Developer's Perspective on Implementing a Tool to Facilitate Development of Robotic Process Automation

Master's thesis in technology
University of Turku
Department of Future Technologies
Software engineering
2020
Sampsa Vuorela

University of Turku
Department of Future Technologies

SAMPSA VUORELA: Developer's Perspective on Implementing a Tool to Facilitate Development of Robotic Process Automation

Master's thesis in technology, 166 p., 0 appendices
Software Engineering
May 2020

Software robots are programs that can mimic human behavior to interact with other desktop or web applications used in various business processes. Robotic Process Automation (RPA) is an emerging technology that aims to automate digital processes with software robots. Compared to software development in general, RPA development requires more emphasis on communication between a project's stakeholders and on the documentation that supports it. Two important pieces of documentation of an RPA project are the Process Design Document (PDD) and the Solution Design Document (SDD). This thesis describes the design and development of a tool that automates the flow of information from a project's PDD to its SDD. The motivation for this PDD-SDD Tool is to facilitate and streamline the processes included in RPA development. Altogether three rounds of interviews with practicing RPA developers were conducted to create the first version of the PDD-SDD Tool. The developers were from MOST Digital Ltd, a company specialized in RPA and software robot development. The final interviews indicated the tool to be successful and it will most likely to be developed further to be taken into production use in MOST Digital. The project is based on a background study, conducted jointly with another master's student Ahmed Abdulghani. The background study is based on scientific publications, companies' white-papers, articles and authors' work experience. Both this and Abdulghani's thesis describe this background study, but are otherwise separate pieces of work.

Keywords: robotic process automation, RPA, software development, process flow, code visualization

Contents

List of Figures

List of Tables

Listings

1 Introduction

Development of Robotic Process Automation (RPA) projects is in many ways similar to the more traditional software development. However, projects are often shorter and contain relatively more communication with the customer. Typically, RPA software is used to automate processes previously done by humans that include interacting with desktop or web applications using keyboard and mouse. In RPA projects the visual workflow of the automated process has an important role as visualizations help both the customer and the developer to understand the process flow better. An idea emerged to create a tool which would help with the communication and the documentation of the short cycles of RPA development. If the tool would succeed, it should also provide help with all of the other development aspects, which include definition, development, testings and maintenance, as they are all tightly interconnected.

This thesis describes the tool's development process. The project was done with Ahmed Abdulghani, who also writes his own master's thesis about the same project, but from a different viewpoint. The foundation for the tool's design and implementation was created by gathering data from scientific publications, companies' white-papers, articles, authors' work experiences and from the interviews conducted during the project. In this thesis the RPA development and the tool designed, researched and implemented are inspected from a developer's viewpoint. In Abdulghani's thesis, the topics are addressed from a definer's and maintainer's viewpoints.

Authors of the both theses work at Ailea Ltd, a software company that is closely collaborating with MOST Digital Ltd, an RPA company located in Finland. MOST Digital Ltd provides ease for many customers by automating their processes with high quality software robots. The project described in the theses is done in collaboration with MOST Digital Ltd. The project contained four development iterations, which included three rounds of interviews. The interviews were conducted in order to gather the interviewees' thoughts about the ongoing project and to adjust its goals during the development process. The interviewees, who had all been working with MOST Digital's RPA projects, were gathered from both Ailea Ltd and MOST Digital Ltd. The interviews provided valuable information and notions of the daily work of a RPA developer, which guided the tool's implementation.

Both theses begin with four background chapters written together, but contain minor differences due to their adaptation to the particular context of each thesis. Chapters describe the general RPA development aspects and their relations to the more traditional software projects. The chapters also provide RPA examples, the initial design for the tool that was developed during this thesis, the presumptions guiding the development, and the Research Questions, which are separated and studied in the separate theses. The reason these background chapters were written is to provide the reader a clear view of the starting point and the prerequisites needed to fully understand the main goals of the separate theses. Both describe and study the same project from different viewpoints, which lean on each other and together form a complete picture of the project and its advancement.

After the background Chapters 2–5, the theses diverge and the authors discuss the project from their own viewpoints. Chapter 6 provides the roadmap for the rest of the project's advancement, as well as enlightens the structure and methodology used in this thesis.

At the time of the writing, the RPA technology is a relatively new subject and there are not many sources available, that describe the actual development procedures of RPA projects. This thesis hopefully provides more insights about the exciting daily life of software robots and their makers.

2 Overview of Robotic Process Automation

The time of robots and automation is upon us, but why now? Automation in general has been present for a long time, but what is the big fuss about it nowadays and why is it getting more attention now than ever before? To understand this, we must dive into Robotic Process Automation (RPA) first. Robotic process automation is a piece of software that executes processes in a computer's operating system, acting in many ways like a human would do.

This chapter provides an overview to RPA based on scientific publications, companies' white-papers, articles and the work done at MOST Digital [3]. The chapter is divided into four categories that examine RPA at a general level, discuss its potential and limitations, and provide current future views. More advanced examples of RPA software, work methodologies at MOST Digital and the goal of this thesis are introduced in Chapters 3, 4 and 5.

2.1 Robotic Process Automation

Robotic Process Automation (RPA) as a term is used to describe tools and processes that operate the user interface of other computer systems in a way a human would do [5]. RPA is especially useful in situations where redesigning and creating integration between systems is considered to be too difficult or costly. RPA gives organizations the ability to automate on the frontend, for example to gather data from applications via user interface. This kind of 'outside-in' approach has the advantage of keeping existing information systems intact. This is one of the main reasons why RPA is currently seen as a way to quickly achieve high Return of Investment (RoI) and that is what the companies that provide Robots as a Service (RaaS) are aiming for when delivering robots to their clients [5, 6, 7] .

Software robots can help us in many different ways. They can completely automate time-consuming and tedious processes performed by humans, or provide help to an employee during the process. RPA software has often been referred to as 'macros on steroids' because it can replicate computer-based tasks as many times as necessary and at a very rapid rate [7]. One thriving idea behind RPA is to supplement human work in a meaningful way so that companies benefit from both automation and more value-added work from their employees.

RPA combines different automation techniques, including screen scraping software, workflow automation and management tools, and artificial intelligence [8]. Software robots can navigate different digital landscapes and also make use of web scraping tools and other scripting tasks [5, 9, 7]. Nowadays, there are different tools to develop RPA software. Some RPA behemoths like UiPath[1] and Blue Prism[2] offer development platforms that focus on visual presentation of the software logic [10, 11]. Visual programming could be easier to work with, especially for people who do not want or know how to program. Along with platforms focusing on visual workflows, RPA is developed with many programming languages which have numerous technologies and libraries for software au-

---

[1] https://www.uipath.com/, 9.5.2020

[2] https://www.blueprism.com/, 9.5.2020

tomation.

RPA platforms are built to ease automation development, especially for people who might not have a programming background. The easiest way to familiarize them with the platforms is by using tools and programs that they are already familiar with. Most of these platforms offer a visual interface in which the developer can drag and drop activities or functions right into the robot's logic, in a Microsoft Visio [12] like fashion. The platform itself generates code automatically as users drag and drop or link activities and icons together on the visual platform. This feature enables the process and subject matter experts to automate their tasks after a few weeks of training. Developing automation without these platforms requires significantly more knowledge in software programming in order to achieve similar automation [6].

At MOST Digital, a generic RPA task is developed in a close collaboration with the customer who is familiar with the process to be automated [3]. This is because a well documented and communicated process is paramount to developing a robust robot. Once the developer is familiar with the process and the systems used in it, the programming can be started. Although, the robot is programmed to do the process in the same way as a human would do, it is sometimes possible and encouraged to optimize and streamline the process, for example calculations can be done without opening additional software. The end result is a software robot 'living' in virtual machine in the cloud or in customer's internal network, working autonomously or collaborating with human workers.

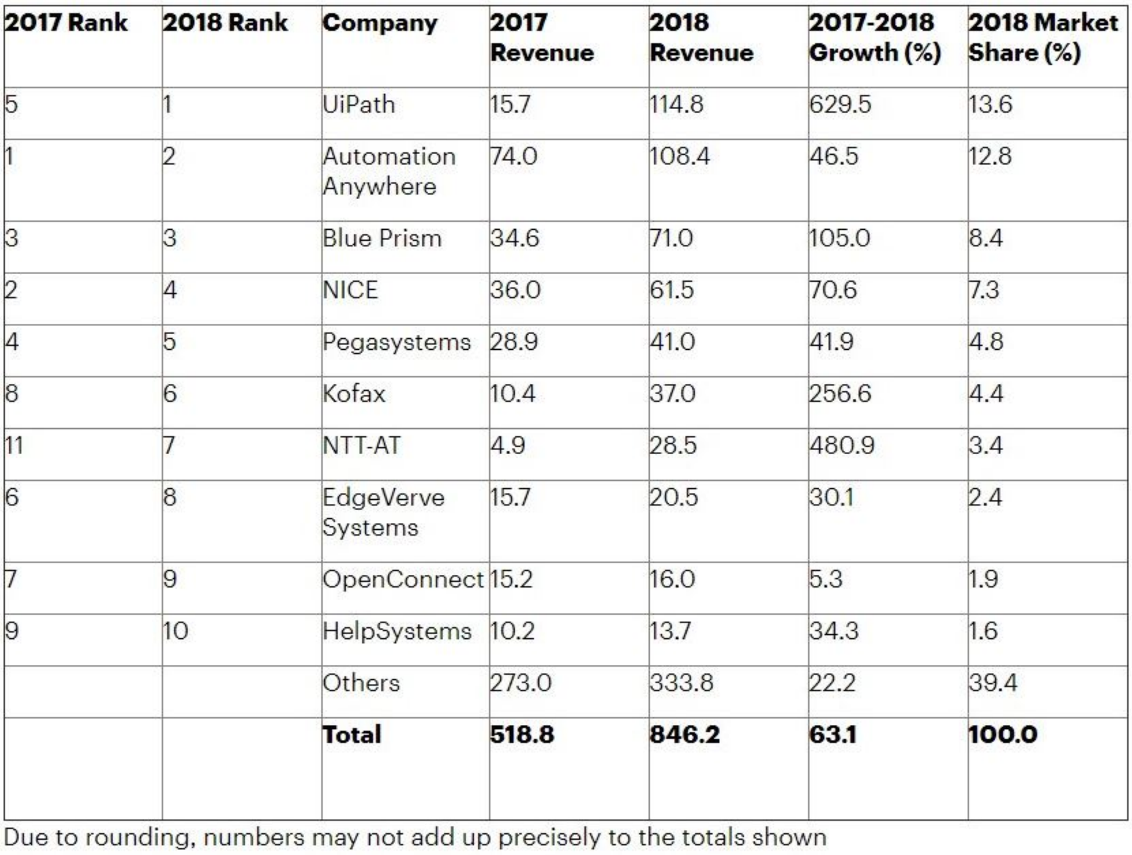
Developers at Most Digital have mainly been using Python[3] and its open source libraries[4]. These allow often a very flexible approach to automation task at hand, and without extra costs from licenses that are often needed with platforms like UiPath and Blue Prism. RPA software programmed with Python can include similar techniques and solutions implemented in these RPA Platforms, such as visual recognition, web scraping and operating system specific scripts to accomplish their goals. Visual recognition is not always the optimal way, but still often needed when the target software can only be accessed through remote desktop protocol (RDP) or programs like Citrix [13]. The robot is able to use scripts saved in the customer's environment and transfer files between machines. These files may be processed to get the necessary data for the automation's next step, or they could be reported directly to the customer, for example via email. Web scraping and browser automation tools are often used when customer's program can be accessed from the robot's own network. Occasionally, software robots may combine these tools, fetching some information via RDP and then forwarding it to another system directly from the robot's environment, or vice versa.

2.2 Potential

RPA is set to be the biggest game-changer for organizations from 2016 and beyond as it has now reached a point where it is mature enough to be easily and cheaply adopted [14]. It is non-invasive and can automate processes that utilize all of companies' line of business systems. These could be old legacy systems [15] or newer applications running in the cloud and desktop environments [7, 14].

As business process management (BPM) [16] has a legacy of long implementations and unclear business cases, RPA aims to achieve quick wins with little investment [5]. Users have gained multiple benefits when humans and robots have started to work together, including cost saving and faster processing with higher quality. Errors have decreased while regulatory compliance and output have improved. Also growth, productivity and satisfaction for both customers and employees have improved. Humans can now focus on the meaningful tasks that require innovation, creation and human interaction while robots handle repetitive and uninspiring tasks.

The 2019 market leader was UiPath according to Gartner's report from June 2019, illustrated in Table 2.1, as UiPath has captured 13,6% of the whole RPA market segment [4]. This is mostly due to their strategy of sharing the platform for free for community use, and by providing free RPA courses and certifications for interested developers on their RPA academy. By offering the knowledge for free, UiPath has gained a lot of interest within the RPA community.



2017 Rank	**2018 Rank**	**Company**	**2017 Revenue**	**2018 Revenue**	**2017-2018 Growth (%)**	**2018 Market Share (%)**
5	1	UiPath	15.7	114.8	629.5	13.6
1	2	Automation Anywhere	74.0	108.4	46.5	12.8
3	3	Blue Prism	34.6	71.0	105.0	8.4
2	4	NICE	36.0	61.5	70.6	7.3
4	5	Pegasystems	28.9	41.0	41.9	4.8
8	6	Kofax	10.4	37.0	256.6	4.4
11	7	NTT-AT	4.9	28.5	480.9	3.4
6	8	EdgeVerve Systems	15.7	20.5	30.1	2.4
7	9	OpenConnect	15.2	16.0	5.3	1.9
9	10	HelpSystems	10.2	13.7	34.3	1.6
		Others	273.0	333.8	22.2	39.4
		Total	**518.8**	**846.2**	**63.1**	**100.0**

Due to rounding, numbers may not add up precisely to the totals shown

Table 2.1: RPA software market share by revenue 2017-2018, worldwide (millions of dollars) [4].

The reason these RPA platform companies are gaining much traction is because of their marketing, and especially by marketing the fact that you do not have to be a programmer to create robots that automate tedious front-office tasks [17]. This marketing strategy was intentionally aimed at desk workers who might not have any programming skills but have a lot of repetitive tasks, thus making them a great target.

2.3 Limitations

While users can achieve many improvements with RPA, there are still areas where robots do not excel. Research shows that they still must be guided by human intelligence. Limits and possibilities for software robots are set by human will and imagination which are yet far too complex for artificial intelligence to replicate [14].

If a customer's process contains many unknown factors and a human worker has to continuously process things differently, the task might not be suitable for automation. Usually these kinds of processes do not have clearly defined input data or they include frequently misbehaving or changing programs, which makes a robust automation much harder to accomplish. However, there might still be a chance for improvement by further optimizing, defining and analyzing customer's processes. This way the problematic issues can be recognized and defined more clearly. The result might be an automated sub process of the original goal which still improves the total process when compared to the initial situation.

As with software development and project management, the project triangle [18] holds true in the world of robots. The quality of the developed robots are tightly linked to allocated time and financial resources. If these are insufficient, it may lead to a robot solution that demands more maintenance later on, thus creating technical debt. This is because the tight time frame imposes restrictions on defining sufficient error handling cases, which help the robot solution to cope with predefined error scenarios. Also, the robots are accessing software that might get updates, which may lead to runtime failures.

Robots can also make incorrect decisions when applied to human interface because of contextual changes. Some errors could be hard to notice and lead to disastrous situations when discovered. Also ethical and security issues arise when software robots impersonate people.

2.4 Future

Technologies that involve learning on the part of the computer are expected to have great potential for the future. When these are combined with RPA solutions the result could be an automated process which could apply a whole new level of learning, judgment and creativity to their work.

Continuously evolving technologies like cloud computing [19] and Internet of Things [20] create unprecedented explosion of data. As workloads keep rising, RPA software will increase productivity and help organisations to cope. At the same time it will be evolving towards truly cognitive automation where software will be able to improve its own performance and make complex decisions.

3 RPA Development

This chapter provides more detailed examples and discusses common characteristics of RPA development. Developing RPA is similar to common software development projects, but emphasizing the value of communication with the customer, who has an important role in the RPA life-cycle. The customer usually has in-depth knowledge of the process, and in addition to the desired result, the customer also instructs the steps in the process and how the the software robot should use required systems. Continuous communication during development ensures that the developer has understood the instructions correctly. In addition, customer guidance is often needed if undocumented exceptions may occur during the development phase.

The main aspects of RPA development are divided and discussed related to academic resources and the concrete work that is done at MOST Digital [3]. A comprehensive example of a simple RPA process is provided at the end of this chapter.

3.1 Definitive Characteristics

The most important aspects throughout the agile RPA development are undoubtedly communication skills and constant communication. Any problematic encounters with the customer's system must be dealt with swiftly by contacting the customer and coming up with solutions together, because they are the owners of the process and usually have the best knowledge of the system.

The aforementioned aspects have great impact on the process definition, thus it is imperative that the participants speak the same language not only on a technical but also on a business level. In order for the project to succeed, communication must be active, as the process definition might change during the development phase.

The following subsections cater a detailed understanding of the RPA characteristics. These contain process definition, communication, documentation, development, testing and maintenance.

3.1.1 Process Definition

Developing automation relies on business process knowledge, which means that a process must exists before it can be automated. Some process optimization might occur before diving into the development part. The starting point of any RPA project is the process definition phase, which usually happens by co-operating with the customer. Initially, the definition process is about transferring the process knowledge, and the way the customer uses their systems, to the developer.

The first time any RPA process is defined, discussions with the customer are initiated to find potential processes to automate. Oftentimes, the process is defined in iterations, as information about the process becomes clearer later on, or the developer might have more technical questions about the initial process documentation, which leads to another definition iteration.

The definition process should be documented in various ways for later use, and it also helps to keep the defined process in scope. Definition documents should enable the use of images, shapes and written text to convey the defined process step by step to the developer. Additionally, a video is advised to be captured of the definition session, for later inspection, as some of the minor details might not find themselves to the process definition document (PDD) [3].

Defined processes are rarely similar, yet it is important to standardize the definition process in order to generate congruent process definition documents. This ensures that the most critical process information is gathered the same way from every process. Challenges may rise if the customer prefers to document the process on their own, without utilizing the standardized approach. Oftentimes, in these cases the customer generated documents lack the essential information for the developer, which in turn demand redefinition iterations. These result in unnecessary costs in both time and resources. [21]

3.1.2 Communication

One major cause of problems in software development is a lack of clarity, not a lack of data [22]. RPA projects tend to be agile, fast paced and rely heavily on customer's instructions about the automated process. Usually in software development, the customer defines the desired final product. In RPA development, the customer also instructs how the process should be done. This does not mean that the developer cannot optimize the process, but the changes made to a customer's original instructions must be based on a clear understanding of how the customer's system is used and what the desired end result should be.

This special situation for many RPA projects means that the importance of efficient communication becomes one of the key factors for successful projects [1]. People perceive and understand things differently, for example customers may give instructions they consider to be obvious but contain a presumption that the developers know the system which is to be automated. Likewise, customers might get confused when developers explain things in too technical manner, with the presumption that the customer knows programming and its vocabulary.

Developers have to communicate with each other and with the customer to form a correct idea of the project, therefore the information flow in software projects has a direct impact on productivity and quality [23]. There are two major forces in software development communication, uncertainty and equivocality, which may occur if the given information is ambiguous and confusing [22]. Especially troublesome issues may arise in projects, where communication's importance is not understood. Project managers will not be able to estimate realistic costs for communication and cannot recognize and solve communication problems when they occur. Communication cannot be improved on the basis of experience if the causes of its problems cannot be identified.

To ensure effective communication, the possibility of misunderstandings, cross-interpretations and lack of information should be minimized [1]. Especially in Agile environments, communication processes should be continuously updated and optimized to the project at hand. In addition, synchronized working hours, unified communication tools and official process verification policies can have a positive impact on the quality of communication [24]. Last but not least, documenting should be done at different levels of formality.

The Media Richness Theory (MRT, Figure 3.1) is a good guide to help organize communication methods according to their media richness and it can be used when choosing the best method for different situations [1]. The theory suggests the usage of high communication media, such as face-to-face conversation, in situations where things tend to be ambiguous and low communication media, such as emails, when it comes to reaching a long-term common understanding. Notes should also be taken from meetings which use high communication media to document decisions made.

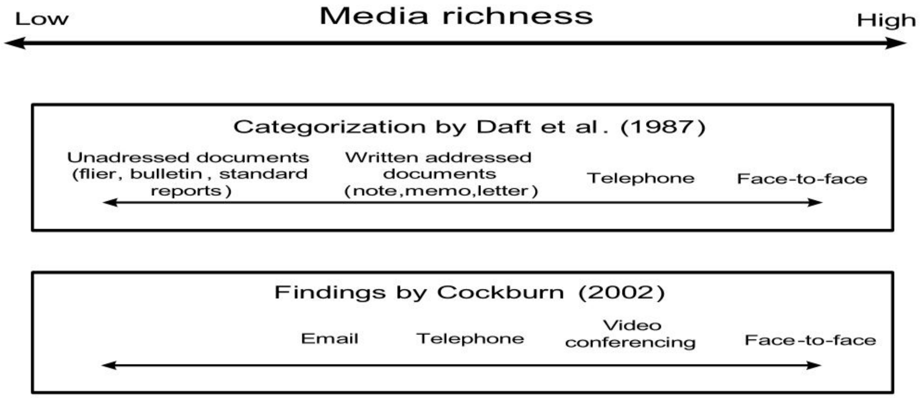

Low
Media richness
High
Categorization by Daft et al. (1987)
Unadressed documents (flier, bulletin, standard reports)
Written addressed documents (note,memo,letter)
Telephone
Face-to-face
Findings by Cockburn (2002)
Email
Telephone
Video conferencing
Face-to-face

Figure 3.1: The Media Richness Theory (MRT) with communication effects of different communication media [1].

3.1.3 Documentation

In addition to communication, good documentation is paramount for a successful RPA project. Although verbal communication may be sufficient for the project's advancement in the short term, good documentation ensures better and more reliable transfer of knowledge and decisions in the future. From this point of view, documentation has a direct impact on the quality of communication [3]. Lack of documentation affects both team collaboration and management decisions to support development in the long term. In these kinds of situations, people have to rely more on tacit knowledge, which could lead to various problems like difficulties to trace down previously made decisions, overall confusion and delays for development [25].

In the short cycles of RPA development, perfection and consistency are by far the most important attributes in good documentation, ensuring that differing assumptions could be minimized [26]. Also unofficial communication should be supplemented with documentation [24]. Unified working practices and applications support consistent documentation, while divergent practices can lead to fragmentation and confusion. Up-to-date and comprehensible documentation should be made available to all stakeholders. The customer should be informed with updates and modifications to the original business logic of the process, while developers and maintainers also need code-level documentation.

Usually high-level RPA documentation should model manual processes previously done by the customer's employees. A process should be documented by the customer in a way that the developers understand it. This is not always an easy task if the customer's employees have a lot of tacit knowledge about the usage of the tools to be automated. This kind of information is not always directly provided, and it needs to be purposefully searched during process definition. Additionally, documentation should be updated and maintained by the developers in a way that customers and maintainers understand it. High-level documentation should contain updates to the original business logic while technical documentation should clarify code-level decisions and structure.

3.1.4 Development

This section is based on authors' work experience at MOST Digital [3].

The development processes can differ depending on the developer and the project. Some recurring aspects can still be recognized that may cause challenges during development processes. Often the cause for delays and misunderstanding results from aspects outside the actual development, from unclear definitions and insufficient initial documentation. The main reason for this is poor communication between the developer, the project's definer and the customer. Therefore, the value of communication and documentation as a part of good development practices should be properly endorsed throughout the company.

At the beginning of project development, the developer studies the initial documentation in order to know what should be developed. The clearer and more precise the initial documentation, the easier it is for the developer to start working with the new project. Delays occur if the developer needs to clarify things he or she did not understand. Another cause for delays and extra costs could be misunderstood documentation caused by tacit knowledge that is not documented clearly.

From the company's point of view the developers should follow good development conventions. Best practices should be regularly instructed, discussed and reported to all developers so that everybody are at the same line. In order to accomplish this, the company's internal communication should work properly.

Changes made to the initial definition during the project development should be informed to both the project manager and the customer. Possible misunderstandings occur more likely at any stage of the development process, if the process definition is not actively updated. Both the developer and the definer should also ensure that outdated versions of the process definition are not left available to cause possible confusion in the future.

Oftentimes, the need for changes occur in older projects due to updates in software, environment or definition. In these situations the developer must study the old documentation and code base in order to become familiar with the project. This step is essential before any changes, and could cause errors in the developed code if not done properly. This kind of further development benefits tremendously from an up-to-date documentation of both high-level business logic and technical level resulting in faster development and less errors.

3.1.5 Testability

Testing is as important in RPA development as in other software development projects. RPA processes often contain visual elements and follow predetermined execution paths, therefore being particularly suitable for black box testing methods [27], where outputs are validated against process specifications without inspecting internal code-level structure. In this section we combine our work experience [3] with testing related studies [6, 27, 28, 29, 2, 30] to discuss testing in RPA development.

Testing helps to find the areas where the program does not meet its specifications and it is therefore a necessary area for software validation. Testing can be done in many different ways and to all kinds of activities, therefore any action that helps to find a violation of specification can be called testing. Usually, software testing includes three activities: testing, debugging and verification. The first can be further separated to static testing, testing during development and testing by execution. Figure 3.2 illustrates the usual phases and testing activities in general software development flow, also present in RPA development. Phases from system requirements to coding can be inspected and tested with static testing methods, which can involve design reviews, code inspections and static analysis of source code.

Verification is needed in design and coding activities to ensure that the program satisfies the original specifications. Verifying can be done by modeling the program in a formal language or using the program itself. Formal specifications derived from system requirements help to verify programs. Specifications can be tested on usage level and functional level to show that the code implements the specifications.

Debugging is usually done during the coding process, later stages of testing and in production or deployment. A bug means a defect, a part of code logic that fails to meet a specification. Debugging is performed to locate and fix faulty code, usually by analyzing and modifying the program's logic. The goal is to diagnose and the precise nature of a known error and correct it. Especially during later stages of development, the cause of bugs may rather come from incorrect specifications than faulty code logic. Locating errors can be a hard process involving double checking known inputs and code logic. Function level test can be done with unit tests to recognize function-level defects. Good coding and logging practices can help tremendously by reducing the need to make extra prints and logging while debugging.

Program usage can also be tested by execution, usually in the last activities: testing, production or development. This could be done with white or black box testing. White box testing inspects and validates the internal inputs and outputs of the program logic, while black box methods use a program without code level inspection [27].

Testing can be a difficult process to be done thoroughly, as Dijkstra points out: ”Program testing can be used to show the presence of bugs, but never to show their absence” [30]. This means that any amount of testing represents only a small sample of all possible computations. Test planning techniques based on partitioning of the functionality, data, end-user operation, et cetera, are very useful and popular and they form the base for many testing technologies.

Testing in RPA development follows the same patterns and practices as in general software development with some emphasised aspects. RPA testing relies heavily on output verification because it is usually performed on the terms of the customer's systems.

Due to restrictions in customer's systems or resources, test environments are not always granted, and the systems to be automated may be flawed. Testing should be done in a way

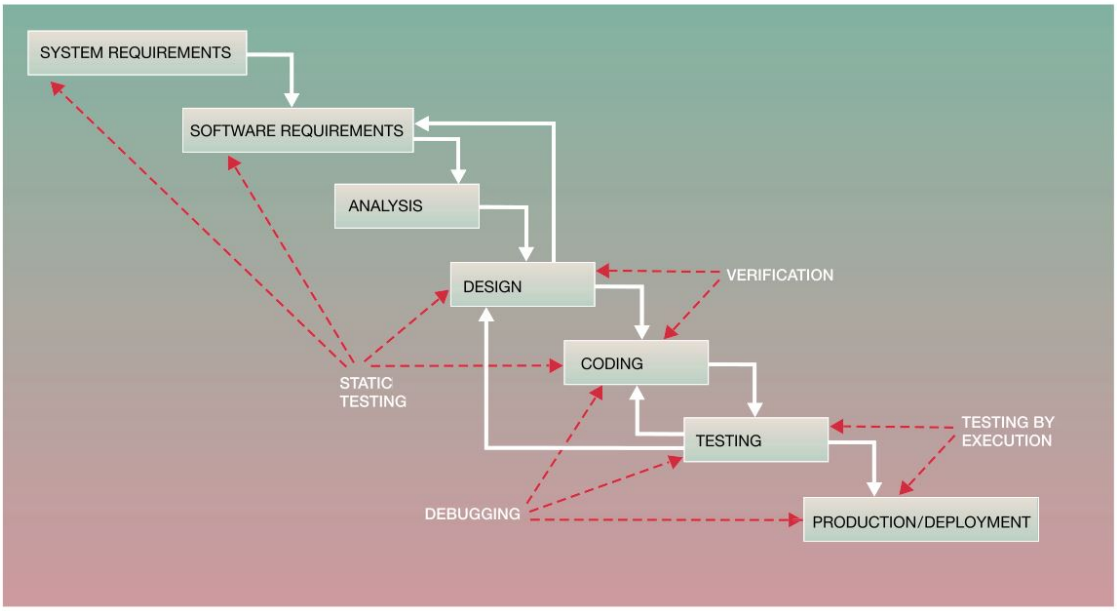


Figure 3.2: The activities in a typical software development process [2].

that production data is not submitted forward, which means that testing needs should be taken into account from the very beginning, when defining the RPA process. The better the definition, the better the implementation. Good PDD allows for better and smarter implementation that can be programmed to recover from possible errors and continue execution even in an unstable environments. If the PDD lacks essential information, it may lead to insufficient testing as well. Therefore, it is paramount to document all known errors and special situations in the definition phase.

3.1.6 Consistency and Maintenance

RPA software is often sensitive to visual changes and crashes may occur more often if the automated software is not stable. In these situations the employee who is responsible for maintaining the tasks has to check the situation and restart or potentially update the software. To do this effectively, some level of understanding about the task logic and the circumstances that caused the error is needed.

Studies have shown that a notable amount of resources used in maintenance is spent in understanding the code [29]. More complex systems are harder to maintain and tend to become even more complicated as time goes on. RPA solutions can differ a lot from each other and completely similar logic between tasks is relatively rare. But when inspected closely, there are similar base functionalities and structures that could be same between different projects. This means that the basic components that are used in different RPA tasks can be unified and maintained more easily.

The maintenance of RPA tasks requires knowledge about the business case in addition to the actual logic, thus an updated Solution Design Document (SDD) is imperative. If RPA tasks follow the same predefined basic guidelines with their high level code structure and basic behaviour implementations as well as unified version control practices, maintenance becomes significantly easier.

3.2 RPA Examples

Automation has already been around before the rise of the RPA platforms. These solutions and tools have been based on pure programmed code without visual representations. These include combinations of scripts and techniques capable of navigating through the visual user interface. Many of these techniques originate from UI automation testing, which are found suitable also for the needs of RPA.

Lately, RPA platforms have included the visual interface to RPA developing, making it easier to use and develop for people without programming background. This has greatly accelerated RPA's fame. In order to understand the differences between tools that require actual programming skills and the platforms based on visual user interface, a simple example is provided with both techniques.

The programmed example is represented with Python which is highly capable in automation with its wide scope of libraries suitable for RPA, thus making it optimal for comparison. It is also used as the main development language at MOST Digital. The visually developed example is provided with UiPath, the market leader which is already discussed earlier in this thesis. The reason these tools are represented in this thesis is that both techniques are valid and can be used for RPA developing, selecting the correct one mostly depends on the needs and preferences of both the customer and the developer.

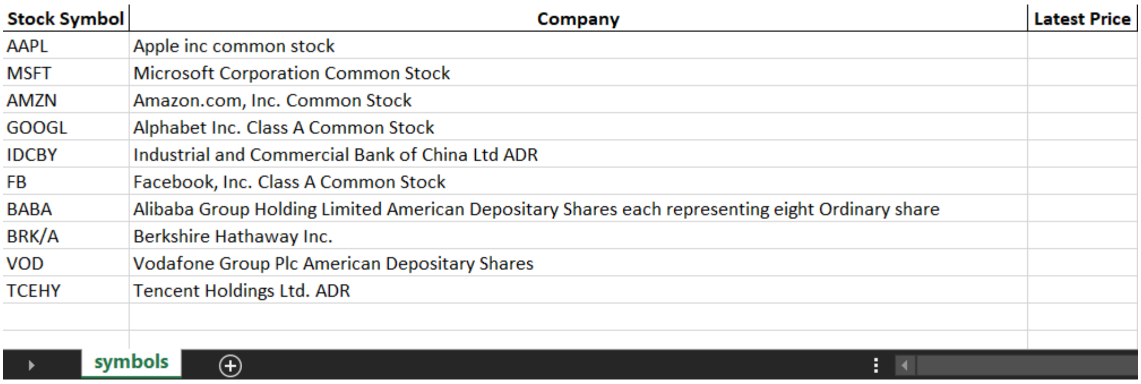
Data is the starting point in most automation projects. In other cases the robot starts the automated task by acquiring the needed data. In this example the robot reads control data from input excel file, which is represented in Figure 3.3 and used with both Python and UiPath. In this case, it is simple and ready to use, but oftentimes cleaning up and fixing the initial data is also part of RPA development. Data may not always be in a coherent state and ready to be utilized as it is.

The example process reads stock symbols from the input excel file, fetches latest stock price from nasdaq.com[1] with Google Chrome[2] and updates the original excel file with the acquired data. Although, the example is rather simple, it aims to demonstrate differences between the selected tools. Also, these steps reflect the basic procedures involved in common RPA solutions. The example's main logic is represented with Python in Listing 3.1 while UiPath's main workflow is displayed in Figure 3.4.

---

[1] https://www.nasdaq.com/, 9.5.2020

[2] https://cloud.google.com/chrome-enterprise, 9.5.2020



Stock Symbol	Company	Latest Price
AAPL	Apple inc common stock	
MSFT	Microsoft Corporation Common Stock	
AMZN	Amazon.com, Inc. Common Stock	
GOOGL	Alphabet Inc. Class A Common Stock	
IDCBY	Industrial and Commercial Bank of China Ltd ADR	
FB	Facebook, Inc. Class A Common Stock	
BABA	Alibaba Group Holding Limited American Depositary Shares each representing eight Ordinary share	
BRK/A	Berkshire Hathaway Inc.	
VOD	Vodafone Group Plc American Depositary Shares	
TCEHY	Tencent Holdings Ltd. ADR	

Figure 3.3: The input excel file for provided RPA examples.

```python
def main():
    # 1 - read stock symbols from excel
    excel_df = read_excel_data()
    # 2 - search latest values from nasdaq.com with Google Chrome
    updated_df = fetch_stock_values(excel_df)
    # 3 - updated stock symbol values in excel
    save_updated_data(updated_df)
```

Listing 3.1: Python main logic.

3.2.1 Reading Data from Excel File

An excel file can be processed multiple ways using different Python libraries. This example uses Pandas, an open source data analysis and manipulation tool [31]. The code, presented in Listing 3.2, reads the excel file's content into a DataFrame object, which is updated in later stages of the process.

In Figure 3.5, reading an excel file in UiPath generates a DataTable and a workbook object, named in this example as WbSymbols. The Read Range activity used inside the Read Excel and Create a DataTable activity is using the workbook object to read the whole of the first sheet in the excel file.

In both cases the code generated, either by coding with Python or by dragging and dropping activities in UiPath, has enabled the user to read the excel file's data into a structured

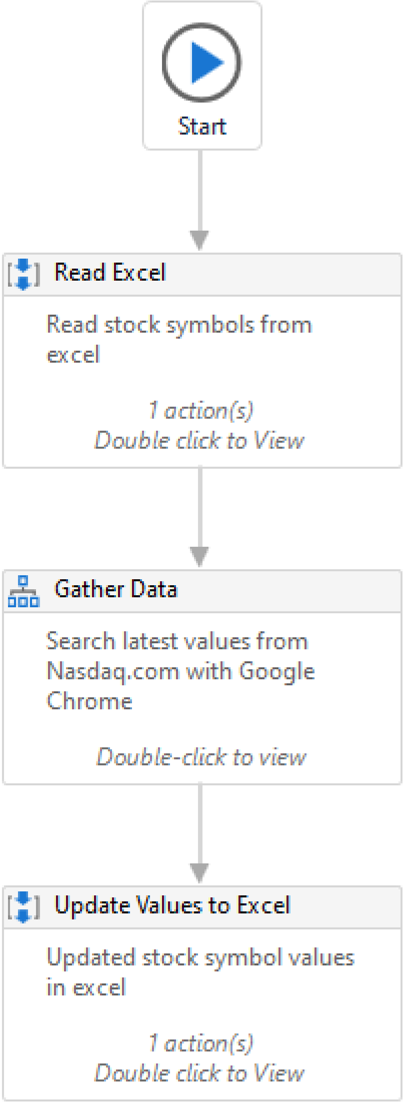
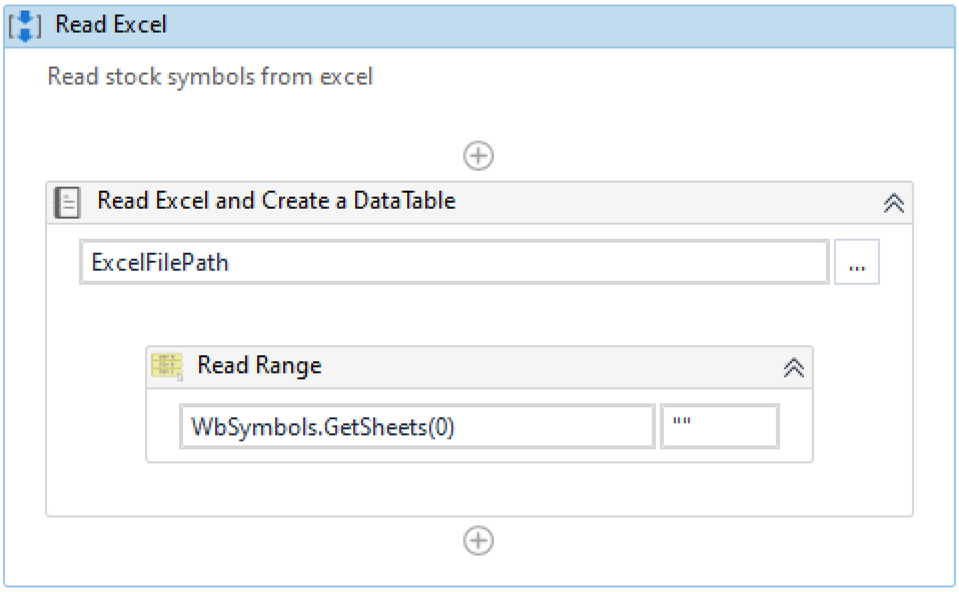


Figure 3.4: The main workflow in UiPath process.



Read stock symbols from excel

ExcelFilePath

WbSymbols.GetSheets(0)

""

Figure 3.5: Reading data from the excel file with UiPath.

object form — In Python's Pandas library it is called a DataFrame, and in UiPath it is called a DataTable.

```python
def read_excel_data():
    EXCEL_VARIABLES = config["EXCEL_VARIABLES"]

    excel_df = pd.read_excel(
            EXCEL_VARIABLES["BLANK_EXCEL_PATH"],
            sheet_name = EXCEL_VARIABLES["SHEET_NAME"],
            encoding = EXCEL_VARIABLES["ENCODING"],
            dtype = object
            )

    print(f"excel_dataframe: \n {excel_df}")

    return excel_df
```

Listing 3.2: Reading data from the excel file with Python.

3.2.2 Fetching Information from the Internet

The next step is to fetch the latest stock price from nasdaq.com. In this example, Python is used with Selenium library[3], originally a framework for testing web applications, but widely used for automation and RPA processes as well [9]. Selenium is able to access Google Chrome via chromedriver.exe, a separate driver that needs to be available in order to get Selenium working.

Python logic in Listing 3.3 shows that querying the stock specific view in nasdaq.com is fairly simple. The correct url address can be constructed from the base url joined with the stock specific symbol at the end. The price can be extracted from the website's HTML structure. The parent element is fetched first and after it has emerged, the price can be read from its inner HTML structure. Finally the fetched price is saved to the symbol's row in the DataFrame object created in the previous step. This procedure is repeated for every stock symbol found in the original input excel file.

The UiPath logic of searching the nasdaq.com website for the correct stock price is illustrated in the Figure 3.6. The first step of the logic is to read the stock symbol that has not yet been updated, as seen in Figure 3.7. In the referenced figure the datatable is looped and a couple of checks are asserted in order to get the needed data.

Next, the logic navigates to the Nasdaq website and retrieves the current stock price, by concatenating the Nasdaq website url and the stock symbol's name, as is used in the Python example. This is presented in Figure 3.8. The Get TStock Price Text activity reads the website's HTML structure and identifies the wanted text element.

---

[3] https://www.selenium.dev/, 9.5.2020

```python
def fetch_stock_values(excel_df):

    SELENIUM_VARIABLES = config["SELENIUM_VARIABLES"]
    EXCEL_VARIABLES = config["EXCEL_VARIABLES"]

    selenium_driver = webdriver.Chrome(
        SELENIUM_VARIABLES["SELENIUM_DRIVER_PATH"]
        )
    selenium_driver.implicitly_wait(20)

    for stock_symbol in excel_df[
        EXCEL_VARIABLES["COLUMNS"]["SYMBOL_HEADER"]
        ]:
        print(f"fetching price for symbol: {stock_symbol}")

        nasdaq_url = "{}{}".format(
            SELENIUM_VARIABLES[’NASDAQ_URL_BASE’],
            stock_symbol
        )
        print(f"google_finance_url: {nasdaq_url}")

        selenium_driver.get(nasdaq_url)

        # getting the specific price element from nasdaq site by its HTML class name
        price_element = selenium_driver.find_element_by_css_selector(
            SELENIUM_VARIABLES["SYMBOL_PRICE_PARENT_ELEMENT_CSS"]
        ).find_element_by_class_name(
            SELENIUM_VARIABLES["SYMBOL_PRICE_ELEMENT_CLASS"]
        )
        # value can be found inside element
        symbol_price = price_element.get_attribute("innerHTML")
        print(f"found symbol_price: {symbol_price}")

        # updating dataframe read from excel
        excel_df.loc[
            (excel_df[
                EXCEL_VARIABLES["COLUMNS"]["SYMBOL_HEADER"]
                    ] == stock_symbol
            ),
            EXCEL_VARIABLES["COLUMNS"]["PRICE"]
            ] = symbol_price

        print(f"updated excel_df: \n{excel_df}")

    print("All stock prices fetched.")
    return excel_df
```

Listing 3.3: Fetching stock symbol data from nasdaq.com with Python.

Lastly, the logic updates the datatable with the gathered data to the stock symbol specific row. The logic is depicted in Figure 3.9. A flow decision is placed at the end of the logic to check if the robot has iterated all of the datatable rows. If not, the robot starts the

”Gather Data” process again until all of the rows are updated, as represented in Figure 3.6.

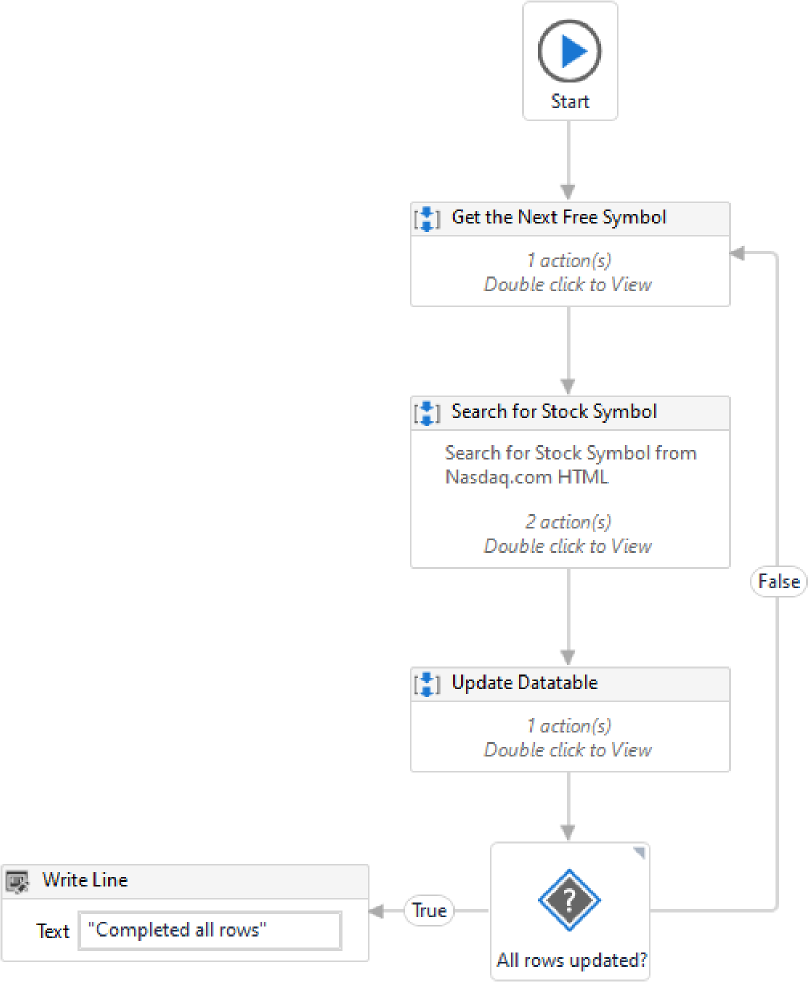
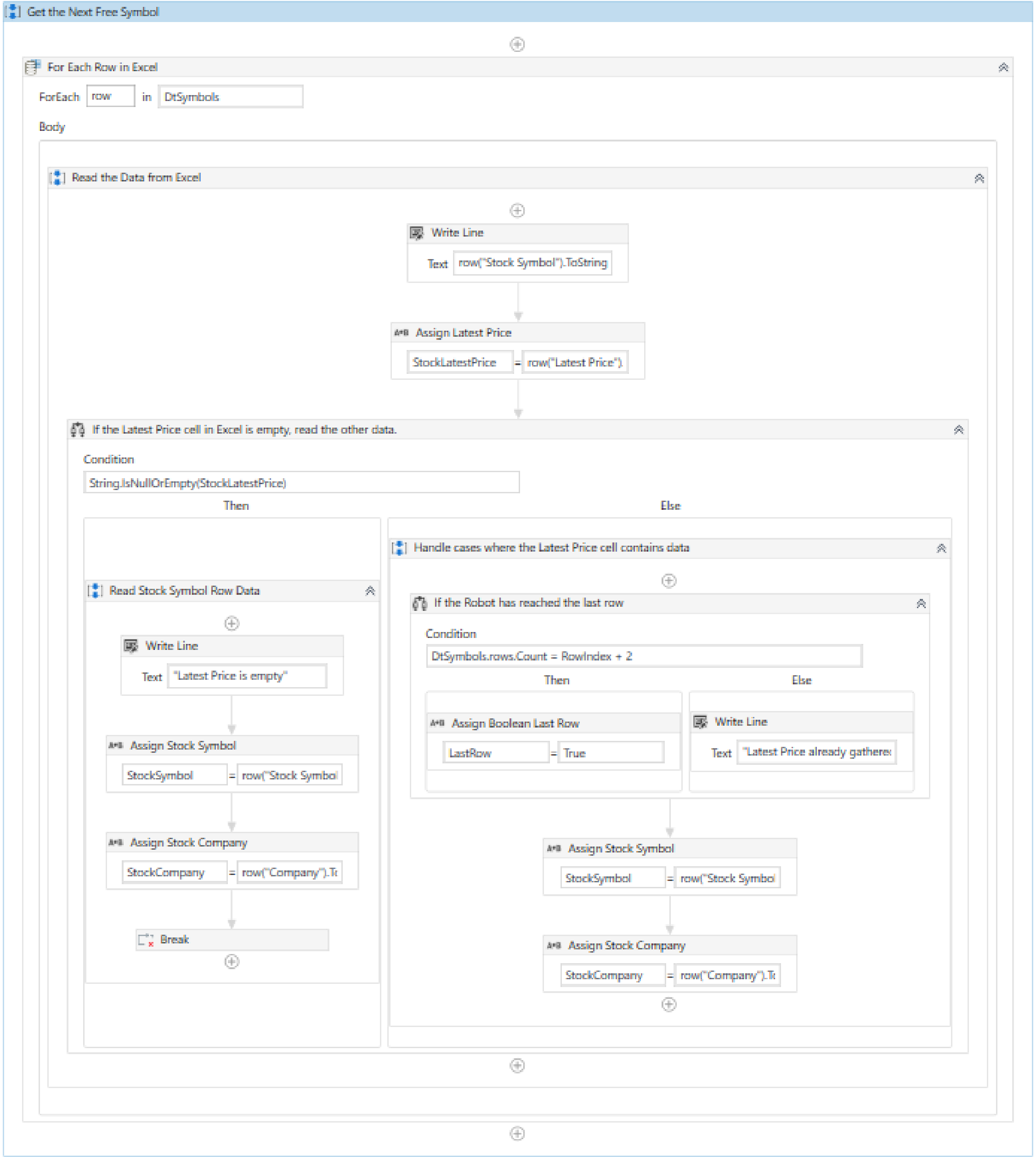


Figure 3.6: Fetching stock symbol data from nasdaq.com with UiPath.

3.2.3 Saving Enriched Data to the Excel File

The final step is to save updated stock prices to the output excel file. Python does this again with Pandas, in Listing 3.4. UiPath writes the updated DataTable to output excel file in Figure 3.10. The resulted output excel file for both examples is represented in Figure 3.11.



ForEach row in DtSymbols

Text row("Stock Symbol").ToString

StockLatestPrice = row("Latest Price")

Condition String.IsNullOrEmpty(StockLatestPrice)

Text "Latest Price is empty"

StockSymbol = row("Stock Symbol

StockCompany = row("Company").T

Condition DtSymbols.rows.Count = RowIndex + 2

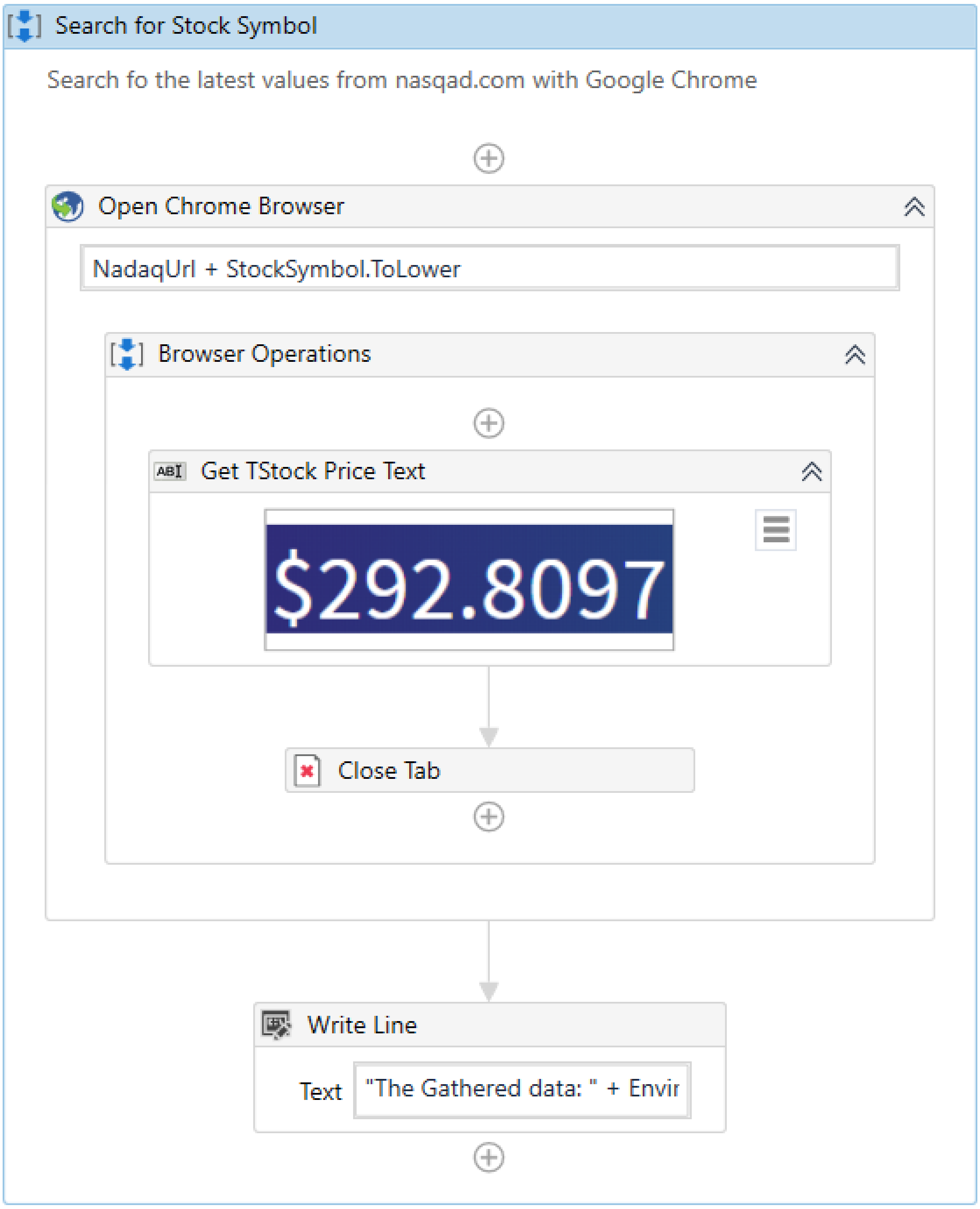
LastRow = True

Text "Latest Price already gathere

StockSymbol = row("Stock Symbol

StockCompany = row("Company").T

Figure 3.7: Looping the DataTable in UiPath.



Search for Stock Symbol

Search fo the latest values from nasqad.com with Google Chrome

Open Chrome Browser

NadaqUrl + StockSymbol.ToLower

Browser Operations

Get TStock Price Text

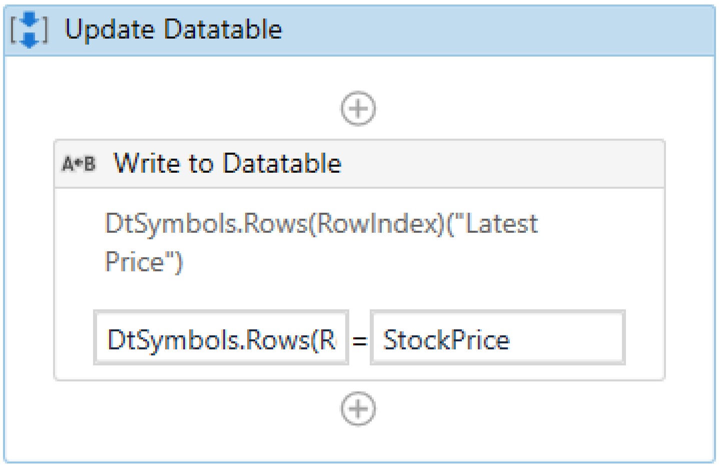
$292.8097

Close Tab

Write Line

Text "The Gathered data: " + Envir

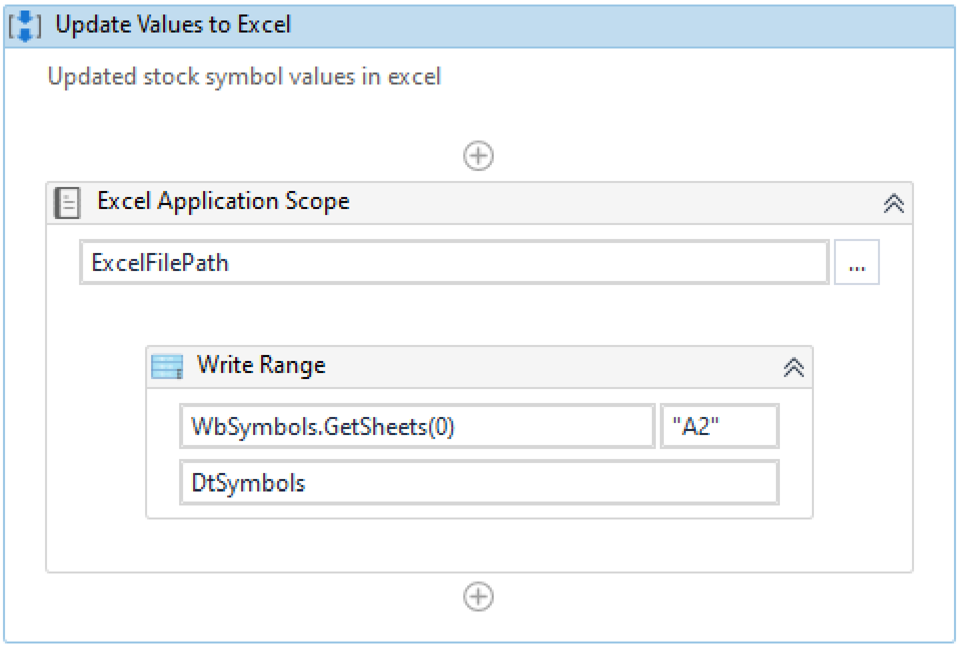
Figure 3.8: Searching for the stock symbol in UiPath.



DtSymbols.Rows(RowIndex)("Latest Price")

DtSymbols.Rows(R = StockPrice

Figure 3.9: Updating the DataTable in UiPath.



Updated stock symbol values in excel

ExcelFilePath

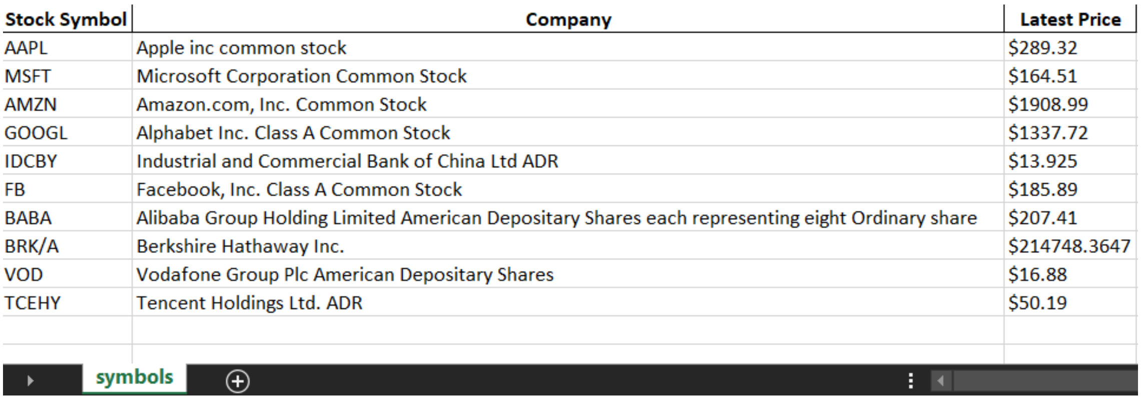
WbSymbols.GetSheets(0) "A2"

DtSymbols

Figure 3.10: Saving enriched data to the excel file with UiPath.

```python
def save_updated_data(excel_df):
    EXCEL_SETTINGS = config["EXCEL_SETTINGS"]

    excel_df.to_excel(
        EXCEL_SETTINGS["RESULT_EXCEL_PATH"],
        sheet_name = EXCEL_SETTINGS["SHEET_NAME"],
        encoding = EXCEL_SETTINGS["ENCODING"],
        index=False
    )
    print("excel saved")
```

Listing 3.4: Saving enriched data to the excel file with Python.



Stock Symbol	Company	Latest Price
AAPL	Apple inc common stock	$289.32
MSFT	Microsoft Corporation Common Stock	$164.51
AMZN	Amazon.com, Inc. Common Stock	$1908.99
GOOGL	Alphabet Inc. Class A Common Stock	$1337.72
IDCBY	Industrial and Commercial Bank of China Ltd ADR	$13.925
FB	Facebook, Inc. Class A Common Stock	$185.89
BABA	Alibaba Group Holding Limited American Depositary Shares each representing eight Ordinary share	$207.41
BRK/A	Berkshire Hathaway Inc.	$214748.3647
VOD	Vodafone Group Plc American Depositary Shares	$16.88
TCEHY	Tencent Holdings Ltd. ADR	$50.19

Figure 3.11: The output excel file for provided RPA examples.

3.2.4 Prerequisites and Settings for the Provided Examples

To be able to use Pandas, Selenium and other libraries with Python, they must be installed in Python environment and imported in the beginning of the script, as presented in Listing 3.5. UiPath requires only the UiPath Studio application as it contains all the necessary dependencies.

```python
import os
import pandas as pd
from selenium import webdriver
import openpyxl
```

Listing 3.5: Imports needed in Python example.

Both techniques can use a separate configuration file. UiPath recommends using a separate excel file, whereas Python can read the configurations from various file types. The provided Python example declares settings in dictionary object as illustrated in Listing 3.6, which could also be read from a separate file.

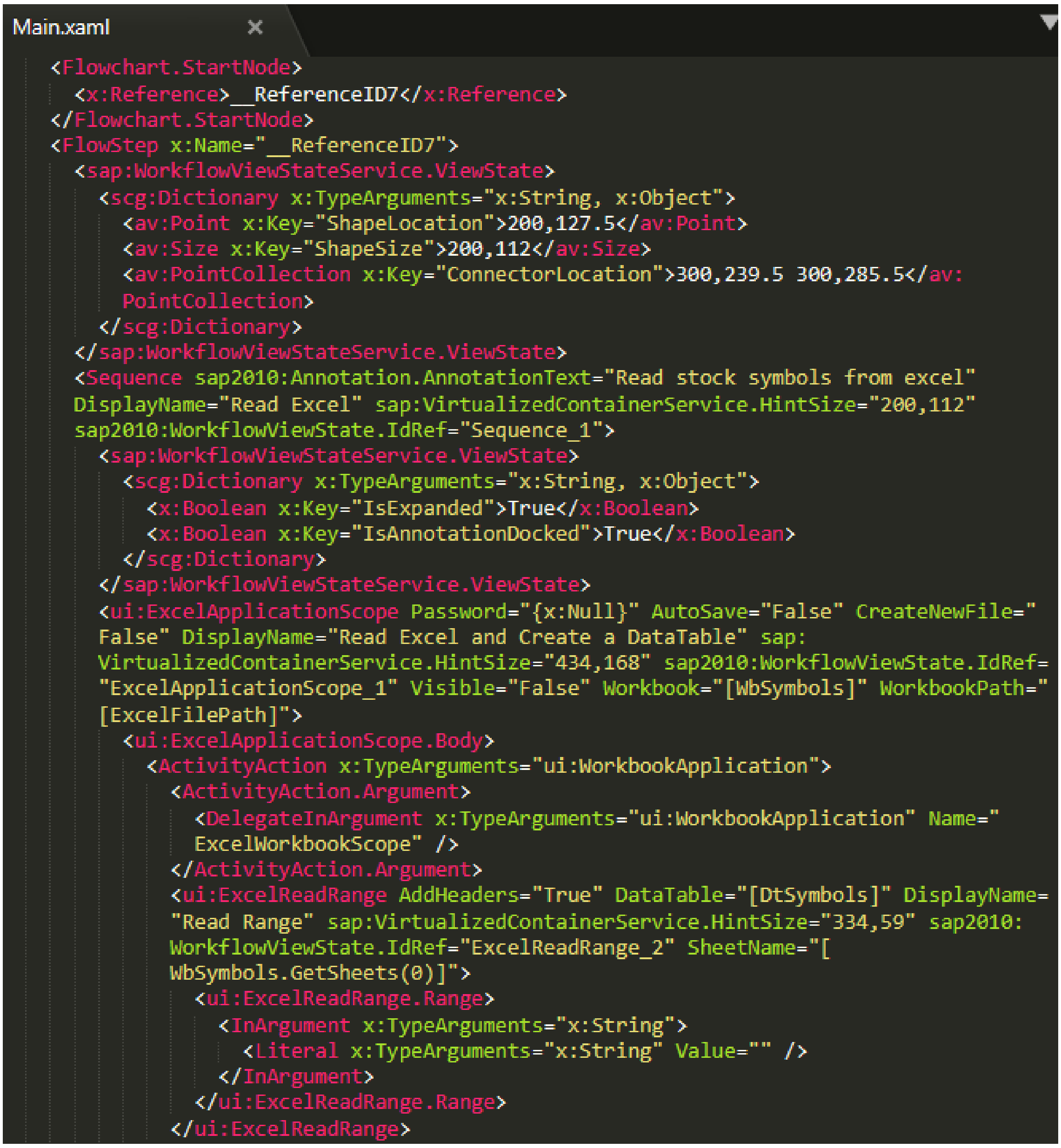
3.2.5 Summary

Both Python and UiPath provide a powerful way to develop automation processes. Choosing the right tool depends on the user's development skills, as UiPath demands less technical knowledge compared to Python. An experienced developer used to programming may find Python more convenient compared to UiPath's drag and drop user interface. Similarly for more business oriented user, UiPath provides faster results without the need to write code.

Programming languages such as Python give the developer more freedom to use custom approaches for the project. This enables more flexibility, but also demands more technical skills and responsibility concerning error handling and security issues. Most of these are already handled with UiPath components, which run in the background. As a closed source application [32], UiPath hides the explicit code logic and links it to the drag and drop user interface with XAML structure, illustrated in Figure 3.12. The structure holds the visual formation instructions for the UiPath Studio and also, function calls with parameters. This way UiPath can provide more ready to use functions. However, it could limit the user's options for wanted, customized functionalities, as the user is unable to access and modify the executable code itself.

```
config = {
    "EXCEL_SETTINGS":{
        "BLANK_EXCEL_PATH":os.path.join(
            "..","blank_stock_symbol_excel.xlsx"
        ),
        "RESULT_EXCEL_PATH":os.path.join(
            "..","stock_symbol_excel_output_python.xlsx"
        ),
        "SHEET_NAME":"symbols",
        "ENCODING":"uff-8",
        "COLUMNS" : {
            "SYMBOL_HEADER" : "Stock Symbol",
            "DESCRIPTION" : "Company",
            "PRICE" : "Latest Price"
        }
    },
    "SELENIUM_SETTINGS":{
        "SELENIUM_DRIVER_PATH":"chromedriver.exe",
        "NASDAQ_URL_BASE":"https://www.nasdaq.com/market-activity/stocks/",
        "SYMBOL_PRICE_ELEMENT_CLASS":"symbol-page-header__pricing-price"
    }
}
```

Listing 3.6: Configuration needed in Python example.



```
<Flowchart.StartNode>
  <x:Reference>__ReferenceID7</x:Reference>
</Flowchart.StartNode>
<FlowStep x:Name="__ReferenceID7">
  <sap:WorkflowViewStateService.ViewState>
    <scg:Dictionary x:TypeArguments="x:String, x:Object">
      <av:Point x:Key="ShapeLocation">200,127.5</av:Point>
      <av:Size x:Key="ShapeSize">200,112</av:Size>
      <av:PointCollection x:Key="ConnectorLocation">300,239.5 300,285.5</av:
      PointCollection>
    </scg:Dictionary>
  </sap:WorkflowViewStateService.ViewState>
  <Sequence sap2010:Annotation.AnnotationText="Read stock symbols from excel"
  DisplayName="Read Excel" sap:VirtualizedContainerService.HintSize="200,112"
  sap2010:WorkflowViewState.IdRef="Sequence_1">
    <sap:WorkflowViewStateService.ViewState>
      <scg:Dictionary x:TypeArguments="x:String, x:Object">
        <x:Boolean x:Key="IsExpanded">True</x:Boolean>
        <x:Boolean x:Key="IsAnnotationDocked">True</x:Boolean>
      </scg:Dictionary>
    </sap:WorkflowViewStateService.ViewState>
    <ui:ExcelApplicationScope Password="{x:Null}" AutoSave="False" CreateNewFile="
    False" DisplayName="Read Excel and Create a DataTable" sap:
    VirtualizedContainerService.HintSize="434,168" sap2010:WorkflowViewState.IdRef=
    "ExcelApplicationScope_1" Visible="False" Workbook="[WbSymbols]" WorkbookPath="
    [ExcelFilePath]">
      <ui:ExcelApplicationScope.Body>
        <ActivityAction x:TypeArguments="ui:WorkbookApplication">
          <ActivityAction.Argument>
            <DelegateInArgument x:TypeArguments="ui:WorkbookApplication" Name="
            ExcelWorkbookScope" />
          </ActivityAction.Argument>
          <ui:ExcelReadRange AddHeaders="True" DataTable="[DtSymbols]" DisplayName=
          "Read Range" sap:VirtualizedContainerService.HintSize="334,59" sap2010:
          WorkflowViewState.IdRef="ExcelReadRange_2" SheetName="[
          WbSymbols.GetSheets(0)]">
            <ui:ExcelReadRange.Range>
              <InArgument x:TypeArguments="x:String">
                <Literal x:TypeArguments="x:String" Value="" />
              </InArgument>
            </ui:ExcelReadRange.Range>
          </ui:ExcelReadRange>
```

Figure 3.12: UiPath example's main XAML file, UiPath Studio Community Edition 2019.

4 Improving Development at MOST Digital

The main motive for this thesis is to design and create a tool that produces more value to MOST Digital, making definition and documentation of RPA projects easier, which should lead to improved communication. These main aspects should lay foundation to ease development, testing and maintaining of RPA projects. At best, the use of the tool would standardize good practices and structures inside the company, which could streamline and accelerate the development phase.

This chapter introduces the life-cycle of RPA projects carried out at MOST Digital. Later, the usual Challenges of RPA development are identified from both MOST Digital's life-cycle and common aspects of RPA development discussed in Chapter 3. The last two sections use these identified Challenges to derive Requirements and to form Research Questions, from which the design for the wanted tool is created and introduced in Chapter 5.

4.1 Project Life Cycle

General RPA project at MOST Digital has four different development phases containing several sub-phases that make up the whole process. As described in Table 4.1 the main phases include definition, development, testing and maintenance. The table is further

		Development Processes			
		Definition	Development	Testing	Maintenance
Actors	Customer	x	(x)	(x)	
	Definer	x	(x)	(x)	
	Developer	(x)	x	x	(x)
	Maintainer			(x)	x

Table 4.1: Development processes and the main actors, marked by x, in RPA development at MOST Digital. Sometimes a process requires an additional actor, marked by (x).

separated between four main actors consisting of one or more individuals, which are the customer, definer, developer and maintainer. The quality of communication and documentation between the actors throughout the development process is paramount in order to achieve the best possible outcome.

At MOST Digital everything starts with the customer. The definition phase consists of meetings between the management of the customer's side and the project managers on MOST Digital's side. This phase involves understanding the customer's process and their needs. Once a process is found to be a good match for automation, the definition of the task begins. In some occasions developers are also asked to provide technical input during the definition process, as seen in Table 4.1.

Once the process is defined and documented, a Process Design Document (PDD) is produced, which should contain all the necessary information about the process in order to start the technical development. Before the RPA developer can begin automating any process, the needed development environments must be set up and additional software installations and access rights must be granted.

Testing and developing RPA solutions go hand in hand. Oftentimes, developing RPA solutions demand continuous process logic testing, as customer's systems may require data to be inputted in order to progress in the process' next step. More comprehensive testing is done at the later stages of development, where the process is validated with the customer. During development and testing, unexpected situations may emerge that were not mentioned in the initial PDD. For these situations the developer usually asks help from the project manager or directly from the customer, in order to solve the task at hand swiftly, as seen in development and testing columns in Table 4.1. The identified changes should be documented and the PDD updated. The solution is validated with the customer by arranging demonstration meetings, where the solution is inspected step by step.

Documentation of the project and the changes that took place during the project are essential for the customer, other developers and for the maintenance team. First of all, the technical documentation of the actual solution must be up to date. Second, a Solution Design Document (SDD) should be created by either updating the original PDD or creating a new and a final version from it, by adding all the changes made to the initial document. Finally, as sharing knowledge helps everyone in the long run, these documents must be shared internally with the support team and with other developers. Additionally, SDDs should be available to the customer.

The last phase of the life-cycle of a general RPA project is ensuring that the solution works consistently on its own, and for this reason MOST Digital offers its customers an alerted support for a fixed time period, assuring that the functional continuity is delivered to the customer. Support is done by the maintenance team, which monitors the solution's status and investigates possible errors, also working with the developers to ensure that the encountered errors are fixed properly. The maintenance team aims to solve problems first by themselves, with the help of documented error scenarios. If they are unable to solve the problem, the error is raised to the developer. This means that the maintenance team is involved with testing and the developer may also take part of the maintenance process as seen in Table 4.1. When the solution's functionality is confirmed, maintaining is continued on a default level. The default level offers customers a continuous support based on the Service-Level Agreements (SLA) [33].

4.2 Identifying Challenges

In order to derive Requirements for the tool to be developed during this thesis, potential Challenges in RPA development are identified from project life cycle discussed in previous section 4.1 and aspects introduced in Chapter 3. Most of the Challenges (C1–C25) presented in Table 4.2 circle around communication and documentation, as they lay foundations to all other parts of RPA project's life-cycle.

From the very beginning of the definition phase, the quality of communication is essential. Most common issues relate to lack of common language, when vocabulary is either too technical or high level for all participants to understand (C4), resulting in vague instructions (C5). Initial challenges in definition phase are associated with missing detailed instructions (C1) and tacit knowledge (C2, C13). More severe consequences may arise if the defined process is not optimized for automation (C3).

Unrealistic customer expectations (C7) are commonly the result of insufficient or inadequate communication (C4), and as the project is usually quick paced and intensive, the communication must flow effortlessly throughout the development, from the developer to the project manager or to the customer. One major obstacle for communication quality is possible availability issues between stakeholders (C9). Without knowing the progression challenges (C6), the customer might think that the initial process definition holds place and development is on schedule (C8). This might be true in some projects, where the process definition is correct throughout the process, but that is rarely the case.

The RPA process documentation should be comprehensive, detailed, precise and formal in order to provide the best known documentation of the process to be automated. Poorly defined PDD, which lacks detailed instructions (C10) may cause inefficient decision making for the developers and raises plethora of questions that end up wasting everyone's time. And, if possible, the PDD should also include known errors and the possible ways of handling them (C24).

Poor technical documentation is another issue RPA projects face (C11), as developers might not have time allocated (C16), or they might not have set the technical documentation to a high priority, which might be caused of overlapping projects. As projects are short and agile, the developers must take care of the technical documentation during and after development. It is vital for the maintenance team, as well as for the other developers to avoid misunderstandings (C18, C19). Additionally, the documents should be unified in both format and file location to ensure the ease of use and availability (C14, C15).

Undefined exceptions are what developers encounter during most RPA development projects, as it is hard to tackle every situation beforehand, thus communication is an integral part of the development. This situation leads to updating the process definition during development based on decisions made. Changes should be also documented in PDD, as it is essential to keep the process documentation up to date (C12). Undocumented changes greatly complicate both further development and maintenance, as well as result in an incomplete SDD (C17, C25). Furthermore, without applying a company wide coding and testing standards, more maintenance is expected (C20).

Immature IT infrastructure is one of the challenges RPA development faces (C21), although the robot does not usually need large systems to work. Missing test environment and data cause additional work to the developer, as they are not able to freely test the system while connected to customer's production systems (C22, C23).

4.3 Deriving Requirements for Improvement Proposal

The identified Challenges listed in Table 4.2 pave the path for the Requirements needed for developing the tool during this thesis. Challenges are spread over the different phases of RPA development and therefore are not solvable with a single solution. However, root and sub Challenges, which could be eased by a single tool, can be identified. The root Challenges are the tip of an iceberg which, when solved, could potentially ease the sub Challenges as well. A Challenge is considered to be a root Challenge when it can not be directly solved with an another Challenge. The derived Requirements (R1–R3) with their root and sub Challenges are listed in Table 4.3.

Seven root Challenges were identified from the definition, documentation and testing aspects while the Challenges in communication, development and maintenance are all considered to be sub Challenges. The first two root Challenges are missing detailed instructions and tacit knowledge during definition in C1 and C2. When the definition is done in high detail, it prevents unoptimized processes in C3, misunderstandings due to communication and documentation in C5, C18 and C19, lack of detailed and tacit knowledge in C10 and C13, poorly documented business logic exceptions in C24, as well as unclear customer expectations due to vague PDD in C7. Furthermore, solving C1 and C2 can prevent immature IT architecture in C21 because the architect can plan a better infrastructure when the process needs are documented thoroughly.

Aspect	**Challenge**	**Description**
Definition	C1	Missing detailed instructions during definition
	C2	Missing tacit knowledge during definition
	C3	Failing to optimize the process during definition
Communication	C4	Using too technical or high vocabulary in communication
	C5	Instructions are too vague, leaving space for misunderstandings
	C6	Stakeholders not up to date of the current level of progress
	C7	Unclear customer expectations
	C8	Unclear timetable
	C9	Availability issues between the stakeholders
Documentation	C10	Lacking detailed instructions in PDD
	C11	Lacking detailed technical documentation
	C12	Decisions made during meetings are not properly documented
	C13	Tacit knowledge not documented
	C14	Documentation not in unified location
	C15	Documentation not in unified format
	C16	No time to update documentation
Development	C17	Outdated PDD
	C18	Misunderstandings due to communication
	C19	Misunderstandings due to documentation
	C20	Divided development conventions inside the company
	C21	Immature IT architecture
Testing	C22	Lack of test material
	C23	Lack of test environments
	C24	Poorly documented exception situations with automated systems
Maintenance	C25	Outdated SDD

Table 4.2: Identified potential Challenges in RPA development grouped by development aspects.

C14, C15 and C16 were identified as root Challenges in the documentation aspect. Solving the issue of decentralized project documentation in C14 could assure that all the updated information is in the correct location. This directly helps with the stakeholders not being up to date with the progress in C6, as well as unclear customer expectations and timetables in C7 and C8. Also, when both C14 and divided documentation format in C15 are fixed, the misunderstandings due to communication and documentation in C18 and C19 are eased as well. Moreover, no time to update documentation in C16 is directly linked to the lack of detailed technical documentation in C11, and outdated PDD and SDD in C17 and C25.

Lack of test material and test environments in C22 and C23 are the last root Challenges, as they are not solvable by other challenges but should be discussed and planned at the early stages of the definition phase and are more related to the customer's ability to provide them. The rest of the Challenges C4, C9, C12 and C20 cannot be solved with a single tool as they are more linked to the company's or the customer's internal processes and best practices.

To ease the identified root Challenges, three Requirements were derived from them. The first Requirement (R1) is to provide guiding questions during the definition phase, in order to answer the most common pitfalls of RPA process' data gathering and testing needs. R1 aims to ensure that the process instructions and tacit knowledge are properly documented, solving C1 and C2, as well as discussing the need for test material and test environments, thus providing ease to C22 and C23.

The second Requirement (R2) states that accessible, unified and up to date documentation should be provided, solving C14. The third Requirement (R3) answers to the rest of the root Challenges with automatic documentation, which should ease C15 and diminish the

Requirement	Description	Root Challenges	Sub Challenges
R1	Providing guiding questions during the definition phase	C1, C2, C22, C23	C3, C5, C7, C10, C13, C18, C19, C21, C24
R2	Providing accessible, unified and up to date documentation	C14	C6, C7, C8 C18, C19
R3	Providing automatic documentation	C15, C16	C11, C17, C18, C19, C25

Table 4.3: Requirements for the tool. Derived Requirements aim to solve identified root Challenges, which are linked to the sub Challenges.

issue of insufficient time resources in C16.

4.4 Research Questions

To validate the value provided by the tool, the effects must be assessed against the Challenges listed in Table 4.2 after the Requirements in Table 4.3 are met. As previous sections have addressed thus far, communication and documentation are the key factors in the RPA life-cycle. The Figure 4.1 illustrates that every process and actor in RPA development listed in the Table 4.1, are leaning on the communication and the documentation. Documentation can be seen as a long term communication, an utility ensuring that tacit knowledge and made decisions would not be forgotten.

The validation is done through three main Research Questions (Q1–Q3) listed in the Table 4.4. The Research Questions are divided between the company's internal actors of the RPA process, containing the definer, developer and maintainer. The tool should provide value for all of them and their viewpoints. The first Research Question (Q1) focuses

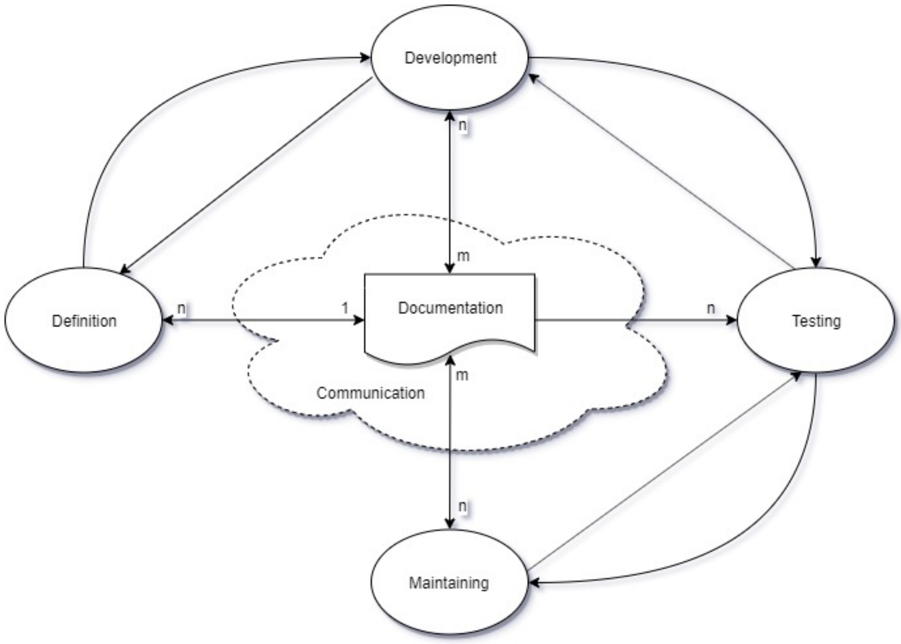


Figure 4.1: Intertwined aspects in RPA development life cycle [3]. All the processes circle around communication and documentation.

on the definition and maintenance phases, where PDD is initially defined or the finalized SDD is inspected. The second Research Question (Q2) investigates the value provided in a more technical level, when developing and testing the RPA process. Finally, the third question (Q3) addresses the overall communication and documentation between project actors.

Research Question	**Description**
Q1	Does the tool ease the definition, project kick-off and maintenance of RPA projects?
Q2	Does the tool ease the development and testing of RPA projects?
Q3	Does the tool ease communication and documentation between stakeholders in RPA projects?

Table 4.4: The Research Questions to be investigated via the tool developed during this thesis.

5 Improvement Proposal and Work Distribution

5.1 PDD-SDD Tool

In order to improve the RPA development aspects, and to address the Requirements listed in Table 4.3, a tool was designed, to automate the documentation flow for the whole project life-cycle, from process definition to maintenance. The tool would form the initial Process Design Document (PDD) and through the project advancement it would automatically be updated to reflect the code logic changes, finally representing the Solution Design Document (SDD).

PDD is first generated in Visual User Interface (VUI) and then processed to the initial code file for the developer. Automated PDD to SDD process is considered to minimize the human effort to maintain process level documentation which in turn supports communication. This should lower confusion and misunderstanding caused by outdated documentation, and address R3. In order to meet R1, guiding questions are added in the VUI's definition phase. After both R1 and R3 are accomplished, the tool needs to be validated and published inside MOST Digital, thus satisfying R2.

The initial design for the tool consists of five main steps (Step 1–Step 5), visualized in

Figure 5.1. In this thesis a standalone version of the tool is developed, ignoring cloud infrastructure, as the tool can be customized for the wanted deploying environment. The user authentication is also ignored to be potentially implemented during further development.

The initial PDD is created by the customer and the project manager at the very beginning of RPA development process. A high level process workflow is crafted in VUI, where parameters and additional information can be inserted and saved (Step 1). The base for code logic is generated automatically, after PDD is ready and submitted in VUI. Additional cloud infrastructure logic would also take place here, if implemented (Step 2).

The development begins after the definition is ready and the initial code base is generated (Step 3). The PDD-SDD Tool reads the code base and updates VUI during the development process, reflecting the latest changes (Step 4). Updating could be done by triggering it manually, or it could be bound to version control system's actions.

The VUI updates itself during the development phase, eventually transforming from the initial PDD towards a continuously more accurate SDD (Step 5). The customer and the project manager cannot make changes to the VUI after the initial definition phase. After that, the changes should be discussed directly with the developer. This feature prevents possibly problematic code logic changes which are not done by the developer.

The VUI should have different levels of views, depending on the user. The customer and the project manager should view the process from a high level, to confirm that the business logic is correct. Initial design dictates this level including three types of views, represented in Figure 5.2, which are divided depending on their information content and amount of detail. The VUI views are not presented all at once, rather they are accessed

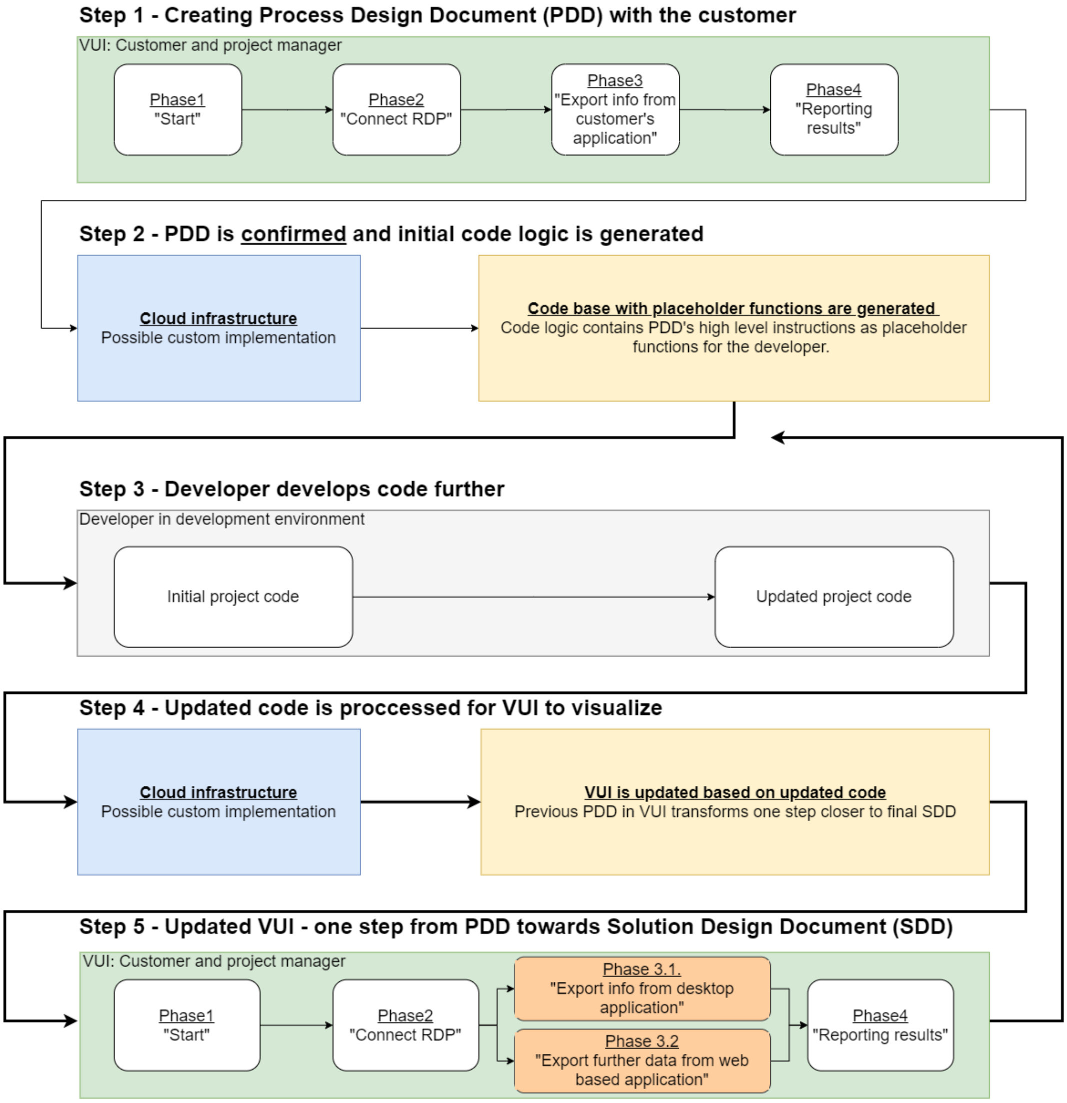

Step 1 - Creating Process Design Document (PDD) with the customer
VUI: Customer and project manager
Phase1 "Start"
Phase2 "Connect RDP"
Phase3 "Export info from customer's application"
Phase4 "Reporting results"
Step 2 - PDD is confirmed and initial code logic is generated
Cloud infrastructure
Possible custom implementation
Code base with placeholder functions are generated
Code logic contains PDD's high level instructions as placeholder functions for the developer.
Step 3 - Developer develops code further
Developer in development environment
Initial project code
Updated project code
Step 4 - Updated code is proccessed for VUI to visualize
Cloud infrastructure
Possible custom implementation
VUI is updated based on updated code
Previous PDD in VUI transforms one step closer to final SDD
Step 5 - Updated VUI - one step from PDD towards Solution Design Document (SDD)
VUI: Customer and project manager
Phase1 "Start"
Phase2 "Connect RDP"
Phase 3.1. "Export info from desktop application"
Phase 3.2 "Export further data from web based application"
Phase4 "Reporting results"

Figure 5.1: Initial design of PDD-SDD Tool.

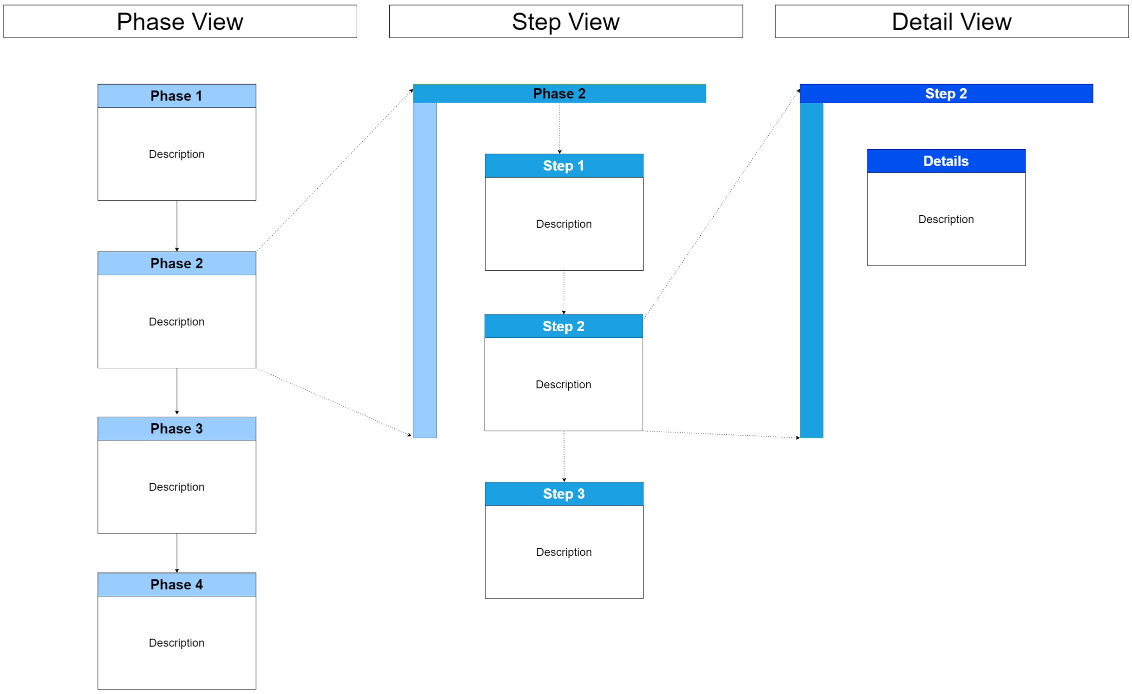


Figure 5.2: Initial rough design of VUI views. The VUI views are not presented all at once, rather they are accessed one level at a time from left to right.

one level at a time from left to right. According to the authors' experiences, three types of views are suitable for most of the RPA projects done at MOST Digital. The Phase level defines the main parts of the process, which are further instructed by the Step and Detail levels. Additionally, fixed amount of VUI views should guide the definer to create PDDs which follow unified structure and are easy to comprehend by other stakeholders.

The Phase level is the first view aiming to define the big picture of the process. The process' high level Phases are separated to be further described in Step and Detail levels. Therefore, the Phase level should not contain more than headers and overall descriptions of the Phase elements. The Step level divides every Phase element into more detailed Steps. This level is meant to behave in a similar way as the Phase level, but in more detailed manner, focusing only on the defined Steps of one specific Phase. The accurate details and descriptions are added into the final view, the Detail level. The Detail level should contain explicit instructions in order to automate the described Step. Instructions may include text and pictures of the automated systems. Additionally, the Detailed view should provide known business errors of the Step. For every element in the Phase, the Step and the Detail levels a placeholder function is generated into the code file, to be later modified by the developer.

Additionally, developers, testers and maintainers should be able to view the process in more detailed manner. For this purpose, a technical view should be implemented, to present the actual code logic and its documentation. The details of the technical view were left to be discussed in more detail in the interviews.

5.2 Solution Compared with Other Mainstream RPA Platforms

Unlike the mainstream RPA platforms such as UiPath, the tool developed during this thesis does not aim to generate finished software. The main motive is to create a tool that unifies communication processes and provides up to date documentation for all stakeholders involved in the development processes. This is considered to result in more unified and straight-forward development practices, creating more value for both the RPA provider and the customer.

The developer has more freedom to use whatever techniques and tools they want, as the PDD-SDD Tool does not create the actual logic. This enables more flexibility and better-suited solutions to different RPA tasks under development. The developer has also the ability to change the project structure, if needed. In these situations, the change should also be validated by the customer and the project manager via updated PDD.

The tool allows custom solutions in the cloud infrastructure (Step 2, Step 4) allowing it to be integrated into the desired infrastructure as a part of other company-specific technical solutions. It should be possible to integrate PDD-SDD Tool into different environments with both RPA and more traditional development projects.

5.3 Work Distribution and Initial Roadmap

As this project is divided into two separate theses developing and researching the provided value of the created tool, the research questions listed in Table 4.4 are divided between Ahmed Abdulghani and Sampsa Vuorela. Abdulghani focuses on Q1, covering the definition and maintenance phases, while Vuorela concentrates on Q2 which includes development and testing. Communication and documentation intertwine with all the other phases, therefore Q3 is investigated in both theses.

The PDD-SDD Tool's development was conducted from Fall of 2019 to Spring of 2020 by both Abdulghani and Vuorela. Two groups of interviewees were gathered from MOST Digital, to provide input for the design and the progress of the tool. With the gathered intel, the tool's design and development was adjusted accordingly. Project's initial roadmap consists of four main development iterations which are illustrated in Figure 5.3.

The Chapters 4 and 5 are part of the first iteration in the roadmap, which were conducted in the beginning of fall 2019. These chapters identified potential Challenges in RPA development, derived Requirements for the PDD-SDD Tool and introduced its initial design. The rest of the iterations in the roadmap are discussed in more detail and validated through interviews in both theses from their separate viewpoints.

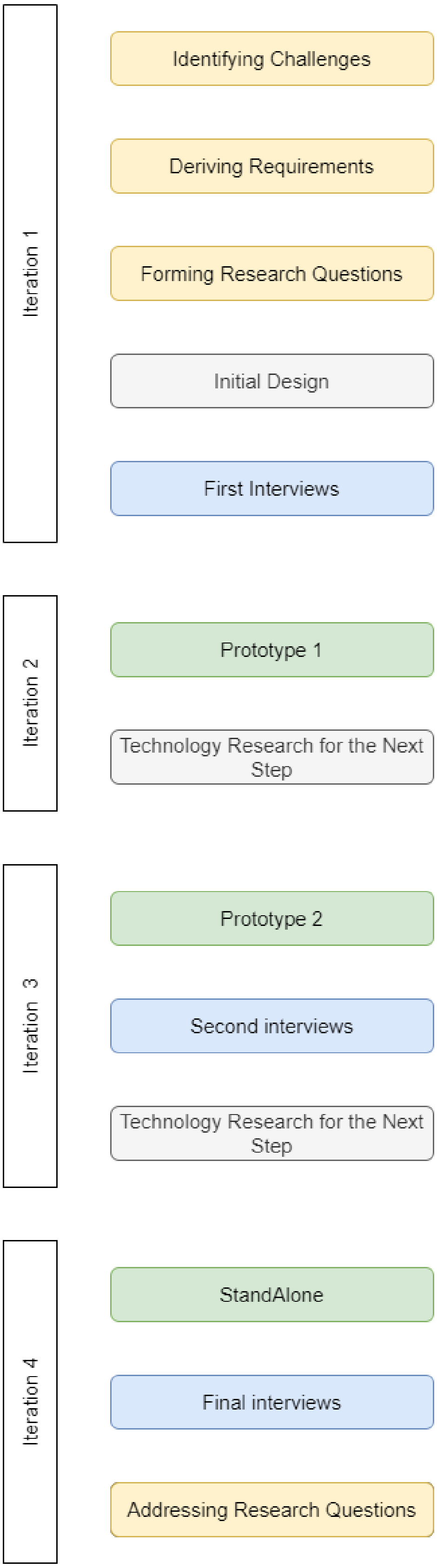


Figure 5.3: Initial roadmap for the PDD-SDD Tool's development.

6 Research Questions Investigated in this Thesis

Research Questions (Q1–Q3) were listed in Table 4.3 in Chapter 4. The Research Questions investigated in this thesis focus on development and testing in Q2 as well as communication and documentation in Q3. Q2 is investigated solely in this thesis, while Q3 is discussed also in Ahmed Abdulghani's thesis, from a viewpoint of definition and maintenance.

The Research Questions can be addressed if the Challenges (C1–C25) and the Requirements (R1–R3) listed in Table 4.3 are met via PDD-SDD Tool introduced in Chapter 5. R1 and R3 can be solved completely during this thesis, while R2 can be truly accomplished only after the tool has been officially published and used in MOST Digital.

This chapter provides an updated version of the initial roadmap, which was depicted in Figure 5.3. The updated roadmap illustrates the PDD-SDD Tool's development from the viewpoint of this thesis. The conducted interviews, research methods, common terminology as well as the future goals for the tool are addressed. Additionally, the interviewees are briefly introduced in the last section of this Chapter.

6.1 Development and Research Methodology

This thesis discusses the development process of the StandAlone version of the PDD-SDD Tool from a developer's viewpoint. The development is divided between four iterations, which were already depicted in Figure 5.3. This section provides an updated version of the roadmap, which introduces the main aspects of the both development and research methods, as well as defines the common terminology used in this thesis. The updated roadmap is illustrated in Figure 6.1.

Two prototypes and the StandAlone version of the PDD-SDD Tool were developed during this thesis. The StandAlone is the first working version of the tool, which is to be potentially developed further after this thesis. The development was quided by the Challenges and the Requirements that were identified from the Chapters 2–5 and updated based on the results of the three interviews. The interviews were conducted to gather intel from the interviewees' experiences in order to potentially update the Challenges and the Requirements if needed, as well as to adjust the design and the development of the tool to meet the actual needs of its users.

The first development iteration depicted in the updated roadmap consists of three parts. The first part was discussed in Chapters 2–5 that introduced the general aspects of RPA development. The chapters also declared the Challenges in Table 4.2, the Requirements in Table 4.3, the Research Questions in Table 4.4 and the PDD-SDD Tool's initial design in Figure 5.1. From this point forward, these Challenges and Requirements are discussed as the Core Challenges and the Core Requirements, as they were identified from the general RPA development aspects, thus providing the core to the initial design of the PDD-SDD Tool.

The first interviews are described in Chapter 7. The general aspects of the RPA devel-

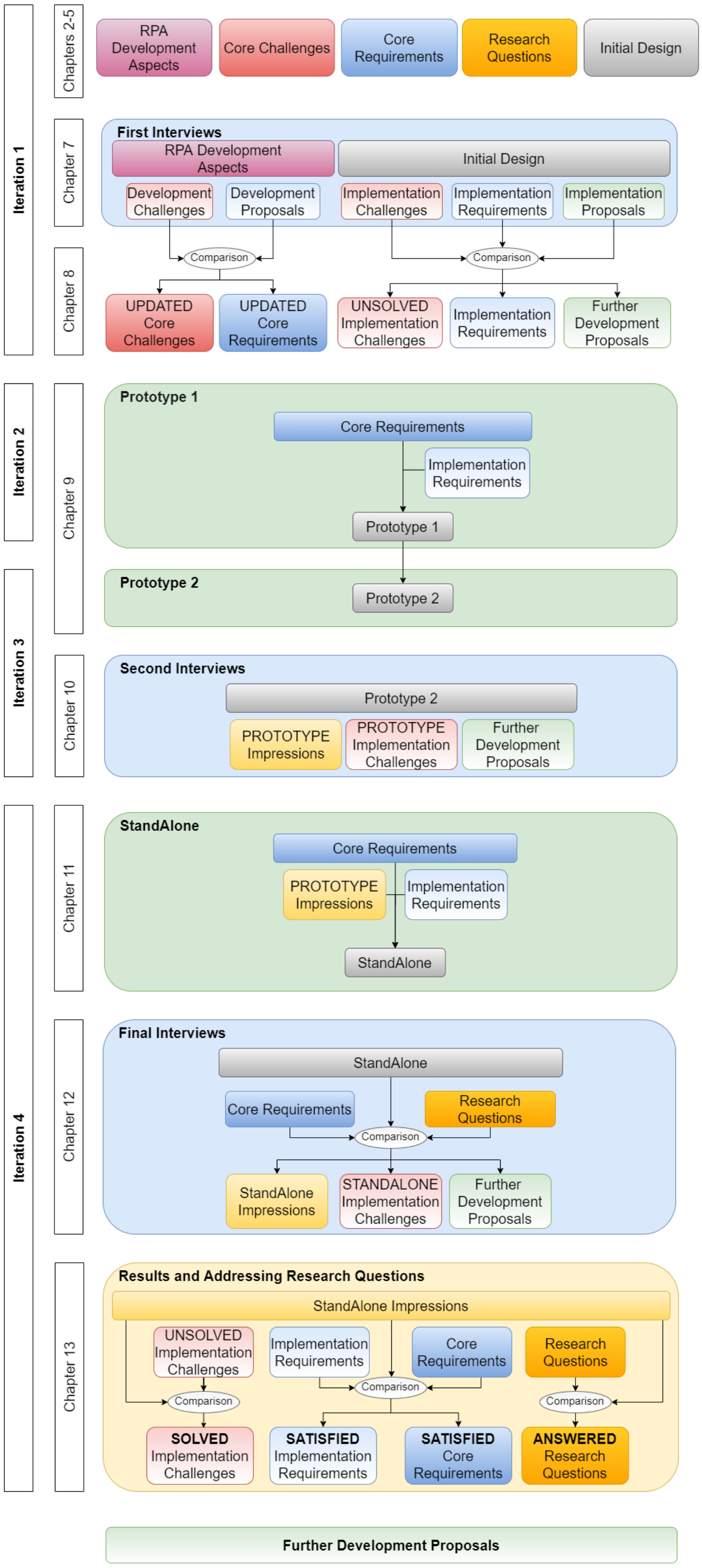


Figure 6.1: Updated version of the initial roadmap in Figure 5.3 illustrating the development and research methods, and common terminology used in this thesis.

opment were discussed with the interviewees, to gather their thoughts of the potential development Challenges related to them. They were also asked to think about possible development Proposals to ease the identified Challenges. The Core Challenges and the Core Requirements were not presented to the interviewees, in order to get unbiased results. The second part of the first interviews presented the initial design of the PDD-SDD Tool, to discuss if it would ease the identified Challenges. The discussions resulted with Implementation Challenges, Implementation Requirements and Implementation Proposals. These are solely related to the provided design of the tool, and are not to be mixed with the Core Challenges and the Core Requirements. The Implementation Requirements are considered to be features that are essential for achieving the desired results with the PDD-SDD Tool, whereas the Implementation Proposals would further enhance its value with additional features.

The first development iteration is completed in Chapter 8, which discusses the results from the first interviews. The development Challenges and the development Requirements identified from the first part of the interviews are compared with each other, as well as with the Core Challenges and the Core Requirements. The comparison aims to identify possible new Challenges or Requirements that were not identified from the Chapters 2–5. The initial design of the PDD-SDD Tool is ajdusted based on the findings and the updated lists of the Core Challenges and the Core Requirements are created.

The second part of the Chapter 8 focuses on the Implementation Challenges, the Implementation Requirements and the Implementation Proposals, gathered from the last part of the first interview, which addressed the initial design of the PDD-SDD Tool. The results are compared with each other in order to match the Implementation Requirements and the Implementation Proposals to the Implementation Challenges. An Implementation Challenge is considered to be solved, if it can be fixed with an Implementation Requirement or an Implementation Proposal. After the comparison, lists of unsolved Implementation Challenges, Implementation Requirements and Further Development Proposals are created.

The next two development iterations focused on the prototype development and are discussed in Chapters 9 and 10. Chapter 9 describes the actual development process and its results, covering the second development iteration. The development is guided with the Core Requirements and the Implementation Requirements from the first development iteration. The prototypes aim to fulfill the core functionalities of the PDD-SDD tool's design.

The third development iteration includes the second prototype development discussed in Chapter 9 and the second interviews in Chapter 10. Once again, the interviewees were asked to think about the possible Implementation Challenges and Implementaion Proposals related to the second prototype. Additionally, the interviewees' Impressions are listed and compared with the Implementation Requirements and unsolved Implementation Challenges in order to identify the potential needs to adjust the StandAlone development to be better suiting the interviewees' actual needs. The Implementation Proposals are briefly discussed and combined with the Further Development Proposals.

The final development iteration started with the development of the StandAlone version of the PDD-SDD Tool, discussed in Chapter 11. The development was guided by the Core Requirements, Implementation Requirements and the interviewees' Impressions of the second prototype. Chapter 12 discusses the final interviews which were conducted after the StandAlone was finished. The StandAlone was presented to the interviewees with the Core Requirements and the Research Questions. The goal of the final interview was to gather Impressions, Implementation Challenges and Development Proposals that helped to identify the solved Implementation Challenges, satisfied Implementation and

Core Requirements, as well as to to answer the Research Questions investigated in this thesis. Additionally, the Implementation Proposals are once again combined with the previous Further Development Proposals and provided at the end of this thesis to illustrate the potential future development to be done with the PDD-SDD tool. The comparison and the results, as well as the Further Development Proposals are discussed and investigated more thoroughly in Chapter 13.

6.2 Introduction of Interviewees

Five developers from MOST Digital Ltd and Ailea Ltd, who are working with MOST Digital RPA projects, were interviewed for this thesis. They all have experience from RPA development, testing, documentation and communication between stakeholders. The interviewees, their work experience in RPA and their job descriptions are briefly introduced in this section.

Heikki Liukkonen

Heikki Liukkonen is a senior developer from Ailea Ltd, who has been working with MOST Digital Ltd's RPA projects. Heikki has been working in the IT industry for 9 years, including 3 years with RPA projects. In addition to RPA development, Heikki has also been working with RPA definition and IT architecture.

Juuso Jaakola

Juuso Jaakola is a developer from Ailea Ltd, who has been working with MOST Digital Ltd's RPA projects. Juuso has been working with RPA projects for 3 years.

Henri Paulamäki

Henri Paulamäki is one of the main developers from MOST Digital Ltd. Henri has been working in the IT industry for 3,5 years, including 1,5 years with RPA projects. In addition to RPA development, Henri has also been working with RPA definition and sales.

Joel Hokkanen

Joel Hokkanen is a developer from MOST Digital Ltd. Joel has been working in the IT industry for 5 years, including 1,5 years with RPA projects.

Niina Mäkelä

Niina Mäkelä is a developer from MOST Digital Ltd. Niina has been working with RPA projects for 1,5 years. In addition to RPA development, Niina has also been working with RPA definition.

7 First Interviews

The first interviews were conducted in September 2019. The interviews were discussing the general aspects of RPA development, which were described in Chapter 3, and the initial design of the PDD-SDD Tool, illustrated in Figure 5.1.

In the first part of the interviews, the interviewees were asked to identify potential Challenges and Proposals for different aspects of the general RPA development. The identified Challenges and Proposals were compared to the Core Challenges and the Core Requirements in order to find out if there was anything still missing from them. The comparison, discussed in Chapter 8, was considered to be paramount for the PDD-SDD Tool's development, since the Core Challenges and the Core Requirements formed the foundation for its initial design.

The second part of the interviews presented the initial design of the PDD-SDD Tool to gather Implementation Challenges, Implementation Requirements and Implementation Proposals relating to the tool's initial design and its concrete development.

The results of the first interview are further analyzed in Chapter 8; they guided the development of the prototypes described in Chapter 9.

7.1 Thoughts on General Aspects of RPA Development

The first part of the interview focused on the different aspects on RPA development, which were introduced in Chapter 3. The interviewees were asked to identify potential Challenges related to them, and potential Proposals that could help solve them. This section is separated by the development aspects discussed, which are communication, documentation, development, testing and PDD updates.

In order to receive unbiased results, the Core Challenges and the Core Requirements were not presented to the interviewees. The identified Challenges and Proposals are presented for every development aspect discussed. The results from the first part of the interview are further discussed in the Chapter 8.

7.1.1 Communication

The discussion related to the communication aspect of RPA development resulted with the Communication Challenges (ComC1–ComC6), listed in Table 7.1, and the Communication Proposals (ComP1–ComP2), listed in Table 7.2. The discussion related mostly to the communication at the beginning of an RPA project and the status reports during the later stages of the development.

Starting a new RPA project has possible communication challenges both internally and with the customer. One of the most common challenge is the difficulty to find a common language between project's stakeholders. In these scenarios the technical and non technical people do not find a common vocabulary, that both understand (ComC1). Technical people may speak in too technical terms while non technical people are not clear enough about what they mean. Also, the communication channels and tools may vary a lot depending on the customer preferences, which could present challenges of its own, when people are communicating via different tools and methods (ComC2). Common, fixed way of communicating with the customer was seen important (ComP1).

Possible challenges with the internal communication were seen to be unclear prioritization and responsibilities (ComC3, ComC4). If the developer is not sure about the priorities between tasks in a RPA project or the priority between two different RPA projects, it is a problem. Similarly if other responsibilities, such as communicating with the customer or updating PDD are not clear, it will cause confusion and delays, possibly resulting in an insufficient end result and documentation. Additionally, unclear or unrealistic timetables were seen very challenging especially if the developer had multiple projects under development simultaneously (ComC5). These caused the developer to have too big a workload with multiple projects and deadlines. To solve these problems, communication should be clearer with both the project management and the developers (ComP2).

Communication Challenge	**Description**
ComC1	No common language between the technical and the non technical people
ComC2	Varying communication tools and channels
ComC3	Unclear internal prioritization
ComC4	Unclear internal responsibilities
ComC5	Unclear timetables
ComC6	The customer does not provide all the tacit knowlegde needed

Table 7.1: Potential Communication Challenges identified in the first interview.

Communication with the customer has its own set of challenges. Oftentimes, the customer does not tell all the tacit knowledge needed by the developer (ComC6). In RPA projects the developer is highly dependent on the customer's instructions, especially if the robot is using the customer's programs.

Communication Proposal	Description
ComP1	Common, fixed way of communicating with the customer
ComP2	General instructions about the official communication tools inside the company

Table 7.2: Proposals to ease communication processes from the first interviews.

The way of updating project status depends a lot of the project manager. Also, the level of detail that the project manager wants to be involved differs. Some may require detailed reports while others are only asking the high level information about the advancement of a project. Communication methods may differ depending on the developer and the project manager. Email and online meetings are used a lot to discuss the projects. Some interviewees consider that the general instructions about the communication tools could sometimes be clearer inside the company (ComP2).

7.1.2 Documentation

The discussion related to the documentation aspect of RPA development resulted with the Documentation Challenges (DocC1–DocC11), listed in Table 7.3, and the Documentation Proposals (DocP1–DocP6), listed in Table 7.4. Documentation was discussed regarding general and technical documentation. General documentation usually consists of higher level information in a README file, while technical documentation means detailed code level documentation, which are called docstrings in Python [34, 35]. The PDD and SDD are both discussed, meaning the same document in its different states in time. PDD is the original document and SDD is its updated form.

Documentation practices differ between interviewees, though they all agree that documentation's role in RPA development is essential. Good documentation should be found in both general and technical documentation. Additionally, up to date visual documentation in PDD was seen paramount in RPA tasks using visual recognition. Even though the value of documentation was not denied, it is not always done properly. Neglecting it is not done by purpose but merely results from different reasons disturbing the developer's workflow. The proposals of making documentation easier related to documentation style and habits, instructions about company's best practices, automating processes between technical and general documentation, and RPA specific documentation ideas.

Up to date documentation was seen to help with both new and old RPA projects. New projects are easier to start when the initial PDD is done properly, with a high level of detail. Older projects may need updates or testing, that require the developer to get familiar with the task's business and code logic. This can be done significantly faster if both the SDD and the technical documentation were properly updated during the previous development cycle.

One of the most common reasons for neglected documentation was insufficient time resources (DocC1). This was often caused by multiple deadlines from different projects closing in simultaneously. Also, sometimes the pressure from the project manager to finish documentation was not high enough and that delayed its creation (DocC2). The general documentation is done more often than the technical documentation, because it is needed also by the support team in the maintenance phase. There was also a notion that finishing the technical documentation was easily neglected if it was not discussed after a project was finished and possible code reviews did not highlight the issue (DocC3).

Both the general and the technical documentation were considered insufficient if they were done too early and not updated later (DocC4, DocC5). Oftentimes, the complete knowledge of a task's requirements were not identified before the first tests in the production environment. This could greatly change some parts of the code logic and therefore require updating the already written documentation. On the another hand, when documentation was done after the project was completed, there was a possibility that the developer did not remember all the important aspects of the created logic anymore, resulting in insufficient documentation (DocC6). One solution for this would be to document at least the most difficult parts of the code logic during the development, to ensure that the essential information is passed forward (DocP1).

There were also potential challenges encountered with older projects. As RPA tasks may differ a lot between customers, there are many different kinds of projects and they do not always follow the same development and logic patterns. Especially custom solutions that differ from commonly used methods are hard to study and understand if they are not documented properly (DocC7). The developer should explain the reasons behind the solution in addition to the documented behaviour (DocP2). Similarly, the developers' old development habits that do not follow the company's best practices will most likely result in inadequate documentation (DocC8). To solve this, the best practices inside company should be continuously discussed (DocP3).

When modifying an older project, most of the interviewees intend to read the documentation. There were notions that if there is no documentation at all or the quality is insufficient, it is making the effort useless (DocC9). The level at which the developer aims to get familiar with the project differs depending on the work task. If the developer needs only to inspect some aspect of the project, for example to help support team in maintenance phase, the developer does not necessarily need to study the whole code base of the

Documentation Challenge	Description
DocC1	Insufficient time resources to create and maintain documentation
DocC2	Documentation not demanded by project management
DocC3	Insufficient documentation instructions in code reviews
DocC4	Documentation becomes insufficient if done too early
DocC5	Documentation becomes insufficient if not updated after changes
DocC6	The developer may forget essential information if documentation is done at the later stages of development
DocC7	It is hard to understand custom solutions that lack a proper documentation
DocC8	The developer having outdated documenting habits compared to company's best practices
DocC9	Insufficient technical documentation of an older project
DocC10	Documentation is limited to general level
DocC11	Updating and old insufficient documentation is a laborious process

Table 7.3: Potential Documentation Challenges identified in the first interviews.

project. If the task is done by the developer, it is usually easier to remember the logic and the reasons behind the development decisions. Leaning on to the tacit knowledge of the developer is of course not preferable.

It was commonly agreed that the most fluent way of creating a code level technical documentation was during the coding process, because then the logic and the development decisions are not yet forgotten. In reality, not everyone followed this method and the technical documentation is often finished just before the maintenance phase. Similarly, the more general level documentation was usually written at the end of the development phase. Notions and thoughts of general behaviour, error scenarios and testing methods were written up during the development. These notions were later added to the transcribed version of the general documentation.

Often the other projects that were scheduled to begin or to be finished in the near future caused the developer's attention to shift towards them rather than to complete the documentation of a just finished project. In this scenario the documentation was often limited to only the general level, to ensure that the maintenance team knows at least the most essential information about the project's error scenarios (DocC10).

Also, correcting older projects' insufficient documentation was seen as a laborious process, because it demanded both time and the proper detailed knowledge of the project (DocC11). Updating old documentation was seen easier if the developer was creating a new major feature to the old process, therefore naturally getting more familiar with the business and the code logic. If, however, the developer was only testing or making minor changes, the updates were not so likely to be documented. These problems were not present if the documentation was already up to date.

Documentation style was seen to be an important aspect of generating good documentation. When the developer adds more context around the code level decisions, a consistent style makes the code more understandable for future readers (DocP4). Additionally, the wider use of docstrings in Python code was seen essential to achieve a proper technical documentation.

Most of the interviewees thought that the common best practices should encourage developers to think more about the quality of the written code and its documentation. A good, consistent documentation style should also help the support team in the mainte-

Documentation Proposal	**Description**
DocP1	Encouraging developers to document at least the most difficult parts of the code logic during development
DocP2	The reasons behind custom solutions should be documented properly
DocP3	Best practices should be continuously discussed in company level
DocP4	Instructing developers to add more context about coding decisions in project code comments
DocP5	Programmatically automating creation of code logic flow from the programmed code
DocP6	Creating more visual documentation about logic and images used in UI based RPA functions

Table 7.4: Proposals to ease documentation processes from the first interviews.

nance phase and the other developers in the further development processes, when they must get familiar with the project. The code reviews were considered to be essential in order to follow the best practices. When the code review is done purposely with every project in the same way, it helps the developers to create consistent documentation and to discuss possible issues with it more easily.

More technical solutions to ease documentation were discussed relating to the connection between the general and the technical documentation. While the general documentation would not necessarily need all the details included in the technical documentation, it could benefit for example the code's logic flow. This could be somewhat automated programmatically (DocP5). This would reduce the laborious documentation work done by the developer.

RPA project have different needs in their documentation. Robots that use visual recognition with their logic, should be documented also visually. This is usually done in PDD and SDD up to some extent, but could also be taken into account in the technical documentation, for example by listing the images used in UI based functions (DocP6).

7.1.3 Development

The discussion related to the actual development aspect of RPA development resulted in the Development Challenges (DevC1–DevC8), listed in Table 7.5, and the Development Proposals (DevP1–DevP2), listed in Table 7.6. The development phase was discussed from two perspectives. The first one was the start of the new project, where the environments are set up for the first time, and the customer is new. The second perspective was the further development of an older project, where environments, systems and documentation are already available.

Many possible challenges relate to inadequate PDD, which force the developer to figure things out later on by asking from the project manager or directly from the customer (DevC1). Additionally, the customer's main motive for automating the process is something that the developer cannot always figure out from the PDD. If the developer is familiar with the motives behind the process instructions, it will help to optimize the solutions the best possible way.

Technical issues encountered with the customer's environments may also cause troubles when starting a new project. If connections or credentials are not properly set up, it could cause delays for development and a project's timetable (DevC2). Additionally, some of the interviewees pointed out the difficulty to reach the customer contacts in bigger organizations (DevC3).

Development Challenge	**Description**
DevC1	Inadequate PDD causing extra work for the developer
DevC2	Authorization issues in the customer's environments and systems
DevC3	Availability issues in larger customer companies
DevC4	Inadequate SDD in the further development
DevC5	Inadequate technical documentation
DevC6	The company's best coding practices are not followed in the older projects
DevC7	Inadequate technical documentation in architectural level
DevC8	Easier to ask help from the previous developer than read the documentation

Table 7.5: Potential Development Challenges identified in the first interviews.

During further development, problems may arise from the outdated SDD (DevC4) or from the insufficient technical documentation (DevC5). If the original PDD was not updated or transformed to represent the outcome of the project, the assumed SDD does not provide the needed value for the developer. The SDD should represent the latest stage of the project, with a proper visual and textual documentation. Additionally, updating the outdated pictures in PDD or SDD is essential especially in projects that navigate the customer's systems with the help of visual recognition.

The technical documentation was seen essential in order to get familiar with the project. Reading only the code base demands significantly more effort. Therefore, the lack of technical documentation usually causes delays because the developer needs more time to study the project. It is essential for the technical documentation to provide context for the solution, in order to tell the developer why the solution is implemented in addition to how it is implemented. Especially the uncommon solutions need to be documented with high detail.

Interviewees agreed that the common coding practices within the company help to get acquainted with an older project. If the coding practices are not aligned or the best practices are not followed, it may cause extra work for the developer to figure out the project's implementation details (DevC6). Additionally, if the architectural solutions are not properly documented in technical documentation, it may cause confusion to the new developer (DevC7).

The new RPA projects tend to lean heavily on the PDD, because it provides the essential business logic information and therefore guides the technical implementation as well. The better the PDD, the better results and faster development. The insufficient PDD usually causes extra work and delays, as already discussed in previous section 7.1.3 (DevC1).

The quality and usefulness of the SDD was seen essential in the later stages of development. The challenge is that developers are not always sure if the SDD is up to date (DevC4). Interviewees read technical documentation in slightly different manner depending on the requirements of the further development task. If the new feature is small, only the essential information tend to be read from the general and the technical documentation. Most of the interviewees familiarize themselves first with the general documentation and check the more technical code level documentation only after that, if needed.

Sometimes it was seen easier and faster to ask help from the previous developers of the project, if they were available. From documentation's perspective this is not a good scenario, because it can be seen to increase tacit knowledge and cause insufficient documentation (DevC8).

Most of the interviewees agreed that more unified and discussed development practices

Development Proposal	**Description**
DevP1	More unified and discussed use of coding and logging practices
DevP2	More unified and discussed use of version control systems

Table 7.6: Proposals to ease development processes from the first interviews.

between company's developers would ease the development processes. These included discussion about coding and logging practices (DevP1) and the use of version control systems (DevP2).

7.1.4 Testing

The discussion related to the testing aspect of RPA development resulted with the Testing Challenges (TesC1–TesC10), listed in Table 7.7, and Testing Proposals (TesP1–TesP8), listed in Table 7.8. Interviewees considered that the potential issues of testing RPA projects usually originate either from initial shortcomings of the system under automation and the customer's environments, or insufficient internal documentation of the older project. An RPA project needs proper test procedures that should be figured out and documented properly. Documenting the test instructions can help significantly when developing new features to older project as well.

The customer needs and systems vary a lot between different RPA projects. From this reason alone, the same testing setup is not possible for every project. Nevertheless, testing should be discussed already in the definition phase, where systems and processes are initially discussed (TesP1). If the requirements for testing are not properly planned ahead, it usually causes extra work and challenges for the developer (TesC1). Testing requirements include information about the customer's test environments as well as test data for the systems under automation.

Testing Challenge	**Description**
TesC1	Testing is not planned in advance to anticipate its needs
TesC2	Testing can only be done with the production data
TesC3	Test environments for automated systems are not available
TesC4	Automated process not able to be tested in smaller segments
TesC5	Insufficient documentation of older project's testing procedures
TesC6	Testing documentation is lacking tacit knowledge
TesC7	Insufficient unit tests in code level
TesC8	Insufficient documentation of the customer's other processes that may be affected of the current task's procedures
TesC9	Unclear logging
TesC10	Unclear commits in version history

Table 7.7: Potential Testing Challenges identified in the first interviews.

Sometimes the complete end-to-end testing of a process can only be done with the actual production data (TesC2). In these scenarios the test environments for automated systems are usually not available (TesC3). Sometimes, when systems' test environments do exist, they may still lack some of the production environments' features, which should also be tested properly.

If the automated process cannot be tested in smaller segments, it results in laborious testing processes for the developer (TesC4). This might be the case for example in a situation where the process' further steps are highly dependent on its previous ones. Sometimes testing can also be challenging because of the insufficient internal documentation of the older process' test procedures and possibilities (TesC5). If the testing procedures are not clearly documented, it is difficult for the developer to know how freely the process can be tested without problems in the customer's end, especially, if the production environment is the only one available. The testing instructions should be documented in high detail, including possible tacit knowledge to avoid problems in the future (TesC6).

Code level testing is challenging if unit tests are not done properly. In these cases the new feature may cause unexpected errors in other parts of the code, which may be hard to locate (TesC7). Also, if the process causes possible conflicts with the customer's other processes, either with different RPA tasks or with the customer's own internal procedures, it should be clearly documented to the architectural level documentation (TesC8). Furthermore, it is sometimes necessary to check previous behavior or changes in order to find out the cause for possible errors in the process. Behavior is hard to inspect if logging is not clear and informative (TesC9). Similarly, if commits in version control are unclear it is harder for the developer to understand them (TesC10).

Interviewees considered documentation to be helpful if it is reliable. Once again, this is not always the case. The more clear testing instructions the documentation includes, the easier the testing is. Often a good documentation was also seen to help with the communication related to the test cases and situations. Interviewees all considered the initial test planning to be essential to anticipate the test requirements and to ease the testing processes during and after the development phase (TesP1). Additionally, the original PDD should be continuously updated towards SDD in order to minimize the risk of an outdated SDD (TesP2).

Best testing practices should be discussed often and improved continuously to ensure that

Testing Proposal	**Description**
TesP1	Test planning should done already in the definition phase to anticipate test requirements
TesP2	PDD should be continuously updated towards SDD
TesP3	Discussing and continuously improving company's best testing practices
TesP4	More consistent use of unit testing
TesP5	Logging following company's best practices
TesP6	Version control messages following company's best practices
TesP7	Implemented testing features to enable testing with production environment without interfering the customer's other systems, if possible
TesP8	Testing procedures should be comprehensively discussed, documented and taken into account during communication

Table 7.8: Proposals to ease testing processes from the first interviews.

the prerequisites for them are at their best possible level (TesP3). Many interviewees agreed that these procedures should include more consistent use of unit testing (TesP4), and better practices in both logging (TesP5) and version control (TesP6). Logging should be clear and informative while version control's commit messages should provide information with high detail, in order to the developer to know what kind of changes took place.

In processes that do not have test environments, there should be testing features implemented at code level, ensuring that the developer may execute the program without the fear of interfering with the customer's other processes (TesP7). If that is not possible, testing possibilities and practices should at least be comprehensively documented in the technical documentation and the general documentation, as well as discussed between developers and testers (TesP8).

7.1.5 Updating PDD During Development

The discussion related to the PDD updates in RPA development resulted with the PDD Challenges (PDDC1–PDDC5), listed in Table 7.9, and PDD Proposal (PDDP1), listed in Table 7.10. The discussion focused on the developer's viewpoint of updating the PDD during development.

Habits of updating PDD differ between interviewees. Some update the PDD only when especially asked to, while others tend to do it more often. Updating the PDD after changes was seen to be essential by all. The value of PDD was highlighted by all as the PDD helps to get familiar with the customer's original problem, to which the developed RPA software should provide help. Additionally, realizing the customer's mindset and goals was seen to help to choose the right development approaches and solutions.

Updating the PDD has its own set of challenges. Especially smaller updates and changes in the business logic may not end up documented in the PDD (PDDC1). The updates are also easily forgotten if they are not especially requested by the project manager or the customer (PDDC2). Often it is not clear if updates should be done automatically by the developer or the project manager (PDDC3). The Insufficient PDD or SDD cause the developer to have an unclear perception of the project's goals and its current status (PDDC4).

Sometimes the distribution of the PDD and its versions may be insufficient. If version control is not done properly, the customer could end up having a different version than the developer, and possible updates may not end up to both of them (PDDC5).

PDD Challenge	**Description**
PDDC1	Minor updates of business the logic are not always documented
PDDC2	Updates are easily forgotten if they are not especially requested
PDDC3	The developer is not sure who is responsible for updating the PDD
PDDC4	The outdated PDD or SDD create unclear perception of the project
PDDC5	Version control of PDD and SDD are insufficient

Table 7.9: Potential PDD Challenges identified in the first interviews.

One proposal to ease the updating procedures of the PDD and the SDD was to better declare the responsibilities between the developer and the project manager (PDDP1). It was considered to be better that the developer was responsible for updating the documentation, in order to ensure that the correct information would be documented in high detail.

PDD Proposal	**Description**
PDDP1	Clearer roles for updating the PDD

Table 7.10: Proposals to ease updating PDD from the first interviews.

7.2 Thoughts on Initial Design of PDD-SDD Tool

The second part of the first interview focused on the initial design of the PDD-SDD Tool, which was illustrated in Figures 5.1 and 5.2. The goal was to find out the possible hits and misses with it, based on the previous discussion about general aspects of RPA development.

The discussion yielded Implementation Challenges (IC1–IC10) listed in Table 7.11, Implementation Requirements (IR1–IR8) listed in Table 7.12 and Implementation Proposals (IP1–IP6) listed in Table 7.13. The Implementation Requirements were considered to be essential for the PDD-SDD Tool's development to succeed, while the Implementation Proposals were merely ideas of what features it could have in the future, after the development phases described in this thesis.

The Initial Design had a positive reception among the interviewees, who thought that the presented tool could ease the aspects discussed earlier in the first interviews. However, there were some challenges that should be solved in order for the tool to work as intended. The interviewees had concerns that the tool would end up being too technical and thus not clear enough for the customer to use (IC1). The division between the technical and the definition view should also be clear, so that the definition view remains easy to comprehend and work with (IR1). It was also pointed out, that Finnish should be supported in the definition view but not in the code file, which should be in English (IC2).

The interviewees thought that the technical view would be useful if it visualizes the whole code logic and its logical structure in high detail (IR2). This would help to form the overall perception of the project in addition to the business logic presented in the definition view. At best, the tool could help in many ways in development and testing. However, even if the tool's implementation would succeed it should still be supplemented with an-

Implementation Challenge	**Description**
IC1	The tool could end up being too technical for the customer to use
IC2	Finnish ending up in code from the definition view
IC3	The definer having too much capabilities to make logical decisions trough the definition view
IC4	The generated code file could include too much information and placeholder functions, making it more unclear than helpful
IC5	How to ensure that the original business process definition is not lost in the updated view that is read from the code file
IC6	How to update the project definition without interfering the code logic
IC7	How to create a technical view with enough detail to actually be useful during development and testing
IC8	How the definition level can include the essential level of information without the logical structures of code
IC9	It is very difficult to create a tool that actually saves time during the RPA development processes
IC10	It is very difficult to create a tool that could actually replace old procedures that people are used to

Table 7.11: Implementation Challenges of the PDD-SDD Tool's initial design identified in the first interviews.

other location for additional documentation. It was proposed that the tool could include links to additional information, such as version control and external documentation (IP1).

Implementation Requirement	**Description**
IR1	The division between the technical and the definition view should be clear
IR2	The technical view should visualize the code logic and its logical structure in high detail
IR3	The developer has to be able to change the code structure after the PDD has been accepted and the code file has been created
IR4	The tool should provide an easy way to include information about the business logic exceptions
IR5	The definition view's information should be linked with the code file only by modifiable tags in function docstrings, separating them from the executable code logic
IR6	The `main` function should include the Phase level function calls to provide the overall logic flow of the defined process
IR7	The tool should provide the original definition view to be compared with the updated ones
IR8	The updated definition view should be able to be reviewed and accepted by the customer

Table 7.12: Implementation Requirements for the PDD-SDD Tool's initial design identified in the first interviews.

The first step of creating the initial PDD with the customer was seen problematic if the definer had too much capabilities to the make code level logic decisions through the tool's definition view (IC3). The interviewees agreed that developers alone should be responsible for code logic and its solutions because they possess better knowledge about it than the project manager or the customer. It was also agreed that the developer should have the power to change the logical structure of the generated code base if necessary (IR3). Additionally, the interviewees discussed the need to inform the developer about the business logic exceptions and how it should be implemented in the definition view of the tool (IR4).

While the automatic code generation was seen as a good thing, there were concerns of how to prevent an excessive amount of information ending up in the code file. Many of the interviewees thought that the potentially large amount of generated Phase and Step level functions would be more annoying than useful from the developer's viewpoint (IC4). The language was also seen challenging for the generated code file, because if the definer is defining the process in Finnish, the declared titles would end up in the code file's placeholder function names. In order to allow the modification of the function names without interfering the definition view's Phase and Step element titles, they should be linked for example with tags in function docstrings (IR5).

The generated code file should provide the main ideas of the defined PDD in a way that is easy to read and comprehend. The `main` function should contain the Phase level function calls to provide the overall business logic flow defined for the process (IR6). It was also pointed out, that the tool should generate code that follows the agreed best styling practices (IP2).

The final part of the designed tool was the updated visual presentation of the code logic in both the definition and the technical views. Most of the interviewees' concerns were linked to this part. One of the main challenges related to the preservation of the original process definition. The tool creates an updated view of how the code logic works, but not necessarily how it should be working. It is essential to ensure that the original business process remains available and the definition has not been changed accidentally by the updated code base (IC5). In order to prevent this the original documentation written between the customer and the project manager should be protected. The tool should provide the original definition view that could be compared to the updated one (IR7). Furthermore, the customer should be able to review and accept the updated definition view (IR8). The tool could also include more comprehensive version control, allowing the user to browse the project's version history (IP3).

Implementation Proposal	**Description**
IP1	The tool could provide links to other sources of material and documentation
IP2	The tool should generate code that follows the company's best styling practices
IP3	The tool could allow browsing the project's version history
IP4	There could be a possibility for the customer or the project manager to suggest changes in the process' business logic in the definition view later on
IP5	Images used for visual recognition could be included in the tool's technical view
IP6	The tool's technical view could be used as an educational tool for the project manager and the customer to get them familiarized with the project in more technical manner

Table 7.13: Further development Proposals for the PDD-SDD Tool's Initial Design identified in the first interviews.

Interviewees pointed out that it should be possible to update the process definition later on by the customer without interfering with the generated and developed code logic (IC6). The suggested procedure for this was to discuss changes separately in meetings or via other media so that the developer can implement the change. The updated code logic could then be reviewed and accepted with the updated definition view in the tool (IR8).

Other possibility would be to include a feature that allows the customer or the project manager to propose business level changes in the definition view (IP4).

There were doubts about the usefulness of the technical view and how it could visualize the actual code flow in enough detail to actually be useful during development and testing (IC7). Additionally, it was wondered how the definition view could have all the essential information without the logical structures of the code file (IC8). It was agreed that the definition view should only contain original or updated instructions of the business logic in text form that is linked to the project's code files with code level comments and tags, while the technical view should be the one tracking the actual code logic flow more closely (IR2, IR5).

In order to create a tool that actually works, it should be saving time for the RPA development processes. Achieving this was seen to be one of the main challenges (IC9). If the tool should be the only way of doing definition and development, its requirements increase a lot. It could end up trying to ease a little bit of every aspect of the development, actually resulting to provide not enough help to any of them. The tool could end up to be too stiff or too unrestricted to use, in both cases resulting to be inadequate. It was seen as a helpful addition to another tools but not necessary being the only one (IC10).

There were also plenty of other proposed features that could be useful in the tool. The tool could include the images used for visual recognition in the technical view (IP5). The technical view was also considered to be useful for educational purposes. The developer could discuss with the customer and the project manager with the RPA process' logic ”as the robot sees it”. This could be helpful when defining further development projects with the same customer, who would now have a better understanding of what kind of information is needed and how the process could be presented to effectively automate processes with RPA software (IP6).

8 First Interviews' impact on Core Challenges, Core Requirements and Tool's Design

In order to continue with the PDD-SDD Tool's development, the initial design introduced in Chapter 5 had to be adjusted with the data gathered from the first interviews, which were discussed in Chapter 7. As the tool's design is based on the Core Challenges and the Core Requirements identified before the first interviews in Chapter 4, they needed to be potentially updated as well. This ensured that the PDD-SDD Tool's design and development would match the actual needs of the interviewed developers.

In this chapter, the Core Challenges, listed in Table 4.2 and the Core Requirements, listed in Table 4.3 are compared to the development Challenges and development Proposals identified from the first part of the interviews, which discussed the general aspects of the RPA development. The possible new Core Challenges and Core Requirements identified from the comparison are filtered further by discussing if they are solvable by a single tool. Finally, the updated lists of Core Challenges and Core Requirements are created.

The second part of the interviews resulted in the Implementation Challenges, the Implementation Requirements and the Implementation Proposals regarding of the PDD-SDD

Tool's design. The last section of this chapter inspects these more closely in order to further adjust the initial design of the tool from a developer's viewpoint. The inspection results in the unsolved Implementation Challenges, the Implementation Requirements and the first version of Further Development Proposals. The unsolved Implementation Challenges are going to be addressed by the upcoming interviews in Chapters 10 and 12, the Implementation Requirements guide the tool's development with the Core Challenges in Chapters 9 and 11, and the first version of the Further Development Proposals are additional features for the PDD-SDD Tool, which are to be implemented after this thesis.

This chapter completes the first development iteration depicted in the updated roadmap in Figure 6.1. The second development iteration and the actual development of the PDD-SDD Tool starts with the two prototypes, discussed in Chapter 9.

8.1 Updating Core Challenges and Core Requirements based on First Interviews

The first part of the first interviews resulted with the development Challenges and the development Proposals relating to the general aspects of the RPA development. The Core Challenges and the Core Requirements listed in Tables 4.2 and 4.3 in Chapter 4 were not presented during the first interviews in order to get unbiased thoughts from the interviewees.

The resulted development Challenges and development Proposals were listed separately for communication, documentation, development, testing and PDD updates. This section compares these results to the previously identified Core Challenges and the Core Requirements in order to recognize the potential need to update them.

The first comparison in Subsection 8.1.1 focuses on the development Challenges and the Core Challenges, while the second Subsection 8.1.2 compares the development Proposals to the the Core Requirements. An development Challenge or an development Proposal is considered to be a new one if it is solvable by a single tool and cannot be matched with the already identified Core Challenges or Core Requirements.

The identified new Core Challenges are further compared to the Core Requirements at the end of this section. The comparison results in an updated list of Core Requirements, which are supposed to cover all of the identified Core Challenges.

8.1.1 Comparing Development Challenges to Core Challenges

The comparison is done separately for every aspect of the general RPA development discussed in the first interviews. Every development Challenge category is compared to the Core Challenges (C1–C25), listed in Table 4.2. The identified new Core Challenges are further discussed in Subsection 8.1.3.

Communication Challenges from First Interviews

There were 6 Communication Challenges (ComC1–ComC6) identified from the first interviews, listed in Table 7.1. All of them match one or more Core Challenge from the definition, communication, documentation and development aspects of RPA development. Relations between the Challenges are illustrated in Table 8.1.

The difficulty of finding a common language between technical and non technical people connects ComC1 to C4 and C18. Varying communication tools in ComC2 can be seen resonating with C20, because the unified development conventions also include communication methods. Issues caused by unclear timetables connect ComC5 to C8. ComC6

	Core Challenges									
	Definition	**Communication**					**Documentation**		**Development**	
	C2	**C4**	**C5**	**C6**	**C7**	**C8**	**C10**	**C13**	**C18**	**C20**
ComC1		x							x	
ComC2										x
ComC3			x	x	x	x			x	
ComC4			x							x
ComC5						x				
ComC6	x						x	x		

Table 8.1: Relations between the Core Challenges and the Communication Challenges identified from the first interviews.

describes the issue of the customer's tacit knowledge not being documented, and correlate directly to C2 and C13, and may also be one the reasons for insufficient PDD in C10.

Issues regarding unclear prioritization in ComC3 and responsibilities in ComC4 can be matched with multiple Core Challenges from the communication and development aspects of RPA development. ComC3 may be caused by too vague instructions as described in C5, unclear level of progress in C6, unclear customer expectations in C7 or unclear timetables in C8, which all relate to misunderstandings due to communication in C18. Similarly, ComC4 relates to C5 but also to the company's development conventions in C20.

Documentation Challenges from First Interviews

There were 11 Documentation Challenges (DocC1–DocC11) identified from the first interviews, listed in Table 7.3. Almost all of them resonate with the Core Challenges, spreading over the communication, documentation and development aspects of RPA development. Relations between the Challenges are illustrated in Table 8.2.

	Core Challenges									
	Communication	**Documentation**						**Development**		
	C5	**C11**	**C12**	**C13**	**C14**	**C15**	**C16**	**C19**	**C20**	**New Challenge**
DocC1							x			
DocC2	x								x	
DocC3	x								x	
DocC4			x	x			x			
DocC5			x	x			x			
DocC6			x	x			x			
DocC7		x						x		
DocC8	x								x	
DocC9		x		x	x	x		x		
DocC10		x	x	x				x		
DocC11										x

Table 8.2: Relations between the Core Challenges and the Documentation Challenges identified from the first interviews.

The insufficient time to create and maintain documentation connects Doc1 to C16. Unclear instructions and divided development conventions discussed with DocC2, DocC3 and DocC8 relate to C5 and C20. Similarly, DocC4, DocC5 and DocC6 can be linked with C12, C13 and C16, which also discuss the lack of documentation about made decisions and tacit knowledge, which might be a symptom of insufficient time to create documentation.

DocC7, DocC9 and DocC10, which were encountered especially during later stages of development, correlate with the Core Challenges C11, C12, C13, C14, C15 and C19, which

	Core Challenges											
	Communication	**Documentation**						**Development**			**Maintenance**	
	C9	**C10**	**C11**	**C12**	**C13**	**C14**	**C15**	**C17**	**C20**	**C21**	**C25**	**New Challenge**
DevC1		x						x				
DevC2										x		
DevC3	x											
DevC4				x	x						x	
DevC5			x									
DevC6									x			
DevC7			x							x		x
DevC8					x	x	x					x

Table 8.3: Relations between the Core Challenges and the Development Challenges identified from the first interviews.

all describe the shortcomings and misunderstandings resulting from insufficient technical documentation.

The laborious updates to old insufficient documentation in DocC11 is a new Challenge because it has not been clearly identified before.

Development Challenges from First Interviews

There were 8 Development Challenges (DevC1–DevC8) identified from the first interviews, listed in Table 7.5. Development Challenges are related to the Core Challenges from communication, documentation, development and maintenance aspects of RPA development. Relations between the Challenges are illustrated in Table 8.3.

An insufficient PDD in DevC1 resonate with C10 and C17. DevC2 is describing authorization issues in the customer's environments, therefore relating to Immature IT architecture in C21. Availability issues create a direct link between DevC3 and C9.

The issue of an outdated SDD in the later stages of development described in DevC4 relates directly to C25 and it can also be caused by the lack of documented decisions and tacit knowledge in C12 and C13. Inadequate technical documentation in DevC5 can be directly linked to C11. Similarly, the issue where the developers are not following the company's best practices is described in both DevC6 and C20.

Even though the inadequate architectural documentation described in DevC7 may be connected to insufficient technical documentation in C11 and immature IT architecture in C21, it is also a new kind of Challenge, because architectural documentation was not especially discussed when identifying the Core Challenges in Chapter 4. DevC8 also describes a new Challenge, where the developer tends to ask help directly from the previous developer without trying to read the documentation first. Reasons for that are likely to be caused by insufficient documentation of tacit knowledge in C13, or documentation not being in unified location or format, as described in C14 and C15.

Testing Challenges from First Interviews

There were 10 Testing Challenges (TesC1–TesC10) identified from the first interviews, listed in Table 7.7. Testing Challenges are related to the Core Challenges from documentation, development and testing aspects of RPA development. Relations of the Challenges are illustrated in Table 8.4.

Lack of test material in TesC2 correlates with C22. Similarly, lack of test environments binds TesC3 and C23 together. Insufficient documentation of testing procedures connects TesC5 to C11 and insufficient documentation of tacit knowledge described in TesC6 relates to C13. Furthermore, both TesC5 and TesC6 correlate with poorly documented

	Core Challenges						
	Documentation		**Development**	**Testing**			
	C11	**C13**	**C20**	**C22**	**C23**	**C24**	**New Challenge**
TesC1				x	x	x	x
TesC2				x			
TesC3					x		
TesC4							x
TesC5	x					x	
TesC6		x				x	
TesC7			x				
TesC8							x
TesC9			x				
TesC10			x				

Table 8.4: Relations between the Core Challenges and the Testing Challenges identified from the first interviews.

exception situations in C24. Additionally, insufficient unit tests in TesC7, unclear logging in TescC9 and unclear version history commits in TesC10 are all somewhat related to the lack of unified development conventions inside the company in C20.

TesC1, TesC4 and TesC8 are new Challenges. TesC1 discusses the need to anticipate testing requirements in advance, while TesC4 describes a situation where an automated system cannot be tested in smaller segments. TesC8 raises the issue of insufficient documentation of the interconnected customer's systems and environments. TesC1 can be seen as a core Challenge, because it may be one reason for both TesC4 and TesC8 as well as the lack of test materials in C22, environment issues in C23 and poorly documented exception situations in C24.

	Core Challenges												
	Communication			**Documentation**					**Development**				**Maintenance**
	C5	**C6**	**C7**	**C10**	**C12**	**C13**	**C14**	**C15**	**C17**	**C18**	**C19**	**C20**	**C25**
PDDC1					x	x			x				
PDDC2	x												
PDDC3	x									x		x	
PDDC4		x	x	x	x	x			x		x		x
PDDC5		x	x				x	x			x		

Table 8.5: Relations between the Core Challenges and the PDD Challenges identified from the first interviews.

PDD Challenges from First Interviews

There were 5 PDD Challenges (PDDC1–PDDC5) identified from the first interviews, listed in Table 7.9. They are all related to the Core Challenges from the communication, documentation, development and maintenance aspects of RPA development. Relations of the Challenges are illustrated in Table 8.5.

PDDC1 relates to decisions and tacit knowledge not being documented in C12 and C13. It can also be one reason for outdated PDD in C17. The forgotten updates in PDDC2 depend a lot on the developer but might also be caused by too vague instructions in C5. Unclear responsibility issues in PDDC3 are similarly linked to C5 but also to misunderstandings due to communication in C18 and the lack of unified developing practices inside the company in C20.

The unclear perception of the project due to oudated PDD or SDD described in PDDC4 correlates with unclear level of progress in C6, unclear customer exceptations in C7, outdated or insufficient documentation in C10, C12, C13, C17 and C25, as well as misunderstandings caused by these in C19. The insufficient version control of PDD and SDD in PDD5 is similarly linked to C6 and C7, relating also to the documentation issues of unified location and format in C14 and C15 and may be one the reasons for misunderstandings due to documentation in C19.

8.1.2 Comparing Development Proposals to Core Requirements

All of the the development Proposals from different aspects of RPA development are combined and compared with the Core Requirements (R1–R3) listed in Table 4.3 in order to find out if there still is a need for additional Core Requirements for the PDD-SDD Tool.

All of the development Proposals were either not solvable by a single tool, or were satisfied with the already derived Core Requirements. The relations between the development Proposals and the Core Requirements are listed in Table 8.6.

Communication Proposals from First Interviews

There were 2 Communication Proposals (ComP1 and ComP2) identified from the first interviews, listed in Table 7.2.

Both ComP1 and ComP2 discuss general instructions about official communication tools and methods to be used internally and with the customer. These Proposals should be handled by discussing the best practices inside a company, not by a single tool.

		Core Requirements			
Aspect	**Proposal**	**R1**	**R2**	**R3**	**Not Solvable by Single Tool**
Communication	**ComP1**				x
	ComP2				x
Documentation	**DocP1**				x
	DocP2				x
	DocP3				x
	DocP4				x
	DocP5			x	
	DocP6			x	
Developement	**DevP1**				x
	DevP2				x
Testing	**TesP1**	x			
	TesP1			x	
	TesP3				x
	TesP4				x
	TesP5				x
	TesP6				x
	TesP7	(x)			x
	TesP8	(x)	(x)	(x)	x
PDD	**PDDP1**			x	

Table 8.6: Relations between the development Proposals from the first interviews and the Core Requirements for the PDD-SDD Tool. Relations marked by (x) partly satisfy the proposal, but not completely.

Documentation Proposals from First Interviews

There were 6 Documentation Proposals (DocP1–DocP6) identified from the first interviews, listed in Table 7.4.

DocP3 addresses the need for continuously discussed best practices at the company level, which is not solvable by a single tool. Additionally, the documentation instructions about custom solutions in DocP2, difficult parts of the code in DocP1, and additional context about made decisions in DocP4 are similarly more tied to company's best practices.

DocP5 and DocP6 in the other hand propose to automate the documentation process, to create a code logic flow from the programmed code and to create more visual documentation. These are similar to R3, which aims to provide automatic documentation.

Development Proposals from First Interviews

There were 2 Development Proposals (DevP1 and DevP2) identified from the first interviews, listed in Table 7.6.

Development Proposals DevP1 and DevP2 discuss the unified coding, logging and version control practices, which similarly to Communication Proposals are more likely to be eased by company's best practices than a single tool.

Testing Proposals from First Interviews

There were 8 Testing Proposals (TesP1–TesP8) identified from the first interviews, listed in Table 7.8.

Once again, the majority of the Testing Proposals relate to company's best practices, which are not solvable by a single tool. TesP3 directly underlines the importance of discussing and improving best testing practices, while TesP4, TesP5 and TesP6 address testing practices from their own viewpoints of unit testing, logging and version control.

Test planning done already in the definition phase in TesP1 and continuously updated PDD in TesP2 could be solved with R1 and R3. R1 aims to provide guiding questions during the definition phase, which could include also specific questions about testing, therefore satisfying TesP1. TesP2 is directly bound to R3, which aims to update documentation automatically.

The implementation and discussion of testing procedures in TesP7 and TesP8 are mostly linked to the company's best practices but could also be partially eased by the automated questions of R1. Additionally, as easing documentation and communication are one of the main goals of the PDD-SDD Tool, TesP8 should be partially eased by R2 and R3 as well.

PDD Proposals from First Interviews

There was 1 PDD Proposal (PDDP1) identified from the first interviews, listed in Table 7.10.

PDDP1 addresses the need to define clearer roles for updating the PDD. This is directly solvable with automated documentation in R3.

8.1.3 New Core Challenges and Core Requirements Identified

The majority of the development Challenges discussed in Subsection 8.1.1 correlate with the Core Challenges in Table 4.2. However, there were six new Core Challenges from the documentation, development and testing aspects of RPA development that were not clearly taken into account before. The comparison between the development Proposals and the Core Requirements discussed in Subsection 8.1.2 did not result with new Core Requirements.

This subsection further inspects the new Core Challenges (NC1–NC6), which are combined from the section 8.1.1 and listed in Table 8.7. The goal is to determine if they are solvable by a single tool and the Core Requirements listed in Table 4.3. If the Core Requirements do not solve all of the new Core Challenges, new ones are derived.

All of the new Core Challenges are all at least partly solvable by a single tool. Automatic documentation required in R3 directly solves NC1 and partially eases NC3. Otherwise, NC3 can be seen as a developer-specific quality and as such, is not solvable by a single tool.

Other Core Challenges are more linked to R1, because the Testing Challenges NC4, NC5 and NC6 can be eased with the automated questions provided during the definition phase. The questions could ensure that the prerequisites for testing are at least discussed as early as possible, giving the customer time to provide the needed test material, as well as to list other processes that may be affected by the defined process.

Similarly, the insufficient architectural documentation described in NC2 could be gathered with the automated questions that ensure that the essential information is documented. In R1, the automated questions are provided in the definition phase. However, the architec-

Aspect	New Challenge	Description
Documentation	**NC1**	Updating old insufficient documentation is a laborious process
Development	**NC2**	Inadequate technical documentation in architectural level
	NC3	Easier to ask help from the previous developer than read the documentation
Testing	**NC4**	Testing is not planned in advance to anticipate its needs
	NC5	Automated process not able to be tested in smaller segments
	NC6	Insufficient documentation of the customer's other processes that may be affected of the current task's procedures

Table 8.7: New Core Challenges identified from the first interviews.

tural information about the process is often company's internal information, and therefore not something to be discussed with the customer. This means that some automated questions should be provided internally, separated from the definition phase. This was not described by the Core Requirements, so a new Core Requirement is created. List of updated Core Requirements with their corresponding Core Challenges are illustrated in Table 8.8.

The new Core Requirement, listed as R4, could also ease other Core Challenges relating to the technical and architectural documentation. R4 could ease the lack of technical and tacit documentation in C11 and C13 as well as to create documentation in a unified format, answering to C15. Additionally, it could help to avoid misunderstandings due to documentation in C19 and prevent setting in an immature IT architecture in C21.

Requirement	Description	Root Challenges	Sub Challenges
R1	Providing guiding questions during the definition phase	C1, C2, C22, C23	C3, C5, C7, C10, C13, C18, C19, C21, C24, NC4, NC5, NC6
R2	Providing accessible, unified and up to date documentation	C14	C6, C7, C8, C18, C19
R3	Providing automatic documentation	C15, C16, NC1	C11, C17, C18, C19, C25, NC3
R4	Providing technical and architectural questions internally after the definition phase	NC2, C15	C11, C13, C19, C21

Table 8.8: Updated Core Requirements based on the results from the first interviews. The table is an updated version of Table 4.3, which is supplemented with new Core Requirement R4 and the new Core Challenges NC1-NC6.

8.2 First Interviews' Impact on PDD-SDD Tool's Initial Design

One of the main goals for the first interviews was to validate the initial design of the PDD-SDD Tool, and to determine if it needed to be adjusted based on the interviewees' thoughts. This should be done if the interviews' results from either the of general aspects of RPA development or from the presentation of the tool's initial design would demand it.

The updated Core Requirements (R1–R4) derived in Section 8.1 and listed in Table 8.8 did not cause changes in the PDD-SDD Tool's initial design. The implementation of R4 was merely an addition with a similar implementation than R1. The specific placement for the questions was going to be defined by the PDD-SDD Tool's progress and the upcoming interviews, which are discussed in Chapters 10 and 12.

The PDD-SDD Tool's initial design was presented and discussed during the final part of the first interviews, which resulted in Implementation Challenges, Implementation Requirements and Implementation Proposals specifically related to the technical implementation of the tool. These are not to be mixed with the previously discussed Core Challenges, Core Requirements and Development Proposals, which related to the general aspects in RPA development.

This section discusses the Implementation Challenges, the Implementation Requirements, and the Implementation Proposals. In order to solve the identified Implementation Challenges, they are compared to the Implementation Requirements and the Implementation Proposals, to find possible solutions for them.

The Implementation Requirements and the Core Requirements guided the prototype development described in Chapter 9. The Implementation Challenges that are left unsolved, are answered later by the the upcoming interviews in Chapters 10 and 12. The Implementation Proposals are left to be further addressed after the second interviews as well, to be combined with the new Further Development Proposals resulting from them.

8.2.1 Comparing Implementation Challenges to Implementation Requirements and Implementation Proposals

The Implementation Challenges gathered from the first interviews possessed a potential threat for the design and success of the PDD-SDD Tool. In order to solve them, they are compared with both of the Implementation Requirements and the Implementation Proposals. An Implementation Challenge is considered to be solved if it can be eased by either an Implementation Requirement or an Implementation Proposal.

The Implementation Challenges (IC1–IC10) are listed in Table 7.11, the Implementation Requirements (IR1–IR8) are listed in Table 7.12 and the Implementation Proposals (IP1–IP6) are listed in Table 7.13. The comparison is illustrated in Table 8.9.

IC1 raises concerns about the usage of the tool if it is too technical, which IR1 and IR5 aim to resolve with clearly divided definition and technical views. The definition view should only contain business level information, leaving the technical details to the technical view.

In order to solve IC2 and prevent Finnish or other unwanted language ending up in code level function names, the IR5 introduces an idea that links information between the definition view and code level with tags included in function docstrings. After this the developer can change function names if needed, without changing the titles in the definition view.

Challenge	**Answer**
IC1	IR1, IR5
IC2	IR5
IC3	IR3
IC4	To be answered in the upcoming interviews
IC5	IR7, IR8
IC6	Communication, IP4
IC7	IR2
IC8	IR5
IC9	IR6, To be answered in the upcoming interviews
IC10	To be answered in the upcoming interviews

Table 8.9: Matching the Implementation Challenges and Implementation Requirements identified from the first interviews.

The link between the definition view and the generated code file raised concerns about the definer's ability to define the code level decisions and logical structures in IC3. IR3 solves this by allowing the developer to have full control over the generated code file and its modifications. Additionally, to ensure that the initial definition does not get lost during code updates as feared in IC5, the tool should provide the original definition view to be compared with the new one in order to validate it, as depicted in IR7.

In addition to the developer's ability to modify the code file, the customer should also be able to update the original definition later, after the development has started. The correct way to do this was wondered in IC6. It was agreed that in order to do this the change should be communicated directly between stakeholders. An additional feature for this in the future development was proposed in IP4.

IC7 raises a concern of how to achieve a technical view with enough detail to actually be useful during development and testing. IR2 aim to resolve this by including detailed logical structures to the code visualization in the technical view. Similarly, IC8 discusses of how to include the necessary amount of information into the definition view without the logical code structures. The answer is to limit the information to the business logic level, as described in IR5.

IC9 and IC10 stated that it is very difficult to actually create a tool that saves time and could replace old procedures. In order to do this, an extra care should be paid to usability features of the tool. Additionally, a clearly constructed main function in code level was seen to help to create overall picture of the code flow and therefore save time when starting a new project, as described in IR6.

Most of the Implementation Requirements and the Implementation Proposals provide solutions for the Implementation Challenges. However, there were also Implementation Challenges that were not directly answered during the interviews, listed in Table 8.10. The concerns regarding generated code's readability in IC4 and the partly answered IC9 and IC10 were left unanswered. These were left to be further discussed the following interviews in Chapters 10 and 12.

Unsolved Implementation Challenge	**Description**
IC4	The generated code file could include too much information and placeholder functions, making it more unclear than helpful
IC9	It is very difficult to create a tool that actually saves time during RPA development processes
IC10	It is very difficult to create a tool that could actually replace old procedures that people are used to

Table 8.10: Unsolved Implementation Challenges for the PDD-SDD Tool from the developer's viewpoint after the first interviews.

9 Prototype Development

The prototype development was conducted after the first interviews. The development was guided by the Core Requirements (R1–R4), listed in Table 8.8 and the Implementation Requirements (IR1–IR8), listed in table 7.12. Two prototypes were developed in order to validate that the code file could be generated from the PDD and, after modifications, processed back to updated SDD. The automatic documentation was one of the core functionalities for the PDD-SDD Tool, demanded by R3.

This thesis focuses only to describe the implemented functionalities and provide screenshots and code file listings from the prototypes and the StandAlone version of the PDD-SDD Tool. The source code of the tool is not provided.

The PDD-SDD Tool's visual user interface consists of the definition view and the technical view. The definition view consists of Phase, Step and Detail levels of the business logic, as defined by the initial design illustrated in Figure 5.2. The first prototypes were supposed to implement only the Phase level functionalities in order to create a basis for R3. Step and Detail levels, as well as the technical view were implemented later, during the StandAlone development, which is discussed in Chapter 11.

The first prototype was supposed to generate the code file based on the definition view. Additionally, it helped to determine the technologies to be used with the implementation of the second prototype and the StandAlone version. The second prototype continued where the first one left off. It was completing the PDD-SDD cycle, by constructing the updated SDD in the definition view, based on the modified code file. In addition to R3, the prototypes aimed to fulfill IR3 by allowing the developer to make changes to the logic via the created code file, and IR6 by generating all the Phase level function calls inside the code file's `main` function. Additionally, the prototypes were laying foundations to IR5 by linking the information between the definition view and code file with tags in the function docstrings.

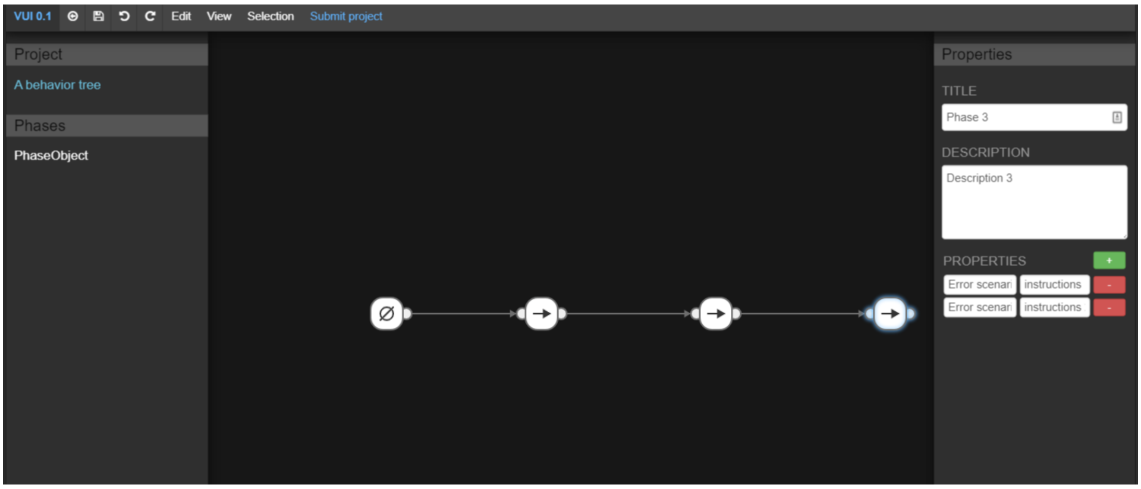
The first section of this chapter describes the whole second development iteration illustrated in the updated roadmap in Figure 6.1. The third development iteration in the roadmap was started with the second prototype, whose implementation is discussed in the second section of this chapter. The last section briefly summarizes the Core and Implementation Requirements that were satisfied by the prototypes. The results were presented to the developers in the second interviews, which were conducted prior to the start of the StandAlone development, discussed in Chapter 10.

9.1 First Prototype

The first prototype was supposed to create a tool that constructs a Phase level code logic with visual user interface and generates a code file based on it. The development also included research on technologies that should be used with the PDD-SDD Tool's implementation. Additionally, the first prototype was planned to fulfill IR6, by creating the code file with all the Phase level functions placed inside the `main` function, in execution order.

A Python web framework Django[1] was selected to provide the back-end for the PDD-SDD Tool, while the front-end implementation was chosen to be built upon the Behavior3 application[2], an open source project created with the JavaScript MVC framework AngularJS[3]. [36, 37]. The Behavior3 application had already implemented an engine to visualize behavior trees, which was considered to be a good starting point for the first prototype and the technical view of the upcoming StandAlone version of the tool [38].

The back-end was created and linked with the modified Behavior3 project with the Django REST Framework[4]. The first prototype did not include any new views but merely refactored the behavior of the original Behavior3 editor, in order to present the primitive definition view. The definition leaned on the node structure, executed from left to right, where the user could create and define Phase nodes by giving them a name and description. The definition view is illustrated in Figure 9.1.



TITLE: Phase 3

DESCRIPTION: Description 3

Error scenari | instructions

Error scenari | instructions

Figure 9.1: The primitive definition view of the first prototype.

---

[1] https://www.djangoproject.com/, 14.5.2020

[2] https://www.behaviortrees.com/, 14.5.2020

[3] https://angularjs.org/, 14.5.2020

[4] https://www.django-rest-framework.org, 21.5.2020

After the user was ready, the project could be submitted to be processed in the back-end where the code file was generated based on the defined node structure. The submit functionality is depicted in Figure 9.2, while the generated code file is represented in Listing 9.1.

The implementation of the first prototype was successful. It also fulfilled IR6 by constructing the `main` function in a way that included the Phase function calls in execution order. No new obstacles were encountered and the second prototype development was started without delays.

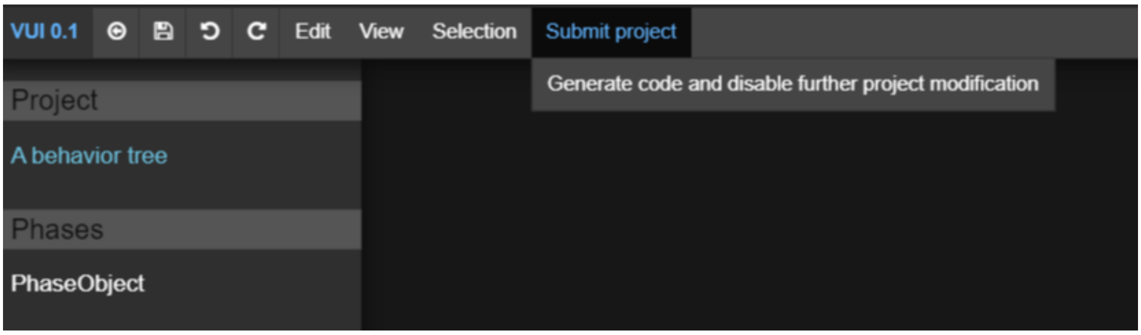


Figure 9.2: Submitting the defined project in the first prototype's definition view.

```
def phase_1():
    '''
    Node name: Sequence
    Node description:
    Description 1
    Node parameteres: {}
    '''
    pass

def phase_2():
    '''
    Node name: Sequence
    Node description:
    Description 2
    Node parameteres: {}
    '''
    pass

def phase_3():
    '''
    Node name: Sequence
    Node description:
    Description 3
    Node parameteres: {'Error scenario 1': 'instructions 1', 'Error scenario 2': 'instructions 2'}
    '''
    pass

def main():

    phase_1()
    phase_2()
    phase_3()
```

Listing 9.1: Generated code file based on the definition view's node structure in the first prototype.

9.2 Second Prototype

The second prototype was planned to implement the full cycle of PDD-SDD automation. The generated and modified code file should be processed and visualized again in the front-end. This feature completes the basis for R3, which aims to create automatic documentation. Furthermore, the second prototype was supposed to allow the developer to change the definition view's business logic via the code file as described in IR3, in Table 7.12.

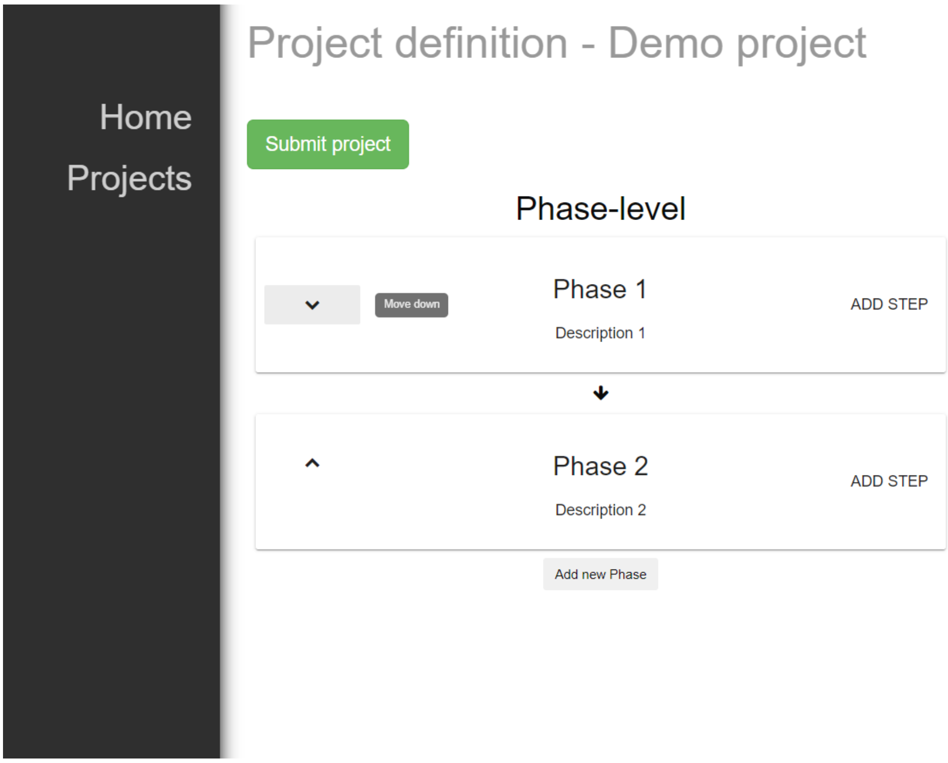
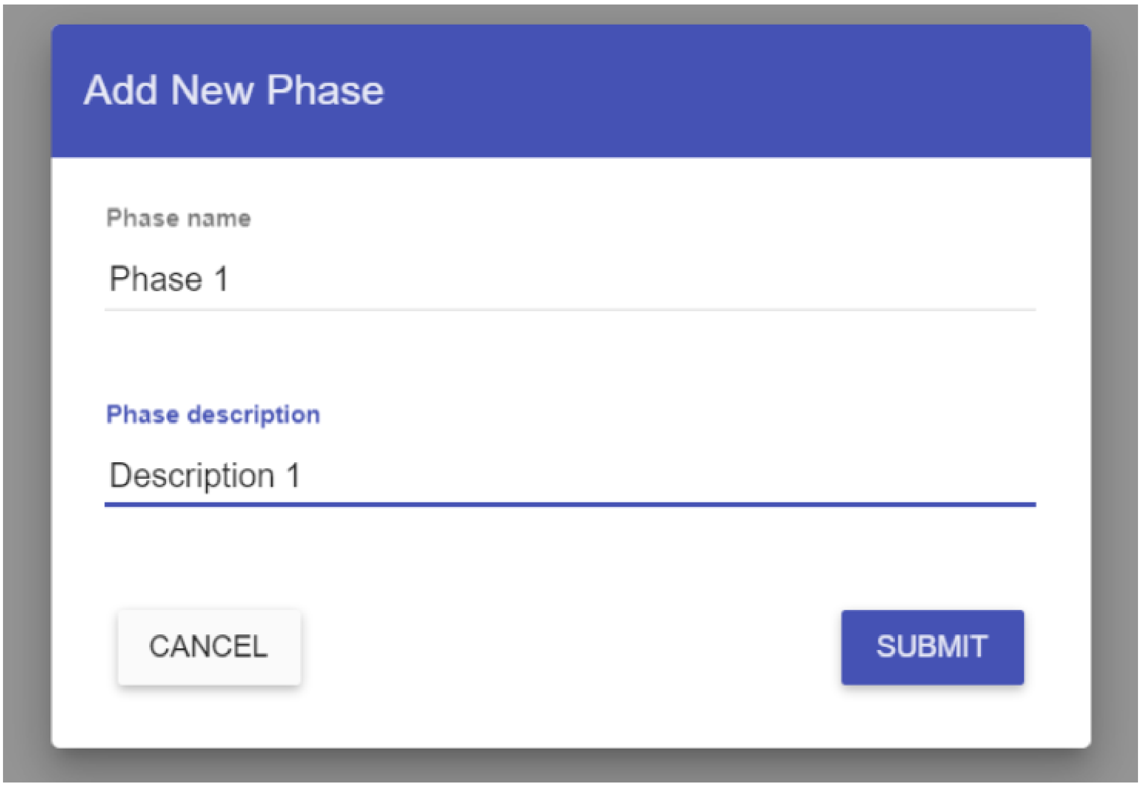


Figure 9.3: The new definition view in the second prototype.

The second prototype implemented a new definition view, which is illustrated in Figure 9.3. It was constructed based on the initial design presented in Figure 5.1, and provided the functionalities to rearrange the order of the Phases as well as to create new Phases via pop up dialog, as depicted in Figure 9.4. A placeholder button for adding Steps was also added to the Phase level elements for demonstration purposes.



Add New Phase

Phase name: Phase 1

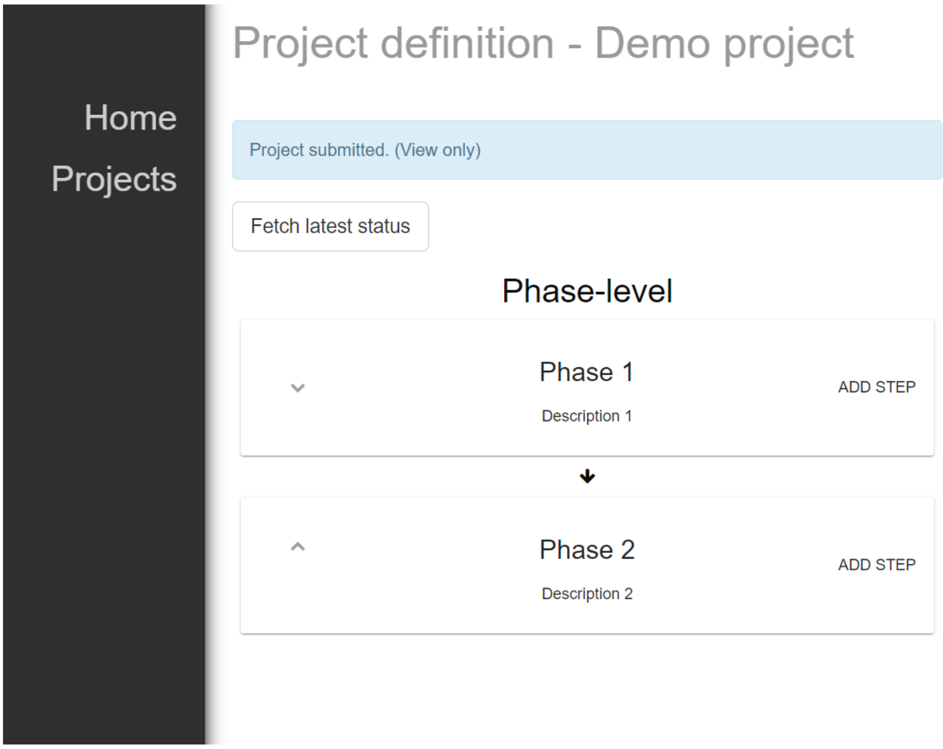
Phase description: Description 1

Figure 9.4: A new Phase's popup dialog in the second prototype's definition view.

After the defined project was submitted, the code file was generated in the back-end in the same manner as in the first prototype, illustrated in Listing 9.2. The generated code file contained the placeholder functions similar to the first prototype, but with pruned docstring content. The parameters included in the first prototype were decided to be implemented in the Detail level, in the upcoming StandAlone version of the tool. Additionally, further modification of the process definition was prevented in the definition view, as depicted in Figure 9.5.

```
def phase_1():
    """
    Description: Description 1

    @PHASE
    """
    pass

def phase_1():
    """
    Description: Description 2

    @PHASE
    """
    pass

def main():

    phase_1()
    phase_2()
```

Listing 9.2: Generated code file based on the definition view's elements in the second prototype.



Project definition - Demo project

Project submitted. (View only)

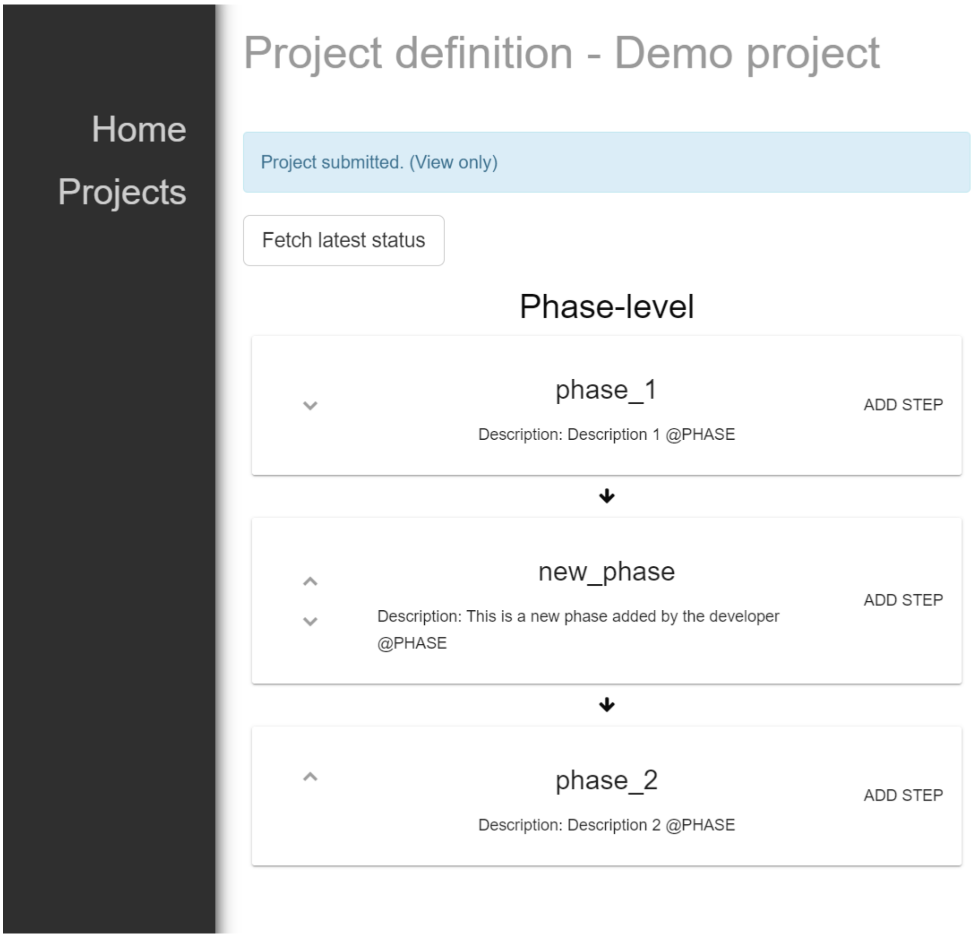
Figure 9.5: The definition view of the second prototype after submit.

After a submit, the updated code logic could be fetched from the file by a ’Fetch latest status’ button illustrated in Figure 9.5. After triggering this command, the back-end processed the code file and created a new definition view based on it. An example of a modified code file is provided in Listing 9.3, while the corresponding updated definition view is depicted in Figure 9.6.

```
def phase_1():
    """
    Description: Description 1

    @PHASE
    """
    pass

def phase_1():
    """
    Description: Description 1

    @PHASE
    """
    pass

def new_phase():
    """
    Description: This is a new phase added by the developer

    @PHASE
    """
    pass

def main():

    phase_1()
    new_phase()
    phase_2()
```

Listing 9.3: Modified code file in the second prototype.

The second prototype was implemented succesfully. The updated definition view contained the Phase level elements in the same order as in the modified code file, therefore fulfilling the basis for R3, as well as IR3.



Project definition - Demo project

Project submitted. (View only)

Phase-level

phase_1

Description: Description 1 @PHASE

new_phase

Description: This is a new phase added by the developer @PHASE

phase_2

Description: Description 2 @PHASE

Figure 9.6: The updated definition view based on the modified code file in the second prototype.

9.3 Fulfilled Core and Implementation Requirements

Both of the prototypes achieved what they were supposed to. They proved implementation of automatic documentation in R3 to be possible. They also fulfilled IR3 and IR6, as the developer was able to change the code structure of the generated code file, which contained the `main` function with the Phase level function calls in executable order.

However, the link between the code level function names and the Phase titles in the definition view was not yet separated, therefore leaving IR5 to be fulfilled in the upcoming development of the StandAlone version. Similarly, the primitive definition view used in the first prototype was left to form the base for the technical view. This way the views could be separated, as required in IR1.

10 Second Interviews

The second interviews were conducted in March 2020 in order to discuss the current progress of PDD-SDD Tool's development. The results from the second interviews helped to confirm that the development was meeting the interviewees' expectations and to potentially adjust the upcoming development of the the StandAlone version, discussed in Chapter 11. Additionally the second interviews were the last part of the third development iteration depicted in the project's roadmap in Table 6.1.

The prototype development was based on the Core Requirements, listed in Table 8.8, and the Implementation Requirements, listed in Table 7.12. The purpose of the second interviews was to make sure that the development had proceeded in the right direction and to find out possible solutions for the still unanswered Implementation Challenges, listed in Table 8.9

The second interviews resulted in Prototype Impressions, prototype specific Implementation Challenges and Implementation Proposals. The results are discussed in the last part of this chapter in order to recognize the fulfilled Core and Implementation Requirements, and to identify potential new Implementation Challenges, which could alter the design of the StandAlone version of the PDD-SDD Tool. Finally, the Implementation Proposals from the first interviews are combined with the prototype specific Implementation Proposals to create a first version of the Further Development Proposals, which are considered to guide the possible development after this thesis.

10.1 Thoughts on Second Prototype

The presentation of the second prototype resulted with Prototype Impressions (ProI1–ProI2), listed in Table 10.1, the prototype specific Implementation Challenges (ProC1–ProC2), listed in Table 10.2, and Implementation Proposals (ProP1–ProP8), listed in Table 10.3. The results are further discussed in Section 10.2 to address the satisfied Core and Implementation Requirements, as well as the unsolved Implementation Challenges.

Prototype Impression	**Description**
ProI1	The tool seemed intuitive and simple to use
ProI2	The Phase View provides and easy way to see the overall flow of the process

Table 10.1: Prototype Impressions from the second interviews.

The reception of the second prototype was positive. The tool's usage was considered to be intuitive and simple (ProI1). Interviewees also thought that the implemented definition view provided an easy way to see the overall flow of the process (ProI2).

Some interviewees were concerned that the instructions written in the definition view could impact the developer's technical decisions too much (ProC1). The interviewees agreed that the possibly insufficient definition should not affect the generated code. A proper instructions of the tool's usage should ensure that the definition view focuses only to the business logic and that the definer would not try to suggest solutions for the techni-

Implementation Challenge	Description
ProC1	Insufficient instructions in the definition view could impact the developer's technical decisions too much
ProC2	Wrong code logic from the code file emerging in the definition view

Table 10.2: Prototype specific Implementation Challenges identified in the second interviews.

cal implementation (ProP1).

Interviewees pointed out that even though the automated creation of SDD was a good feature, it possessed a risk of wrong code logic emerging in the definition view (ProC2). This feature was also seen essential to be instructed in a clear manner to the users of the tool (ProP2).

The instructions for the PDD-SDD Tool should also guide the users to protect the initial definition created in the definition view (ProP3). It was considered essential that the developer must protect the generated definition in the code level comments and modify them only if especially instructed to do so. Additionally, it was proposed that additional images could be added to the other levels to provide additional information (ProP4).

Another feature suggested by the interviewees was to provide a way for the customer to request changes to the developed code and previously accepted business logic (ProP5). It was also proposed that the definition view could include a similar way for the developer to require the customer's acceptance for the updated view (ProP6). This could be especially useful in the later stages of the development, after the initial definition.

In addition to the definition view, the implementation of the technical view was also dis-

Implementation Proposal	**Description**
ProP1	A proper instructions for the definition view should be provided
ProP2	A proper instructions for the code file tags should be provided
ProP3	Clear rules for modifying the definition in the code file tags should be provided
ProP4	Additional images could be added to the Phase and the Step levels to provide additional information
ProP5	The definition view could provide a way for the customer to request changes to the previously accepted business logic
ProP6	The definition view could provide a way for the developer to ask the customer's acceptance for the updated view
ProP7	The Technical view should visualize the imported code logic as well
ProP8	There could be a need to modify information also in the technical view

Table 10.3: Prototype specific Implementation Proposals from the second interviews.

cussed during the interview. A good project structure was considered to contain several files and folders. From this viewpoint, the tool's technical view should also be able to visualize the imported code logic as well (ProP7). Potential needs to modify information in the technical view were also discussed (ProP8).

10.2 Results

The results from the second interviews had no significant impact on the development of the StandAlone version of the tool. In this section, the Prototype Impressions, the prototype specific Implementation Challenges and Implementation Proposals are discussed further and compared to the previously listed Implementation Requirements and unsolved Implementation Challenges.

The first part of this section uses the Prototype Impressions to recognize the satisfied Implementation Requirements and to address the unsolved Implementation Challenges. The Implementation Requirements are also used to solve the prototype specific Implementation Challenges with the Core Requirements.

The last part of this section compares the Proposals from the both Interviews conducted, in order to form a list of Further Feature Proposals, which are considered to be implemented after this thesis.

This section completes the third development iteration illustrated in the project's roadmap in Figure 6.1. The results from the second interviews lay foundation to the fourth and final development iteration that starts with the StandAlone development, discussed in Chapter 11.

10.2.1 Impact on Tool's StandAlone Version

The Prototype Impressions (ProI1 and ProI2) are used to recognize both the satisfied Core Requirements (R1–R4) in Table 8.8 and the satisfied Implementation Requirements (IR1–IR8) in Table 7.12. Furthermore, the Prototype Impressions are used to address the unsolved Implementation Challenges (IC4, IC9 and IC10) in Table 8.10. An Implementation Requirement is considered to be satisfied if an Impression states that the desired result is achieved. Similarly, an unsolved Challenge can be solved by an Impression's statements. Finally, at the end of this subsection the Implementation Requirements are used to solve the prototype specific Implementation Challenges (ProC1–ProC2) with the Core Requirements (R1–R4) from Table 8.8.

Comparing Prototype Impressions to Core and Implementation Requirements

ProI2 states that the Phase level function calls listed in the `main` function in the code file were seen to be informative, thus satisfying IR6. The progress towards automatic documentation in R3 had also advanced significantly, but the implementation still lacked the remaining functionalities of the PDD-SDD Tool's design. Therefore R3 remained unsatisfied. The other Core and Implementation Requirements were also left to be addressed by the StandAlone version, which was expected because the prototypes were only laying the foundation for it.

Comparing Prototype Impressions to Unsolved Implementation Challenges

IC9 raised the issue that it is very difficult to create a tool that actually saves the developer's time. The way in which the second prototype generated the initial code file from the definition view was considered to be useful, easing the developer's work at the beginning of a new project. ProI1 described the tool to be intuitive, which was also considered to imply that the tool's development was proceeding in the right direction. However, all the unsolved Implementation Challenges still needed further inspecting after the StandAlone development.

Comparing Prototype Specific Implementation Challenges to Core and Implementation Requirements

Both of the prototype specific Implementation Challenges were solved with the Core Requirements and the Implementation Requirements, therefore they had no effect on the development of the StandAlone version.

R2 would minimize the risk for ProC1 by providing accessible and unified documentation, therefore no changes for the ongoing development plans were made. This should be achieved when the developer has full control over the code files, as described in IR3.

As required in IR5, the definition should be separated from the executable code logic, therefore Phase or Step level functions should emerge in the definition view only if they are especially tagged by the developer. With the proper instructions about the tool's usage, this should solve the ProC2.

10.2.2 First Version of Further Development Proposals

Implementation Proposals for the tool's further development (IP1–IP6) were already introduced in the first interviews, listed in Table 7.13. Additionally, there was 8 new Implementation Proposals gathered from the second interviews (ProP1–ProP8), listed in Table 10.3. The new Proposals were either new or aligning with the previous ones.

In this section all of the identified Proposals are combined and listed as the Further Development Proposals (p1–p11), in Table 10.4, divided between the definition view, the code file and the technical view. The Implementation Proposals are considered to be useful for the PDD-SDD Tool, but not relevant for the core functionalities, therefore they should be implemented after this thesis, when the StandAlone version is successfully completed and validated.

ProP1, Prop2 and ProP3 all discuss the importance of proper instructions about the PDD-SDD Tool's usage. These are combined to cover all aspects of the tool and listed as p1.

There are 4 Proposals that are solely considering the definition view. ProP4 is transferred directly to p2. IP4 was combined with ProP5 and listed as p3, discussing a way for the customer to request changes to the previously accepted definition view. Additionally, p4 is cloned from ProP6 and p5 is identical with IP3, which describes the need to include project's version history into the definition view.

Providing links to external documentation and other sources of information described in p6 is related to the definition view, the code file and also to the technical view. The Proposal is taken from IP1 from the first interviews. Additionally, p7 suggests that the generated code should follow the company's best styling conventions, as described in IP2.

The last 4 Proposals consider only the technical view. The visualization of the imported code logic, discussed in ProP7 is directly listed as p8 and the possible need to modify information also in the technical view links p9 to ProP8. IP5 suggests that the images used in visual recognition could be included in the technical view, listed as p10. The final p11 is taken from IP6, which considers the technical view being a good educational tool for improving the technical mindset among different stakeholders.

The Further Development Proposals are going to be addressed again after the final interview, when the StandAlone version of the PDD-SDD Tool is presented and validated. The possible new Implementation Proposals from the final interview are compared and added to the ones discussed in this section.

Proposal	**Description**	**Def View**	**Code File**	**Tech View**
p1	Proper instructions should be provided to ensure that the tool provides the best possible value	x	x	x
p2	Additional images could be added to the Phase and the Step levels to provide additional information	x		
p3	The definition view could provide a way for the customer to request changes to the previously accepted business logic	x		
p4	The definition view could provide a way for the developer to ask the customer's acceptance for the updated view	x		
p5	The definition view could allow browsing of the project's version history	x		
p6	The tool could provide links to other sources of material and documentation	x	x	x
p7	The tool could generate code that follows the company's best styling practices		x	
p8	The technical view should visualize the imported code logic as well			x
p9	There could be a need to modify information also in the technical view			x
p10	Images used in visual recognition could be included in the technical view			x
p11	The technical view could be used as an educational tool for the project manager and the customer to get familiar with the project in more technical manner			x

Table 10.4: Further Development Proposals after the second interviews.

11 StandAlone Development

The StandAlone version of the PDD-SDD Tool was developed after the second interviews had been conducted. The StandAlone was supposed to implement all the features in the initial design depicted in Figure 5.1 and fulfill the Core Requirements in Table 8.8 and the Implementation Requirements in Table 7.12. Furthermore, the unsolved Implementation Challenges in Table 8.10 were supposed to be answered based on the StandAlone version in the final interview, which is discussed in Chapter 12.

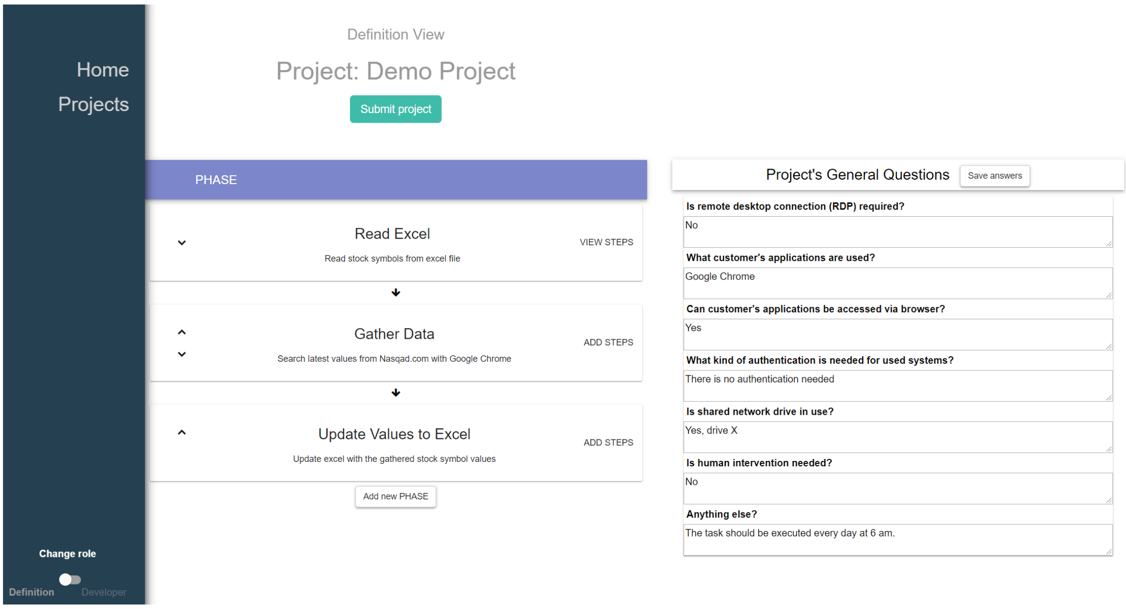
The StandAlone development is the first part of the final development iteration depicted in the project's roadmap in Figure 6.1. The first part of this chapter discusses the actual implementation of the StandAlone version, while the second part addresses the Core Challenges and the Implementation Challenges which can be declared to be fulfilled by the developed tool. The remaining Core and Implementation Challenges, are addressed via the Impressions from the final interviews, in Chapter 13.

11.1 Implementation

The StandAlone version contains the definition and the technical views. These are demonstrated with the generated and updated code file in four subsections that are divided between the definition view, the generated code file, the updated definition view and the technical view.

11.1.1 Definition View

The first visible addition compared to the second prototype were the general questions about the project, illustrated in Figure 11.1. The definition view has been splitted vertically, the left side contains the Phase level elements and the right side provides the general questions for the user. The questions were supposed to fulfill R1 from the Core Requirements, listed in table 8.8.



Definition View

Project: Demo Project

Read Excel
Read stock symbols from excel file

Gather Data
Search latest values from Nasqad.com with Google Chrome

Update Values to Excel
Update excel with the gathered stock symbol values

Project's General Questions

Is remote desktop connection (RDP) required?
No

What customer's applications are used?
Google Chrome

Can customer's applications be accessed via browser?
Yes

What kind of authentication is needed for used systems?
There is no authentication needed

Is shared network drive in use?
Yes, drive X

Is human intervention needed?
No

Anything else?
The task should be executed every day at 6 am.

Figure 11.1: The definition view of the StandAlone version including Phase level elements at the left side and the general question at the right side.

The StandAlone version also reinforced the Phase level with the Step and Detail levels, depicted in Figures 11.2 and 11.3. The Step level was similar to the Phase level, whereas a new layout was implemented for the Detail level. The creation, modification and deletion functionalities were also added to the Phase, Step and Detail elements. Additionally, the whole project can be deleted from the projects tab, which is illustrated later in this chapter.

The detail level includes the place for instructive image to be uploaded by the user, the

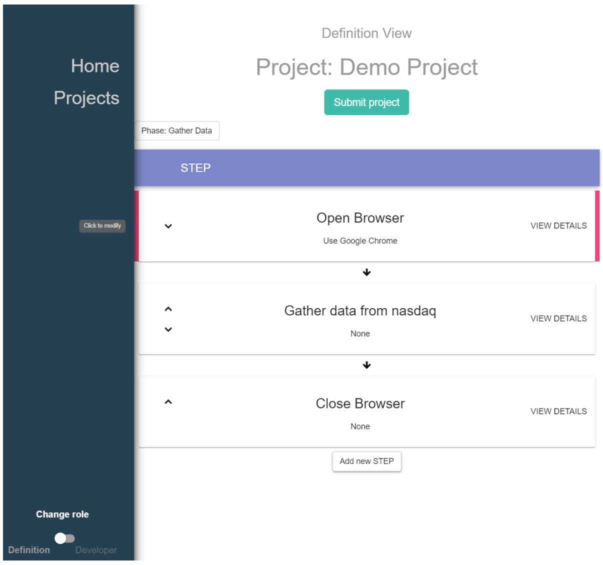
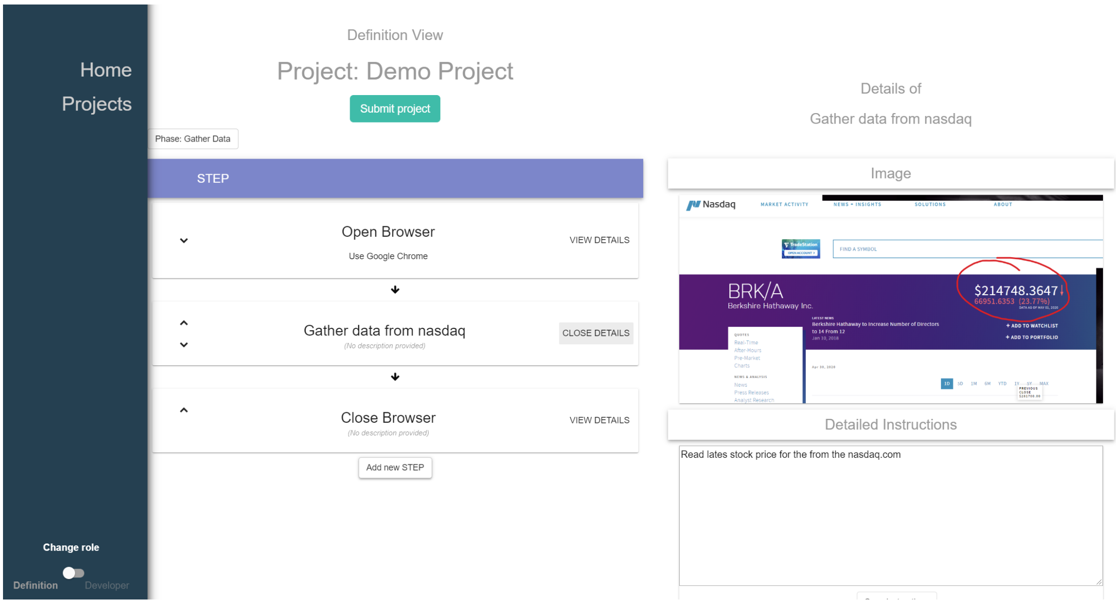


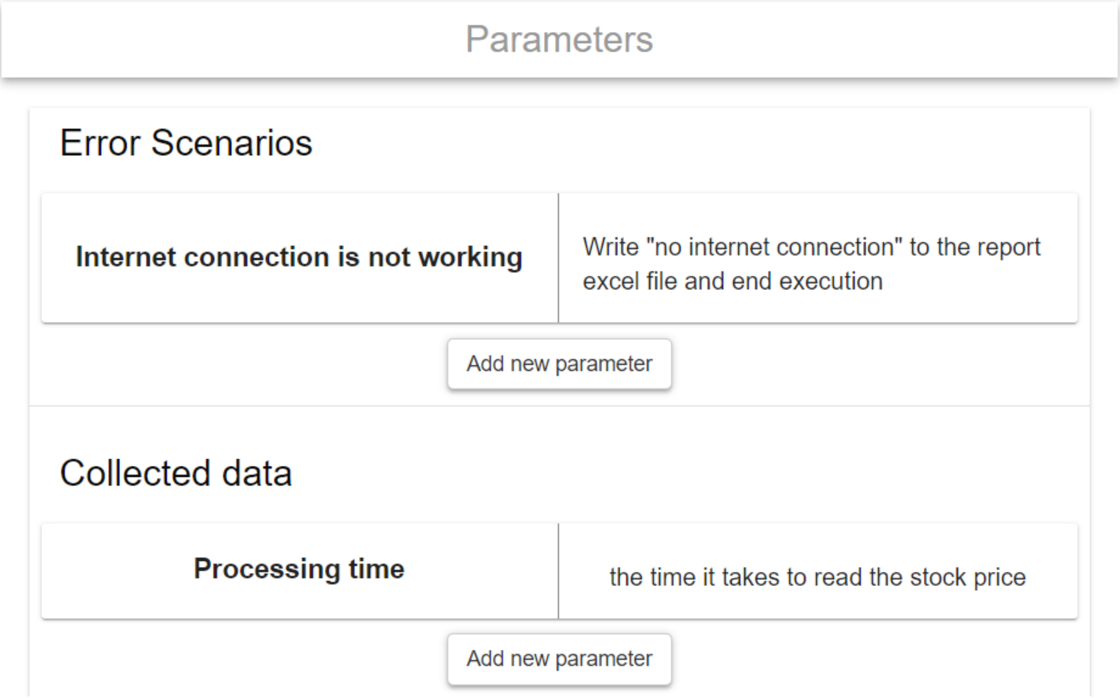
Figure 11.2: The Step level of the opened Phase "Gather Data". The right side of the view remains blank if no Step is selected.



Read lates stock price for the from the nasdaq.com

Figure 11.3: The Detail level of the opened Step "Gather data from nasdaq". The Detail level contains image, detailed instructions and parameters. The parameters are located below the detailed instructions, and are not visible in this screenshot.

detailed instructions about the selected Step element and possible error and data collation parameters. The image container, shown at the right side of the Figure 11.3 can be enlarged for more detailed inspection when clicked. The detailed instructions beneath the image can be freely edited by the user, given that the project is not yet submitted. The parameters of the Detail level are located under the detailed instructions, which can be accessed when the view is scrolled down. Parameters are depicted in the Figure 11.4.



Error Scenarios	
Internet connection is not working	Write "no internet connection" to the report excel file and end execution

Collected data	
Processing time	the time it takes to read the stock price

Figure 11.4: Error scenarios and data collation parameters under the detailed instructions in the Detail level.

11.1.2 Generated Code File

When the project is submitted by clicking the submit button under the project header in the definition view, the back-end starts to process the code file content in similar manner than in the second prototype. The generated code file, illustrated in Listing 11.1, includes both the Phase and the Step level functions. Furthermore, the Step level functions include tagged Detail level information in the docstring comment.

```python
def update_values_to_excel():
    """
    @TITLE: Update Values to Excel

    @DESCRIPTION:
    Update excel with the gathered stock symbol values

    @PHASE
    """
    update_excel()


def update_excel():
    """
    @TITLE: Update Excel

    @DESCRIPTION:
    None

    @INSTRUCTIONS:
    Update excel with gathered information

    @PICTURE:
    None

    @ERRORS:
    ----

    @DATA_COLLATION:
    ----

    @STEP
    """
    pass

def main():

    read_excel()
    gather_data()
    update_values_to_excel()
```

Listing 11.1: Part of the code file generated from the definition view illustrated in Figures 11.3 and 11.4. The `main` function contains function calls to the Phase level functions, which in turn call the Step level functions. Step level functions contain the data for the Detail level as well.

The Phase function `update_values_to_excel()` in Listing 11.1 contains three types of tags that are tied to the definition view. Text after the tag `@TITLE` is the visible title shown to the user in the definition view, separated from the code level function name. Similarly, `@DESCRIPTION` binds the defined description to the code level, and the final tag `@PHASE` specifies the type of the function.

The Step and Detail level information is bound to the functions in a similar manner, illustrated in Listing 11.2. In addition to the tags also included in the Phase functions, a Step function contains `@INSTRUCTIONS`, `@PICTURE`, `@ERRORS` and `@DATA_COLLATION` tags, therefore covering all the information needed in the definition view's Step and Detail levels.

```
def gather_data_from_nasdaq():
    """
    @TITLE: Gather data from nasdaq

    @DESCRIPTION:
    None

    @INSTRUCTIONS:
    Read lates stock price for the from the nasdaq.com

    @PICTURE:
    http://localhost:8000/pdd-sdd-rest/media/images/temp_T13Anw2.png

    @ERRORS:
    ----
    description: Internet connection is not working
    instructions: Write "no internet connection" to the report excel file and end execution
    ----

    @DATA_COLLATION:
    ----
    description: Processing time
    instructions: the time it takes to read the stock price
    ----

    @STEP
    """
    pass
```

Listing 11.2: The Step function `gather_data_from_nasdaq()` *generated from the definition view elements illustrated in Figures 11.3 and 11.4.*

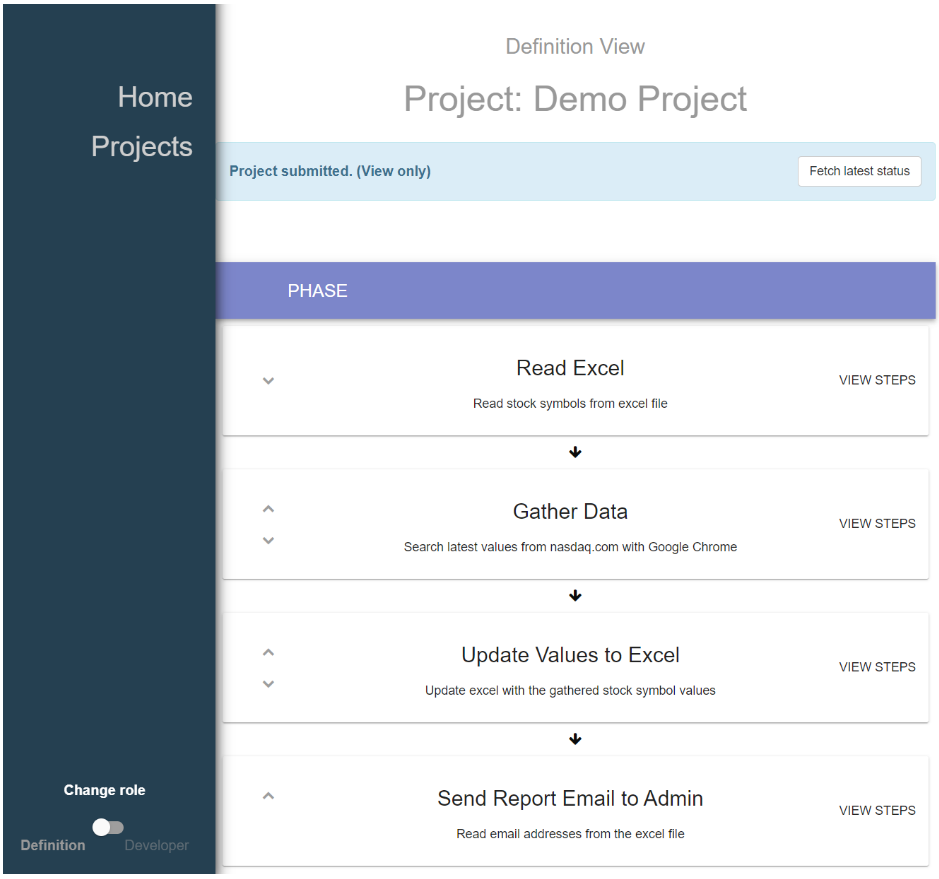
11.1.3 Updated Definition View

The modified code logic can be fetched in a similar manner than in the second prototype. The definition view is updated based on the processed code file, visualizing the Phase, Step and Detail level elements in the execution order. The business logic visualized in the definition view cannot be modified in the tool after it has been submitted and the code file has been generated. After submit, only the developer has an ability to make updates to the business logic via the code file, based on the customer's wishes. This ensures that updates do not break already programmed logic. An example of an modified code file is illustrated in Listing 11.3. The updated Phase level can be seen in Figure 11.5 and the updated Step and Detail levels are depicted in Figure 11.6.

The developer programs the software robot based on the provided business logic, which is fully modifiable in the generated code file. This ensures that the PDD transforms toward SDD every time the developer updates text associated with the tags in the docstring comments, as illustrated in Listing 11.3. The developer can also add new Phase, Step and Detail elements to the generated code, which are processed and visualized in the definition view in execution order. The definition view is only for business logic, therefore it does not include logical structures of code logic.

```
def construct_report_excel():
    """
    @TITLE: Construct Report Excel
    @STEP
    """
    pass


def send_report_email(error=False):
    """
    @TITLE: Send Report Email
    @INSTRUCTIONS: Report Emails can be found from the Excel file
    @STEP
    """
    pass


def report():
    """
    @TITLE: Send Report Email to Admin
    @DESCRIPTION: Read email addresses from the excel file
    @PHASE
    """
    try:
        construct_report_excel()
    except:
        error_msg = "constructing the report excel failed!"
        send_error_email(error_msg)
        raise

    recipient_list = []
    for recipient in recipient_list:
        send_report_email()


def main():
    stock_symbol_list = read_excel()
    result_list = []
    if(stock_symbol_list):
        result_list = gather_data(stock_symbol_list)
        update_values_to_excel(result_list)
    report(result_list)
```

Listing 11.3: Part of a modified code file with new functions and logical structure. A new Phase level function `report` *and new Step level functions* `construct_report_excel` *and* `send_report_email` *were added. The* `@TITLE` *tag in* `report` *defines the title visualized in the definition view in Figure 11.5*



Project submitted. (View only)

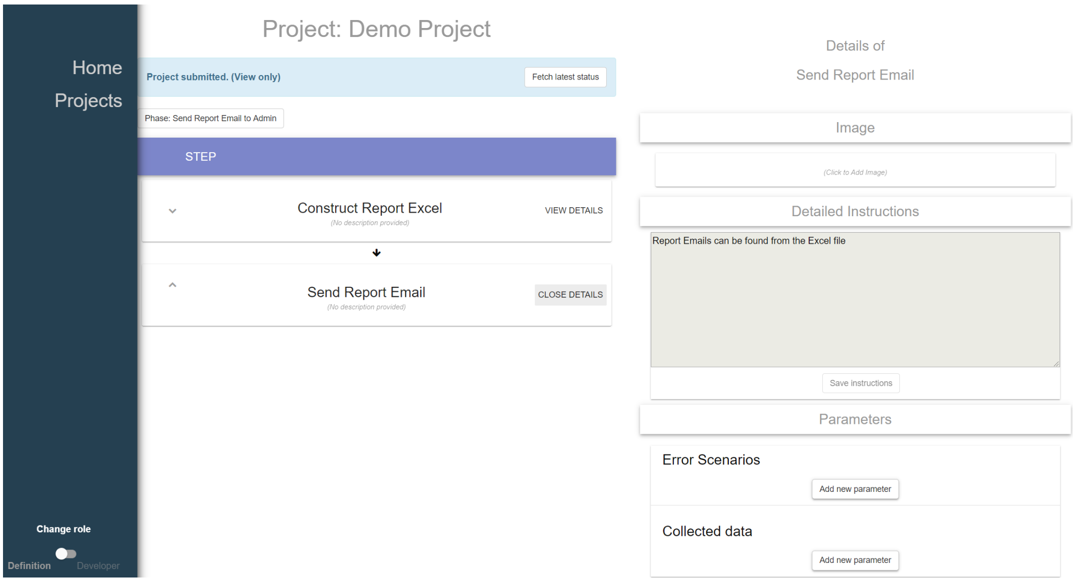
Read Excel
Read stock symbols from excel file

Gather Data
Search latest values from nasdaq.com with Google Chrome

Update Values to Excel
Update excel with the gathered stock symbol values

Send Report Email to Admin
Read email addresses from the excel file

Figure 11.5: Updated Phase level view based on the modified code file in Listing 11.3. A new Phase level element for the "Send Report Email to Admin" was added.



Project submitted. (View only)

Construct Report Excel
(No description provided)

Send Report Email
(No description provided)

Report Emails can be found from the Excel file

Figure 11.6: Updated Step and Detail level views based on the modified code file in Listing 11.3.

11.1.4 Technical View

The technical view was also implemented in the StandAlone. The tool added a switch in the user interface in order to change the role from the definition to the developer, depicted in Figure 11.7. When the developer role is active, the user has an option to access both the definition view and the technical view from the projects tab, illustrated in Figure 11.8.

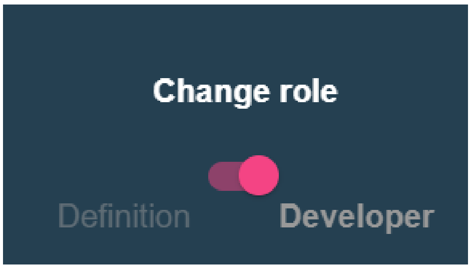


Figure 11.7: Role switch in the StandAlone user interface.

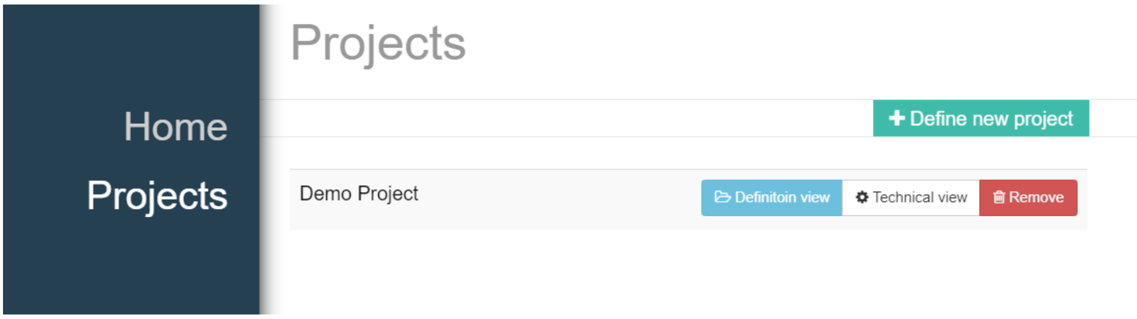
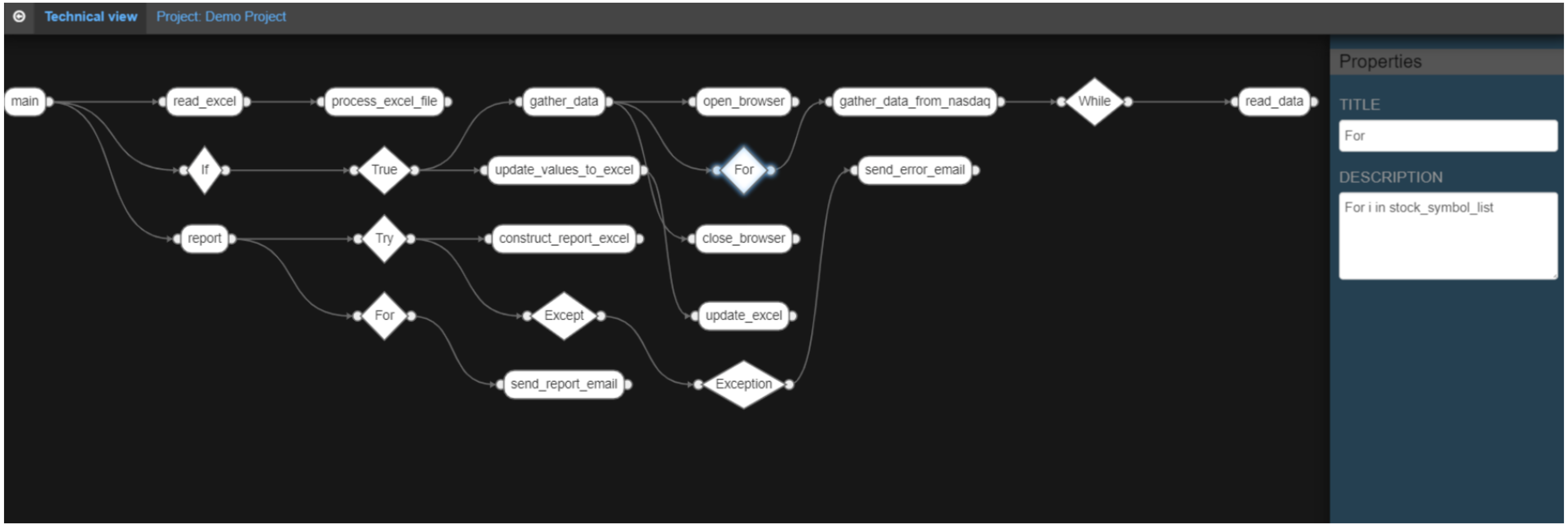


Figure 11.8: User can access the technical view from the Projects tab if the developer role is activated. Otherwise, the option is hidden.

The technical view was constructed based on the original Behavior3's editor view, previously discussed during the first prototype development in Chapter 9. In order to achieve the wanted level of technical detail, the logical structures were read from the code file in execution order and the corresponding node structure was created. The technical view, based on the modified code from Listing 11.3, is illustrated in Figure 11.9.



TITLE
For

DESCRIPTION
For i in stock_symbol_list

Figure 11.9: The technical view with the function names and the logical structure based on the modified code file in Listing 11.3. The selected node's detailed content is visualized in the right side of the view.

The technical view visualizes every function and logical structure that is executed in the code level, which include also the functions containing the definition view's business logic. In addition to the functions, the provided Figure 11.9 contains logical nodes for `if`, `try-except`, `for` and `while` statements, which are written in the modified code, partly depicted in Listing 11.3. These don't cover all possible structure types available, but illustrate some of the most common ones.

The StandAlone version's technical view is capable of visualizing any python code contained in a single file, executed from `main` function. This means that the feature is not limited only to the RPA projects but could be used to other kinds of software projects as well. The technical view is updated every time that the project status is fetched via the definition view.

11.2 Fulfilled Requirements

Most of the Core Requirements and the Implementation Requirements were fulfilled by the StandAlone version of the PDD-SDD Tool. However, due to the limited time resources and the need for additional feedback, the implementation of some features were postponed to be done after this thesis, during further development.

11.2.1 Core Requirements

The StandAlone fulfilled most of the Core Requirements (R1–R4) listed in Table 8.8.

The guiding questions were provided in the definition view, as required in R1. R2 stated that accessible, unified and up to date documentation should be provided. The tool achieves this, if it ends up being used with the actual RPA projects in MOST Digital.

The StandAlone managed to complete the visions of the initial design, to automate code base creation from the PDD, and reading the updated code file back to the SDD, therefore satisfying R3, which required automatic documentation.

The importance of documenting the most important technical and architectural information was recognized during the first interview. R4 states that the technical and architectural questions should be provided after the definition phase, to help further development and testing. During the StandAlone development, it was agreed that its implementation would be relatively similar to R1, but the placement for it could be in the code level or in the technical view. Because the placement needed more feedback from the interviewees, which could not be received due to the limited time resources, it was added to the Further Development Proposals, to be implemented after this thesis.

11.2.2 Implementation Requirements

Two of the Implementation Requirements, IR3 and IR6, were already satisfied by the Prototypes, as discussed in Chapter 9. The first prototype managed to include the Phase level function calls in the code file's `main` function, as required in IR6. IR3 stated that the developer should be able to change the code structure after the PDD has been accepted and code base created, which was later enabled by the second prototype. The StandAlone version managed to fulfill most of the remaining Implementation Requirements, listed in Table 7.12.

The definition view and the technical view were separated, therefore satisfying IR1. Against the interviewees original exceptions, the technical view was successfully implemented in a manner that visualized the exact logical structure of the project code, as required in IR2. The Detail level in the definition view added parameters, which included possible business error scenarios, satisfying IR4. Additionally, the division between the definition view and the technical view also resulted with separated business and code logic, which satisfies IR5.

IR7 stated that the original definition view should be available for the user to be compared with the updated view. Additionally, the updated definition view should be able to be reviewed by the customer as required in IR8. Due to the limited time resources, both of these Implementation Requirements were postponed to be implemented after this thesis. They are going to be included in the Further Development Proposals in Chapter 13.

The usability and the overall success of the StandAlone version is validated by the interviewees during the Final Interview, which is discussed in Chapter 12.

12 Final Interview

The project described in this thesis culminated with the final interviews, which were conducted on April 2020. The first part of the interview presented the developed StandAlone version of the PDD-SDD Tool to the interviewees. The second part addressed the Core Requirements in Table 4.3 and the Research Questions in Table 4.4.

This chapter describes the final interview, where the interviewees were asked to tell their Impressions about the StandAlone and if it fulfills the Core Requirements and answers to the Research Questions. The interviewees also identified new Further Development Proposals and StandAlone specific Implementation Challenges.

The resulted the Further Development Proposals (StaP1–StaP15) are listed in Table 12.1, the StandAlone Impressions (StaI1–StaI19) in Table 12.2, and the StandAlone specific Implementation Challenges (StaC1–StaC6) in Table 12.3. The results, the success of the StandAlone version of the PDD-SDD Tool, as well as the fulfillment of the both Core and Implementation Requirements and the Research Questions are addressed in Chapter 13.

12.1 Thoughts on StandAlone Version of PDD-SDD Tool

The overall impression of the StandAlone was positive among the interviewees. The tool was considered to be successfully implemented, supporting unified development practices. It was separately mentioned that the outdated documentation has been the biggest potential weakness encountered with RPA projects. The issue could be significantly eased if the PDD-SDD Tool would end up being used company-wide.

Once again, the interviewees were asked to think about possible challenges and further development proposals that could occur with the developed StandAlone version. The interviewees were also asked to try the tool during the interview. It was agreed that a proper instructions for the tool's usage should be made in order to get the best possible value out of it (StaP1). It was also noted, that a proper trial with a real RPA project should be conducted, which would help to identify needs that cannot be covered or identified with the interviews (StaP2).

12.1.1 Definition View

Interviewees agreed that the usability issues were paramount for the tool to actually be useful. The definition view was considered to be easy to comprehend and fluent to navigate (StaI1). It was seen to endorse an unified PDD structure and to provide a clear outlook of the overall flow of the defined process, even though the visual layout could be further developed (StaP3).

The general questions, which were implemented to answer the R1 from the core Requirements, were considered to be successfully implemented and that they will help with the important questions at the start of the new project (StaI2). However, it was also noted that some answers may vary depending on the Phase or Step, thus some questions should be implemented in these levels of the definition view as well (StaP4).

The Detail level was the biggest update to the definition view after the second prototype.

Interviewees thought that it managed to include the essential content and seemed easy to use (StaI3). However, sometimes one step could need more images, for example to illustrate possible error scenarios (StaP5). On the other hand, it was agreed that if one Step contains too many images, it could end up being more confusing than helpful. The acceptable amount should be decided within a company and added to its best practices.

The provided parameters in the Detail level were also considered to be good and useful (StaI4). The tool could also provide an option to add new custom parameters in addition to the error scenarios and data allocation (StaP6). Additionally, reordering the Phase and Step level elements were considered to need drag and drop features to be more user friendly (StaP7).

The interviewees agreed that updating PDD would be significantly easier with the tool (StaI5). The further possibilities of the tool and the definition view also sparked a lot of conversation. It was proposed that a new tag could be added to signal the current development status of the Phase or Step element, to be visualized in the definition view. This way the project manager and the customer would instantly see the current progress (StaP8).

12.1.2 Generated Code File

The generated code file followed the same structure that was already presented with the second prototype. The interviewees thought that the content of the generated comments was good. It linked the information between the definition view and the generated code file in a clear manner and provided an easy and fast way to read the Phase or Step instructions (StaI6).

Some interviewees told that the tool almost identically automates the starting procedures

StandAlone Proposal	Description	Def View	Code File	Tech View
StaP1	A proper instructions must be provided to ensure the tool provides the best possible value	x	x	x
StaP2	In order to continue the development in the right direction, a trial with a real RPA project is needed	x	x	x
StaP3	The visual layout could be further developed	x		x
StaP4	General questions could be separated between Phases and Steps	x		
StaP5	The Detail level in the definition view could need more images	x		
StaP6	It would be useful to allow custom parameter types in the Detail level in the definition view	x		
StaP7	Reordering the Phase and Step level elements would be easier with drag and drop features	x		
StaP8	The development status could be visualized in the definition view with an additional tag	x		
StaP9	It should be possible to hide the generated code file comments		x	
StaP10	It could be better to save the code file comments into separate files		x	
StaP11	The separated code file comments could be viewed with a custom plugin for the code editor to minimize the need to move between the files		x	
StaP12	The code file generation should result with multiple code files, if required by the company's best practices		x	
StaP13	The testing status could be visualized in the technical view functions		x	x
StaP14	The technical view could visualize the test coverage			x
StaP15	The technical view could provide more information about the code without the need to separately click its elements			x

Table 12.1: StandAlone Proposals from the final interviews.

for a new project that they have done (StaI7). The way that the main placeholder functions are generated was seen to significantly speed up the start of the development. Also the need to switch between the PDD and the code file decreases, when the instructions are already written in the function comments.

Even though the value of the generated content was recognized, the high amount of generated text was considered to be a problem (StaC1). It was agreed that the content should be at least possible to hide, or it would be annoying during the coding process (StaP9). Many of the code editors have a functionality that can collapse code elements, and that could be used to help with the issue.

Another solution would be to generate separate files for the comments, as discussed during the previous interviews as well (StaP10). The separate files could be bound with code functions that would contain only one tag, minimizing the needed content. Additionally, the content from the separated files could be visualized with a customized code editor plugin, for example with a hover dialog, to make its examination more fluent (StaP11). The amount of generated definition comments was also considered to differ between tasks. Smaller projects should not have so many Phases and Steps defined in the definition view, therefore the generated comments would be less dominant.

The interviewees also recognized possible challenges in the generated code file. There was a change that the developer would accidentally misuse the code level comments and affect the business logic in the definition view (StaC2). It should also be ensured that the generated comments would not conflict with any additional development tools used with Python (StaC3).

The project structure was considered to need updates as well. Project generation should be in line with the company's best practices, and possibly separate the Phases to different modules, which could be imported to the main code file (StaP12). It was agreed that the issue will need further discussion. In order to generate more code files, the tool should be able to read them as well, during the PDD-SDD update process.

12.1.3 Technical View

The technical view was received with positivity, even though it confused the interviewees before the proper instructions to interpret it were provided (StaC4, StaP1). The visual presentation of the code functions and logical structures were considered to be useful. The interviewees agreed that the technical view will help to get familiar with an already existing project (StaI8).

In addition to provide a way to get acquainted with an existing project, the technical view was seen to help with test planning as well (StaI9). In order to add even more value to the view, additional features were proposed. The testing status could be visualized in the technical view elements with the current test coverage (StaP13, StaP14). Additionally, the usability would improve if the view could provide more information without clicking the elements (StaP15).

12.2 Comparing Tool to Core Requirements and Research Questions

At the end of the final interviews, the Core Requirements (R1–R4), listed in Table 8.8, and the Research Questions (Q2 and Q3), listed in Table 4.4, were presented and discussed with the interviewees, to determine if the StandAlone managed to fulfill its goals.

StandAlone Impression	**Description**	**Def View**	**Code File**	**Tech View**	**Requirements**	**Q2**	**Q3**
StaI1	The definition view is easy to comprehend and use	x					
StaI2	The provided questions are implemented successfully	x			x		
StaI3	The Detail level in the definition view is useful	x				x	
StaI4	The provided parameters are useful	x	x				
StaI5	Updating the PDD is significantly easier with the tool	x	x				x
StaI6	The content of the generated code file is good and provides a quick way to get acquainted with the business logic		x			x	x
StaI7	The generation of the code file already automates some procedures previously done by the developers		x			x	
StaI8	The content of the technical view is good and provides a quick way to get acquainted with the code logic			x		x	x
StaI9	The technical view helps test planning by providing the overall view of the process			x		x	
StaI10	In order to fulfill R2 the tool should be taken into company-wide use				x		
StaI11	The automated content in both the Details View and the technical view fulfill R3				x		x
StaI12	The tool would ease development and testing of at least the smaller projects					x	
StaI13	The tool would help to enhance better development practices					x	x
StaI14	It is useful when most of the project information is available from one place					x	x
StaI15	The tool saves the developer's time					x	x
StaI16	The tool has potential to replace the old PDD practices					x	x
StaI17	The communication between different stakeholders would be eased by the tool						x
StaI18	The creation of documentation is significantly easier with the tool						x
StaI19	The automated documentation reminds the developer to complete the technical documentation as well						x

Table 12.2: StandAlone Impressions from the final interviews.

12.2.1 Core Requirements

All of the Core Requirements were discussed with the interviewees. The implementation of R4 was postponed as discussed in the Chapter 11. The fulfillment of the Implementation Requirements are discussed in Chapter 13 based on the results of the final interviews.

R1: Providing guiding questions during the definition phase

Interviewees agreed that R1 was fulfilled completely (StaI2).

R2: Providing accessible, unified and up to date documentation

R2 was considered to be partially fulfilled. The complete fulfillment would require that the PDD-SDD Tool is taken into company-wide use and added to the company's best practices (StaI10).

R3: Providing automatic documentation

Automatic creation of the SDD was considered being very useful feature. Automatic documentation in the technical view needs further development, but the needed data is accessible and therefore it has fulfilled the R3 (StaI11).

12.2.2 Research Questions

The Research Questions are divided between different aspects of the RPA development. Q2 is discussed from the development and testing viewpoints, whereas Q3 includes com-

Implementation Challenge	Description	Code File	Tech View	Q2
StaC1	The code file has too much text due to the generated comments	x		
StaC2	The developer could misuse the generated comments in the code file	x		
StaC3	The generated comments may break some development tools used with Python	x		
StaC4	The technical view is confusing without proper instructions		x	
StaC5	Developers could give up with the tool quite fast if they find it too laborious to use			x
StaC6	The tool might conflict with the previous best development practices	x		x

Table 12.3: StandAlone specific Implementation Challenges from the final interviews.

munication and documentation.

Q2: Does the tool ease development and testing of RPA projects?

The feedback from the final interviews was positive and the interviewees thought that the tool would ease development and testing. However, it was also mentioned that the actual usage was the only way to prove it completely. The prerequisites where there, but the trial usage was needed (StaP2). The developers attending to the trial should be co-operative and to really want to improve the tool. It was considered that some developers could give up with it relatively fast if its usage is too laborious (StaC5). Additionally, the trial would help to identify possible previous best practices and development techniques conflicting with the new tool (StaC6).

According to the interviewees, the tool would definitely ease the development and testing of the smaller projects (StaI12). The bigger the project, the more challenging it is. The tool could enhance best practices in both coding and definition levels (StaI13). When the most of the project information is available from one place, it is both useful and speeds up the development (StaI4, StaI5). Creating the concrete test procedures in code level is not eased by the tool but the proactive test planning, such as designing test cases could benefit from it (StaI9).

Q3: Does the tool ease communication and documentation between stakeholders in RPA projects?

All interviewees agreed that the PDD-SDD Tool has potential to replace the old PDD practices and that the communication between the different stakeholders would definitely be eased with it (StaI16, StaI17).

The documentation processes was seen to clearly benefit from the tool as well (StaI18). As intended, the PDD would continuously update itself towards the SDD. The tool provides an easy way to get acquainted with a project, which was one of the biggest benefits recognized by the interviewees (StaI6, StaI8). The automatic documentation was also seen to provide a reminder for the developer to complete the technical documentation as well (StaI19).

13 Results

The StandAlone version of the PDD-SDD Tool was successfully implemented in a way that corresponded its initial design in Chapter 5. All three rounds of interviews guided the development with useful feedback about both the Core and the Implementation Requirements, Challenges and Proposals for the tool.

The results of the final interviews are further discussed in this chapter, which summarizes the final outcome of the project, as well as concludes the last development iteration illustrated in the project's roadmap in Figure 6.1. The first section describes the satisfied Core and Implementation Requirements. The second section seeks answers to the so far unsolved Implementation Challenges. After that, the Research Questions investigated in this thesis are answered in the third section. Finally, in the last section of this chapter, a combined list of the Further Development Proposals is formed to guide the potential further development of the PDD-SDD Tool.

13.1 Satisfied Requirements

The Core Requirements (R1–R4) for the PDD-SDD Tool were updated after the first interviews and listed in Table 8.8. The Implementation Requirements (IR1–IR8) addressed the needs of the concrete implementation of the tool from a developer's viewpoint, listed in Table 7.12.

The core and Implementation Requirements are discussed in relation to the StandAlone Impressions (StaI–StaI19) from the final interviews, listed in Table 12.2.

Core Requirements

The Core Requirements were formed from the initial Core Requirements listed in Table 4.3 and the additional R4, which was identified from the first interviews in Chapter 8. The core Requirements were formed based on the general aspects of the RPA development, not by the specific PDD-SDD Tool's design.

All of the initial Core Requirements were successfully implemented and confirmed by the final interviews, discussed in Chapter 12. StaI2 noted that R1 is implemented successfully. Interviewees agreed that R2 was to be completely fulfilled when the tool is taken into company-wide use, as stated in StaI10. StaI11 confirmed that the automatic documentation demanded by R3 was also implemented successfully.

R4 was postponed to the Further Development Proposals, as stated in Chapter 11. This was not considered to be a significant shortcoming, because the implementation of the technical and architectural questions would be relatively close to the general questions in R1.

Implementation Requirements

Implementation Requirements were identified from the first interviews, in Chapter 8. Some of them were satisfied by the technical implementation while others were validated with the StandAlone Impressions from the final interviews. The Implementation

Requirements were formed based on the PDD-SDD Tool's design, and included features that were considered to be important for the tool's implementation from a developer's viewpoint.

As discussed in the Chapter 9, IR3 and IR6 were satisfied by the first two prototypes. The developer has an ability to change the code structure after the PDD has been accepted and the code base created, as required in IR3. The generated code file has a `main` function that contains the Phase level function in execution order, which satisfies IR6.

The remaining Implementation Requirements can be addressed via the identified Impressions from the final interviews. The clear division between the technical and the definition view, demanded by IR1 was validated by StaI1, StaI2, Stai3 and StaI8, which stated that both the definition view and the technical view are useful and implemented successfully. The code logic visualization required by IR2 in the technical view was similarly confirmed by StaI8 and StaI9.

An easy way to include business logic exceptions demanded by IR4 is validated by StaI4, which considered the parameters of the Detail level in the definition view to be successfully implemented. Also the tags that link the definition view with the generated code files were demanded by IR5, implemented in the StandAlone version and validated in StaI6.

IR7 and IR8 were postponed to the Further Development Proposals, as discussed in Chapter 11.

13.2 Solved Implementation Challenges

The final interviews resulted with 6 StandAlone specific Implementation Challenges (StaC1–StaC6), listed in Table 12.3. In this section they are compared to the previous 3 unsolved Implementation Challenges (IC4, IC9 and IC10), listed in Table 8.10, which were not answered in the second interviews. The goal is to find solution to them from the StandAlone Impressions and The Further Development Proposals gathered from the final interviews.

The concern that the generated code file could contain too much content is discussed in both IC4 and StaC1. Even though the StandAlone did not manage to solve the issue, the interviewees provided potential Further Development Proposals that could solve it in the future. The content could be hidden as described in StaP9, or generated into separate files as proposed in StaP10. In order to improve the usability even more, an additional plugin could be developed to view the separate files, as described in StaP11. Additionally, in order to identify the best option, the StandAlone tool should be given a proper trial with actual RPA projects, as stated in StaP2.

The difficulty of creating a tool that actually saves time and could replace old procedures were previously identified in IC9. The feedback from the final interviews was positive and the interviewee's thought that the tool could indeed save the developer's time during development, as described in StaI15, thus satisfying IC9. StaI16 answers to IC10 by stating that the tool was considered to posses better potential than the previously used format to document the PDD and the SDD.

The proper instructions and a trial usage of the tool, which were described in StaP1 and StaP2, provide also an ease to the other Challenges identified in the final interviews. With a proper instructions, the code file comments should be used correctly, therefore solving StaC2. Also the technical view would become easier to comprehend, answering to StaC4.

The trial usage helps to investigate if the tool indeed breaks some previous development tools or best practices, as feared in StaC3 and StaC6. It would also help to gather feedback from the developers to determine if the code file usage is too laborious, as described in StaC5.

13.3 Answered Research Questions

This thesis was seeking answers to Q2 and Q3 of the Research Questions (Q1–Q3) from Table 4.4. The remaining Q1 is to be answered in Abdulghani's thesis. Additionally, Q3 is discussed in both theses, as it concerns all aspects of RPA development. The Research Questions are answered with the help of the StandAlone Impressions from Table 12.2.

Q2 asked if the tool could ease development and testing of RPA projects. The results from the final interviews strongly suggest that the development and testing could be eased with the implemented tool. StaI12 stated that the tool would ease the development and testing of at least the smaller projects. It would also enhance better development practices, provide most of the needed material in a one place and save the developer's time, as described in StaI13, StaI14 and StaI15. Additionally, the interviewees thought that the tool could replace the old PDD practices, as described in StaI16.

Q3 asked if the tool could ease communication and documentation of RPA projects. The interviewees thought that the tool would indeed do that as described in StaI17. StaI18 and StaI19 stated that the documentation was considered to be significantly easier with the tool, and that it also reminds the developer to complete the technical documentation of the code logic.

13.4 Further Development Proposals

The final interviews yielded 15 Proposals (StaP1–StaP15) for the potential further development, listed in Table 12.1. Some of them were used to solve potential Implementation Challenges, as discussed in the previous sections, but the rest of them were new additions, which were considered to be useful in the future versions of PDD-SDD Tool. Additionally, the R4 from the Core Requirements and IR7 and IR8 from the Implementation Requirements were postponed to be implemented in the future.

This section discusses the new Proposals and compares them with the previous ones (p1–p11), listed in Table 10.4, in order to form the final list of Further Development Proposals that will guide the potential development after this thesis. The combined list of the final Further Development Proposals (P1–P27) is illustrated in Table 13.1. The Proposals are ordered by the PDD-SDD Tool's aspects, which are the definition view, the code file and the technical view.

The need for a trial usage in StaP2 was proposed during the final interviews by many interviewees. It should also be conducted prior to other major updates, therefore it is listed as P1. Additionally, StaP1 and p1 both discuss the importance of proper instructions for the tool's usage, listed as P2. The postponed Core and Implementation Requirements are listed next, to be first addressed after the trial and instruction Proposals. R4 is listed as P3. Similarly, IR7 and IR8 are listed as P4 and P5.

There are still 2 Proposals from the final interviews that are similar to the previous ones. StaP4 proposes the automated questions to be separated among the Phase and Step elements, listed as P3. Additionally, the amount of images in the Detail level of the definition view was discussed in both StaP5 and p2, listed as P4.

The rest of the Proposals gathered from the final interviews are totally new ones, and are combined with the previous list of Further Development Proposals without modifications.

Proposal	**Description**	**Def View**	**Code File**	**Tech View**
P1	In order to continue the development in the right direction, a trial with a real RPA project is needed	x	x	x
P2	A proper instructions must be provided to ensure the tool provides the best possible value	x	x	x
P3	The tool should provide also technical and architectural questions internally after the definition phase	x	x	x
P4	The tool should provide the original definition view to be compared with updated ones	x		
P5	The updated definition view should be able to be reviewed and accepted by the customer	x		
P6	The tool could provide links to other sources of material and documentation	x	x	x
P7	The general questions could be separated between Phases and Steps	x		
P8	The Detail level in the definition view could need more images	x		
P9	The visual layout could be further developed	x		x
P10	It would be useful to allow custom parameter types in the Details level in the definition view	x		
P11	Reordering the Phase and Step level elements would be easier with drag and drop features	x		
P12	The development status could be visualized in the definition view with and additional tag	x		
P13	The definition View could provide a way for the customer to request changes to the previously accepted business logic	x		
P14	The definition view could provide a way for the developer to ask the customer's acceptance for the updated view	x		
P15	The definition view could allow browsing of the project's version history	x		
P16	It should be possible to hide the generated code file comments		x	
P17	It could be better to save the code fiule comments into separate files		x	
P18	The separated code file comments could be viewed with a custom plugin for the code editor to minimize the need to move between the files		x	
P19	The code file generation should result with multiple code files, if required by the company's best practices		x	
P20	The tool could generate code that follows the company's best styling practices		x	
P21	The testing status could be visualized in the technical view functions		x	x
P22	The technical view could visualize the imported code logic as well			x
P23	The technical view could visualize the test coverage			x
P24	The technical view could provide more information about the code without the need to click elements			x
P25	There could be a need to modify information also in the technical view			x
P26	Images used in visual recognition could be included in the technical view			x
P27	The technical view could be used as an educational tool for the project manager and the customer to get familiar with the project in more technical manner			x

Table 13.1: Further development Proposals to be implemented after this thesis.

14 Discussion

The StandAlone version of the PDD-SDD Tool can be considered a success. Most of the Challenges were solved and the Requirements were fulfilled. The ones that were left unanswered can most probable be fixed with the gathered further development Proposals. Similarly, both of the Research Questions were answered in a manner, which strongly suggests that the StandAlone version of the PDD-SDD Tool was successfully implemented, and that it does possess a serious potential.

The first interviews in Chapter 7 gathered intel about the general RPA development and presented the initial design for the PDD-SDD Tool. The interviews resulted with the new Core Challenges, Core Requirements and Further Development Proposals, which helped to adjust the design and implementation of the prototypes. Two prototypes were developed in Chapter 9 to test the core functionalities designed for the tool, which were automated code file generation based on the PDD, and automated PDD updates towards SDD that corresponded to the updated code file.

The second interviews, discussed in Chapter 10 presented the second prototype in order to validate the development direction with the interviewees. Once again, Implementation Challenges, Implementation Requirements and Further Development Proposals were gathered. Additionally, the Further Development Proposals were combined with the ones from the first interviews.

The implementation of the StandAlone in Chapter 11 was conducted in a manner that solved the majority of the Core Challenges and satisfied almost all of the Core and Implementation Requirements. The final interviews, discussed in Chapter 12 presented the StandAlone as well as discussed the Core Requirements and the Research Questions with the interviewees, who thought that the tool had successfully achieved its goals.

Even though the results from the conducted interviews aligned well with the original presumptions from Chapters 4 and 5, they were still essential parts of the development process. The interviews ensured that the made presumptions of the RPA development Challenges and the tool's Requirements were correct and that the tool would actually provide ease for RPA development. Additionally, the development process and the discussions from the interviews further underline the importance of communication and documentation in RPA projects.

The development process of the StandAlone did not differ from a general software project. The development greatly benefited from the prototypes and short iteration cycles, which ensured that the tool was implemented in a manner that corresponded with the interviewees' needs. Overall, the interviews provided tremendous help with the development of the tool, and if the interviewees had identified any major problems, they would probably have been solved effectively, without much extra work. Such development practices, which collect end-user opinions to ensure that the initial assumptions of the designers are correct, are likely to be necessary in all types of software projects that develop new products.

The StandAlone version of the PDD-SDD tool is not a standard documentation tool because it is specifically designed for RPA development. The way the tool documents business logic was designed to suit all stakeholders in RPA project. This means that the distinction between high-level and technical documentation had to be clear. High-level information is easy to document for a customer, who has the option to describe the business logic verbally and with instructive images. In addition to the general documentation of the definition view, the technical view was implemented to meet the technical documentation needs of developers and maintainers.

The interviewees agreed that the next step for the StandAlone version of the PDD-SDD Tool would be a proper trial. The trial would provide more information about the use of the tool with real RPA projects, which would help adjust the direction of further development to ensure that the tool achieves even better results, one robot at a time.

References

[1] I. Jacobson. What they dont teach you about software at school: Be smart! In P. Abrahamsson, M. Marchesi, and F. Maurer, editors, *Agile Processes in Software Engineering and Extreme Programming*, pages 1–4, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[2] B. Hailpern and P. Santhanam. Software debugging, testing, and verification. *IBM Systems Journal*, 41(1):4–12, 2002.

[3] A. Abdulghani and S. Vuorela. Work experience from software and RPA development, 2016-2020.

[4] S. Moore. Gartner Says Worldwide Robotic Process Automation Software Market Grew 63% in 2018. https://www.gartner.com/en/newsroom/press-releases/2019-06-24-gartner-says-worldwide-robotic-process-automation-sof (visited on 10.2.2020), 06 2019. Gartner Press Release.

[5] W. Van der Aalst, M. Bichler, and A. Heinzl. Robotic Process Automation. *Business & Information Systems Engineering*, 60(4):269–272, Aug 2018.

[6] L. Willcocks, M. Lacity, and C. Andrew. The it function and robotic process automation. *The Outsourcing Unit Working Research Paper Series*, (15/05):1–39, 10 2015.

[7] Automation Anywhere. The Complete Starter Guide to RPA. https://www.automationanywhere.com/images/guides/rpa-starter-guide.pdf (visited on 9.5.2020), Automation Anywhere.

[8] N. Ostdick. The Evolution of Robotic Process Automation (RPA): Past, Present, and Future. https://www.uipath.com/blog/the-evolution-of-rpa-past-present-and-future (visited on 9.5.2020), UIPath.

[9] C. Saxena S. Gupta, S. Kumar. Review Paper on Comparison of Automation Testing Tools Selenium and QTP. *MIT International Journal of Computer Science and Information Technology*, 5:55–57, 08 2015.

[10] C. Le Clair. The Forrester Wave™: Robotic Process Automation, Q2 2018. Forrester Research, Inc., 06 2018.

[11] O. Ylönen. Blue Prism- ja UiPath-vertailu ohjelmistorobotiikassa. Thesis, Metropolia University of Applied Sciences, 2018.

[12] S. Helmers. *Microsoft Visio 2016 Step By Step: MS Visio 2016 Ste by Ste_p1*. Step by Step. Pearson Education, 2015.

[13] G. James. *Citrix XenDesktop Implementation: A Practical Guide for IT Professionals*. Elsevier Science, 2010.

[14] L. Willcocks. Why Robots May Not Be Taking Your Job – at least, not in the next 10 years. https://www.europeanbusinessreview.com/how-organisations-can-embrace-automation/ (visited on 9.5.2020), European Business Review, 2016.

[15] N. Weiderman, J. Bergey, D. Smith, and S. Tilley. Approaches to Legacy System Evolution. Software Engineering Institute, 12 1997. Carnegie Mellon University, Pittsburgh.

[16] W. van der Aalst, A. ter Hofstede, and M. Weske. Business Process Management: A Survey. In *Business Process Management*, pages 1–12, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[17] I. Chen and K. Popovich. Understanding customer relationship management (CRM): People, process and technology. *Business Process Management Journal*, 9:672–688, 01 2003.

[18] R. Atkinson. Project management: Cost, time and quality, two best guesses and a phenomenon, its time to accept other success criteria. *International Journal of Project Management*, 17(6):337–342, 12 1999.

[19] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, J. Stoica, and M. Zaharia. A View of Cloud Computing. *Commun. ACM*, 53:50–58, 04 2010.

[20] F. Xia, L. Yang, L. Wang, and A. Vinel. Internet of Things. *International Journal of Communication Systems*, 25, 09 2012.

[21] I. Luukkonen, J. Mykkänen, T. Itälä, S. Savolainen, and M. Tamminen. *Toiminnan ja prosessien mallintaminen: Tasot, näkökulmat ja esimerkit*. Solea Project, University of Eastern Finland and Aalto University, 01 2012.

[22] R. Daft and R. Lengel. Organizational information requirements, media richness and structural design. *Management Science*, 32:554–571, 05 1986.

[23] C. Seaman and V.Basili. Communication and organization in software development: An empirical study. *IBM Systems Journal*, 36(4):550–563, 1997.

[24] B. Ramesh, L. Cao, K. Mohan, and P. Xu. Can distributed software development be agile? *Commun. ACM*, 49:41–46, 10 2006.

[25] R. Hoda, J. Noble, and S. Marshall. Documentation strategies on agile software development projects. *International Journal of Agile and Extreme Software Development*, 1(1):23, 2012.

[26] D. Spinellis. Code Documentation. *IEEE Software*, 27(4):18–19, 2010.

[27] G. Myers, C. Sandler, and T. Badgett. *The Art of Software Testing*. ITPro collection. Wiley, 2011.

[28] T. Heinonen. Ohjelmistorobotiikan testaus. Master's thesis, University of Turku, 2019.

[29] G. Gill and C. Kemerer. Cyclomatic complexity density and software maintenance productivity. *IEEE Transactions on Software Engineering*, 17(12):1284–1288, 1991.

[30] E. Dijkstra. Notes on structured programming, 04 1969. Technological University Eindhoven.

[31] W. McKinney. *Python for Data Analysis*. O'Reilly Media, Inc., 2 edition, 10 2017.

[32] J. Joseph, S. Hissam, B. Fitzgerald, and K. Lakhani. Collaboration, conflict and control: the 4th workshop on open source software engineering. In *Proceedings. 26th International Conference on Software Engineering*, pages 764–765, 2004.

[33] A. Correia, F. Brito e Abreu, and V. Amaral. SLALOM: a Language for SLA specification and monitoring. *ArXiv*, abs/1109.6740, 2011.

[34] G. Prana, C. Treude, F. Thung, T. Atapattu, and D. Lo. Categorizing the Content of GitHub README Files. *Empirical Software Engineering*, 24(3):1296–1327, Jun 2019.

[35] D. Goodger and G. van Rossum. *Docstring Conventions*, pages 303–307. Apress, Berkeley, CA, 2010.

[36] S. Liawatimena, H. Hendric Spits Warnars, A. Trisetyarso, E. Abdurahman, B. Soewito, A. Wibowo, F. L. Gaol, and B. Abbas. Django Web Framework Software Metrics Measurement Using Radon and Pylint. In *2018 Indonesian Association for Pattern Recognition International Conference (INAPR)*, pages 218–222, 2018.

[37] M. Ramos, M. Valente, R. Terra, and G. Santos. AngularJS in the Wild: A Survey with 460 Developers. In *Proceedings of the 7th International Workshop on Evaluation and Usability of Programming Languages and Tools*, PLATEAU 2016, page 9–16, New York, NY, USA, 2016. Association for Computing Machinery.

[38] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren. Towards a unified behavior trees framework for robot control. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5420–5427, 2014.