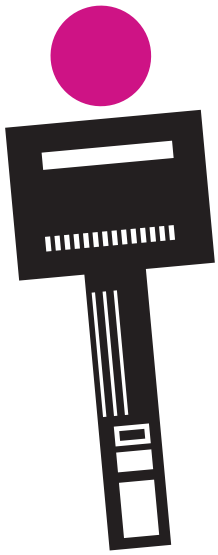

RAM ROBOTICS AND MECHATRONICS

EVALUATION OF A FEATURE-BASED DESIGN METHOD FOR RAPID DEVELOPMENT OF HARDWARE IN CYBER-PHYSICAL SYSTEMS

W. (Wouter) Horlings

MSC ASSIGNMENT

Committee:
dr. ir. J.F. Broenink
T.G. Broenink, MSc
dr. ir. G.M. Bonnema

March, 2021

018RaM2021
Robotics and Mechatronics
EEMCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | TECHMED CENTRE UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

Summary

Testing is an incredibly important part of the design process. Before a quality product is put into production, it has gone through extensive testing procedures. Likewise, new design methods have to be tested before they can be used in a design project.

The Rapid Iterative Design Method (RIDM) is a proposed feature-based design method for rapid development of Cyber-Physical Systems (CPS). Using RIDM the system is divided into a set of features. Each feature represents a part of the system functionality. By implementing and testing one feature at the time it provides a structured method to deal with the complexity of CPS.

This thesis evaluates if the RIDM is a suitable design method for the hardware-side of CPS. For the evaluation, a system is designed using the RIDM as a case study. Prior to the case study, some adaptations are made in order to use the design method for hardware. These adaptations add steps to create the set of features for a given design problem.

The RIDM focusses more on how to implement the features and less on how to define the features. However, the case study showed that the method of defining features is crucially important to the outcome of the design process. Another important finding is that a feature cannot be described with functionality alone. To be able to implement and test a feature, it must describe requirements and components as well.

Overall, the RIDM shows real potential to improve the design process of CPS. The approach to determine the order in which features are implemented greatly reduces the impact of design failures. Unfortunately, most of the RIDM is currently hindered due to a lack of tooling.

The main findings in this thesis suggest that the RIDM must incorporate a holistic design process. This design process describes all development steps needed to get from a problem description, via the set of features, to a finalized product. Furthermore, tooling to organize and test the development is required to utilize all advantages the RIDM provides.

Contents

Chapter 1

Introduction

1.1 Context of this Thesis

Cyber-Physical Systems (CPS) contain systems that control and monitor their included physical system parts (Rajkumar et al., 2010). This physical system is often a system of mechanical components which are deeply intertwined with the software components. Automobiles, robots, medical devices and even the smart grid are examples of CPS. The complexity of CPS has gone from an embedded system that improved the fuel consumption of a car engine to a fully autonomous vehicle. Although the complexity opens up more design possibilities, improved efficiency, and better safety, it has downsides as well.

Major downsides with the increasing complexity are the increasing developing cost and the decreasing reliability. Broenink and Broenink (2019) introduced a new design method for CPS that aims to deal with the downsides of the complexity. Throughout this thesis, the term *Rapid Iterative Design Method*, abbreviated to RIDM, is used to refer to the design method by Broenink and Broenink (2019).

The RIDM adopts a design technique called rapid development that splits the development process into small individual steps, where each of these steps are implemented and tested separately. Testing each individual step creates feedback on a short interval, finding errors in the design as early as possible. When a test reveals an error in the design, the worst case scenario is that all resources invested since the error was made are lost. Errors are unavoidable, but detecting them as early as possible reduce the amount of lost resources.

As part of the research, Broenink and Broenink performed a small case study. In this case study, they have designed a controller, and implemented the controller in software for a physical off-the-shelf system. Developing CPS incorporates the computational software side and the physical dynamic side. However, the case study by Broenink and Broenink only covers the software side of a CPS. For this design method to be suitable for a complete design of CPS it must apply to the physical part of the system as well.

1.2 Research Objective

Broenink and Broenink (2019) present a case study in their paper, developing a software based control system following the RIDM. About the result of that case study they state that ”this [case study] does not

mean that the same techniques cannot be applied to the physical part of the system." In this thesis, I research whether the RIDM applies to the physical part of a CPS, to come to a design method that apply on both the physical and cyber (software) part of a CPS.

The paper makes no attempt to offer a comprehensive design method to be used out of the box. The RIDM does not provide information about bringing a system into being, it does not address problem definition, requirements or initial design steps. Another weakness is that the RIDM gives no explanation of how the design steps are executed, only specifying that they are used. The design method would have been more useful if the authors had made a complete design method available to accompany their paper. To assess the RIDM as a design method for CPS, I set the following research objectives:

- Extend the RIDM with a preliminary design phase, focussing on the physical part of CPS.
- Refine the RIDM to make the design steps more explicit with improved instructions.
- Develop and perform a case study that tests and evaluates the RIDM as a design method for the physical part of CPS.

Evaluation of the RIDM as a design method is done with the results of the case study as the following objectives:

- Assess the influence that applying the RIDM has on the design process for CPS.
- Describe which adaptations are required for both the RIDM and the design method to establish a competent design process for CPS.

1.3 Approach

The goal of this thesis is to evaluate the RIDM, in the form of a case study. The case study consists of a *design process*, developing a CPS according to the RIDM. Based on the results of the design process, the RIDM is evaluated. However, there are a couple of steps required prior to the start of the case study.

The first step is to produce a concrete *design plan* based on the design method. The concrete design plan improves the evaluation of the design techniques. The design method is presented in an abstract form which leaves room for interpretation. This abstract form hampers the evaluation process, as the ambiguity of the design method makes it difficult to point out flaws in the design method. Therefore, I assess the design method and add detail to make a more concrete design plan. Because the RIDM focusses on rapid development principles and modelling techniques, it does not cover the design steps outside of that focus. These steps, like problem definition and system requirements, are a crucial part of the design process and are added to create the concrete design plan. The added steps are based on the steps from the *Systems Engineering (SE)* approach (Blanchard and Fabrycky, 2014).

With a design plan to use in the case study there are two steps of preparation left. The first step is to develop an *evaluation protocol* to ensure complete and consistent feedback during the case study. The evaluation protocol consists of a list of questions that are evaluated for each design step. The protocols contains questions about the design method itself, thus evaluating the instruction of each design step.

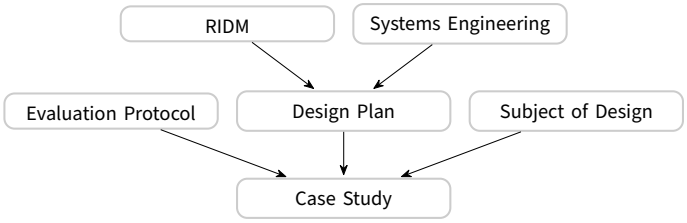

Figure 1.1: The case study is consists of something to be designed (subject of design), how to design that something (design plan), and how to evaluate the design process. The design plan itself is a combination of the RIDM and SE.

Other questions are about the design process, covering the execution of the instructions. The other step is to provide the *subject of design* to develop in the case study, essentially defining a problem that has to be solved. How all these components combine into the case study is shown in Figure 1.1.

Normally, the design process focusses on delivering the end product in the most effective manner. However, the goal of this research is to use the design process to evaluate the design method, not to develop a product. A possible pitfall is that during the design process the developer finds a simple solution, such that the design techniques to deal with the increased complexity are left untouched. Therefore, it is important to guarantee a minimum level of complexity. Instead of defining a problem that is very complex, I decided to require a minimum complexity to the solution. This makes the design process complex enough, without requiring an excessive amount of development time or compromising the quality of the evaluation.

Together with some other practical requirements, the best subject of design found is ”Writing a tweet on a whiteboard”. The subject of design is interesting because it has multiple design solutions that are complex but not unpractical. Furthermore, it has some interesting dynamics, requires a control law, and can easily be constructed into a prototype.

With a subject of design that requires a solution in the form of an object that incorporates both physical and cyber parts to develop; a design plan which describes how to develop this solution; and a protocol to evaluate the design plan and the development of the solution; the case study is executed. From the results of the case study I propose multiple improvements to the design method, not only for the physical part of CPS but also the cyber part.

1.4 Structure

The thesis is structured as follows: The first two chapters introduce the design methods. Chapter 2 gives a background of the RIDM and SE approach and how this is combined into the design plan. The design plan is presented in detail in Chapter 3, where each step is explained.

The next three chapters cover the case study: Chapter 4 explains the method of the case study, the subject of design and the evaluation protocol. Chapter 5 documents the execution of the case study, showing the development during the design process. All the questions and observations that were administered by following the evaluation protocol during the case study are analysed in Chapter 6.

The last two chapters reflect on the design plan that is evaluated in this research. Chapter 7 uses the evaluation results of the case study to reflect on the design plan in this thesis. And finally, the research is concluded in Chapter 8.

Chapter 2

Starting Point

The goal of the design plan is to develop a CPS. Due to the nature of CPS, it involves a multi-domain design approach. Therefore, the subject of SE is relevant to this approach. Furthermore, the RIDM is discussed in more detail in this chapter.

The RIDM does not initiate from the problem description step. As this step is required a design from scratch, the RIDM is combined with the approach from SE establish the required design steps.

2.1 Systems Engineering

Blanchard and Fabrycky (2014) describe SE in their book as: ”an interdisciplinary approach and means to enable the realization of successful systems.” Their book extensively covers multiple design methods and design steps in detail. For this thesis, their approach on *Bringing a Systems into Being* and *Preliminary System Design* are especially relevant. SE is a complete field of engineering on its own and only the top of the iceberg is used in this thesis.

2.2 Rapid Iterative Design Method

The RIDM by Broenink and Broenink (2019) describes a methodology using two core components for the implementation: the rapid development cycle and the variable-detail approach. The design method also describes the preparation steps that are required prior to this implementation. In short, the preparation prepares a list of features. These features are implemented one by one with in the rapid development cycle using the variable-detail approach. The following sections discus each of these three design steps.

2.2.1 Rapid Development Cycle

The goal of the rapid development cycle is sequential implementation of the features in a system. Each iteration of the rapid development incorporates the following steps:

1. Create an initial design and corresponding tests for the next feature.
2. Implement and test that feature.

The first step is to create an initial design and tests that are used to verify the requirements of the current feature. During the second step, the initial design is developed into a detailed design of the feature. This detailed design of the feature is develop with the variable-detail approach, in which the level of detail is stepwise incremented. When the second step is completed, the implemented feature contains all the required details and passes all the tests as defined in the first step. From this point the rapid development cycle is repeated for the next feature, or, when no features are left, finish the development.

2.2.2 Variable-Detail Approach

The variable-detail approach starts with a low-detailed model and increases the detail discretely over multiple iterations. The low-detailed model is for example a single transfer function of the system. In the following iteration, the detail of the model is increased by adding, for example, non-linearity, non-continuity or parasitic elements.

The tests, as specified in the first step of the rapid development cycle, are performed after each addition of detail. If the tests show that the added detail is not conform the requirements, the added detail is reviewed or redesigned. When the added detail passes the tests, the process is repeated to add more detail. The variable-detail approach is finished when all the tests are passed and all the detail is added.

2.2.3 Preparation steps

Although the RIDM does not specify the complete steps for the preparation, it does state some requirements. The rapid development cycle requires a list of features that can be implemented and tested individually. These features are gained by partitioning the functionality of the system.

For each feature it is required to specify the feature requirements and the corresponding test protocol. The feature requirements are based on the system requirements and the tests are used to validate that the feature meets its requirements. About the order of implementation, the RIDM states that critical features must be implemented first, as these features have an increased chance of invalidating the complete design. Would such a feature fail, the investment loss is limited, because the development is still in an early stage.

Following the feature separation step is the system test protocol. The goal of this step is to describe how the requirements of the system are tested. These tests can cover a single feature or multiple features.

2.3 Combination

To create a complete design plan, design approaches from both SE and the RIDM are combined. The RIDM requires an initial design that is then split into features. To meet this requirement, a linear set of steps from problem description to initial design is used from SE. These three steps are shown at the top of Figure 2.1 in the Systems Engineering group. The steps show some similarity with the first steps of a V or Waterfall model.

The requirements and initial design are used in the four steps of RIDM continue the design process. These four steps are grouped at the bottom of Figure 2.1.

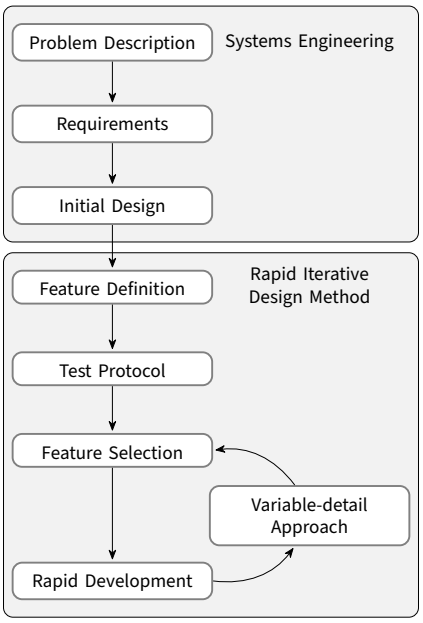

Figure 2.1: Combined design plan, where the first three steps are based on the SE-approach and the other four steps are taken from the RIDM (Broenink and Broenink, 2019)

Chapter 3

Design Plan

The goal of this chapter is to define a concrete design plan that is used in the case study. All of the steps in the design plan must be specific such that each of these steps can be evaluated after the case study is finished. The previous chapter introduced how two design methods are combined to form the basis of the design plan.

The design plan consists of two parts: The first part is the Preliminary System Design and contains the linear set of steps from problem description to feature definition. The second part is the Development Cycle, which contains the features selection, variable-detail approach and rapid development cycle.

3.1 Preliminary Phase

The goal of the preliminary design phase is to create a set of features for the design solution. Although these design steps in SE play a crucial roll in the success of the development, they are, however, very exhaustive. A major part of this complete design process is the required documentation to ensure agreement about the design between the different stakeholders. Resulting in a process that can take months or even years, which is not feasible for this thesis.

In this thesis, this design plan is only used for evaluation and has only one stakeholder, the author. This allows for a simple implementation of the SE approach, as it not possible to create a false start due to misunderstanding, saving valuable time.

The first three steps of the preliminary phase are based on the SE approach by Blanchard and Fabrycky (2014). As the evaluation of SE is not in the scope of this thesis, this chapter only covers the minimal description of the design steps in SE. These three steps deliver requirements and an initial design. The last two steps define the set of features and tests based on these deliverables.

3.1.1 Problem Description

Before any design process can start, the ”problem” has to be described. In other words, why is the function of the system needed? This is described in a *statement of the problem*. In this statement of the problem it is important to describe ”what” has to be solved, not directly ”how”. Blanchard and Fabrycky (2014) also note that ”defining the problem is often the most difficult part of the process”. It is important to ensure

good communication and understanding between the different stakeholders. Otherwise, it is possible that the designed product is not up to the customers expectations. It furthermore involves defining the subjects like what are the primary and secondary functions? When must this be accomplished? What is not a function? For this thesis, however, the problem definition is limited to a short statement of the problem, covering some required functions with corresponding requirements.

3.1.2 System Requirements

The system requirements are derived from the problem definition, and describe the characteristics of the system. As these characteristics form the foundation of the system, the requirements must be defined without any ambiguity, vagueness or complexity. The requirements are written according to the Easy Approach to Requirements Syntax (EARS) (Mavin et al., 2009). EARS was chosen for this design method due to its simplicity, which fits the scope of this thesis. Later in the design, these requirements are distributed over the subsystems. Any issues, like ambiguity, in the requirements, propagate through these subsystems. This might lead to a redesign of multiple sub-systems when these requirements have to be updated.

3.1.3 Initial Design

In the initial design step, the ”what has to be solved”, is expanded with a solution on ”how it is solved”. To find the best solution it is important to explore the different solutions and design space. Often, there are many possible alternatives but they must be narrowed down to the solutions that fit within the schedule and available resources. The best alternative is materialized in a design document together with the system requirements. This design document is used in the next phase of the design.

3.1.4 Feature Definition

During the feature definition step, the initial design is split into features as preparation for the rapid development cycle and the variable-detail approach. The RIDM does not provide a particular approach to define the features of the design. But, the goal is to have features that can be implemented and tested individually. The approach in this design plan aims to provide a more guided and structured way to split the features.

The approach to define features in this design plan is based on the separation of levels principle (*RobMoSys* 2017). This principle defines different levels of abstraction. This starts from the top with the *mission*, for example, serving coffee. Followed by less abstract levels such as: a *task* to fill the coffee mug; a *skill* to hold that mug; and a *service* allows the hand to open or close.

The different levels allow the features to be split multiple times in a structured way. Take the coffee serving example, to fill the coffee mug, it is not sufficient to only hold the mug. The system also has to pour coffee into the mug, and maybe add sugar or milk. This results in a hierarchical tree of functions as shown in Figure 3.1. Each of the levels have a many-to-many relation with each other.

With this approach, features are defined top-down and are implemented bottom-up. Thus a *skill* is defined as one or more *services*. When all the

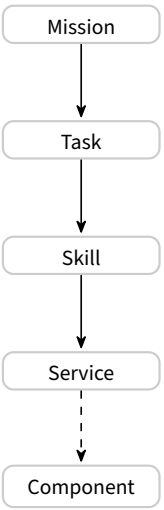

Figure 3.1: Hierachical structure of functions and components. Each arrow represents a many-to-many relation.

services are implemented, they are combined into a *skill*. The advantage of this is that the *skill* defines a milestone to combine the relevant *services*. Or looking at the example: the system must at least be able to grab, stir, and pour before it can fill a mug with coffee, milk and sugar.

Another advantages is that multiple *skills* can have a *service* in common. This would be the case if our system also needs to serve tea. The system can already hold a mug and only needs the ability to add a teabag. Even though there is no exact level of abstraction required for each of the features, it does create a structure for the developer. In the end, the developer must rely on its engineering judgement to chose the optimal division between features.

The bottom level of the hierarchy is a special case as it describes hardware instead of functions. The components are used to execute the functionality of the system with. For example, having a mobile robot arm near a coffee machine does meet the hardware requirements, it does not have any functionality if that is not yet implemented. This also creates a clear division for the developer as the functions cannot be mixed with the hardware.

3.1.5 Test protocol

During the rapid development cycle and the variable-detail approach, the system is tested constantly. This is to make sure that the design still performs as expected. The tests are based on the requirements. Each requirements must be covered with at least one test. The tests consist of a description which specifies how to perform the test and what the result of the test must of must not be. Together with the description, there is a list of required features to perform the test and a list of requirements that are met if the test passes.

3.2 Development Cycle

The development cycle consists of three steps, which are repeated for each individual feature. These three steps form the core of the RIDM. This starts with selecting the feature that is to be implemented, which then is implemented with the rapid development and variable-detail approach.

3.2.1 Feature Selection

The goal of this section is to improve the features selection criteria of the RIDM The RIDM states that critical features, those with a high *Change of Failure (COF)*, must be implemented first. If a critical feature fails, it is at the start of the design process, thereby invalidating only a portion of the design process. Features that are (time) expensive to implement, must be implemented as late as possible. These expensive features have a high *Cost of Change* and placing them at the end of the development avoids making changes to the features.

The *Change of Failure* and *Cost of Change* are a good starting point for selection criteria. However, this creates an interesting situation for features with both a high change of failure and a high cost of change. The rest of this section provides a structured approach for feature selection.

An example that shows the importance of the order of features is the development of a car. To have a critical damped suspension in a car,

the weight distribution of the car must be known. If the suspension of the car is designed before all the features that determine the weight distribution, it is likely that the suspension design is not up to requirements. Resulting in a redesign of the suspension feature and thus increasing the overall development cost. This example is caused by the dependency between different features.

To determine the order of implementation of features, a dependency graph and a comparison table is made. The dependency graph and the comparison table for a theoretic system is shown in Figure 3.2 and Table 3.1 respectively. In general the dependency of the features is inherited from the hierarchical structure that is made in the feature definition step.

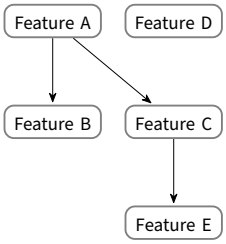

Figure 3.2: Dependency graph for features.

The comparison table has a dependees column, that describes the number of features that are depending on that specific feature, and are derived from the dependency graph. The tests column describes the number of tests that are covered by implementing this feature. These tests are defined during the initial design and the feature definition, the number represents the amount of tests that pass after implementation of the feature.

The COF per time score is calculated by dividing the COF score with the time score. The COF score indicates the likeliness of unforeseen difficulties during the implementation of the feature. The time score is an estimation about the required time for implementation. This time score is strongly connected with the *Cost of change*, but for readability I chose to refer to time instead. Due to the limited scope of this thesis, it is not possible to give a good metric for determining COF and time. Nevertheless, it is strongly advised that the developer defines some metric that fits his project best.

It seems logic to always implement the feature with the highest COF, but it is possible that the combined COF of multiple features is higher for the similar time investment. This is visible in Table 3.1: In a time span of 6 days it is possible to implement feature E or features A, C, and D. The COF for E is 45 % which is significantly less than the combined 65 %[1] of A, C and D.

[1] This is not a valid approach to calculate the combined chance, but suffices for the goal of this example.

With a completed comparison table, the order of implementation for the features is determined by the following rules:

1. Features that are dependencies of others must be implemented first.
2. Features that complete more system test than other features when implemented have priority.
3. Features with the higher *COF per time* score than other features have priority.

The rules are applied in order. If one rule reduces the set to a single feature, the rest of the rules are skipped. The third rule is a sorting rule,

	Dependees	Tests	COF	Time	COF over time
Feat. A	2 (B, C)	2	15%	3 days	5
Feat. B	0	3	40%	5 days	8
Feat. C	1 (E)	5	25%	2 days	12.5
Feat. D	0	4	15%	1 day	15
Feat. E	0	4	45%	6 days	7.5

Table 3.1: Comparison of features with their corresponding COF and time. The last column is the COF value divided by the planned number of days.

and the feature that fits best is implemented. In case of a draw or in special cases the developer decides what feature to implement next.

Looking at an example of 5 features: As shown in Figure 3.2, features B and C depend on feature A; feature D does not have any dependency connections; and feature E is dependent on C. Together with the information in Table 3.1, the order of implementation is:

Feature A: has two features that are dependent on this feature, more than any other.

Feature C: has one feature that is dependent on this feature, most dependees after A is implemented.

Feature D: has the same number of tests as E, but D has a significant higher COF per time score than E

Feature E: has the most number of tests.

Feature B: only one left to be implemented.

Note that this example assumes that nothing changes. In case of a feature not being feasible during the implementation, the design has to be reviewed. This also means that the dependency graph and comparison table change, possibly resulting in a different order of implementation.

3.2.2 Rapid Development

Each iteration of this rapid development cycle implements one complete feature. The feature that is implemented is selected in the prior feature selection step. The goal of this step is to lay the foundation for the development of the feature. This foundation consists of a basic model, a set of detail elements and a list of tests. The set of detail elements is a collection of design aspects that are added to increase the detail during the next design step. These detail elements can represent behavior, parasitic elements, or components. How these detail elements are implemented and what the basic model consists of is based on the initial design of the selected feature.

The initial design of the feature is similar to the system-wide approach in Section 3.1.3. It consists of a design space exploration, but with more detail, which is possible as the feature is significantly smaller than the complete system. From the design space exploration, the developer selects the optimal design choice for the current feature. For this design choice, a design document is made that illustrates the rough shape and dynamics of the implementation.

The basic model and the detail elements are based on an initial design of the feature. The basic model consists of only the most basic elements of the design. As the basic elements that make the basic model differ strongly per system, there is not a specific approach. In general, the basic elements should only represent dominant and essential behavior of the system. A good starting point for the dominant behavior is to identify the interesting energy states of the system. The energy states of interest can include the energy states that are dominant, but also the states that are chosen by the developer. These last states could represent the output states or status that have to be measured. In the end, the developer decides which states are required and implements them in the basic model. All the elements that are part of the initial design but are not part of the basic model are classified as the detail elements.

Lets take a motorized double inverted pendulum for example, which consists of two arms with motorized joints. Both pendulum arms are dominant energy states. The electrical motors have also internal states, but store significantly less energy than the pendulum arms. An basic model would in this case only consists of the arms, possibly even without any dynamic behavior. The dynamic behavior, motor characteristics, resistance, or gravitational force are examples of detail elements to be added to increase the detail.

3.2.3 Variable-Detail Approach

With the variable-detail approach the basic model is developed into a refined model of the feature. This is done by adding the detail elements over the course of multiple iterations. Each iteration produces a new model with more detail than the previous. The newly added detail is evaluated by performing the tests that were defined during the rapid development cycle. Not all tests are expected to succeed from the start, as not all details are implemented. For example, if the internal resistance of a electric motor is not yet implemented in the model, the motor can draw unlimited current, and this would exceed the maximum current draw of the system. The decision flowchart in Figure 3.3 determines whether the design must be reviewed or can continue on a failed test. The decisions are made with the following questions:

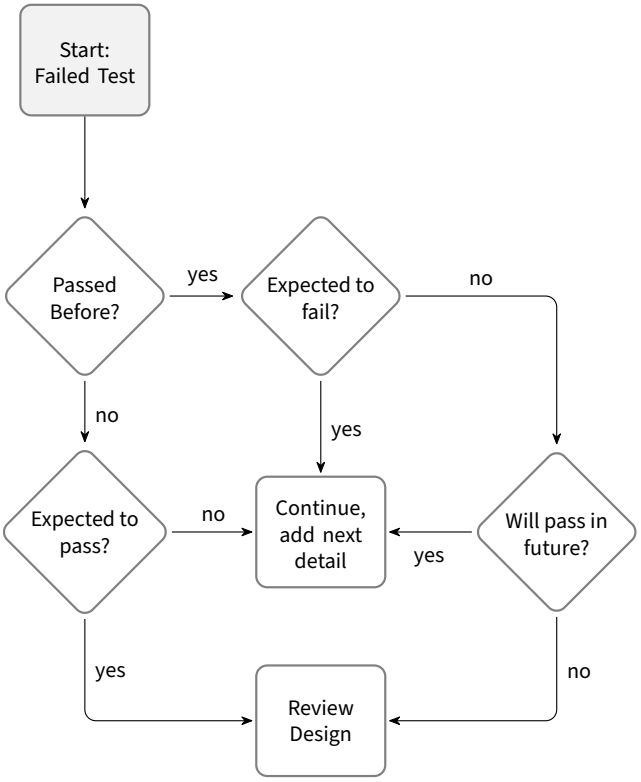

Figure 3.3: Decision flowchart to follow for failed tests on each detail level. Decision tree starts at the top left rectangle. Depending on the questions, the next step of action is to continue with the design or review the design.

Passed Before? The current test of the current design failed, but was there a previous detail level where it passed?

Expected to fail? Does the test fail as a direct result from the added detail and was that intentional?

Expected to pass? Should the added detail to the model result in a pass of the test?

Will pass in future? Is there an element to be implemented that results in a pass of the test?

In the case that the implementation of a detail element fails multiple times, the developer has to investigate if implementing the feature is still feasible. This could result in a redesign of the feature or system. When and how this decision has to be made differs per situation and is outside the scope of this thesis. The developer must evaluate if there are feasible alternatives left for this element, feature or system, and apply these alternatives if possible.

When all detail elements are implemented; all tests pass; and the basic model has evolved into a refined model of the feature, the design cycle moves back to the feature selection. In the case that this is the last feature to implement, this concludes the development.

3.3 Summary of Design Plan

The steps from SE and the RIDM are combined to create the design plan as shown in Figure 3.4. The first five steps of the design process form the preparation phase: problem description, requirements, initial design, feature definition, and test protocol. The initial design step creates a holistic design based on the prior problem description and requirements step. The last step of the preparation is the feature definition, where the initial design is split into different features. The resulting initial design and its features together form the design proposal for the development steps. The last step of the preparation phase is the test protocol step, where the tests are defined to monitor the design process and validate that the system meets the requirements. The development cycle consists of the feature selection, rapid development, and variable-detail steps. These three steps are applied to each feature in the system individually.

With each iteration of the development cycle a new feature is added to the complete system. All the tests of the individual features are performed in the complete system as well. This ensures that the one feature does not break a another feature. The design is finished when all the features are implemented, tested and combined.

In the optimal situation the preparation phase is only performed once at the start of the design, and the development cycle is performed for each feature. However, if features prove to be infeasible, some steps have to be revisited.

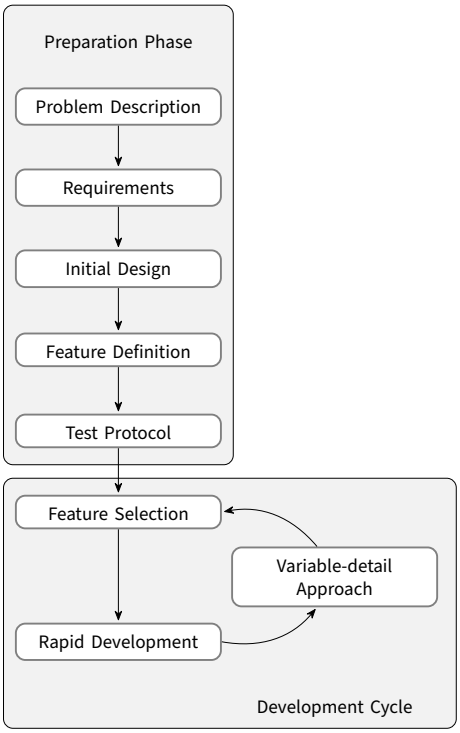

Figure 3.4: Combined design plan, based on the SE and RIDM approach.

Chapter 4

Case Study: Method

The goal of this case study is to evaluate the design plan as presented in the previous chapter. The evaluation is done by developing a system according to the design plan. In general, the method of the case study follows all the steps of the design plan. Additionally, an evaluation protocol ensures that the development is evaluated consistently. The last important thing is a subject of design that is developed as the case study. The next sections present the evaluation protocol and explains the choice for subject of design.

4.1 Evaluation Protocol

The evaluation protocol ensures that the different steps, decisions and changes of the design are consistently evaluated. This protocol contains a questionnaire and model validation. The full questionnaire is administered during each step in the design plan. It covers questions about the design process as well as the design plan. The model validation is performed at the end of the development. This validation is done by creating a physical prototype of the design and comparing the operation with the designed model.

4.1.1 Questionnaire

The questionnaire consists of two sets of questions. The first set of questions is shown in Table 4.1. This set consists of pairs of questions and focusses specifically on the execution of the design step. Each pair embodies a theme, with one question answered before, and the other question answered after the execution of the design step. The goal of these pairs is to compare the expected and resulting outcome of the design step. The second set of questions focusses on the described method of the design step. These questions are shown in Table 4.2.

To answer and record the questions consistently, there is a template document with these questions. The document with the answers is stored in version control and is automatically typeset into a PDF document.

4.1.2 Model validation

The design plan focusses on the modelling of the system. It is, however, not given that passing all the tests does also results in a work-

Table 4.1: Table with questions to evaluate the steps. With corresponding questions ordered in pre and post step columns.

Prestep Questions prior to the execution of the step to set a baseline.	Poststep Questions after the execution of the step to check if the implementation met the expectations. In hind-sight, what should have been executed differently?
What was the previous step? Does this influence this step? Is this a review?	**What is the next step?** Moving forward or is a review required of previous step(s)?
Describe the plan of action. What is the next step going to be? How is it going to be executed?	**Explain any deviations from the plan of action.** What changed during the execution, what deviations where taken and why?
What is the method of testing? What is the protocol to review the result of this step?	**How did you test/validate the step?** How was the evaluation done? Did it reveal something new?
What is the expected workload? How many hours are required for the execution of the step? Also give a range in your uncertainty.	**Was the workload different than expected?** How much time was invested in the step? Why was it different than expected?
What is the expected result of the step? At the end of the step, what is the expected result?	**Is the result as expected?** Does the result match the description made pre-step? Why does it not match?
What is the expected feasibility? What are the problems you expect during implementation? Why?	**Was the expected feasibility of the implementation accurate?** Even if the implementation succeeded, the feasibility does not have to be 100%. Give an estimate for the feasibility now that the implementation is finished. Explain the difference with the pre-step feasibility.

Table 4.2: Table with questions to evaluate the design method itself

Is this method a suitable approach for the hardware design? Are there aspects in the hardware design that is not covered by the method? Or is the method not suited at all for hardware? Why not? How to do it different?
Does the method contain counter-intuitive steps? Are there steps that feel not optimal? Is it appealing to execute the step different? Is it just getting used to? Or really inefficient?
What is the feasibility of this step in the method itself? Not the execution of the step, but the step itself. Are those steps realistic? How can the method be improved? Why?

ing design. If the tests are incomplete or complications in the design are overlooked, the design process is worthless. Because the design method would be unreliable.

Therefore, the model is validated with a physical prototype of the design. This shows whether the model is correct and whether all assumptions about the system are correct. The prototype does not only show where the design process went wrong, it can also be used to improve the design plan to prevent these modeling problems.

4.2 Subject of Design

The choice in subject of design has a strong influence on the effectiveness of the evaluation of the design plan. To ensure the best subject of design a list of requirements is composed. Based on this list the best subject of design I could come up with is a ”Tweet on a whiteboard writer”, which is referred to as system. Other subjects were considered, but did not meet the desired requirements.

The most important requirement is that, while developing the system, the different aspects of the design plan are used. Taking into account that there is a limited time budget and that the system must fit within the scope of the thesis, the set of possible subjects of design is slim. The time budget is set to 10 weeks of development and the system must have a dynamic system that is actuated via a software controller. The tweet on a whiteboard fits within these requirement as it can have interesting dynamics and has multiple features. Although it is possible that the system is seen as a wall plotter with basic XY-movement, there are alternative implementations that achieve more complex movement. This provides the required complexity and allows for different levels of detail. The XY-movement is the basic feature and detail is added in the form of other features. More detailed features are, for example:

1. Lifting/lowering the marker from/on the board
2. Erasing: End effector manipulation
3. Changing color: Switching Marker
4. Speed improvement

Similar to the XY-movement, these features have multiple implementations that add complexity to the system. This gives the possibility during the case study to go with a more or less complex design, allowing to fit the case study in the time budget without compromising the quality of evaluation.

Although a finished product is not required, a partial prototype is part of the testing and validation procedure. As the design method focuses on the physical component, a mechanical prototype is important. The prototype would originally been constructed with the rapid prototyping facilities at the university. However, the COVID-19 pandemic forced the university to close, and me to work from home. This limited the rapid prototyping to DIY-tools and a 3D-printer. It is expected that this set of tools is sufficient to construct a prototype of the tweet on a whiteboard system.

Other options that were also considered but did not meet the requirements. One of these options was a 3D calibration system for a position measurement system. This idea was rejected because the complexity originated from the required accuracy instead of the dynamics. In other words, choosing interesting dynamics would degrade the accuracy of

the system. A peg-in-hole problem, was also considered as a system. But that is mainly a complex sensing and control problem, and not dynamically interesting.

Chapter 5

Case Study: Execution

This chapter presents the execution of the case study. Where the goal of the case study is to evaluate the design plan as presented in Chapter 3. To achieve this goal, I develop a system according to the design plan and document this design process. As described in Section 4.2, the subject of design is a ”Tweet on a Whiteboard Writer”. Documenting the process is done by following the evaluation protocol as described in Section 4.1. To start the case study unbiased, during the preparation I did perform as little preliminary research as possible on the design options of the whiteboard writer.

The chapter begins with the section about the preparation phase, which contains the five steps from problem description to test protocol step as shown in Figure 3.4. This is followed by two completed development cycles in the later two sections. Both of these sections cover the feature selection, variable-detail approach and rapid development steps as shown in Figure 3.4. Each design step is described in a separate section. Herein, the result of each design step is presented and concluded with an evaluation section at the end. This evaluation section discusses the pairs of questions that were answered according to the evaluation protocol (Table 4.1). The questions regarding the design method itself are discussed in Chapter 7.

5.1 Preparation Phase

The preparation phase contains four design steps. It begins with a problem description. The problem description is used to create a list of system requirements. Based on the requirements, a number of design solutions proposed and eventually one of these solutions is chosen as initial design. Splitting the initial design into features is done in the feature definition step.

5.1.1 Problem Description

The problem description describes the need for a solution or system. In this case, I want a robot that can write a tweet on a whiteboard. A specific requirement is that the system must be complex enough, such that the specific aspects of the design method are used. These specific aspects are the ones that deal with complexity and are subject to the evaluation. The system must meet the following requirements:

- Write a twitter message, or tweet, on a whiteboard.

- Remove the tweet from the whiteboard.
- Write the tweet within three minutes on the board.
- The text must be readable across a meeting room.
- The solution must contain interesting dynamics.

Evaluation

The problem description is very brief, which is not a complete surprise. As most of the work for the problem description was already done by choosing the subject of design. However, it was not expected to be this minimal. Perhaps the most serious disadvantage is the absence of stakeholders. Normally, a good problem definition focusses on getting the stakeholders on the same line (Shafaat and Kenley, 2015). However, this case study does only have one stakeholder, the author, defeating the purpose of getting everyone on the same line. Creating a more elaborate problem description would not improve the evaluation of the design process, but it does cost valuable time.

5.1.2 Requirements

The next step is to create requirements based on the problem description. The goal is to write and erase a tweet on the whiteboard. Originally a tweet had a character limit of 140, but this was doubled to 280(Rosen, 2017). However, the decision is made to keep the limit at 140, as it does not improve the case study but can increase the construction cost. The text is limited to fifty characters per line, with a total of three lines. For the readability, the distance to a whiteboard in a meeting room is taken as four meters. The operating speed must allow the tweet to be written within three minutes. Therefore, the goal is to write one character per second. The last requirement is that the dynamics of the system must be sophisticated. Meaning that a solution with complex or non-trivial behavior is preferred. Using EARS to define these requirements gives:

System Requirements:

1. The Writer must be able to write at least fifty characters per line.
2. The Writer must be able to write at least three lines of text.
3. The Writer must plot characters with a size that is readable from 4 meters for a person with good eyesight.
4. The Writer must plot in a regular used font with corresponding character spacing.
5. When a new tweet is send to the Writer, the Writer must wipe the existing tweet and write down the new tweet.
6. If the Writer is not wiping or writing then the Writer must not obstruct the view of the whiteboard.
7. While writing, the Writer must have a writing speed of at least one character per second.
8. The dynamics of the Writer must be complex/sophisticat-

ed/interesting.

Some other requirements that are related to the operation of the system are:

System Requirements:

9. If the Writer is tasked to wipe the tweet, the Writer must wipe the tweet within sixty seconds
10. When a reset-signal is send to the Writer, the Writer must recalibrate its position on the board.
11. When a wipe-signal is send to the Writer, the Writer must wipe the board clean.
12. The Writer must not damage itself.

Additionally there are some restrictions on construction. As the rapid prototyping facilities at the university are closed due to the Covid-19 pandemic, the available tooling in reduced to my personal tools:

System Requirements:

- The Writer shall not exceed a total cost in materials and/or tools of €200.
- The Writer shall be constructed with simple tools in the following list:
  - Screwdrivers (Hex/Inbus, Torx, Philips, etc)
  - Drill
  - Screwtaps
  - Jigsaw
  - Wrenches
  - Soldering iron
  - Various Pliers
  - PLA 3D printer

Evaluation

The requirements step was performed without problems. Defining the requirements for the problem description did not present any difficulty. Due to the simplicity of the problem description, there were no contradictory requirements, which would complicate the requirements step. Furthermore, a single stakeholder takes away any negotiation between stakeholders. Where the stakeholders are a combination of engineers on the design team and/or the project client.

Although the requirements itself are not difficult to define, ensuring that they are complete is difficult. Discussion between team members and stakeholders helps to spot any ambiguity or problems with the validity. EARS was very useful in this case as it gives a strong template to help avoid ambiguity.

5.1.3 Initial Design

The initial design started with a design space exploration. The goal was to collect possible solutions and ideas for the implementation.

The exploration resulted in a lot of whiteboard writing robots ideas. These robots are sorted in four different configurations. Each configuration explained in the following sections. From the possible configurations, the one that fits the requirements best, is made into an initial design.

Cable-Driven

The cable-driven robot is suspended with multiple cables. The end-effector that contains the marker is moved along a board by changing the length of the cables. The cable-based positioning system results in an end-effector with a large range and high velocities. A basic setup is shown in Figure 5.1. This given setup contains two cables that are motorized. The big advantage of this system is that it scales well, as the cables can have almost any length.

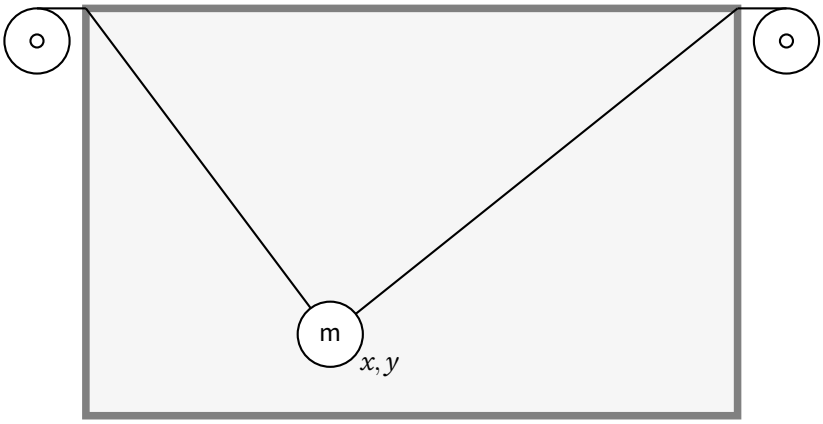

Figure 5.1: Planar view of cable-driven robot. This setup contains two motorized pulleys in both top corners. From these two cables a mass is suspended at position x, y. By changing the length of the cables, the mass is moved over along the whole board.

Although it is possible to achieve high velocities, this system is limited by the gravitational acceleration. In case of vertical acceleration, the maximum downward acceleration or upward deceleration is limited by $9.81\,\mathrm{m\,s^{-2}}$. The horizontal acceleration depends on the relative angle of the suspending cable. The closer the end-effector is below the cable pulley, the lower the pure horizontal acceleration becomes. Figure 5.2 illustrates the horizontal acceleration for different angles.

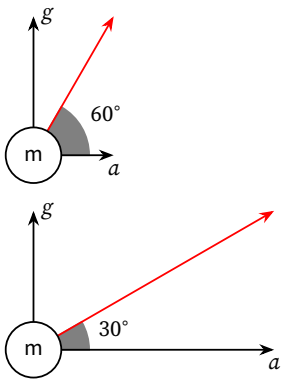

Figure 5.2: Illustrating the limit for pure horizontal acceleration a, for different angles to compensate for gravitational acceleration g. The red arrow represents the acceleration as a result of the pulling force of the cable, which is vectorized in a vertical acceleration that compensates g and a vertical acceleration a.

A possible solution to this is to add one or two additional wires to the system. These can pull on the system to 'assist' the gravitational force. Depending on the implementation, the extra cables make the system over-constrained. Nevertheless, the extra cables allow for higher acceleration limits in vertical and horizontal direction.

Cartesian-coordinate robot

This configuration is a very common design for plotters and shown in Figure 5.3. This setup is also known as a gantry robot or linear robot. It normally consists of two sliders, which behave as a prismatic joint. Because each slider covers a single X or Y axis, the control and dynamics of this system are rather simple. The biggest challenge is in the construction of the system, especially when the size of the system is increased. The larger system requires longer sliders, which are expensive. Another difficulty is the actuation of both horizontal sliders, if these sliders do not operate synchronous the vertical slider would slant and likely jam.

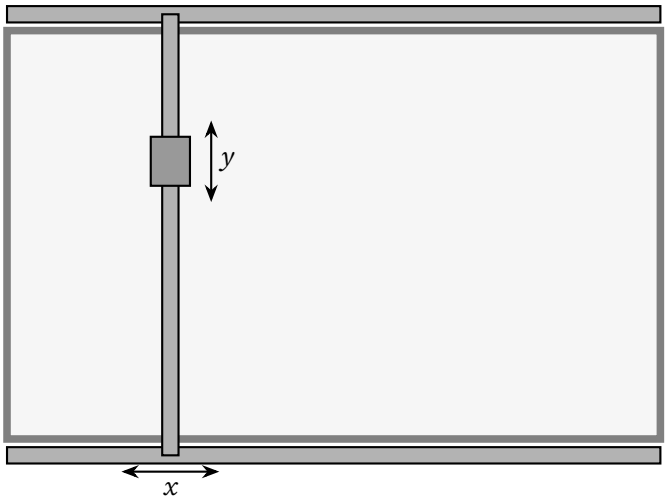

Figure 5.3: This Cartesian plotter consists of two horizontal sliders to provide the x-movement and one vertical slider to provide the y-movement.

Polar-coordinate robot

This robot is a combination of a prismatic and a revolute joint. Where the revolute joint can rotate the prismatic joint as shown in Figure 5.4. With this it can reach any point within a radius from the rotational joint. This is a little more complex design than the Cartesian robot.

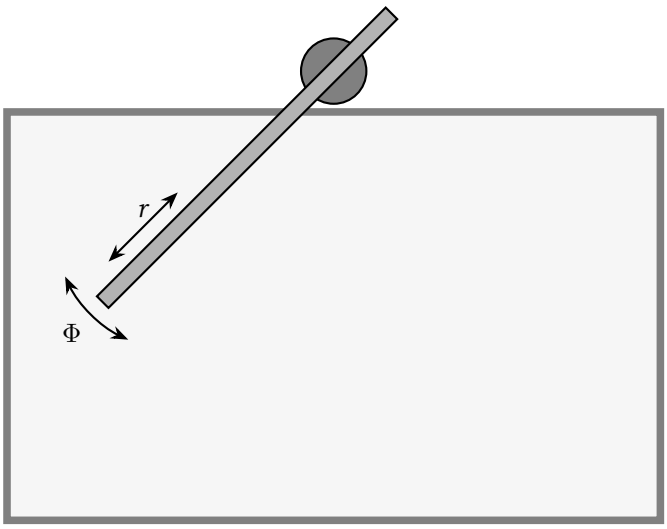

Figure 5.4: A combination of a revolute joint and a prismatic joint, creating a polar-coordinate robot.

This robot has multiple disadvantages. The range of the robot is defined by the length of the prismatic joint. Thus when the operating range is doubled, the robot size has to be doubled or even more than that. Furthermore, when the arm of the robot is retracted, it protrudes on the other side. Therefore, the complete radius around the revolute joint cannot have any obstacles. Figure 5.5 gives an impression of the required area. Even with this area, the arm cannot reach the complete board. This makes the required space of the setup very inefficient. Another disadvantage is that a long arm increases the moment of inertia and the gravitational torque on the joint quadratically. Furthermore, the long arm introduces stiffness problems and it amplifies any inaccuracy in the joint.

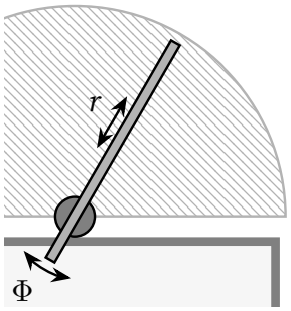

Figure 5.5: The diagonal lined section shows the part of the protruding area that is used by the arm.

SCARA

The selective compliance articulated robot arm (SCARA) robot is a configuration with two linkages that are connected via rotational joints. It

compares to a human arm drawing on a table as shown in Figure 5.6. Similar to the polar robot it can reach all points within a radius from the base of the robot. But the SCARA does not protrude like the polar arm (Figure 5.5). Depending on the configuration of the arm, it is possible to keep the arm completely within the area of operation. A downside is that the mass of the additional joint and extra arm length increase the moment of inertia and gravitational torque similar to the polar robot. This makes the SCARA configuration convenient for small working areas as that keeps the forces manageable. Additionally, as the arms of the SCARA have a fixed length, it is possible to create a counter balance. This can be used to remove any gravitational torque from the system. It would however increase the moment of inertia even further. For current requirements, the working area is too large for any practical application of the SCARA.

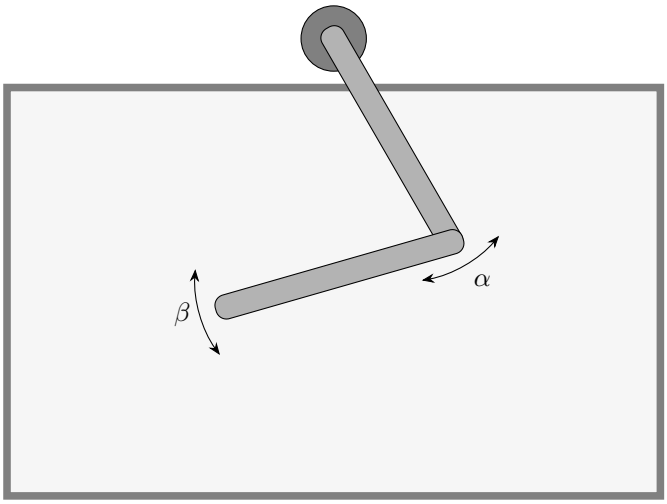

Figure 5.6: Schematic example of a SCARA, consisting of two rotation linkages. This setup can be compared to a human arm, where the gray base above the whiteboard represents the shoulder and the connections between both linkages the elbow.

Combining

A fifth option is to combine two of the discussed configurations, wherein the best properties of two configurations are used. The most interesting combination is the cable bot together with the SCARA. In this combination, the SCARA is small, only able to write a couple of characters. The smaller size of the SCARA makes it quick. To write full sentences the SCARA is placed on a carriage that is suspended by the cable bot. An example of this cable driven carriage (CDC) with the mounted SCARA is shown in Figure 5.7.

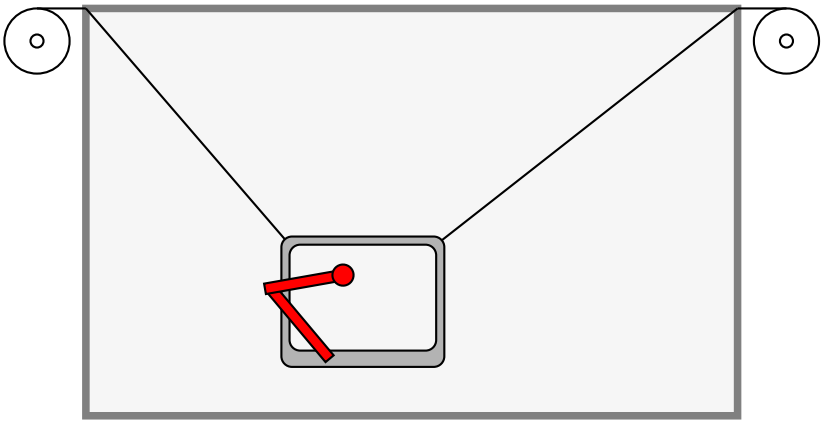

Figure 5.7: Combined system that integrates the cable bot together with the SCARA. The SCARA in red is mounted on the CDC.

This increases the complexity of the dynamics of the system, by having four degrees of freedom. Furthermore, the movement of the SCARA

also causes movement of the CDC. Shrinking the SCARA also decreases the challenges regarding construction, as long and unstable arms are out of the picture.

Choice of system

The previous sections have shown four different configurations. These configurations are compared in Table 5.1. Each of the systems are scored on range, speed, cost, obstruction, effective area, and the interesting dynamics:

Range
: The range scores the system on the practical dimension of the system, larger is better. The cable, cartesian, and combined configuration scale very well, the cables or slider rails can be made longer without real difficulty. The SCARA or polar configuration run into problems with the arm lengths, as forces scale quadratically with their length.

Speed
: Except for the cable bot, all configurations score sufficient on speed. The cable bot can reach high velocities, but the acceleration is limited, depending on the configuration, to the gravitational acceleration.

Cost
: For the cost, all systems fit within the €200 budget, except for the Cartesian setup. All systems require DC or stepper motors, but the cartesian setup also requires linear sliders which are expensive, especially for longer distances.

Obstruction
: The obstruction score depends on the capability of the system to move away from the text on the board, such that the system does not obstruct the written tweet. All systems except for the cable and combined configuration can move themself outside of the working area. It is possible that the wires of the cable or combined configuration obstruct the view. However, the wires are expected to be thin enough to not block any text.

Scalability
: For the scalability, the cable bot and the combined system score high. The cables make it possible to easily change the operating range of the system, only requiring reconfiguration. The cartesian system scales poor because the length of the sliders is fixed, and longer sliders are expensive. For the polar system and SCARA, the forces on the joints scale quadratically with the length of the arms. However, the SCARA can be build with counter balance making it scale less worse than the Polar system.

Effective Area
: With the effective area, the system is scored on the area it requires to operated versus the writable area. The polar configuration has a low score due to the protruding arm.

Interesting Dynamics
: The last metric, scores the system on the complexity of the dynamics. This is a more subjective metric, but also a very important one. In the problem description, the complexity of the dynamics was determined as one of the core requirements. The cartesian configuration is trivial, both sliders operate completely separate from each other and the position coordinates can be

mapped one to one with the sliders. The combined configuration excels for this metric, as it has 4 degrees of freedom and the SCARA movement can cause the carriage to swing.

	Cable bot	Cartesian	Polar	SCARA	Combined
Range	++	+	−−	−	++
Speed	−	+	+	++	+
Cost	++	−−	+	+	+
Obstruction	−	+	+	+	−
Scalability	++	−	−−	−	+
Effective area	++	+	−−	+	++
Interesting dynamics	−	−−	−	+	++
Total	+5	−1	−4	+4	+8

Table 5.1: Table with comparison of the four proposed configurations and a combined configuration of the cable bot and the SCARA.

The comparison in Table 5.1 shows that the combined configuration as preferred. Which is not surprising as it combines the advantages of both the cable bot and SCARA. Although those systems have a good score of their own, they have disadvantages. The cable bot has low acceleration and no challenging dynamics. The main difficulty for the SCARA is being able to build it large enough.

The combined configurations, complement each other. The range of the CDC allows for a small SCARA. The small size of the SCARA makes it quick. This compensates for the low acceleration of the cable bot and removes the need for a SCARA with long arms. Therefore, the choice of configuration is the combined system of the SCARA and CDC.

Evaluation

This was the first step that felt really productive in the design process. It created a enormous amount of information and insight of the design. In hind sight, it would have been useful to have this information during the requirements step. However, as the requirements step are mainly on the ”what” to solve, and specifically not on ”how” to solve it, this information was avoided on purpose during the requirements step.

This step did result in an initial design that is used in the next steps. However, I noticed that none of the previous steps have a clear start or end. For the problem description and the requirements steps the question is when all required information is collected. In the initial design it is always possible continue researching design options to come up with an even better design. Especially with complex system, it is unrealistic to create complete requirements before making design decisions. Resulting in the question: at what point do we have enough information and must we move to the next design step? This is also known as the *requirement versus design paradox* (Fitzgerald, Larsen, and Verhoef, 2014).

5.1.4 Feature Definition

This step divides the requirements and initial design into features. These features are implemented one by one during the development cycle, later in the process. As described in Section 3.1.4, the functionality of the system is split over four different levels of abstraction. The result of this split is shown Figure 5.8.

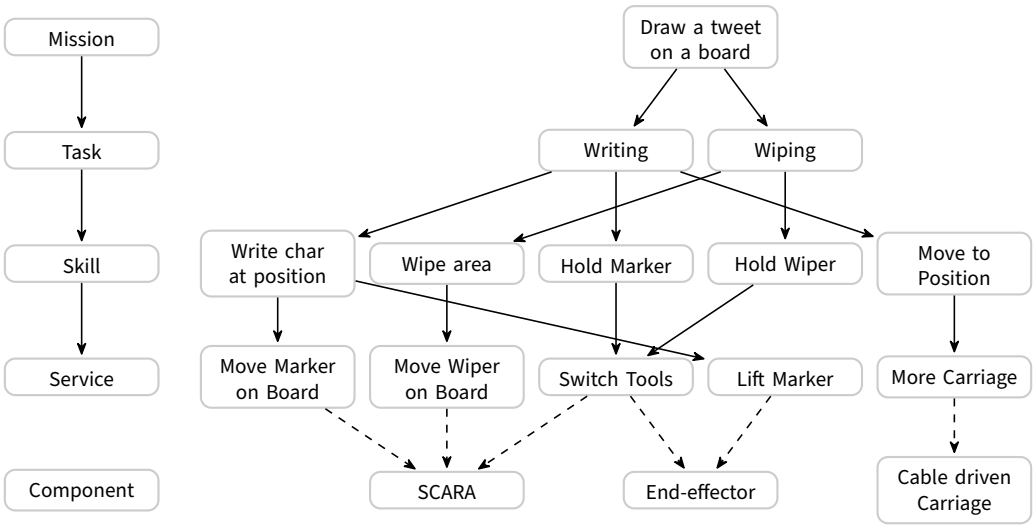

Figure 5.8: Hierarchical tree of different functions and components.

Evaluation

The mission and task features are easy to define. The skill and service features become a bit more vague, often features fit in both categories. Sometimes it is difficult to split a skill into a service, as they are already very specific. Additionally, I attempted to keep the feature tree a bit compact, to keep in scope with this thesis.

The components also use a similar approach as the functions, resulting in a hierarchical structure of sub-components, where the SCARA would have motors and electronics as sub-components.

5.1.5 Test Protocol

The last step of the preparation phase is to implement a test protocol. The tests are designed to validate if the system meets its requirements.

While defining the tests, it became clear that part of the requirements was not sufficiently defined. The current requirements apply to the complete system and is not updated for the design choices made in the initial design. If the tests are made based on the current requirements, they can only be performed when the complete design is implemented. To create tests that apply to specific features or compoments, the requirements had to be updated. This update adds order of operations and additional requirements, which are explained in the following two sections. The third section explains how the tests are formed based on the up-to-date requirements.

Defining the Order of Operation

There are two modes of operation: writing and erasing. In these situations, the end-effector holds a writing or erasing tool respectively. By defining the order of operation for each mode, specific requirements are assigned to the SCARA and the CDC.

The current design uses the SCARA to write characters on the board at a static position. When these characters are written the CDC moves to the next position:

Order of operation: Writing

Precondition: Board marker as tool in end-effector.

1. Move CDC to position of characters.
2. Write three characters with the SCARA.
3. Repeat step 1 and 2 untill the Tweet is on the board.
4. Move CDC away from the text on the board.

The second order of operation is about the erasing. Removing text from the board is done by the following steps:

Order of operation: Erasing

Precondition: Board eraser as tool in end-effector.

1. Move CDC to position of characters.
2. Clear the area in reach of the SCARA.
3. Repeat step 1 and 2 till the Tweet is removed from on the board.

A possible third order of operation is tool switching using the end-effector. At this point, the design of the end-effector is not definitive enough to determine an order of operation. Additionally, not having an order of operation for the end-effector did not hinder the definition of tests.

Improving Requirements

The defined order of operations add more requirements to the system. Two of the new requirements specify the operational behavior of the SCARA and CDC, based on the order specified above. To ensure that the overall system can still write one character per second, the SCARA and CDC must both complete their task in three seconds. Therefore, two seconds are assigned to the SCARA to write three characters and the CDC gets one second to move the SCARA to the new position.

A fifth requirement is added because it was overlooked during the previous steps. The five system requirements are in addition to those in Section 5.1.2:

System Requirements:

13. While writing, the SCARA must have a writing speed of at least 1.5 characters per second.
14. When the CDC is at a static position, the SCARA must be able to write at least three characters at that position.
15. When the SCARA finished writing at their current position, the CDC shall move the SCARA to it's next position where it can write the subsequent characters.
16. When the SCARA has to be moved to a new position, the CDC shall perform this movement within one second.
17. When the system changes from writing to erasing or vice-versa, the SCARA and End-effector should change the tool within ten seconds.

These additional requirements take into account that the current design consists of a SCARA, end-effector and CDC. Where each of these components has a different role and thus a different responsibility in the system.

Setting up the tests

Based on the updated requirements, a set of test cases is created. In total there are five small and five large test cases. The small tests cover a single compoment or feature and the large tests combine multiple features. Each test specifies for which requirements they apply and include a description that explains the test. The smaller test cases also indicate on which feature or component they apply. All ten tests are included in Appendix A. The following is a small and a large test case from these ten.

One of the small tests focusses on the speed and range requirements of the SCARA:

System Test 1: Small rectangle

During this test, a rectangle will be drawn on the whiteboard using the SCARA. This rectangle is 50 mm high and 70 mm wide, such that three characters fit within the rectangle. To test the speed requirements, the rectangle should be drawn within one second.

Features: SCARA
Requirements: 3, 7, 11, 13, 14
Results: The test passes when:
- Rectangle height is at least 50 mm
- Rectangle width is at least 70 mm
- Completion time is less than 1 s

Repeatability is tested in one of the large system wide tests:

System Test 6: Repeatability

This tests if the Writer can draw repeatedly on the same position, for different approach angles, on the board. The system will start with drawing multiple 60 mm squares on the board in a random location. To test the repeatability, a circle with a 55 mm diameter must be drawn inside of the square. This should be done with twenty squares in an area of at least 1000 mm x 300 mm. The drawing order of each square must be different from the drawing order of circles, this ensures that the Cable bot makes a different approach path.

Features: SCARA, Cable Bot
Requirements: 3, 4, 9, 11, (12)
Results: The test passes when:
- Each square has a circle drawn inside.
- The squares and circles are within 5 mm of their given dimensions.
- All the circles are completely within their corresponding square.

Evaluation

This step was completed without many difficulties. Eventhough this step included an unexpected revision of the earlier requirements and definition of order of operations. Indicating that I overlooked details while defining the requirements in Section 5.1.2. According to the design plan as described in Chapter 3, I have to go back and review those

requirements. Followed by reviewing all steps after the requirements. However, this complete review is not practical and extremely time consuming. The point here is, looking at the evaluations of this and previous steps, that chosen design strategy is not feasible, especially as a novice designer. In Chapter 6 I evaluate this with more detail.

During the analysis, I expected more specific tests. Each test can then be used as a milestone during the development of the system. Every time detail is added, an additional test passes. Or a test fails, notifying that something went wrong and has to be investigated. Creating such specific tests relies on the details in the design of the system. The current design is basic and these details are added during the feature implementation. Nevertheless, this step resulted in a set of tests that cover all requirements and features that are specified in this preparation phase. When all the tests pass, the system should meet all the requirements.

5.2 First Development Cycle

With the preparation phase completed, the development cycle is next. This consists of three steps: Feature selection, Rapid Development and Variable-detail Approach. The current section explains the first development cycle during the design. For this first cycle of the design process, I design the end-effector. However, not long after the start of the development process, the implementation of the end-effector proved to be too complex. This led to the decision to abort the implemention of the end-effector. Eventhough no progress was made in the design, this attempted implementation did provide valuable insight in the desing process.

5.2.1 Feature Selection

For each feature in the system the dependees, tests and COF/time factor is determined, as explained in Section 3.2.1. These values are combined into Table 5.2.

Feature	Dependees	Tests	COF	Time	COF/Time
SCARA	–	3	40%	10 days	4
End-effector	SCARA	2	60%	8 days	7.5
CDC	–	2	30%	10 days	3

Table 5.2: Overview of the different features and their dependencies, number of tests that are covered and the COF/time factor. The COF/time factor is calculate as COF divided by time.

The SCARA depends on the end-effector, as explained in the initial design. However, for the CDC no dependency was defined even though it has to lift the other two components. This is mainly because the torque and range requirements of the SCARA depend on the implementation of the end-effector. Especially the required range depends on the method of grabbing and releasing tools. For the CDC it only changes the mass that has to be lifted. Upgrading the motor torque is a minor parametric change and the dependency is therefore deemed insignificant.

The testing number is directly the number of tests that apply to that feature. The COF and time values are not determined with a specific protocol, but with simple engineering judgement. The estimated COF is high for the end-effector due to the collision dynamics of the operation. It has to grab something and that is difficult to model. Furthermore, it was not known if that design would work. The SCARA has the most

moving parts, but no difficult dynamics and has therefore an estimated COF of medium. For the CDC there was no real COF and got therefore a low COF indication.

Based on Table 5.2, the end-effector is implemented first. The end-effector has the most dependees, and is therefore chosen above the other two.

Evaluation

This first step of the detail design phase did go well. Although COF and time assessment is always depend on some engineering judgment, this human factor introduces uncertainty in the assessment. However, an improved approach for the COF assessment can drastically reduce this human factor. Within a design team a form of planning poker (Grenning, 2002) could be a good option.

5.2.2 Rapid Development of the End-Effector

This section explains the process of the development of the end-effector. The first step is to create an initial design of the model. In subsequent steps, detail is added to this model.

The previous section explained the relative high COF assessment for the end-effector. Which was not exaggerated as the implementation proved to be troublesome. Eventually, the implementation was unfeasible and was therefore cut short. Nonetheless did it result in useful evaluation points on the design method. The process of this step is explained in the following sections.

Initial design

The end-effector is mounted on the SCARA and acts as an interface for the tooling. The SCARA and end-effector combined are able to grab and release the write and erase tooling. There are multiple approaches to handle the tooling. However, there is a trade-off to be made with the SCARA feature, the heavier the end-effector is, the more force the SCARA must deliver. And because the goal is to make the SCARA light and quick, this end-effector must be light-weight as well.

The best options in this case is a simple spring-loaded clamp. To release the tool, the clamp is forced open, pushing it against the holder. As the end-effector is connected to the SCARA, the SCARA is responsible for the pushing force. Because the actuation force of the SCARA is used, it removes the need for an additional servo in the end-effector. Resulting in a simpler and lighter design.

The initial design of the clamp and the operation is shown in Figure 5.9. Although this design requires the SCARA to deliver more force. The relative low mass of the end-effector also keeps the moment of inertia small. Therefore, the current design reduces the impact on the acceleration of the SCARA.

Behavior Modelling

The next step is to implement this design with the corresponding behavior in a dynamic model. The challenge in this case is the modelling of the contact dynamics. Based on some experience in modelling with

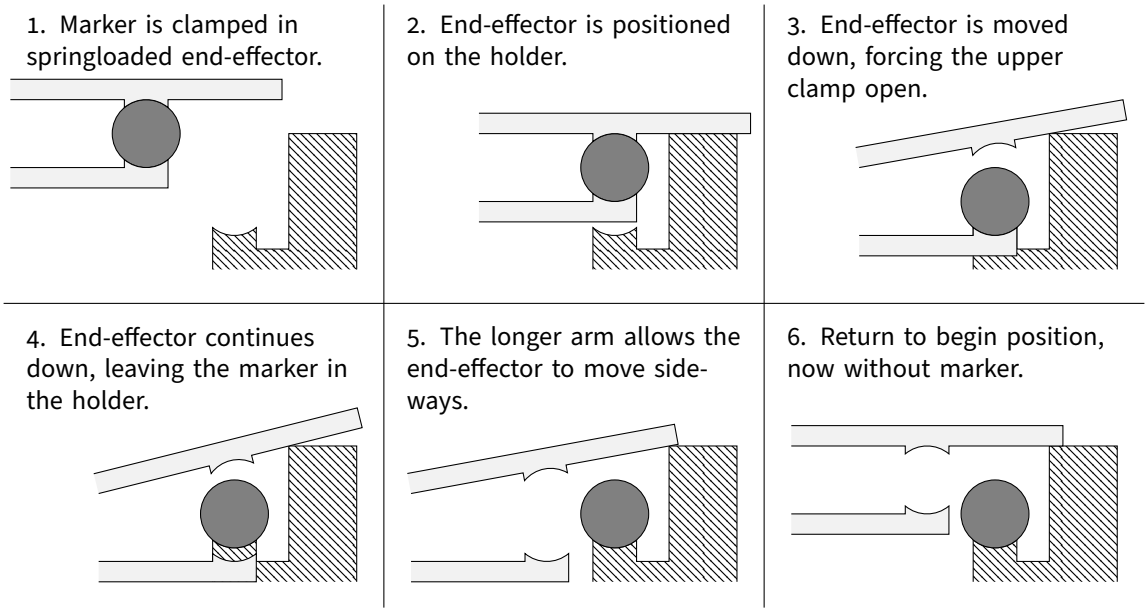

Figure 5.9: Operation of the end-effector. The clamp is forced open against the holder to release the marker. Instead of releasing, the marker is grabbed by reversing the order of executing for these steps.

collisions, I decided to use the 20-sim 3D mechanics editor. Unfortunately, there is little tooling available and there are no debugging options if the model does not behave as expected. The marker kept falling trough the gripper or flew away.

With the small amount of progress made in two days the implementation was not promising. A system freeze caused the model to corrupt, where the complete configuration of the shapes and their collisions was lost. Based on the loss of work and the low feasibility of the implementation, the decision was made to remove the end-effector from the design.

With the end-effector removed, the SCARA gets a direct connection with the marker. Lifting and lowering the marker is included in the SCARA feature as well. Unfortunately, this means that switching to the eraser is not longer possible as functionality.

Evaluation

The lost progress of the model is unfortunate, but the implementation did not go as expected anyway. It was probably for the best as it forced an evaluation of the design and avoided a tunnel vision while trying to get it to work. However, it did show the value of the COF per time analysis. This early failure resulted in changes for other components. But as none of the other components are implemented yet, no work is lost.

5.3 Second Development Cycle

As the previous development cycle was aborted prematurely, the development cycle is repeated for the next feature. Starting with a feature selection process. Followed by the rapid development.

5.3.1 Feature Selection

The implementation of the end-effector proved to be unfeasible and was therefore removed from the design. This means that only two features are left. Table 5.3 shows an updated feature comparison. Compared with the previous feature selection in Table 5.2, the number of tests for the SCARA decreased and the COF/Time increased. This is because System Test 5 relied on both the SCARA and the End-effector which is no longer applicable. Based on the feature comparison, the next component to implement is the SCARA.

Feature	Dependees	Tests	COF	Time	COF/Time
SCARA	–	2	50%	12 days	4.2
CDC	–	2	30%	10 days	3

Table 5.3: Comparison of the two remaining features in the design process. This table is an updated version of Table 5.2.

Evaluation

The feature selection for the second cycle is an updated selection process of the first cycle (Section 5.2.1). This resulted in a quick and effortless feature selection process, as most of the work was already done.

5.3.2 Rapid Development for SCARA

The goal is to present a functional model of the SCARA. The requirements state that it must be able to write three characters within two seconds. And to pass System Test 1 it must draw a 50 mm by 70 mm rectangle within one second. The basic design principle is based on the initial design as shown in Figure 5.7. For the lowest detail level of the design, I decided on a kinematics model. The model is very simple as it does not implement any physics. However, the model enables me to tinker with the design parameters, such as the lengths of the linkages and joint angles. In the following steps, the level of detail is gradually increased to arrive at an elaborate model. Planning all the different steps in advance is difficult as design decisions still need to be made. Nonetheless, I can describe at least the following levels of detail for the model:

1. **Basic kinematics model:** forward and inverse kinematics, no physical behavior.
2. **Basic physics model:** ideal 2D physics, ideal joints and rigid bodies with mass and inertia.
3. **Basic motor behavior:** joint actuation with non-ideal DC motor.
4. **Basic control law:** path planning.

After these steps the optimal order of implementation for the levels of detail becomes vague. However, the following elements are required to make an elaborate model:

- Improved motor model
- 3D physics model

When the first design decisions are made, the succeeding levels of detail for these and other elements are laid out.

Evaluation

The current steps in the rapid development are difficult to perform. There is, unsurprisingly, lack of a clear vision of the end-product, which makes an explicit description of every level of detail not realistic. However, it was still possible to describe steps for the initial levels of detail in the design.

The remaining elements, that are essential to the design, take shape in a later stage of the development. Apart from this small deviation, the deliverables of this step are a good start of this development cycle.

5.3.3 Variable-Detail Approach

The following steps is to increase the level of detail of the model. The initial model together with the set of steps in the detail level is inherited from the previous design step. To start, I implement the basic model and implement the different levels of detail. Based on the model after those steps, it is possible to make more detailed design decisions. The decisions make it possible to plan the subsequent levels of detail. Implementing these details results in a competent model.

Basic Design Implementation

The development starts with the basic model shown in Figure 5.10. The model consists of the forward and inverse kinematics of the design. With this kinematics model it was easy to find a suitable configuration of the SCARA. I tested if the SCARA reaches the required operating area, to satisfy system requirement 14. The operating area is a couple of centimeters away from the base of the SCARA. This is to avoid the singularity point that lies at the base of the SCARA. Resulting in the arms being longer than strictly necessary but it reduces the operating range for the angles of the joints, allowing for simpler construction.

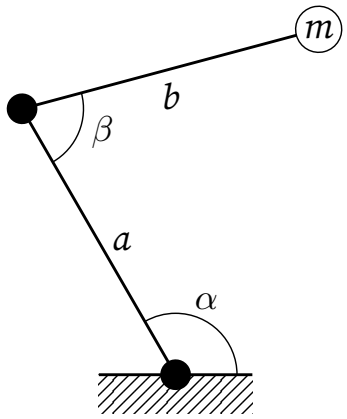

Figure 5.10: Basic kinematics of the SCARA. The arm consists of two linkages a and b; two joints α and β; and a point mass m which represents the end-effector/tool.

At this point, there are already multiple design decisions made about the position of the operating area and the arm lengths. As second detail iteration the basic physics of the model are implemented. The model is in the form of a double pendulum, with two actuated joints. The ideal motors in the joints give the SCARA unlimited acceleration. Replacing the ideal motors with a DC-motor gives an indication about the torque required for operation. Implementing a simple PID-controller allows the SCARA to follow the rectangular path as described in System Test 1. The simulation allowed me to determine the minimum requirements of the motors. The motors must be able to deliver at least 0.2 N m of torque and reach an angular velocity of at least 12 rad s^{-1}.

Detailed design decisions

The basic model gave some valuable insight about the dynamic behavior of the system. However, the current configuration is very simple but requires a motor in the joint. In Figure 5.11, this setup is shown as configuration 1. The disadvantage is that a motorized joint is heavy, which has to be accelerated with the rest of the arm.

Other configurations in Figure 5.11 move the motor to a static position. Configuration 2 is a double arm setup, but has a limited operating range, caused by a singularity region in the system when both arms at the top are in line with each other. Configuration 3 also has such a singularity, but due to the extended top arm this point of singularity is located outside of the operating range. However, this configu-

ration requires one axis with two motorized joints on it. Even though this is possible, it does increase the complexity of the construction. By adding an extra linkage, the actuation is split as shown in configuration 4. Configuration 4 is the preferred option for the SCARA.

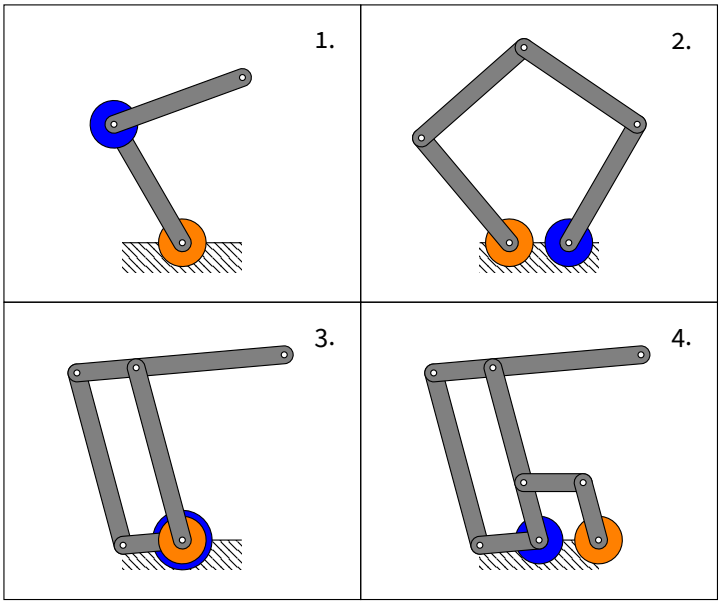

Figure 5.11: Four different SCARA configurations. The colored circles mark which of the joints are actuated. Configuration 3 has two independently actuated joints on the same position.

The current implementation with DC-motors require a feedback controller that compensates for external forces. Such feedback control requires a position sensor for each motor. A simpler solution is to use stepper motors instead. The advantage of a stepper motor is that it is designed to maintain a specific angle. The stepper motors make it possible to use a feedforward controller. This removes the need for a position sensor.

The stepper motors are havier than the DC-motors However, as the new configuration places the motors on the CDC, the additional mass is benificial. The rapid movement of the SCARA creates a reaction force on the CDC. With a heavier CDC, the reaction force results in less movement of the CDC

Unfortunately, the stepper motors are more expensive than simple DC-motors. Nonetheless, the extra costs are easily compensated as it saves development time due to the simplified control law, and the removed need for extra angle sensors used in feedback control.

Due to the aborted implementation of the end-effector, the SCARA must also lift the marker of the board. With the fourth configuration (Figure 5.11), it is possible to add an extra joint in the linkage. As the marker only needs to be moved a couple of millimeters from the board, a simple hobby servo suffices.

Advanced Detail Implementation

The design decisions made in the previous sections, make it possible to plan the next steps of adding detail. The following steps are an addition to the steps as described in Section 5.3.2:

5. **Advanced motor behavior:** Stepper motor behavior.
6. **Advanced physics model:** Updating physics model to 3D physics.
7. **Advanced marker lifting:** Marker lifting behavior, servo lifts marker of the board.

Starting by replacing the DC-motor with a stepper motor model, which is based on a model by Karadeniz, Alkayyali, and Szemes (2018). The controller is updated as well, to accommodate for the behavior of the steppers. The next step is to implement a dynamic model of configuration 4 in Figure 5.11. The dynamics of the SCARA are based on a serial link structure (Stramigioli and Bruyninckx, 2001). This serial link structure makes it easy to add and extend joints, bodies and mass points to the system. Therefore, the last detail, the marker lifting, was added without any difficulty. The servo is connected via a linkage with the marker such that it rotates away from the board.

Component Design

At this point the development has reached a detailed design together with a dynamic model representing that design. The dynamic model is a useful tool to test and evaluate the system behavior. However, it does not include the shapes of the components and can therefore not be used to evaluate clearance or collision between components.

By implementing the design using CAD software, it is possible to inspect for collisions. Furthermore, this model is then also used to print the custom parts. For the mechanical part I used OpenSCAD as CAD software, based on prior experience with the software. With this it was possible to implement all the custom components as well as the commercial off-the-shelf (COTS)-components. To inspect how the components moved, the inverse kinematics model is implemented in the CAD drawing as well. The inverse kinematics made it possible to insert cartesian coordinates, resulting in a dynamic CAD design. Using different orientations of the end-effector allowed me to inspect the clearance between the different components.

Following the rectangular path as defined in System Test 1 revealed that collisions occurred between parts. These collisions were resolved by adding an indentation in one linkage and moving another linkage. These changes are shown in Figure 5.12 The complete setup with the custom parts and the COTS-components, such as stepper motors, servo and marker, is shown in Figure 5.13.

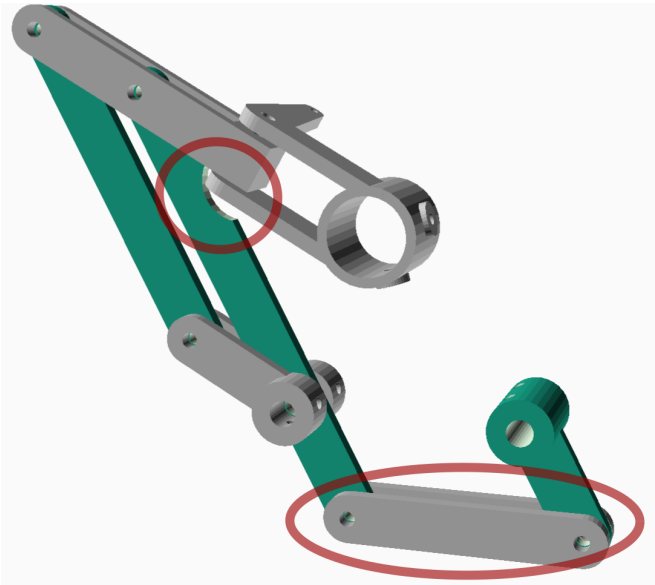

Figure 5.12: CAD of the SCARA configuration, with the end-effector oriented in the lower left corner of the operating area. The configuration has been adapted at the two circled points, to resolve collisions in this orientation. An indentation was made to ensure that the arm can make the required angle. The bottom linkage was located above the joints as depicted in the fourth configuration in Figure 5.11. This was moved to below the actuated joints as it did collide with the end-effector.

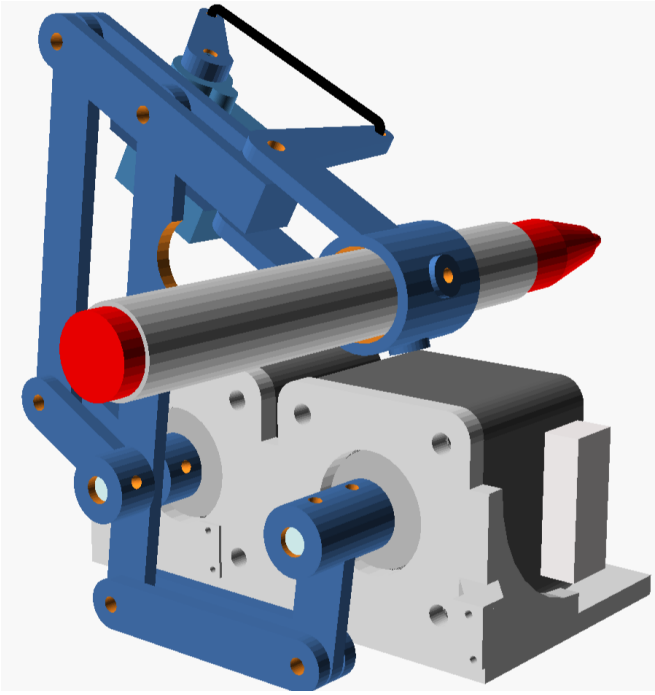

Figure 5.13: Rendered 3D model of the SCARA, including steppers, marker and servo.

Evaluation

The complete development was rather smooth. However, this was not without deviating from the original design plan. It was not feasible to define all different levels of detail before the start of the development. Prior to the design, it was possible to plan 4 levels of detail. After implementing these levels of detail, the design decisions taken made it possible to define additional levels of detail.

In total there are seven predefined levels of detail in the design, meaning that there must also be seven test cycles. However, I noticed that testing occurred more often than seven times. During the design, running the simulation of the dynamics is easy. Resulting in extremely short feedback loops, sometimes even minutes. For example, changing the arm lengths and evaluate the new behavior. Did it improve? Is this as expected?

These small intermediate tests were often implicitly created and are not the tests as specified in the test protocol (). Nonetheless, they provide insight that is valuable for the design process. The interesting question here is whether these small tests should be part of the design process and what it would add to the design process.

5.3.4 Conclusion of Development

At this point, the development of the SCARA is completed. According to the design plan, the next step for the development is the implementation of the CDC feature. However, the evaluation of the development until this point resulted in enough information to draw conclusions about the design plan. I expect that executing this development a third time is not beneficial to the case study, given the additional effort. Time is better spent on the realization of a prototype and evaluating the current design method. Therefore, the next section goes into the construction of the prototype instead of the development of the CDC.

5.4 System Design Validation

To validate the dynamical and mechanical models, I have build a prototype of the current design. For the mechanical design, the CAD model is used to print the custom parts. Other components, such as steppers, microcontroller, screws and miscellaneous electronics, are ordered. To test the dynamics, the steppers and servo have to be actuated. To achieve this actuation a control law has been written in software.

5.4.1 Mechanical Construction

With the 3D printed parts the SCARA was easy to construct. To connect the bodies on the joints, a bolt with washers is used. Although this is clearly not the ideal technique to build joints, it sufficed and was by far the easiest option.

During assembly I noticed that the bolts of a joint and those that hold the stepper motor in place collided. This was possible because the bolts were not included in the CAD-model. In hindsight this should have been included. Fortunately there was enough clearance to mount the SCARA slightly further on the axle. Resulting in an operating SCARA without having to redesign the mechanics.

5.4.2 Control of the SCARA

Although the focus of the design plan was specifically not the software, it still forms an important part of the development. To run the software, I chose for a STM32 microcontroller (MCU), which is a powerful processor with sufficient IO available. The servo motor is directly connected to the IO of the MCU while the stepper motor is connected via a stepper driver board[1], see Figure 5.14. RIOT-OS was chosen as an operating system due to prior experience and available support. To write characters on the board the following tasks are implemented in software:

- Software driver for the stepper controller
- Software driver for servo motor
- Inverse Kinematics Function
- Control/Path planning

[1] IC with H-bridges to power the stepper motor.

The task of the software driver is to handle the communication to the hardware stepper drivers. At initialisation of the software, the hardware stepper driver is configured over hardware. When a new set-point is set in the software driver, the time between each step is calculated. The software driver creates a time-out event with a callback that sends a step signal to the hardware stepper driver. The use of time-out events make it possible to run multiple stepper drivers in software asynchronous.

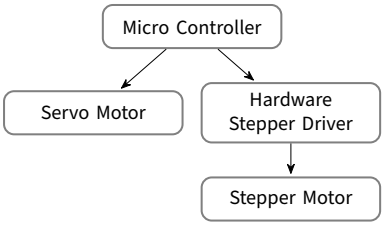

Figure 5.14: Hardware connections. The servo motor connected to an IO-output. The stepper controller connected via UART and IO-pins. The stepper controller provides the correct current for the stepper motors.

The set-point for the software driver is calculated by interpolating the path between the current position and the desired position. This interpolation is necessary to draw a straight line between two points with the SCARA, as a linear movement of the angle would create curved paths. Because the software stepper driver counts the steps send to the stepper motor, which gives the current position of the SCARA. The calculation and update of the next set-point is done with a fixed interval. To calculate the angles that are needed for the set-points a lookup table is used, which replaces expensive trigonometric calculation needed for

inverse kinematics. An advantage of this approach is that it can cope with missed or late event callbacks for the software stepper driver.

The path planning is responsible for the desired position. This can be a rectangle or a set of three characters. The font for the characters is made by Hudson (2015) and consists of a header file with an array of coordinates for each character. When the current position of the SCARA is within range of the desired position, the desired position is updated with the next coordinate of the character.

There are two special elements in the array of coordinates: up and down. These specify whether the marker should be lifted from or lowered on the board. In the transition period of lifting or lowering, there is a short builtin wait for the stepper movement, to avoid unwanted drawing. When the marker is lifted, the interpolation is disabled and the stepper drivers move directly to the position where the next line starts.

For the lifting of the marker the servo on the arm is used. The angle of the servo motor is controlled by the pulse length of a square wave. The software servo driver switches the pulse length when it is ordered to lift or lower the marker. The code for the servo driver is a provide module in RIOT-OS.

In summary, the path planning uses the coordinates of the characters to determine the next desired position and the state of the marker. When a line must be drawn the marker is lowered and the path to the end of the line is interpolated. The position from the interpolation is then converted to angles using the look-up table. The angles are pushed to the software stepper driver, which are used to calculate the interval between steps. The data path for drawing a line is shown in Figure 5.15.

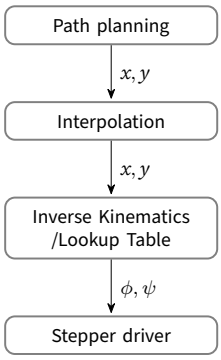

Figure 5.15: Simplified data flow in the software. The path planning generates a vector to the next set point. The interpolation reduces the length of this vector. With the inverse kinematics the required stepper motor angles are generated. These angles are set as new set point for the stepper driver.

5.5 Result

In the end, the development produced eight models with increasing levels of detail and one prototype. The different levels of detail and how they are modelled are shown in Figure 5.16.

As the CDC was not finished, a small stand was build to test the SCARA. The assembled SCARA prototype on the stand is shown in Figure 5.17. This prototype is able to execute the small rectangle as described in System Test 1, and thus passes the test. In addition, it was possible to write three characters. Therefore, passing System Test 4.

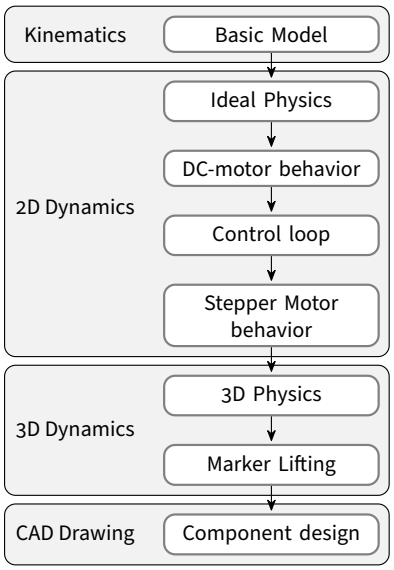

Figure 5.16: Levels of detail of the design are shown on the right, starting with the least detail at the top and most detail at the bottom. Through out the development different types of models are used, these are shown on the left.

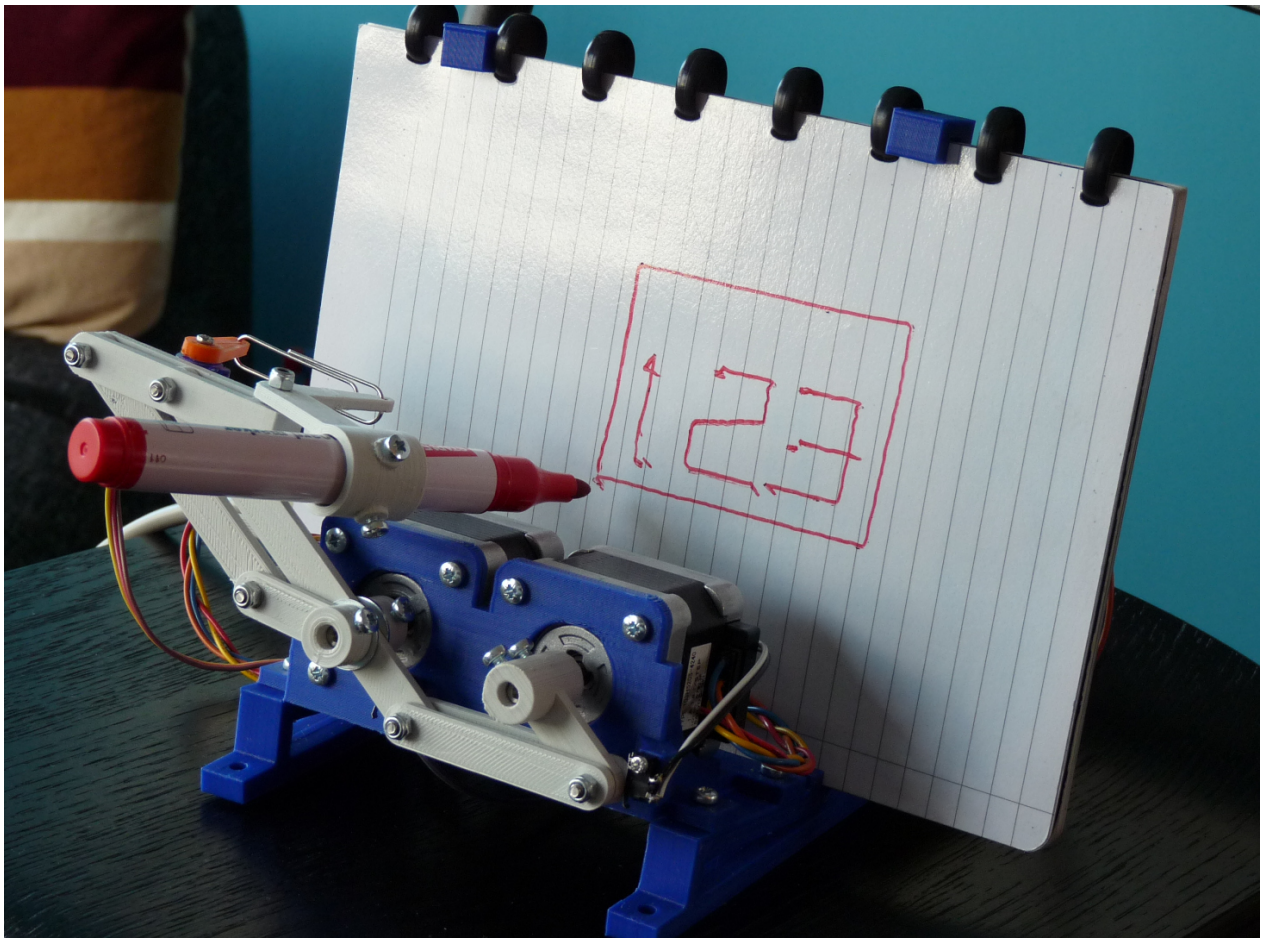

Figure 5.17: Assembled prototype of the SCARA. The SCARA passed System Test 1 drawing the perimeter and passed System Test 4 with the characters 123.

Chapter 6

Case Study: Evaluation

The previous chapter described the development and implementation process of the Whiteboard Writer. This chapter focusses on the evaluation of the development during the case study. The design method itself is evaluated in the next chapter. However, some of the topics discussed in this chapter have a strong overlap with those in the next chapter.

The first section is about the time spend on the development. Followed by a section about the role of stake holders and one about the use of modelling languages during a development. The last section is a more personal reflection about the case study.

6.1 Time Investment

Prior to each step in the development, I made an estimation on the workload of that particular step. In Figure 6.1 the planned and spend time on each step are plotted next to each other. In addition to the steps, time was also spend on the hardware construction and software development.

The initial approach for the feature definition did not result in a statisfactory set of features. Therefore, the approach for the feature definition was reformulated. Before new approach was formulated, multiple attempts were made to get a representative set of features. The time spend on performing the feature definition is not representative as formulating the new approach and creating the set of features were performed in parallel. The real execution time is estimated to be around 3 to 5 days ([1] in Figure 6.1).

Furthermore, there is a significant time difference both development cycles. Prior to the first development cycle I was not confident about the feasibility of the end-effector implementation. Based on that, I decided to spend about three days on the basic model of the end-effector to collect more information. This let me to the conclusion that the end-effector was too time-consuming for this case study. Therefore, first development cycle was cut short ([2] in Figure 6.1)

For the second cycle, I also planned three days to create the basic model. This time, the basic model was finished within a couple of hours. Based on this successful implementation and prior experience, I planned an additional two weeks of development time for this cycle.

To validate the design and model of the SCARA, I build a prototype. This consisted of building the hardware and writing software. Acquir-

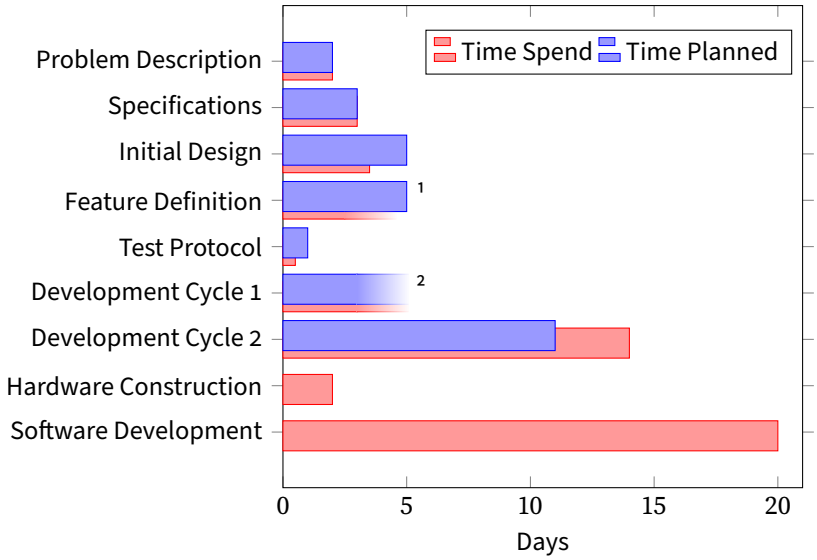

Figure 6.1: Overview of the planned and spend number of days for each step during the case study. Some of the values in this do not represent the time requirements of this design method:
[1] During the feature definition the design method was reviewed. 13 days were spend on this review and execution, obfuscating the actual execution time. The execution time is an estimated 3 to 5 days.
[2] The first cycle was cut short due to its complexity.

ing and assembling the hardware took about two days. This was mainly due to CoViD-19 restrictions which made part ordering and printing more challenging. Without these restrictions I think it would be a day of work.

The time required to get the software to a viable state was four weeks. Even though the focus was not on the software, this timespan of four weeks was too significant to ignore. Especially when the software is compared to the developed models. As explained in the previous section, I build a total of eight models. Each of these models includes documentation and an evaluation of the design process. The software, on the other hand, is in a bare minimum state; I skipped documentation and evaluation; and the code quality relatively low. Still, the software was more time consuming than the hardware modeling and development.

6.2 One-man development team

The case study was performed by me as a single developer. Against all expectations, this one-man development team made the preparation phase more difficult instead of easier. The goal of the problem description and the requirements step is to get the stakeholders on the same line (Shafaat and Kenley, 2015). This involves creating agreed-upon requirements for the system, but with only one stakeholder, this agreement is implicit. Moreover, it undermines the incentive of the problem description and requirements step. Part of this is that there is no penalty for future reviews of the requirements, as I already agreed.

Furthermore, specific details and decisions were often made subconsciously, while I was commuting, waiting in line, or even showering. Making structured documentation of these decisions at a later point in time without missing any of them is impossible. The social interaction within a design team stimulates this documenting process as it improves the recall and interpretation of information. It also improves the judgement and selection of design alternatives (Lamb and Rhodes, 2008).

6.3 Switching Modelling Language

The initial idea of the development was to start with a basic model and extend that model by adding more detail. Meaning that one design and one model would develop in parallel with each other. However, the development of the SCARA resulted in four major model versions. The basic model started with a kinematics model. To take the physics of the design into account, a 2D dynamics model was created. Multiple steps of detail into the development, the 2D model was not adequate anymore. Therefore, the design was remodeled with 3D physics. Although this 3D physics model was able to implement the dynamic behavior, the modeling language was not suitable to design the shape of the mechanical components. Resulting in a fourth model which represents the mechanical component design, in the form of a CAD drawing.

There are a couple of problems with this approach. Implementing the same model with two different modelling approaches, makes both models incompatible with each other. This removes the possibility to switch back to a lower detail implementation. Additionally, it creates the possibility to transfer parameters incorrectly from one model to another. Such a switch is also labor intensive as the complete model has to be build from scratch again. Furthermore, there is the possibility that this new model has been for nothing, as the planned detail proves to be unfeasible. The point is, a future iteration of the design method must minimize these type of model switches to reduce the chance of implementation errors.

6.4 Reflection

In the following section, I reflect on my own impact on the development. The preparation and development phase are discussed separately.

6.4.1 Preparation phase

During the preparation phase often I had difficulty with getting the required information. The information was often not specific enough or it was overlooked. Even though attempting to be thorough, requirements were never really specific. As explained in the previous section, the lack of stake-holders is one of the reasons for information not being specific.

Information that was overlooked created a situation where I needed information that should have been the result of a previous step, which was not the case. In most situations it was possible to continue with the execution of the step. However, during the test protocol step (Section 5.1.5) it was not possible to continue. Resulting in additional requirements added to the design, before continuing with the design process.

One of the main causes that attributes to the information shortage is that I am a novice designer/developer. The experience that I have is from a graduate course and two extracurricular projects. Being inexperienced does definitely not aid the design process. Needless to say, more experience would improve the information situation. However, it does not solve the problem. Further improvements for the design method are required to improve the information process during development.

6.4.2 Development phase

For the development phase I have significantly more experience compared to the preparation phase. Creating models is something that I really enjoy and this improves the process significantly. Even though the development phase went smoother than the preparation phase, there is still room for improvement. Originally I attempted to create separate sub-models for each component. These sub-models can then be combined into larger models. For example, the SCARA and the CDC both include two stepper motors. When I add detail to the stepper motor model, the SCARA and the CDC would then be updated as well. However, each sub-model has to be updated manually. In total four times in case of the stepper motor, which makes this workflow very labor intensive.

A workflow that enables easy combination and interchange of sub-models is beneficial with this design method. It makes it easy to evaluate the latest changes, by comparing them with previous versions. In addition, it makes it possible to lower the detail on some models during the development. The lower detail of the sub-models can improve the simulation speed significantly. And during the final test use the full detail to ensure that every thing is performing as expected.

6.4.3 Continuation of this Case Study

At the point that the SCARA was implemented, I gathered so much new information that some of the design choices felt obsolete. In this case study, the prototype is used to validate the design. However, the current prototype contains so much information that it would improve the requirements and initial design significantly.

Following the current design plan, the next step would be to develop the CDC. In theory, if I would continue the case study, my proposal is to consider the current design as an actual prototype and revisit the preparation phase. However, it is very important to note that this decision relies on the fact that the prototype is already created. In other words, the work is already done and resulted in useful information for a next design iteration.

In case of a different system, I doubt that creating a prototype, followed by a full repeat of the design method is an efficient approach. Therefore, the choice to revisit the preparation phase must not be considered as an improved design method but as an argument to improve the preparation phase itself. However, I think that an improved preparation phase must be shorter and incorporate a prototype.

Chapter 7

Design Method Evaluation

This chapter evaluates the design method as described in Chapter 3. The first section is about the system complexity of CPS. The second section evaluates the elements of a feature. The third section discusses the difference between model and design. The preparation phase and the RIDM are discussed in the last two sections.

7.1 System Complexity

Section 6.1 explains the time resources required for the development of the software in the system. Even though the focus was creating a hardware focussed solution for the ”Tweet on a whiteboard”, the complexity of the software required for this system was underestimated. Royce (1970) also acknowledges this difference in complexity for soft and hardware. He expects 50 pages of software documentation for each page of hardware documentation in projects with comparable budget.

Although the focus was on complex hardware solution, this solution was only possible with the use of software. The interaction between the SCARA and CDC is only possible with software that can switch states. Furthermore, the path planning that writes characters on the board is completely dependent on software as well. Sheard (1998) discusses that pure-hardware solution are relatively simple in their problem space perspective. However, the hardware solution is often complex in the solution space perspective And indeed, during the initial design in the case study, the choice was made for the most complex hardware solution. Even though the hardware is more complex, without software, the SCARA and CDC have no functionality.

Another point on system complexity is prototyping. Because hardware tends to be relatively simple, building a hardware prototype such as the SCARA is cheap and quick. An initial hardware prototype is easily constructed with COTS readily available. Because the hardware transfers power the interfacing between components is trivial. For example, linear actuation can be achieved with a rack and pinion construction, linear motor, gear and chain link, or a connecting rod. This might not be part of the final product, but it is useful to investigate the feasibility of the project.

Furthermore, the changes are also easily made to hardware. It is possible to weld or glue on new parts or remove them with the angle grinder. Adding components to software is tedious and can lead to unwanted behavior. However, this is difficult to test because the software is more

complex. Moreover, unwanted behavior of the hardware is discoverable, and when it breaks it is often destructive. The software can run for multiple days before crashing, as a result of integer, stack or buffer overflows for example.

As long as the development is still in progress, one hardware prototype is more malleable than the software in terms of resources. However, when the designed system is put into production, changing multiple hardware systems becomes economically unviable. A design method for CPS must acknowledge that the inherent complexity of software comes with a high *cost of change* and a high *chance of failure*. Additionally, the design method must use the hardware prototype low *cost of change* to its advantage.

7.2 Elements of a Feature

The design plan as described in Chapter 3 improves the feature definition by adding more structure. The goal of this extra structure was to make a design from scratch possible. A distinction was made between functional features and component features. The functional features are obtained by splitting the functionality of the system, which are then organized in a hierarchical tree. The hardware, which provides a platform for the functionality, is split into component features. These component features form the bottom layer of the hierarchical tree.

Still, the current approach does not provide sufficient structure to define the features of the system effectively. The evaluation of the feature definition (Section 5.1.4) points out that it does not provide any structure for components. It is currently not possible to define subcomponents for components. Furthermore, making connections between a task or mission and a (sub-)component would make the hierarchical structure unclear.

Another point is that the current approach creates a set of requirements and a set of features. The original plan was to distribute the requirements along the features. However, this was more complex than expected and ended up in the background.

A more suitable approach for the definition of features is a SE process that is known as functional decomposition. Kordon et al. (2007) describes this process as a method for structured decomposition of the functionality of a system. Instead of one hierarchical structure that contains functions and components, the process results in three separate hierarchical structures. Each of these structures describe the elements and sub-elements for functionality, physical components and system requirements separately. Between the elements in these structures are connections created that describe the relationships. These relationships describe the link between functions, components and the rationale for requirements.

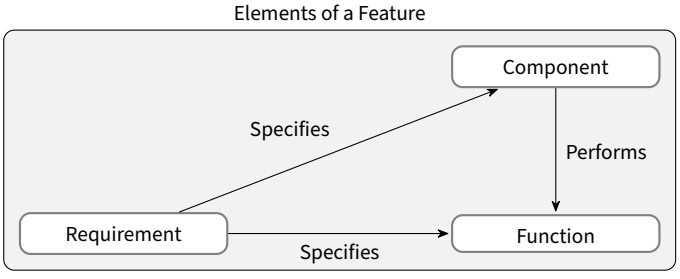

Figure 7.1: Relations and elements within a feature. (Kordon et al., 2007)

Using the structure provided by functional decomposition has a couple of advantages. The current design plan as described in this thesis, considers the feature as a component or a function. As explained in the previous section, the hardware gets its function from the software. Implementing an individual function or component does not deliver a testable feature.

With this new approach, a feature can be formed by grouping the elements that are connected via the defined relationships (Figure 7.1). This feature describes a function that is performed by a component. Furthermore, the requirement specifies both the function and component, and the requirement defines the test of the feature.

Contrary to the design plan in this thesis, the SE process decomposes the functionality of the system over multiple iterations. This is a significant improvement compared to the current approach, in which all requirements were determined before any features was defined. The feature definition during the case study, showed that specific requirements were overlooked. Later, while defining the test protocol, it became clear that no order of operation was specified.

Functional decomposition, or a similar SE process, would not only improve the feature definition step, but the preparation phase as a whole. Future implementations of the RIDM must consider such a process, as it provides a structured method to develop a solution for a problem. Whereby the solution is split in to a elaborate set of features.

7.3 Model and Design Relation

The RIDM as well as the design method in this study do not make an explicit distinction between the model and the design. This implicitly resulted in a model that represents the complete design. Over the course of the development the complexity of the design increased, resulting in more complex modelling as well. The model used in the case study was first implemented as a kinematics model, and as the design became more complex it was represented with 2D and 3D physics, and a CAD drawing.

There are two issues with this approach: first, that the approach does not comply with the general model properties; second, the parameters of the design are represented by multiple models.

7.3.1 Model properties

According to Stachowiak (1973), three general properties apply for a model: first is that the model is always representative to its original; second, the model must only include attributes of its original that are relevant to the respective developer or user; and third, the model must be pragmatic to the original, meaning that models are an adaptation of the original with respect to a specific purpose.

The first property applies to the model because the model represents the full design, and is therefore always representative. However, because the model represents the full design, it violates the second and the third property. The models used in the case study include all attributes of the design and it lacks a specific purpose.

Consequently, the models become overly complex. Especially for larger designs the model complexity drastically decreases simulation speed for dynamic systems. But it also complicates finding bugs in the model.

7.3.2 Design Parameters

The design of the SCARA is currently represented by two types of models: a dynamics model and a CAD drawing. Both these modelling types have different purposes and represent different aspects and parameters of the design. However, both models share parameters of the design as well.

For the SCARA design, the dynamics represent mostly the motor and controller behavoir. The CAD drawing represents the shape of the components. But the kinematics play an important role for both models. A direct result from this is that it increases the *cost of change*. When the design changes, the changes must be applied for both models, increasing the amount of work.

This distribution of design parameters has more disadvantages, as copying the parameters between different models is error-prone and labor-intensive. The case study in this thesis is small, but did already involve 8 different models spread over 4 different modelling approaches. For larger design projects, it is almost given that copying of parameters would result in problems with the design process. Having design parameters distributed across different models is, without any doubt, undesired.

7.3.3 Structured design and models

To solve these problems, the design method must have a strategy for organizing the design and the corresponding models. This consist of a centralized design, which is validated with the use of models. Important is that every model inherits from the design.

The three general properties must apply to every model made. Instead of creating a model of the complete design, only small parts of the design are modelled.

Additionally, a method to organize all design parameters reduces the *cost of change*. The goal is that all the models automatically use the design parameters from a centralized location. Any changes to the design are made at that centralized location, each model can than be tested automatically with the updated parameters. This eliminates copying of parameters and allows for automated testing. It removes the human factor and produces direct feedback about the design change.

7.4 Preparation Phase

Initially adding a preparation phase to the RIDM was not within the scope of the thesis. Causing me to underestimate the role that the preparation phase had for the RIDM. In hindsight it is clear that the preparation phase is crucial for the design process, and thus also for the evaluation of RIDM. The linear set of steps were chosen as it was trivial to put those in front of the RIDM. However, the linear set of steps proved to be inapt for the development of complex CPS.

Without a concrete approach[1] to get from a *problem* to a functional design with features, the RIDM is unsuitable for the development of CPS. Describing such an approach is far outside of the scope of this thesis. Nonetheless, several SE processes offer a possible solution, such as functional decomposition, state analysis (Ingham et al., 2005) or spiral model (Boehm, 1988). Furthermore, the advantages of modern

[1] Here, I specifically use the term approach because preparation phase implies that it must be a phase prior to the RIDM.

techniques of rapid prototyping should also be considered to aid the design process.

A possible candidate for the approach is to use a spiral model as basis and apply the techniques of RIDM, rapid prototyping and functional decomposition in that basis. Another option is to extend the development cycle of RIDM with functional decomposition and rapid prototyping. Further research is required to determine the optimal approach for the RIDM. This research should use data and experience from existing design projects. Above all, the design of such an approach needs direct involvement of experienced systems engineers.

7.5 Rapid Iterative Design Method

This chapter began by a breakdown of the elements of a feature, argued the importance of distinction between design and model, and explained the need for an integrated preparation phase. The commonality between these three issues is that they all stem from the rapid development cycle, which was introduced in Section 2.2.1 as part of the RIDM. It is apparent that the current implementation of the rapid development cycle is not suited for the design of a cyber-physical system. Further studies, which take these issues into account, should be undertaken.

Even though, these issues have a large impact on the overall performance, they must not overshadow the rest of the design method. The feature selection step and variable-detail approach did show a positive contribution to the design method. The following sections discuss their performance and what potential impact an improved rapid development cycle introduces.

7.5.1 Feature Selection

The goal of the rapid development is to process a list of features into a competent model. In this case, the list of features was produced in the preparation phase. The features are then, one by one, implemented according to the variable-detail approach. To determine the order of feature implementation, I specified a feature selection protocol, which is explained in Section 3.2.1. Based on this case study, the feature selection is a suitable addition to the design method. Especially for the failed feature implementation as described in Section 5.2.2. Would the SCARA have been implemented first, a failure in the end-effector might result in a required redesign of the SCARA feature. However, with only two uses during this case study, caution must be applied.

Of the criteria used in the selection, the COF-time factor and the dependees are, in my opinion, most relevant. The dependees is a hard criterium: if there are any features that it depends on, but not yet implemented, it cannot be selected. Otherwise the feature would be implemented before the required information is available. As explained in Section 5.2.2, the feature selection approach aims to clear the largest amount COF in the smallest amount of time possible. However, between the dependees and the COF-time factor, there is a criterium for the number of tests, which could hinder this approach. The current approach would result in the situation where a feature with lots of easy-to-pass tests, is implemented before a features with less, but more difficult tests. It is then possible to spend a lot of time on something that is very likely to pass anyway.

This does not alter the fact that to complete the design all tests have to pass. That all test have to pass is also the reason for this criterium in the first place: give priority to the feature that passes the most tests on completion. Even though it is difficult to draw concrete conclusions about the feature selection, a recommendation is to use the number of tests and the change of failure for each test as a metric to calculate the COF-time value. In addition, other metrics and approaches that can improve the COF-time calculation are: number of dependees, the number of tests of those dependees, and planning poker. Further work is required to establish which metrics are most suitable to calculate the COF values.

7.5.2 Variable-Detail Approach

The variable-detail approach is a very practical development tool. A note of caution is due here since the variable-detail approach has not been used to its full potential. The goal was to add detail to a feature in strictly defined steps. Between each step the tests are applied to the updated model. Based on the test, the development continues or the model is rolled back to an earlier version. In addition, the models, independent of the level of detail, can be reused in other models.

However, multiple difficulties were encountered during the case study that hindered the variable-detail approach. As was mentioned in Section 6.4.2, the lack of good version control made it difficult to work with multiple versions of a model. This made it difficult to switch or revert to other levels of detail. However, the greatest difficulty is due to the model representing the design, as discussed in Section 7.3. Because the design contains a high level of detail and the model is a full representation of the design, it is difficult to make a simple implementation or to switch back. This strong relation between the model and the design, also caused the complete model to be switched to a different representation.

Even though the variable-detail approach did not perform as planned, I expect this approach to be a very strong part of the design method, given that a solution is found to the problems described above.

Chapter 8

Conclusion

8.1 Case Study

Extend the RIDM with a preliminary design phase, focussing on the physical part of CPS.
To get from a given problem or idea, to an initial design that can be used by the RIDM, a linear set of steps was added. This set consists of a problem definition, requirements and initial design step. These steps are based on the SE-approach.

Refine the RIDM to make the design steps more explicit with improved instructions.
To perform a reproducible evaluation of the RIDM, the method of the different design steps were defined more explicit. The RIDM specifies the development cycle and the variable-detail approach with sufficient detail, making them ready to use. How to define features and tests for the development cycle, were not as clearly defined. In this thesis, two steps were added to the design method: one with a method to define the set of features and one that is used to specify the test protocol. Two design steps were added in this thesis that describe a method to define the set of features and create a test protocol. Furthermore, a feature selection step was added to aid with the development.

Develop and perform a case study that tests and evaluates the RIDM as a design method for the physical part of CPS.
The case study consisted of the development of a *Tweet on a Whiteboard* writer. This development is performed according to the design plan, that was the result of the first two research objectives. The *tweet on a whiteboard* writer was chosen as subject of design based on a set of requirements. The goal of these requirements is to find a subject of design that evaluates most aspects of the RIDM when implemented.

A list of questions was formed to monitor the progress of the case study. The questions are answered before and after each step of the design process. The list was created to ensure a consistent documentation of the expected outcome and the actual outcome of each step. Both the expected and actual outcome are used to evaluate the design step.

8.2 Rapid Iterative Design Method

Assess the influence that applying the RIDM has on the design process for CPS.

The core of the RIDM consists of the development cycle and the variable-detail approach. Both of these methods have specific influence on the design process.

The development cycle introduces a feature-based approach to the development process. With the development cycle the system is implemented feature by feature. This requires the development team to split the functionality of the system into features. It forces the developers to go through the design in a structured manner. Furthermore, to determine in what order the features are implemented, the developers must establish the *cost of change* and *chance of failure*-metric for each feature.

Based on the *chance of failure* and *cost of change* metrics, the features are ordered with the aim to reduce the impact of a design failure. Even though the case study only applied the feature selection twice, it proved itself useful by selecting the end-effector feature first. By prioritizing the end-effector, its failure had only a minor impact on the design.

During each iteration of the development cycle, the selected feature is implemented according to the variable-detail approach. However, the ability to assess the influence of the variable-detail approach is limited by the absence of tooling for model organization and testing. Without tooling that is compatible with version control, it is difficult to switch between model versions, undo design changes, or run automated testing. Furthermore, as the development did not distinct between design and model, the models used often contained more detail than strictly necessary. Both these limitations resulted in models that would surpass the minimal required level of detail; therefore, it is not possible to assess whether the minimum required level of detail can be established with passing all the tests. Nevertheless, the variable-detail approach introduces a step wise addition of detail that enforces a structured method similar to the development cycle.

It is unfortunate that the development cycle did not include a structured method to define the features nor their order of implementation. The performance of the variable-detail approach is currently hindered by the absence of tooling. Consequently, this limits the accuracy of the assessment on the actual influence of the RIDM. Notwithstanding these limitations, the results of the case study suggest that the structured approach of RIDM reduces the impact of design failures and reduces the development cost for CPS design.

Describe which adaptations are required for both the RIDM and the design method to establish a competent design process for CPS.
The RIDM required adaptations before it could be evaluated. The adaptations made in this thesis showed variable degrees of success during the case study. To create a competent design process, some adaptations must be improved and some new ones must be added.

The produced result by the development cycle depends strongly on the provided features to implement. To ensure a consistent result for the design of CPS, RIDM must incorporate a design process to define these features. Moreover, the features must not only describe functionality, but each feature must also include a physical component and a requirement. These three elements together make it that features can be implemented and tested individually.

The design process must describe a complete method from problem description till the set of features. In this design process, the solution to the problem is established in the form of the functionality of the system. The design process determines what components perform

that functionality, and puts requirements on the components and the functionality. All design decisions made during this process shape the final product. Therefore, the design process to determine the features is urgently important in the ability of RIDM to successfully develop CPS from scratch.

The variable-detail approach requires adaptations to fully utilize the advantages that the short cycle and testing provide. Therefore, the models must be separated from the design. This requires a centralized design including a database for all design parameters. Models are no longer required to represent the complete design, allowing for more specific models. Moreover, the models can conform to the general model properties. Because the models are more specific, more of them are required to cover all aspects of the design.

To manage the increased number of models a form of version control is needed. The version control makes it possible to organize, and if necessary to combine and integrate, different models. Furthermore, it makes it possible to revert design changes and switch to different model versions.

The final adaptation is the ability for automated testing. Automated testing provides a major advantage on top of the previous adaptations. As the models inherit their parameters directly from the centralized design database, every design change propagates to all models. With automated testing, all model are simulated after a design change. This highlights any unwanted behavior caused by the design change. As the models are made more specific, a failed simulation of a model automatically pin points the area where the problem occurs.

Implementing and evaluating the adaptations as described above are required to determine if these adaptations are sufficient. The next section describes recommendations that must be considered before implementing these adaptations.

8.3 Recommendations

Before any of the adaptations are applied to the RIDM, further research on the exact format of these adaptations is recommended. The recommended steps taken in further research are:

- *Make the application area and purpose of the RIDM specific*: To design a good design method, the design process must start with a clear problem description. Currently, the RIDM does provide a design method, but does not state clear requirements such as:
  - *Type of CPS*: mainly hardware, software or control?
  - *Design focus*: improve reliability, real-time guarantee, reduce complexity, shorten development-time?
  - *Internal or external use*: is the client directly involved?
  - *Development team*: number of developers from what background?

  These requirements improve the focus of further research. This focus could also help to attract and involve other organizations. Above all, it prevents the RIDM of becoming a ”master of none”.
- *Explore existing design projects that share the application area and purpose*: To avoid inventing the wheel or or provide a solution none wants, it is recommended to explore existing design

project. Involve projects from companies and universities, successful and unsuccessful. Evaluate all the projects with at least the following questions:

  - What type of design paradigm or model is being applied?
  - Where is the complexity in the project and how is it dealt with?
  - How are the metrics of *cost of change* and *chance of failure* defined?
  - How are the design and models connected?
  - Which design tools are used by the design team? Why are they used?
  - Are there common design problems between the different projects?
  - How is the client involved in the development process?
  - What considerations are made to chose between modelling or hardware prototyping?

- *Hypothesize the improvements provided by RIDM for existing design projects, and vice versa*: Based on the evaluation of the design projects:
  - How could the RIDM improve those existing design projects?
  - What lessons can be drawn from the existing design projects?

Depending on what the results and conclusions of the recommended research topics are, a strategy has to be created to further develop the RIDM. Currently, there are two likely scenarios:

- Make the RIDM part of an existing design model, such that the advantages are integrated with existing design models.
- Develop the RIDM into a complete design method, such that it can be used for the development of the complete product life-cycle.

Independent of what strategy is chosen, it is recommended to:

- Implement the adaptations as described in this thesis.
- Perform the adaptations and improvements of the RIDM with a multi-disciplinary design team.
- Evaluate the RIDM with projects that are within the application area of the RIDM.

The recommendations result in a more focussed development of the RIDM. But these recommendations are only the top of the iceberg of what is required to develop RIDM as a complete design method for CPS Expected is that a full development of the RIDM takes multiple years and many developers and researchers to complete.

The RIDM does bring some techniques that show potential. These techniques could improve existing design methods. Based on this thesis, the following research topics are recommended:

- *A technique or protocol for to organize the parameters of a design, such that the parameters can be used in modelling*: Can the current modelling tools be adapted to read parameters from a database and can design tools be adapted to write parameters to a database?

- *Tooling for modelling software, to allow for unit testing*: Software development applies unit testing, where behavior of each function is tested separately. In modelling this would allow every submodel to be tested separately.

Appendix A

Test Specifications

System Test 1: Small rectangle

During this test, a rectangle will be drawn on the whiteboard using the SCARA. This rectangle is 50 mm high and 70 mm wide, such that three characters fit within the rectangle. To test the speed requirements, the rectangle should be drawn within one second.

Features: SCARA
Requirements: 3, 7, 11, 13, 14
Results: The test passes when:

- Rectangle height is at least 50 mm
- Rectangle width is at least 70 mm
- Completion time is less than 1 s

System Test 2: Perimeter

The Cable bot must move along the outer edges of the text area. This area consists of three lines of text with fifty characters each. Resulting in a perimeter of 1000 mm wide and 250 mm high. This proves that reach of the system is sufficient to write the text. Furthermore, the Cable bot should move outside of the perimeter as well. Moving outside of the text area is to prove that Cable bot has a position where it does not obstruct the written text.

Features: Cable Bot
Requirements: 1, 2, 6, 11, (12)
Results: The test passes when:

- The Cable bot moved along the edge of the text area.
- The Cable bot moved outside of the text area.

System Test 3: Cable Bot Speed

The Cable bot must be able to move a distance of 80 mm in horizontal direction within a second. At the start and the end of the movement the speed of the Cable bot must be zero. This is to ensure that the SCARA can then write three characters at the given position.

Features: Cable Bot
Requirements: 7, 9, (12), 14, 15, 16
Results: The test passes when:

- At the start and end of the test, the Cable bot does not move relative to the board.
- During the test, the Cable bot has moved 80 mm within 1 s.

System Test 4: Triple Chars

The SCARA together with the end-effector must write 3 characters without moving the Cable bot. This extends on the small rectangle of System Test 1 but the end-effector must now be able to lift the marker of the board. The three characters should be written on the board within two seconds.

Features: SCARA, End-Effector
Requirements: 3, 4, (12), 13, 14
Results: The test passes when:

- The SCARA wrote three characters on the white-board within 2 s.
- The Cable bot did not move more than 10 mm.

System Test 5: Tool Change

The system has to switch in some way between the marker and a wiper, or a different color. For this test the system must switch a tool within 10 seconds.

Features: SCARA, End-Effector
Requirements: (5), (12), 17
Results: The test passes when:

- A tool is released from the end-effector and stored for later use.
- A different tool is attached to the end-effector.
- The tool switch is completed within 10 s.

System Test 6: Repeatability

This tests if the Writer can draw repeatedly on the same position, for different approach angles, on the board. The system will start with drawing multiple 60 mm squares on the board in a random location. To test the repeatability, a circle with a 55 mm diameter must be drawn inside of the square. This should be done with twenty squares in an area of at least 1000 mm x 300 mm. The drawing order of each square must be different from the drawing order of circles, this ensures that the Cable bot makes a different approach path.

Features: SCARA, Cable Bot
Requirements: 3, 4, 9, 11, (12)
Results: The test passes when:

- Each square has a circle drawn inside.
- The squares and circles are within 5 mm of their given dimensions.
- All the circles are completely within their corresponding square.

System Test 7: Linearity

The system must draw a grid on the drawing range (1000 mm x 300 mm), with the horizontal and vertical lines spaces 100 mm from each other. The distance between two horizontal or two vertical lines cannot be smaller than 90 mm or larger than 110 mm. The lines are not allowed to deviate more than 10 mm in a line section of 300 mm.

Features: SCARA, End-Effector, Cable Bot
Requirements: 1, 2, 3, (12)
Results: The test passes when:

- All lines are drawn, 11 vertical and 4 horizontal lines.
- All lines in parallel separated from their neighbor by atleast 90 mm and atmost 110 mm.
- Each line does not deviate more than 10 mm.

System Test 8: Writing

To test the complete writing abilities the following text must be written on the board:

```
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG!?@,.-
0123456789101112131415161718192021222324252627282
the quick brown fox jumps over the lazy dog!?@,.-
```

This is a full 150 character area. It must be readable and write all the characters correctly. It must be completed withing 150 seconds. Which is 2 minutes and 30 seconds.

Features: SCARA, End-Effector, Cable Bot
Requirements: 1, 2, 3, 4, (5), 6, 7, 12, 13, 14, 15, 16
Results: The test passes when:

- The text as described is readable from a atleast 4 m distance.
- The text is writen on a clean whiteboard within 150 s.

System Test 9: Wiping

The complete board must be cleared of any marking within 60 seconds. This is without the change of tool.

Features: SCARA, End-Effector, Cable Bot
Requirements: (5), 10, 11, 12
Results: The test passes when:

- The system cleaned the board within 60 s.

System Test 10: Complexity

The last test is that the design has to be complex enough. This has to be evaluated by the developer of the system.

Features: SCARA, End-Effector, Cable Bot
Requirements: 8
Results: The test passes when:

- The developer can motivate that the system is complex enough to evaluate the case study.

Appendix B

System Requirements

This appendix gives an overview of all the system requirements for the Whiteboard Writer that defined during the Case Study. Table B.1 gives an overview over which test correlates to which requirements.

System Requirements	1. Small rectangle	2. Perimiter	3. Cable bot speed	4. Triple Chars	5. Tool Change	6. Repeatability	7. Linearity	8. Writing	9. Erasing	10. Complexity
1		●					●	●		
2		●					●	●		
3	●			●		●	●	●		
4				●		●		●		
5					○			○	○	
6		●						●		
7	●		●					●		
8										●
9			●			●				
10									●	
11		●				●		●	●	
12	○	○	○	○	○	○	○	●	●	
13	●			●				●		
14	●		●	●				●		
15			●					●		
16			●					●		
17					●					

Table B.1: Correlation between system requirements and tests. A closed dot indicates a singular relation. An open dot represents a spread relation. In other words, to meet a requirement atleast one test with a closed dot must pass or all tests with an open dot must pass.

System Requirements:

1. The Writer must be able to write at least fifty characters per line.
2. The Writer must be able to write at least three lines of text.
3. The Writer must plot characters with a size that is readable from 4 meters for a person with good eyesight.
4. The Writer must plot in a regular used font with corresponding character spacing.
5. When a new tweet is send to the Writer, the Writer must wipe the existing tweet and write down the new tweet.

6. If the Writer is not wiping or writing then the Writer must not obstruct the view of the whiteboard.
7. While writing, the Writer must have a writing speed of at least one character per second.
8. The dynamics of the Writer must be complex/sophisticated/interesting.
9. If the Writer is tasked to wipe the tweet, the Writer must wipe the tweet within sixty seconds
10. When a reset-signal is send to the Writer, the Writer must recalibrate its position on the board.
11. When a wipe-signal is send to the Writer, the Writer must wipe the board clean.
12. The Writer must not damage itself.
13. While writing, the SCARA must have a writing speed of at least 1.5 characters per second.
14. When the CDC is at a static position, the SCARA must be able to write at least three characters at that position.
15. When the SCARA finished writing at their current position, the CDC shall move the SCARA to it's next position where it can write the subsequent characters.
16. When the SCARA has to be moved to a new position, the CDC shall perform this movement within one second.
17. When the system changes from writing to erasing or vice-versa, the SCARA and End-effector should change the tool within ten seconds.

Bibliography

Blanchard, Benjamin S and W. J Fabrycky (2014). *Systems engineering and analysis*. ISBN: 978-1-292-03839-1. URL: http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=1418193 (visited on 10/19/2020).

Boehm, B. W. (May 1988). "A spiral model of software development and enhancement". In: *Computer* 21.5, pp. 61–72. ISSN: 1558-0814. DOI: 10.1109/2.59.

Broenink, T. G. and J. F. Broenink (June 11, 2019). "Rapid development of embedded control software using variable-detail modelling and model-to-code transformation". In: *Communications of the ECMS: Proceedings of the 33rd International ECMS Conference on Modelling and SimulationECMS 2019*. 33rd International ECMS Conference on Modelling and Simulation 2019, pp. 151–157. URL: https://research.utwente.nl/en/publications/rapid-development-of-embedded-control-software-using-variable-det (visited on 11/12/2019).

Fitzgerald, John, Peter Gorm Larsen, and Marcel Verhoef, eds. (2014). *Collaborative Design for Embedded Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-642-54117-9. DOI: 10.1007/978-3-642-54118-6. URL: http://link.springer.com/10.1007/978-3-642-54118-6 (visited on 09/03/2020).

Grenning, James (2002). *Planning Poker or How to avoid analysis paralysis while release planning*. URL: https://wingman-sw.com/articles/planning-poker (visited on 03/22/2021).

Hudson, Trammell (2015). *Asteroids font*. Asteroids font. URL: https://trmm.net/Asteroids_font/ (visited on 01/25/2021).

Ingham, M. et al. (2005). "Engineering Complex Embedded Systems with State Analysis and the Mission Data System". In: *J. Aerosp. Comput. Inf. Commun.* DOI: 10.2514/1.15265.

Karadeniz, Ahmet, Malek Alkayyali, and Péter Szemes (Apr. 6, 2018). "Modelling and Simulation of Stepper Motor For Position Control Using LabVIEW". In: *Recent Innovations in Mechatronics* 5. DOI: 10.17667/riim.2018.1/7.

Kordon, Mark et al. (Mar. 2007). "Model-Based Engineering Design Pilots at JPL". In: *2007 IEEE Aerospace Conference*. 2007 IEEE Aerospace Conference, pp. 1–20. DOI: 10.1109/AERO.2007.353021.

Lamb, Caroline T. and Donna H. Rhodes (2008). "2.2.1 Collaborative Systems Thinking Research: Exploring systems thinking within teams". In: *INCOSE International Symposium* 18.1, pp. 222–233. ISSN: 2334-5837. DOI: 10.1002/j.2334-5837.2008.tb00802.x. URL: http://onlinelibrary.wiley.com/doi/abs/10.1002/j.2334-5837.2008.tb00802.x (visited on 09/23/2020).

Mavin, Alistair et al. (Aug. 2009). "Easy Approach to Requirements Syntax (EARS)". In: *2009 17th IEEE International Requirements Engineering Conference*. 2009 17th IEEE International Requirements Engineering Conference, pp. 317–322. DOI: 10.1109/RE.2009.9.

Rajkumar, Ragunathan (Raj) et al. (2010). "Cyber-Physical Systems: The next Computing Revolution". In: *Proceedings of the 47th Design Au-*

tomation Conference. DAC '10. New York, NY, USA: Association for Computing Machinery, pp. 731–736. ISBN: 978-1-4503-0002-5. DOI: 10.1145/1837274.1837461. URL: https://doi-org.ezproxy2.utwente.nl/10.1145/1837274.1837461.

RobMoSys (May 5, 2017). RobMoSys. URL: https://robmosys.eu/approach/ (visited on 01/21/2021).

Rosen, Aliza (Nov. 7, 2017). *Tweeting Made Easier*. Tweeting Made Easier. URL: https://blog.twitter.com/official/en_us/topics/product/2017/tweetingmadeeasier.html.

Royce, Dr Winston W (Aug. 1970). "MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS". In: *IEEE WESCON*, pp. 1–9.

Shafaat, Ali and C. Robert Kenley (2015). "Exploring the Role of Design in Systems Engineering". In: *INCOSE International Symposium* 25.1, pp. 357–370. ISSN: 2334-5837. DOI: 10.1002/j.2334-5837.2015.00068.x. URL: http://onlinelibrary.wiley.com/doi/abs/10.1002/j.2334-5837.2015.00068.x (visited on 09/23/2020).

Sheard, Sarah A. (1998). "7.18. Systems Engineering for Software and Hardware Systems: Point-Counterpoint". In: *INCOSE International Symposium* 8.1, pp. 928–936. ISSN: 2334-5837. DOI: 10.1002/j.2334-5837.1998.tb00131.x. URL: http://onlinelibrary.wiley.com/doi/abs/10.1002/j.2334-5837.1998.tb00131.x (visited on 09/09/2020).

Stachowiak, Herbert (1973). *Allgemeine Modelltheorie*. Wien: Springer. 494 pp. ISBN: 978-3-211-81106-1 978-0-387-81106-2.

Stramigioli, S. and H. Bruyninckx (May 2001). "Geometry of dynamic and higher-order kinematic screws". In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164). Vol. 4, 3344–3349 vol.4. DOI: 10.1109/ROBOT.2001.933134.