel with the microscope. Multiple episodes of bugs occurred, many

times impossible to diagnose. The product was delivered but with a high risk in quality policies (not fully tested, and with ongoing bugs still in detection and treatment).

At this moment, there are six operating Alpha microscopes, spread across the world. The project is still in the planning phase for mass production which will imply a redesign of the equipment and new software development for the production line. Each microscope is different and needs specific software configuration, according to its purpose of utilization, for industry or investigation.

The profile of the user is important at this stage. Knowing the customer and their requirements makes it easier to plan and develop a trustworthy product. For industry purposes, the results are the most important, together with the reliability of the equipment, without failures and with an uptime of 99,5% or more.

A question that remains is if the project should prioritize production tools for the mass production of the microscope or not, depending on the decisions made in this phase of planning.

6.2 Case Study Analysis

This case study focuses on the difficulties that occur inside a project's team, when different positions have distinct visions of the product, and of how it should evolve and be developed.

The conflict between stakeholders in the definition of requirements for the long term and having a clean requirements architecture is the main problem. It is difficult for them to comprehend each other and come to terms with small increments of functionality that could be beneficial to the final outcome. This could generate value not only for the scientists, that need the application for their daily work on improving and developing the machine, but also for possible clients that may want to buy the microscope.

When both sides have low communication channels and do not fully comprehend the work being done on the other side, problems surface. The project was rushed and delivered with very high risk in quality policies. That was something unusual for software developers, who always have the goal to deliver applications tested and fully working, without quality problems.

With this case, we understand the distinct mindsets of the product development team and software developers in the metrology for industry and science. The first is very methodic in presenting results and observing them but is chaotic in the implementation of solutions, and the second, with a much more open mind and agile style.

Analyzing the project, it is possible to comprehend the importance of direct contact and communication between all involved in the project. The process of defining requirements, if done only by the laboratory people, becomes very formal, and rigid, and features come out according to their goals, not taking into consideration other applications for the microscope or the software developers' opinion on the matter. When we look at the goal of each group, the product development team and developers, they differ, and so do the requirements for the control of the microscope and the application.

For instance, the product development team has a requirement to have a tool in the software to perfect the alignment of an electron beam. They want an interface that allows them to have this tool and experiment with different algorithms. The same tool is needed by the application team but from the perspective of the client. Thus, it must have a corporate aspect and have configuration options that are quite different from those used in the lab.

It is key that the software developers understand the application of the microscope as a product and the vision that the lab has of it, as is the opposite, that the product development team comprehends the view of the developers for the control application. When both sides are aligned, both can negotiate, suggest alternatives, and come up with innovative solutions for the project.

Communication is the base for better final product quality, a successful project, and an increase in speed of development. The finished application can be used in conjunction with the scripts created by the product development team in the laboratory and in accordance with the needs of both parties.

Another relevant conclusion was the lack of leadership and a strong voice of the PO to set everything in motion and clearly define milestones for feature implementation. The PO should be able to understand the requirements of the several stakeholders in order to prioritize and redefine the overall project design. Both the functional and the technical aspects of the requirement have to be understood. Technical specialists and application specialists will often lack understanding of one another so the PO role has to be fulfilled by someone overarching the domains.

6.3 Key Takeaways

The main takeaway from this project is, when having distinct profiles inside a project, like the product developers and software developers, it is key to find common ground between them. If not, the project can quickly become involved in conflicts. Another important aspect of this project is how rushed it was, which led to issues with the product's quality.

This project clearly lacked defined leadership. The team is unable to work with defined goals and vision because no PO was not prepared for the job. The PO should be the one who could make sense of the various profiles and persuade them to cooperate in order to achieve a common objective. These issues could be avoided with clear requirements and direction from someone who is mindful of all parties involved.

In brief, the biggest problems found in this project are:

- Misunderstanding between teams involved - communication issues;
- Requirements definition not according to stakeholders' wants and needs;
- Project was rushed to deliver the product to the client;
- Lack of defined leadership.

What should have been done?

1. Prioritize communication between different teams working on the same project;
2. Track progress and schedule periodic meetings to share the current status and come up with solutions to resolve possible problems;
3. Take time in process and requirements definition. A well-defined project increases transparency and helps to better understand the goals.

Chapter 7

Beta: Applications of Alpha Microscope in Semiconductor Industry

The Beta Project corresponds to the application development of the Alpha microscope that was designed in Chapter 6. At this stage, a development team continues to exist, but the final product is not yet fully designed. There is software that appears to be similar to what the client wants but lacks the tools required for the production level and service.

7.1 Case Study Description

The requirements imposed for this phase include process development for possible industrial applications. There is a greater concern for optimization and ease of alignment in the microscope, which is one of the characteristics available with its use.

Here, the PO requires to perform several tests with a precise list of points with different types of conditions and wants the software to be configurable, with the help of auto functions. This is transmitted to the rest of the team as the goal of the phase. Software development and microscope development occur at the same time.

The application goal was to be able to create an automation language inside the software that would allow the creation of sequences of tasks, something close to the application from the Automator Project in Chapter 4. In this scenario, we needed something so simple and straightforward that even someone who isn't a physicist or developer could utilize and put together their method.

After many iterations of development and evaluation of similar tools in at least three other company products, the development team concluded that developing something too generic, that works for every single use-case and system, has unsupportable costs, both monetary, in time, and in support.

To provide a balanced solution in terms of quality, cost, and speed to delivery, the first option was to give low-level interfaces to a team of application support developers who worked with MatLab/Python languages. There was no capacity to look into automation because the focus was on the reliability and testability of the solution.

This alternative worked for the first results but soon proved to be insufficient because the scripts were not configurable or robust. There came the second alternative: implementing the solution in the final software, without generalization, with a specific user interface.

With a cost reduction of fifty times more than the first option, this turned out to be the best alternative, the one still in development now.

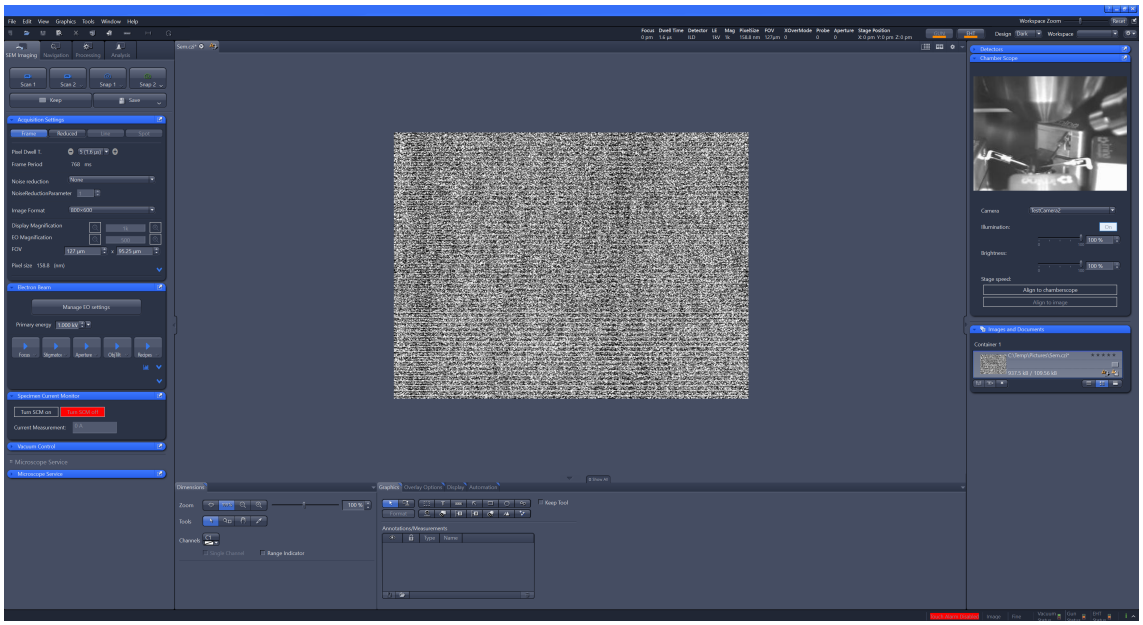


Figure 7.1: Beta Application Interface: Design View

Figure 7.1, shows the main interface of the Beta Application. The interface in view is the one that captures images directly from the microscope, including all the features regarding the control of the microscope camera in order to generate an image with all the settings intended.

The panel to regulate the mode of image capture and every command that makes it possible to take a picture of the microscope are displayed on the left. The image that was shot is shown in the middle, and on the right is the view of the microscope chamber. Finally, you may modify the picture sizes, colors, and other characteristics at the bottom.

7.2 Case Study Analysis

The second phase of the Beta Project presents a different view than the first phase. This is a good example of how decisions in project management can impact the project for the best outcome. When the PO saw that the first alternative was taking too long and was not sufficient to meet the client's requirements, the decision to change to another alternative proved to be the best judgment.

The key factor is having an Agile methodology that is efficient and a PO that follows the structure and status of the project at all times. When the PO and the team have a good communication basis, the project flows and requirements are fulfilled rapidly.

When the requirements needed to be changed, according to the results from the first alternative, Agile proved to be adaptable, and with low effort, the team quickly changed its mindset to meet the new requirements.

This is a good example of how Agile works well if all parties are invested and motivated. Above that, constant communication with stakeholders proved to be the decisive factor for success.

7.3 Key Takeaways

The main idea of this project is, contrary to others, to demonstrate an example where product management was a success. It is from this type of projects that we can learn and collect information for future ones.

In brief, the biggest triumphs found in this project are:

- Agile works well when the whole team is invested and motivated;
- Communication is the key factor for every project;
- Right from the start, project goals were communicated and PO made sure every team involved understood them;
- Multiple iterations with continuous testing and assessment against other similar tools were helpful to quickly grasp the project's flow and come to team-based decisions. This allowed for an immediate response and alteration in line with the conclusions;
- Strong leadership with a good PO, that guided the team to success and clearly defined project goals, requirements, and prioritized what needs to be done first.

Chapter 8

Conclusions

This chapter presents an overview of all the projects described and analyzed in this dissertation. From their comparison, conclusions can be drawn and a best practices guide can be produced in order to help with the decision-making in product management for future projects.

8.1 Overall and Comparative Conclusions between Case Studies

This dissertation's major objective was to learn more about four case studies and comprehend the key ideas while examining project and product management in the software development industry. The business would be able to recognize and handle the current difficulties by contrasting them.

The ultimate objective was to improve the future decision of priorities, requirements, and system design when confronted with stakeholders and use cases with very different profiles.

To achieve all of the goals proposed in this dissertation and to answer every question that prompted this work, thorough theoretical research was conducted and presented, followed by a very detailed description of each project, as well as an analysis of the main impediments or conflicts for each.

This conclusion seeks to answer the main concerns of LSI by giving comparative research on project disputes. To address question number one, what particular challenges and bottlenecks are influencing the company's project management and software development processes, a comparison table is created by compiling the major problems identified in previous chapters.

For question number two, what has been discovered to be the genuine reasons for the issue, a comprehensive investigation of each case study and specific issue is required in order to deduce what can be the sources of such conflict.

Finally, for question number three, how can the company resolve these concerns, a best practices guide is created to assist the organization by providing recommendations on how to avoid those issues from occurring and negatively impacting the project's conclusion.

The major purpose of the dissertation is successfully accomplished with these measures.

Given the examination of each case study, it is feasible to analyze the project management events inside each project and deduce similar issues, choices, and team evolutions. These conclusions will give key insights for future projects at the company and also aid other similar organizations in the market.

Table 8.1 is provided to help with understanding the main difficulties in each project and to compare them. It lists the most significant project management issues identified through analysis in earlier chapters, along with the projects for which they were discovered.

Table 8.1: Problems found in each Project. Comparison between Projects

Problem	Automator Project	Tomorrowland Project	Alpha Project
Lack of a Defined Leadership	X	X	X
Communication Issues	X	X	X
Inaccurate Requirements Gathering	X	X	X
Lack of Project Goals	X		
Lack of Market Research	X		
Issues with Expectations Management	X		
No Proof of Concept Definition		X	

The three main problems, found in every project, were related to the lack of a defined leadership, communication among the teams involved, and issues with the definition and fulfillment of requirements. This finding is not unexpected given that several studies conducted recently identify these factors as the most frequent causes of project failure.

Numerous issues with the start of the planning and development process exist in addition to these. Project goals, market research, and proof of concept issues are connected to choices made early in the implementation process, sometimes motivated by a desire to complete the process quickly. These are crucial because if project managers don't take their time, in the beginning, to define all of these components and devote time and effort to all of this, additional difficulties will surface along the way.

The main causes for each of these problems were found to be due to:

- **Lack of a Defined Leadership** - lack of a strong PO that provides guidance and decisions when needed, as well as prioritizing the requirements according to goals and due dates
- **Communication Issues** - insufficient engagement between teams and stakeholders; absence of communication plan; no project progress track; one team does not understand the vision of another

- **Innacurate Requirements Gathering** - absence of project goals; no communication plan; too many requirements from different teams that do not fit together; lack of planning meetings to discuss and reach a consensus about requirements together
- **Lack of Project Goals/Market Research** - project rushed to bring faster value to the client
- **Issues with Expectations Management** - Lack of thorough planning; incorrect estimates of the costs, materials, and time needed; poor communication amongst all parties engaged in the project

When it comes to team communication, it is critical to first understand the various profiles inside the projects. Since we are dealing with microscope development with applications, each project has numerous teams, each with its own set of requirements and goals. Usually, there are four main profiles coordinated by one Project Leader:

1. **Scientists** - these have the goal to develop research and investigation regarding the microscope. Their requirements involve improvements and tests on the tool. These usually take long periods of time and focus only on creating the best microscope possible
2. **Software Developers** - these create the main application for the microscope. Their requirements include creating the key elements, such as the UI, user controls, and microscope controls, as well as multiple interfaces that link to all equipment inside the microscope. Their focus is on building a quality application, that can be used by the stakeholders and sold to customers as part of the microscope's pack.
3. **Application Engineers** - these serve as a link between customers and technical teams. They employ client feedback and sales data to build and solve complicated software challenges and applications. They give technical help and knowledge to customers by testing apps and reacting to comments.
4. **Product Developers** - these work on single components that belong in the microscope. For example, the microscope has an SEM column and an OM column. Each has a product development team that focuses on enhancements and the construction of those columns. Their primary focus is on testing and enhancing the quality of their column as well as data collection in order to increase the precision of the microscope.

With such different profiles inside the main project, it might be hard to find common ground and for project managers to understand how to connect all these people. In reality, if one of these profiles is given too much authority and influence, an imbalance arises and the project loses track. Too much power in the hands of scientists slows development; too much power in the hands of application engineers causes requirements to become overly comprehensive and full of minor details, as seen in the case studies.

It is important for project managers to take the time to understand which requirements fit together, and which do not but above that, frequent meetings with all the teams are important, not

only to keep track of performance and development but also to work on requirements, plan what needs to be done by which team, and have space for all to talk and resolve problems together.

Project teams that collaborate can achieve a common goal more effectively and efficiently than people working alone. It is impossible to please every entity, but it is possible to please the project as a whole.

8.2 Best Practices Guide

A best practices guide can assist project managers grasp the essential points to consider and suggestions to keep the project moving in order to overcome and prevent challenges like the ones listed.

Table 8.2: Best Practices Guide

Problem to Avoid	Guidance Tips
Lack of a Defined Leadership	• Select a PO who can be upfront and articulately state what is required, what is lacking, and the course of action • The PO should, first of all, clarify the main goal of the project to all team members and make sure everyone understands • Choose the right leadership style for each team. In this case, an Agile PO is required to be involved in the whole development process • The PO should focus on maximizing the value of the product and delivering it to the client • PO should take responsibility for the success of the product and the project

Communication Issues	• Have a thorough strategy for communication that includes what needs to be said, how often, the channels to use, and who is responsible • Invite stakeholders to regular check-in meetings to learn how the project is progressing and whether it is meeting the criteria • Encourage team contact by holding social events and coffee breaks in addition to regular meetings and scrum reviews to boost participation • Track progress by keeping an updated backlog and sprint board • Showcase and modify leadership traits to meet the demands of the team and the individual • Be open to feedback and make adjustments along the way
Inaccurate Requirements Gathering	• Identify the relevant stakeholders and their requirements, prior to the execution of the project • Establish proper project goals • Document and confirm requirements by holding discussions with all parties • Prioritize requirements based on goals, then plan for incremental improvements in capability to meet each of those criteria • Monitor progress and make modifications as needed based on customer feedback

Problems regarding Project Goals	• Before beginning the project, define the objectives, goals, and milestones. This is the only method to ensure project success. • Understand the project's core vision and ensure that the team does as well • Use SMART Goals: Specific, Measurable, Achievable, Relevant, and Time-Bound. By defining these criteria in relation to your goal, you can be sure that your goals can be attained in a set amount of time. This method removes generalizations and hunches, establishes a precise time frame, and makes it simpler to monitor progress and spot missing milestones. • Set up a planning session with each team engaged to convey the goals and be open-minded to changes
Lack of Market Research/Proof of Concept	• Do not assume that conducting market research or providing proof of concept is easy and do not skip these procedures • Perform a proof of concept to demonstrate the project's worth and ability to truly alleviate the customer's problem • Before any development, perform complex market research to understand the market and take notions of existing products and competition
Issues with Expectations Management	• Do not rush the planning phase. Setting specific and reasonable expectations based on the team's skill level, the resources at hand, and the anticipated results is crucial • Include team members in the planning process to establish roles and responsibilities, timetables, metrics, and process estimation • Have effective communication with the tips above • Focus on milestones and work towards each increment

In terms of future work, it is crucial for the company to start by reading the guide and, before anything else, start by implementing an agile methodology where every employee has a voice and fosters teamwork. Additionally, it is crucial to develop a strategy from the outset and instruct each team on the project's values and methods of operation.

The management group should transparently teach its members using this guide. They can be better taught and prepared to respond when the first indicators of difficulties appear if they are aware of the primary issues that frequently arise from the beginning.

Most important, the company should always foster good communication between all teams involved in every project, and work with motivation towards success. The corporation can begin by setting up daily meetings to monitor the progress of each project inside the organization and occasionally setting up team-building activities like outdoor games such as football or trekking or, for instance, escape rooms to promote interpersonal collaboration.

The feedback received from the company regarding the findings and the best practices guide produced in this report was positive. They intend to implement this guide to improve the outcomes of future projects and improve their project management.

In what relates to this dissertation, future developments could come from studying a larger variety of projects and supervising the implementation of these tips and their impact on the future. By requiring teams to provide fresh advice and jointly produce a unique guide for each organization, these additional projects might not only validate the best practice guide but also improve it.

Bibliography

Aitzaz, S., Samdani, G., Ali, M., and Kamran, M. (2016). A comparative analysis of in-house and outsourced development in software industry. *International Journal of Computer Applications*, 141:18–22.

Akbar, R. (2019). Tailoring agile-based software development processes. *IEEE Access*, 7:1–1.

Aliyu, A. (2013). Project management using critical path method (CPM): A pragmatic study. *Global Journal of Pure and Applied Sciences*, 18:197–206.

Altameem, E. (2015). Impact of agile methodology on software development. *Computer and Information Science*, 8:9–14.

GeeksforGeeks (2023). Software development models & architecture. `https://www.geeksforgeeks.org/software-engineering-spiral-model/?ref=lbp`. Accessed: 2023/04/13.

Grant, A. (2020). High-level and low-level programming languages. `https://www.makeuseof.com/tag/high-level-low-level-programming-languages/`. Accessed: 2023/03/24.

IBM (2020). What is continuous integration? `https://www.ibm.com/topics/continuous-integration`. Accessed: 2023/04/04.

Islam, A. Z. and Ferworn, D. A. (2020). A comparison between agile and traditional software development methodologies. *Global Journal of Computer Science and Technology*, pages 7–42.

Jira (2023). Jira software. `https://www.atlassian.com/software/jira`. Accessed: 2023/04/14.

Keil, M., Lee, H. K., and Deng, T. (2013). Understanding the most critical skills for managing it projects: A delphi study of it project managers. *Information Management*, 50:398–414.

Khan, A. and Mir, M. (2021). Critical path method (CPM). *Research in Medical Engineering Sciences*, 9:1031–1033.

Kirovska, N. and Koceski, S. (2015). Usage of kanban methodology at software development teams. *Journal of Applied Economics and Business*, 3:25–34.

Lopez-Martinez, J., Juarez-Ramirez, R., Huertas, C., Jimenez, S., and Guerra-Garcia, C. (2016). Problems in the adoption of agile-scrum methodologies: A systematic literature review. *Proceedings - 2016 4th International Conference in Software Engineering Research and Innovation, CONISOFT 2016*, pages 141–148.

Mahalakshmi, M. and Sundararajan, M. (2013). Traditional sdlc vs scrum methodology – a comparative study. *International Journal of Emerging Technology and Advanced Engineering*, 3:2–6.

Mathur, A. and Acharya, A. (2015). A comparative study on utilization of scrum and spiral software development methodologies: A review. *International Journal of Engineering and Technical Research*, Vol. 4:90–94.

Mens, T., Cataldo, M., and Damian, D. (2019). The social developer: The future of software development [guest editors' introduction]. *IEEE Software*, 36:11–14.

Mirza, M. and Datta, S. (2019). Strengths and weakness of traditional and agile processes - a systematic review. *Journal of Software*, 14:209–219.

Nimble (2023). Swiftkanban. `https://www.nimblework.com/products/swiftkanban/`. Accessed: 2023/04/28.

Paulk, M., Curtis, B., Chrissis, M., and Weber, C. (1993). Capability maturity model, version 1.1. *Software, IEEE*, 10:18–27.

PeerGroup (2023a). Equipment automation. `https://www.peergroup.com/products-services/oems/equipment-automation/`. Accessed: 2023/04/19.

PeerGroup (2023b). Peer tool orchestrator (pto). `https://www.peergroup.com/products-services/oems/equipment-automation/pto/`. Accessed: 2023/04/25.

Peterson, T. M. (2007). Motivation: How to increase project team performance. *Project Management Journal*, 38:60–69.

PMI (2017). Success rates rise | pulse of the profession 2017. `https://www.pmi.org/learning/thought-leadership/pulse/pulse-of-the-profession-2017`. Accessed: 2023/03/24.

PMI (2023). What is project management? `https://www.pmi.org/about/learn-about-pmi/what-is-project-management`. Accessed: 2023/04/17.

Professional, D. (2019). Continuous integration best practices. `https://www.devonblog.com/continuous-delivery/continuous-integration-best-practices/`. Accessed: 2023/04/10.

ProgramizPRO (2022). High-level vs. low-level programming: What's the difference? `https://programiz.pro/resources/high-level-vs-low-level/`. Accessed: 2023/04/20.

Rehkopf, M. (2023a). Kanban vs. scrum: which agile are you? `https://www.atlassian.com/agile/kanban/kanban-vs-scrum`. Accessed: 2023/05/16.

Rehkopf, M. (2023b). Scrum sprints. `https://www.atlassian.com/agile/scrum/sprints`. Accessed: 2023/04/17.

Reiff, J. and Schlegel, D. (2022). Hybrid project management – a systematic literature review. *International Journal of Information Systems and Project Management*, 10:45–63.

Rozhnova, T., Tomachynska, V., and Korsun, D. (2022). Life cycle models, principles and methodologies of software development. *Scientific Collection «InterConf+»*, (28(137)):394–401.

Saeed, S., Jhanjhi, N. Z., Naqvi, M., and Humayun, M. (2019). Analysis of software development methodologies. *International Journal of Computing and Digital Systems*, 8:445–460.

Salameh, H. (2014). What, when, why, and how? A comparison between agile project management and traditional project management methods. *International Journal of Business and Management Review*, 2:52–74.

Schwaber, K. (2004). *Agile Project Management with Scrum*. Developer Best Practices. Pearson Education.

SEMI (2023). Semi standards. `https://semi.org/en`. Accessed: 2023/04/18.

Shylesh, S. (2017). A study of software development life cycle process models. *SSRN Electronic Journal*, pages 1–7.

Wells, D. (2013). Extreme programming: A gentle introduction. `http://www.extremeprogramming.org/`. Accessed: 2023/04/13.