UTRECHT UNIVERSITY

Designing the Automated Greenhouse

Matching Requirements and Architecture for Startup Product Specification Using Epic Stories

by
Remmelt Blessinga
(remmelt@aruku.io)

Supervisors
Prof. Dr. Sjaak Brinkkemper (s.brinkkemper@uu.nl)
Dr. Fabiano Dalpiaz (f.dalpiaz@uu.nl)

A thesis submitted in partial fulfillment for the degree of
Master of Business Informatics

Faculty of Science
Department of Information and Computing Sciences

In cooperation with
Aruku B.V.

December 2018

Abstract

Within the requirements engineering domain, user stories have become a commonly used lightweight approach to document requirements. However, critics highlight the limitations of user stories. To overcome these shortcomings, epic stories and Jobs-to-be-Done were developed. This study investigates if epic stories can be used to specify the functional software architecture of a startup by applying the Requirements Engineering for Software Architecture method. A case study is conducted to test the RE4SA method at the Dutch startup Aruku B.V. by documenting requirements and creating a functional software architecture. The results are used to analyse the limitations of RE4SA and its potential as a tool for software specification.

Keywords: Epic Stories, Jobs-to-be-Done, Software Requirements Engineering, Functional Software Architecture, RE4SA.

Contents

1. Introduction

The world population is continuing to grow at a tremendous speed. By the end of this century planet Earth is expected to host between 9 and 13 billion humans [0], up from the 7.2 billion currently alive and a tremendous increase from the 2.5 billion alive in 1950 [1].
In addition to a higher population, more people than ever will be living in cities, giving rise to an increasing number of megacities that need to be supplied with all sorts of resources [5].
This larger and wealthier urbanized population will have to overcome, amongst other threats, environmental degradation, climate change, and resource depletion [3]. This in addition to having to grow an ever increasing amount of agricultural products on a diminishing supply of land with increasingly scarce fresh water [4].
This poses new challenges, and thus economic opportunities for businesses. The Netherlands already holds the key to overcome some of these challenges with its advanced greenhouse technology. This was highlighted by the article "this tiny country feeds the world" in National Geographic Magazine [6]. Indeed, despite having limited natural resources, including land, the Netherlands has managed to become the second biggest net value exporter of agricultural products in the world, exporting more than Russia and China combined and only second to the United States of America [2].
The author of this work stipulates that innovation is required to overcome the challenges of the 21st century. Various routes of innovation are available, such as reducing environmental footprint by improving production efficiency, overcoming labor shortages through robotization, and finding new ways of managing greenhouses and trade in produce using artificial intelligence. In addition, these solutions need to be exportable to other nations without loss of competitive advantage.
This stipulation resulted in the idea of the startup Aruku B.V. to construct components for a fully autonomous greenhouse that can be gradually developed and introduced within the Netherlands in cooperation with local farmers, who traditionally have worked together very well to share knowledge and innovate. This gradual approach will allow Aruku to quantify implicit knowledge of farmers, and automate human labor processes that require a high degree of education.
Eventually, when proven to work in the Netherlands, Aruku products can be exported to other nations to enable local production without the need for long distance flights or shipping, and without the need for highly trained staff, an ideal situation for megacities that do not have access to knowledge readily available in the Netherlands.
To get to this point of automation, first an understanding has to be obtained of what processes exist and need to be automated and how such automation can satisfy the needs of farmers. This poses a challenge for both requirements engineer and software architect who both approach the design of such a complex system in a different way, making the development of a roadmap a difficult challenge with little scientific literature to aid them.

1.1 Problem Statement

The design of components for the autonomous greenhouse through the unification of existing hardware systems, software, and knowledge processes within an Aruku platform is a complex long term project that requires extensive experimentation and research. Often, the requirements of the end user of these solutions are still uncatalogued and difficult to transform into a software architecture effectively.
The identification of user stories has proven to be an effective lightweight approach to requirements engineering [9], which in turn is a solid base for agile software development [10]. However, some criticism highlights limitations of user stories. The Utrecht University research group GRIMM has defined job stories as an alternative that embeds goal-oriented principles by emphasizing situation, motivation and the expected outcome [11]. Job stories were renamed epic stories to better differentiate between the concepts of Jobs-to-be-Done, jobs,

and job stories, and unify the existing concept of epics with that of job stories. As such, the term epic stories is used from now on to describe job stories [20]. One proposed method for using epic stories as a basis for software development is by adopting them as a basis for a software architecture similar to how user stories are often linked to software features. Just like epic stories can be seen as roadmap themes for user stories, so we can see software architecture modules as themes for features. The same parallelism can be tested by linking epic stories to software architecture modules as we link user stories to features. However, currently no research has been done to validate the hypothesis that epic stories can be mapped to software architecture modules, and no case study has been conducted to investigate what obstacles might be encountered.
This study is important as it can help define how a requirements engineer and software architect are able to help each other by using epic stories as a basis for software architecture modules to better satisfy the needs of stakeholders. In addition, it is possible that such a relation will form a foundation for product roadmapping. Currently little research has been done on epic stories, although various industry thought leaders have written about this concept, yet a solid scientific base has been established to work from. In addition, no direct link between Jobs-to-be-Done theory and software architecture was found in the reviewed works on Jobs-to-be-done and epic stories, more on this in the literature review.

1.2 Objective, Scope, and Structure

The objective of the proposed research is to study whether epic stories are an effective way to come to a software architecture using the Functional Architecture Model [12] and subsequently developed utrecht Architecture Definition Language (uADL) [14]. This will be done as a case study, allowing Aruku B.V. to effectively plan ahead for its design and development process in a goal oriented fashion. The end result of this study is a method and case study to show that epic stories can be used as a basis for software architecture design by making an attempt to use epic stories, linked to jobs from the Dutch greenhouse sector, as a basis for functional software architecture modules of a specific Aruku product under development. In turn this will show if user stories, grouped under epic stories, are suitable for a similar mapping to architectural features that are grouped under the modules of a software architecture. A secondary result of this study will be the insights gained into this method as a tool for roadmapping product development.

The study maintains the following structure; first a literature review is conducted in chapter 2. This is followed by the definition of a research method, defined in chapter 3. Chapter 4 outlines the proposed solution in detail. Chapter 5 describes the results of the case study. Chapter 6 contains an analysis of the results. The conclusions are written in chapter 7, followed by a discussion of the study in chapter 8.

2. Research Approach

2.1 Research Questions

The goal of the proposed study is to investigate whether epic stories are a good basis to design a software architecture. This is tested with a case study of the autonomous greenhouse components Aruku B.V. is developing and will result in a generic architecture for a specific product. The study will be exploratory in nature and contain a large design element to validate conclusions and recommend further work.

The main research question that will need to be answered is formulated as follows:

> MRQ - How can the requirements and software architecture for the software specification of a startup be based on epic stories?

This question, when answered, will tell if there is indeed a applicable relation between epic stories and software architecture modules. The startup component was explicitly mentioned as startups, in this case Aruku B.V., have unique limitations, specifically that they often design products from scratch, which allows for a more comprehensive review of this method than for example a redesign of software would at a large corporation, where a plethora of stakeholder concerns might influence the results of this study.

Before this question can be answered, several sub-questions will need to be answered first as a better understanding of the domain needs to be obtained and several hypotheses need to be tested.

First, a better understanding is required on what epic stories are, how they relate to requirements engineering and what is already known about their relation to specifying an architecture. This gives the appropriate background context for the study and forms the basis of the literature review. In addition, an understanding is required of what constitutes a satisfactory product specification for a startup.

> RQ1 - What is currently known about requirements engineering in relation to specifying a software architecture?

A good view has to be obtained of how epic stories are placed in the requirements engineering domain and how this domain is connected to that of software architecture specification. The answer to this research question will provide an overview of current practices used to define software architectures based on stakeholder requirements. It is expected this will relate primarily to the mapping of user stories to features, as the domain of epic stories in relation to software architecture is relatively new.

> RQ2 - What are the requirements of a startup when defining product specification?

The answer to this question will allow the definition of criteria to validate the case study and establish if, and when, the use of epic stories satisfies the requirements of a startup.

To be able to use epic stories, and the related user stories to specify an architecture for Aruku B.V. a set jobs, user stories and epic stories should be identified as part of a case study. Not everything is known on how to best relate epic stories to software architectures, resulting in the formulation of the following research questions.

RQ3 - How can epic stories be employed to create a software architecture using the utrecht Architecture Definition Language?

The answer to this question will establish the basis to answer the main research question. It will tell if there exists a connection between epic stories and architecture. When a better understanding is obtained of, if, and how epic stories can be mapped to a software architecture, the theory can be validated by answering this question as part of the case study.

When the architecture is completed, an analysis should tell if the architecture is actually usable to startups as a product definition, for this a level of detail needs to be established to see if anything is missing or redundant. This is done in cooperation with the coding team at Aruku B.V. and by utilising the knowledge obtained from RQ2 to answer the following question.

RQ4 - How effective are the results of this study when used for startup product specification?

To further solidify the mapping between requirements engineering and software architecture, a high-level look is taken at the relation between jobs and product as supported by the mapping of epic stories and architecture modules to see if a similar relation exists here as on the lower level of user story and feature.
The answer to this final question will determine if the end result of the case study meets all the requirements of the startup and determine if the established process is suitable as a basis for further development of the end product.

2.2 Research Design

A logical structure for solving practical problems is the regulative cycle. This structure originates in design science. Design science emphasizes the connection between knowledge and practice by showing that we can produce scientific knowledge by designing useful things. The regulative cycle was developed by Wieringa [21]. It starts with an investigation of a practical problem, itself the outcome of solving earlier practical problems; then specifies solution designs, validates these, and implements a selected design; the outcome of which can then be evaluated, which could be the start of a new turn through the regulative cycle.

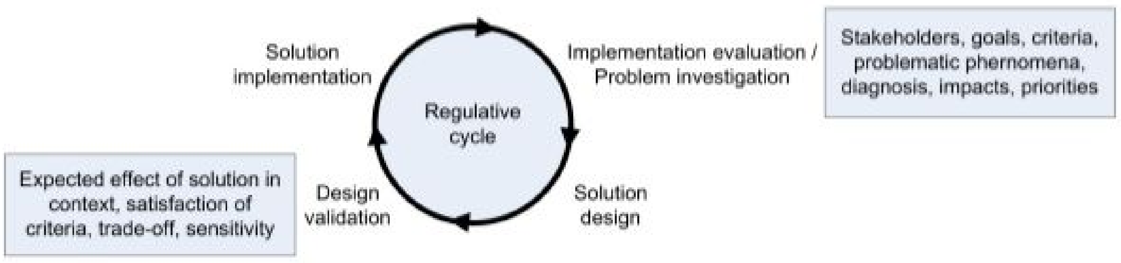

Solution implementation
Implementation evaluation / Problem investigation
Stakeholders, goals, criteria, problematic phernomena, diagnosis, impacts, priorities
Regulative cycle
Expected effect of solution in context, satisfaction of criteria, trade-off, sensitivity
Design validation
Solution design

Figure 1 - Regulative cycle by Wieringa

2.2.1 Problem-driven investigation

Stakeholders experience problems that need to be diagnosed before solving them [21].
In the first phase of the regulative cycle the problem investigation starts with a description and explanation of these problems. In the case of this research this leads us to the main research question.

MRQ: Can epic stories be used to define architecture modules for product specification of a startup?

In this study, the problems are the shortcomings of user stories and how epic stories can be a solution to these shortcomings. In particular it is unclear if epic stories can serve as a basis for defining a software architecture. To properly understand the domain and be able to explain the problems experienced by stakeholders, an explanation will be given of what constitutes a user story, an epic story and how these relate to specific software architecture concepts as defined in the utrecht Architecture Description Language. This is done in the first phase of this study as part of the literature review and will answer research question 1 *RQ1 - What is currently known about requirements engineering in relation to specifying a software architecture?*
When this is finished, criteria will be defined that will have to be met before we can accept the solution to the main research question. To define these criteria, first an understanding must be developed of what constitutes a satisfactory outcome for the subject of the case study, in this case a startup. This is the answer to research question 2, *RQ2 - What are the requirements of a startup when defining product specification?*

This part of the study will take the form of a literature review that will first look at scientific publications on user stories and epic stories, and how these are currently used in requirements engineering. Then several papers are investigated to obtain an overview of software architecture models and principles that can be of use to this study. Papers will be grouped according to common themes and gathered using a rudimentary snowball approach.

2.2.2 Solution design

In this part of the study, a solution is designed that will provide an answer to the main research question once it is tested. In other words, it will state how to solve the problem defined in the first phase.

A definition is made of method to formulate epic stories based on the work of van de Keuken et al. [11], and a theory is proposed on how to map these to software architecture modules. A decision is made on what architecture components will be modeled to test the mapping of epic stories to FAM modules. Reasoning is established to determine how detailed the architecture needs to be to be of use to the stakeholders as a solution to the stated problem. This will result in the core theory implemented in the case study.

2.2.3 Design validation

Once the solution has been designed it will be tested for internal and external validity and trade offs will be considered. By looking at internal validity an assessment is made if the design, when implemented in this context, will satisfy the criteria identified in phase 1.
Following the internal validity, a look is taken at external validity, in other words, would the designed solution work in a different context than the one being used in this study?
Finally, trade offs are considered and the design is compared to different designs or variations on the current design, what are the trade offs? Will these alternative solutions also meet the criteria?

Feldt and Magazinius [59], go as far as to consider seven different types of validity threats in empirical software engineering research commonly addressed in research. Five of these are relevant to this study.

1. Internal validity: Did the treatment/change we introduced cause the effect on the outcome? Can other factors also have had an effect?
The effect in this research will be the working theoretical basis for a method of mapping requirements to an architecture. It is possible that architecture on its own can serve as a roadmap, but it will lack the link to requirements as explicitly formulated in stories and jobs. As such the researcher of this paper sees no other factors that could have an effect on the outcome.

2. Construct validity: Does the treatment correspond to the actual cause we are interested in? Does the outcome correspond to the effect we are interested in?
The relation between requirements engineering and software architecture will be shown by applying the method in practice during the case study, this will result in a set of jobs, epic and user stories, and architectural views. In turn these will be analysed by the Aruku code team. The outcome of this analysis will show that the theory can be accepted as valid.

3. External validity, Transferability: Is the cause and effect relationship we have shown valid in other situations? Can we generalize our results? Do the results apply in other contexts?
4. Dependability: Are the findings consistent? Can they be repeated?
Because the study will result in a applicable method, applied to a specific case to demonstrate its validity, with a solid theoretical framework to support the conclusions, it will be possible for other practitioners to replicate the results of this study using different cases by replicating the steps from this study.

5. Confirmability: Are the findings shaped by the respondents and not by the researcher?
The end results of the study will be analysed with a team of coders working at Aruku B.V., their contribution will be a vital independent view of the study outcome and allow for findings shaped independently from the researchers opinion.

2.2.4 Implementation

In the implementation phase the solution is applied to the domain context. In the case of this study, the design is used to create a software architecture of the Aruku autonomous greenhouse components using epic stories as a basis. This will result in a catalogue of jobs, epic stories and user stories and their mapping to software architecture using the uADL.
For the definition of jobs and epic stories the method by van de Keuken et al. [11] is employed, using the quantitative path shown in the analysis section of the PDD shown in figure 2.

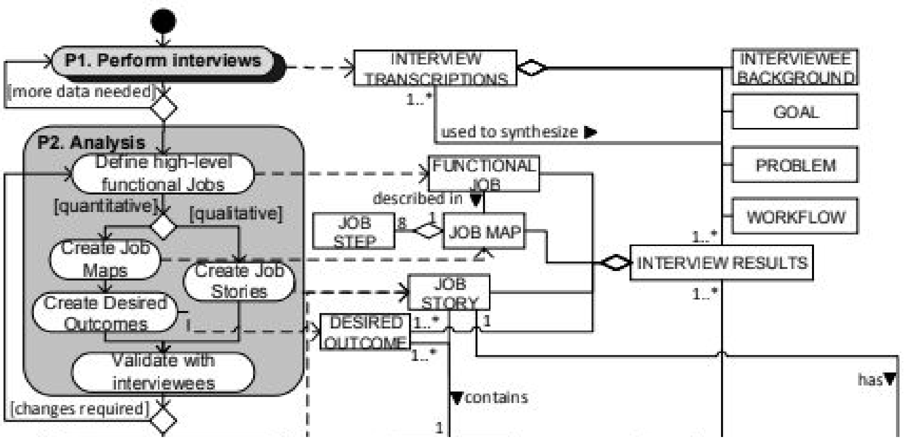


Figure 2 - epic story analysis process deliverable diagram by van de Keuken et al. [11]

First information is gathered from stakeholders in the form of document review and interviews. This information is then analysed to discover high-level functional jobs that using the template; *[Help me] [verb] [noun plural].* Using a quantitative approach epic stories are written using the template; *When [problematic situation], I want (to) [motivation], so that (I can) [expected outcome].*
Once the jobs and epic stories have been established, they are described using a product brief. Which will show what links epic stories have to specific jobs and should give a good indication of functionality groups for the software architect.
When the project briefings have been finished, the method defined in the design is employed to create a software architecture using the utrecht Architecture Description Language [14]. This is done partially by the

Aruku product manager and the Aruku software architect. Table 1 shows what viewpoints will be modeled and by whom. Be executing the theory it will allow the formulation of an answer to the third research question, *RQ3 - How can epic stories be used to create a software architecture using the utrecht Architecture Definition Language?*

	Product Manager	**Software Architect**
Context	x	
Functional Architecture	x	x
Feature Diagram		x
Scenario Overlays	x	
Deployment	x	x

Table 1 - viewpoint development task division overview

The results are used to discuss with all Aruku stakeholders if the solution is deemed adequate for the problem, this evaluation will give new insights on the limitations of the proposed theory and any possible concerns with its implementation. This will answer the fourth research question, *RQ4 - How effective are the results of this study when used for startup product specification?* This answer will allow to formulate an answer the main research question. This part of the research will result in the results and analysis section, where new findings are presented in the results and interpretations are presented in the analysis.

2.2.5 Repeat if implementation hits obstacles.

If obstacles are encountered during the case study, these are used as problems to repeat the cycle by finding a solution for the obstacle. This allows to further refine the research and explore in depth the limitations of the implementation. The findings from this process will be added to the results section, or if so required, in the discussion section.

3. Literature Review

For this literature review a total of 56 papers, shown in table 2, were investigated to answer research question 1, *What is currently known about requirements engineering in relation to specifying a software architecture?* The literature review is divided in four categories, first the domain of epic stories in relation to requirements engineering and software engineering is explored. Subsequently, available literature in relation to user stories is reviewed. Finally, a look is taken at software architecture practices. A base set of five papers is identified using google scholar, internal documentation used at Utrecht University, and literature recommended by various courses offered by the university curriculum. These papers allow for a further snowball approach, selecting relevant literature from the reference sections where necessary. Various other papers were recommended or found outside of the snowball approach. For example relevant course materials for related courses at Utrecht University were consulted to uncover relevant information.

Original	**Snowball**
[11] G. Lucassen, M. van de Keuken, F. Dalpiaz, S. Brinkkemper, G. Sloof and J. Schlingmann, "Jobs-to-be-Done Oriented Requirements Engineering: A Method for Defining Job Stories", *Requirements Engineering: Foundation for Software Quality*, vol. 10753, pp. 227-243, 2018.	33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48
[22] G. Lucassen, F. Dalpiaz, J. Werf and S. Brinkkemper, "The Use and Effectiveness of User Stories in Practice", *Requirements Engineering: Foundation for Software Quality*, vol. 8997, pp. 205-222, 2016.	49, 50, 51
[23] G. Lucassen, F. Dalpiaz, J. van der Werf and S. Brinkkemper, "Improving agile requirements: the Quality User Story framework and tool", *Requirements Engineering*, vol. 21, no. 3, pp. 383-403, 2016.	52, 53, 54
[24] G. Lucassen, F. Dalpiaz, J. van der Werf and S. Brinkkemper, "Improving User Story Practice with the Grimm Method: A Multiple Case Study in the Software Industry", *Requirements Engineering: Foundation for Software Quality*, vol. 10153, pp. 235-252, 2017.	55, 56
[25] G. Lucassen, F. Dalpiaz, J. van der Werf and S. Brinkkemper, "Visualizing User Story Requirements at Multiple Granularity Levels via Semantic Relatedness", *Conceptual Modeling*, vol. 9974, pp. 463-478, 2016.	57, 58

Table 2 - Snowball overview

3.1 User and Epic Stories

To answer *RQ1 - What is currently known about requirements engineering in relation to specifying a software architecture?,* a better understanding is required of what epic stories are. Due to this domain being very young in terms of scientific research, it is required to further investigate the origin of the concept, namely user stories and Jobs-to-be-done theory, as well as various articles by industry thought leaders. User stories in particular, when investigated, will offer a better understanding of the requirements engineering domain, allowing the answering of RQ1.

3.1.1 User Stories

Lucassen et al. present in the paper titled The User and Effectiveness of User Stories in Practice [22], an exploration of how practitioners employ user stories and perceive them. Based on 182 valid responses and follow-up commentary the researchers came to various key findings. The most relevant findings for this study are that the template proposed by Connextra, shown in figure 3, is both prevalent and more effective for requirements engineering than not using a template, that technical stakeholders are less positive on the use of user stories than non-technical stakeholders and that overall user stories are considered to be a good medium to ensure the creation of the right software.

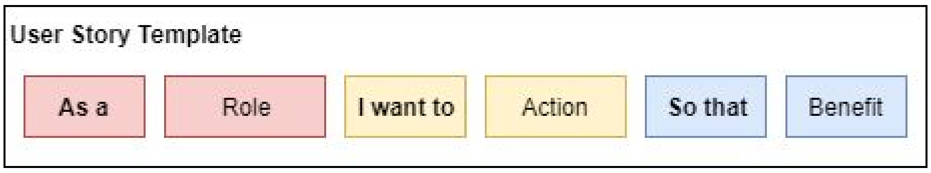


Figure 3 - User Story Template

In his book on applying user stories [34], Cohn identifies various factors that come into play when writing good user stories. Most important of these are that user stories have to be written in an active voice, for one user, be unnumbered, keep the purpose in mind and contain a role. On his website [70], Cohn gives the following example of a good user story:

*“**As a** user, **I can** indicate folders not to backup **so that** my backup drive isn't filled up with things I don't need saved.”*

Cohn states that anyone can write user stories, but that it is generally the product owner’s responsibility to make sure a product backlog of agile user stories exists. He also states that user stories are written throughout a project, and that no specific moment exists for their definition in the project lifecycle.

Bik et al. work with sixteen software producing organizations to construct a reference method for user story requirements. The study states that little was known about activities and artifacts preceding and following user story related activities, making it unclear for organizations how to fully utilise their user stories. Captured in a process deliverable diagram the study identified five phases, requirements gathering, user stories, development, testing, and deployment [28].

Klement [33] claims that one of the biggest problems with user stories is that too many assumptions are made and that they do not acknowledge causality. He states there is no room to ask the why, and that you are essentially locked in a sequence without context. In addition he argues that the persona element does not add anything of value, although this is highly disputed. One reason for this dispute is the survey by Wang et al. [35]. The survey shows that agile requirements engineering practices, including the use of user stories, play a crucial role in agile development and that they are an important prerequisite for projects’ success though many agile methods advocate coding without waiting for formal requirements and design specifications. As a solution Klement proposes the epic story describing situation, motivation and expected outcome, this results in the template shown in figure 4.

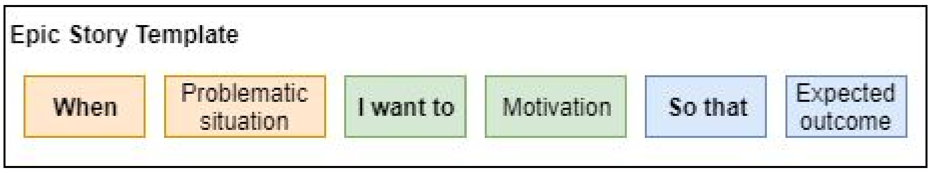


Figure 4 - Epic Story Template

While very similar, Mike Cohn, illustrates that user and epic stories can be different: “As a technique to represent what a software system is supposed to do, user stories are argued to be superficially the same as alternative solutions: requirement statements, use cases and scenarios. However, the large amount of alternatives suggests, that the small differences between the solutions are in reality very important. The variation consists of what is written down, and when” [34]. On his website he does not specify a particular template for an epic story, but defines the concept as *“generally too large for an agile team to complete in one iteration”* [70], giving the following example of what he considers to be an epic:

*“**As a** user, **I can** backup my entire hard drive.”*

.

3.1.2 The origin of epic stories

At the base of epic stories is the Jobs-to-be-done theory. Intercom has made this theory the foundation of its product and marketing strategies, conducting exhaustive research to learn more about their customers, they too illustrate the problems that Klement sees with user stories [36]. Perhaps one of the more influential works in the inception of epic stories and jobs theory, is the book *When Coffee and Kale Compete*, by Klement [37]. In this book an overview is offered on the origin of jobs theory followed by the perceived challenges and benefits of using jobs, such as distributed decision-making, insights into customer requirements and improved understanding of the data required for innovation. Jobs are The book contains three case studies, which give insights into the applicability of jobs to be done theory, such as the challenges a requirements engineer will encounter.

3.1.3 Epic stories in detail

One work in particular is of great importance to this review, that of van de Keuken et al. [11, 15]. In which epic stories, called job stories by van de Keuken as based on the work of Klement [33], are defined and analysed through a case study.
Most important in this study, is the conceptual model of an epic story, shown in figure 5, and the integrated job story method, shown in figure 2.

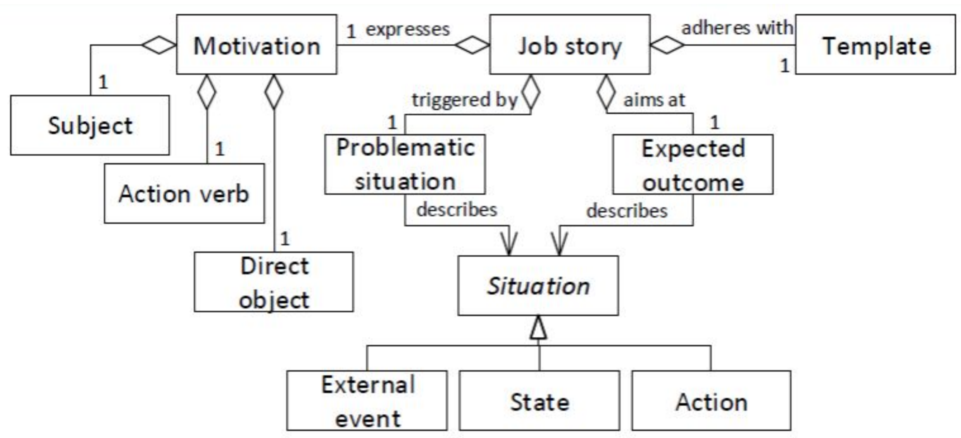


Figure 5 - Conceptual Model of Epic Stories (after Van de Keuken et al. [11])

The study concluded that product managers considered that epic stories emphasize understanding of why and how customers use products. Epic stories help scope a product and improve communications with stakeholders. When exploring a new market segment they considered the possibilities for opportunity prioritization very valuable and project briefs to help promote a creative design process. However, they considered practical applicability limited due to dependence on customers to do time-consuming analyses. Developers considered epic stories too high level to be very useful, yet marketeers thought the method to be very valuable, especially to convey why a feature is valuable for a target audience [15].
In an article on the website Medium by Tessman [45], epic stories in practise are discussed. The main benefit is marked as a greater understanding of the design problem without leading the design. Tessman gives various examples of epic stories, such as:

***"When** I'm looking at perspective campuses, **I want to** know what the campus looks like, **so I** can envision myself walking to class or studying on the grass."*

One of the bigger challenges is stated to be granularity of the epic. Prepositions, such as using *"in the video"* or *"during the video,"* can be a clue that you need to go deeper into the epic and take another look at workflows, for example by creating a sub-epic. A second challenge is the scope of the outcome, which often has multiple solutions, the author suggests using empathy with the subject to see if an optimal choice exists, but that the various outcomes will often give you valuable insight into your epic. In another Medium article [42], Klement claims that a well described 'when' is essential for an epic story, as otherwise the epic story becomes too vague. He refers to this as the struggling moment. Epic stories that contain problematic situations such as *"When I'm hungry"*, or *"When I want to check my email"* are too vague to be helpful. Instead he suggests to rewrite these stories into *"When I'm hungry, running late to get somewhere, not sure when I'm going to eat again, and worried that I'll soon be tired and irritable from hunger…"* and *"When I want to check my email, but don't want anyone around me to know I'm checking my email becuase they'll think I'm being rude…"*

3.1.4 Jobs-To-Be-Done

In their MITSloan management review article, Christensen et al., citing the Jobs-to-be-done theory, describe the benefits executives can reap when segmenting their markets by job. Secondly, they describe the methods that those involved in marketing and new-product development can use to identify the job-based structure of a market. Finally, they show how business plan details can become more coherent when jobs to be done are properly understood [38].

Ulwick takes a similar approach in his Harvard Business Review article, claiming that customers should not be trusted to come up with the solutions to their problems, and as such should only be asked for outcomes. In the method developed at his company, Cordis, the focus is explicitly on the desired outcomes of certain tasks [39]. In another article by Ulwick, the relation with Jobs-to-be-done theory is much more prominent. The article discusses a universal job map, which is created to define vision and direction, to discover opportunities, and guide customer needs gathering [40]. This, and other Jobs-to-be-done theory work by Ulwick is culminated in his book, *Jobs To Be Done [41]*. The book discusses in detail the theoretical framework behind Jobs-to-be-done and several frameworks and methods for employing the theory in practice, illustrated by case studies. The construction of an outcome driven organisation is deemed ideal by Ulwick, who states it can be achieved in three phases: understanding customers using jobs to be done, discovering hidden opportunities in the market, and using new customer insights to drive growth. Ulwick and Hamilton [44] propose the Jobs-to-be-done strategy growth matrix. In this work they consider that success of an innovation is dependent of the selection and successful execution of the right strategy. The matrix they propose can show companies which product strategies are best for them once they know the dynamics of their market.

In a Linkedin article [43], Blair takes a contrarian approach from Ulwick and states his biggest problem with jobs theory and epic stories are its simplicity. Blair states that he believes many practitioners will not differentiate properly between jobs, needs and benefits, and as such the value will get lost and people will get stuck in their own paradigms. The biggest issue he believes, is that anything qualifies as a job.

There are three main currents in Jobs-to-be-done publications, the first shown is the scientific approach which shows the practical applications but also limitations of the Jobs-to-be-done theory, such as in the work by Christensen. The second current is that of Klement and Ulwick, who actively advocate and expand Jobs-to-be-done and have published books on the theory. The third is the current in which Blair presides, that of the critics, which doubt the practicality of the theory and wonder if it isn't too ill restrained.

3.1.5 Understanding User Needs

At the basis of Jobs-to-be-done theory are user needs, which are attempted to be captured in jobs, this section looks at how user needs can be understood as the basis of the jobs theory.
To help designers develop a limited set of salient scenarios, Potts [46] proposes a schema similar to story schemata. Like stories, scenarios have protagonists with goals, they start with background information already in place, and they have a point that makes them interesting or tests the reader's understanding. The scenario schema provides a structural framework for deriving scenarios with slots for such teleological information. Scenarios are derived from a description of the system's and the user's goals, and the potential obstacles that block those goals. Potts describes the scenario schema and a method for deriving a set of salient scenarios, illustrating how these scenarios can be used in the analysis of user needs for a multi-user office application. In his work, van Lamsweerde reviews various research efforts undertaken into the domain of goal-oriented requirements engineering, using a case study to suggest what a method might look like. Lamsweerde concludes that there are many advantages to goal-oriented requirements engineering. Object models and requirements can be derived systematically from goals, goals provide rationale for requirements, a goal graph provides traceability from high level strategic concerns to low level technical details and more [47]. Bebensee looks as how product managers can sometimes be confronted with a continuous stream of incoming requirements and examines how to prioritize requirements. The paper demonstrates a technique that can be used to this end, comparing it with Wiegers matrix. The conclusion states that the developed technique is suitable for prioritization, as tested at two small software development companies. However, limitations were seen with missing consideration of dependencies between requirements and a one dimensional criterion that determines prioritization [48].

3.1.6 User Story Methods

Wautelet et al. state that standard user story templates can be problematic to use as none of them define any semantic related to a particular syntax precisely or formally leading to various possible interpretations [58]. The study studies templates in order to reach unification in the concepts syntax, an agreement in their semantics as well as methodological elements increasing inherent scalability of user story based projects.
Through a multiple case study, Lucassen et al. propose the Quality User Story Framework [23], consisting of 14 quality criteria that user should strive to conform to. This framework allows practitioners to discern between high and low quality user stories in an empirical fashion using natural language processing. The criteria could be a basis for this study to evaluate the quality of written user stories. The effect of applying the Grimm Method's QUS framework and the AQUSA tool into existing user story practices was examined through a multiple case study by Lucassen et al. [24]. Although the overall number of quality defects decreased, no meaningful change was perceived by participants. As such the researchers could not conclude anything meaningful to this study.
Software has become so complex and it evolves so quickly that we fail to keep it under control. A paper by Huisman et al. [26] proposes intents, fundamental laws that capture a software systems' intended behavior. Summarizing, intents have the goal to have all stakeholders express, understand, and prioritize their expectations of the software. Intents must be verifiable, and this verification should bridge the gap between detailed code and

its high-level, emerging properties. Ultimately intents are intended to regain a grasp on rapidly complexifying software, providing insight and making it possible to hold developers accountable for their product.
The paper agile human-centered software engineering by Memmel et al. [27] bridges the gap between software engineering and human-computer interaction by indicating interfaces throughout the different phases of SE and HCI lifecycles. The study concludes that using use scenarios, a basis for user stories, and prototypes are fundamental for the design process. A cycle was developed to assist in bridging the gap between the domains, but a practical study is yet to be done on its implementation.
Ryan looks to identify some phases and tasks where natural language processing may be usefully applied in requirements engineering [52], specifically the formulation of user stories. The study concludes that the results suggest the validation of requirements must remain an informal, social process.

3.1.7 Visualizing User Stories

Despite the readability of text, it is hard for people to build an accurate mental image of the most relevant entities and relationships. As the number of requirements and concepts grows, obtaining a holistic view of the requirements becomes increasingly difficult. In visualizing user story requirements at multiple granularity levels via semantic relatedness by Lucassen et al. [25], the researchers implemented an automated approach to visualizing user story requirements at different granularity levels via semantic relatedness. The paper is primarily focused on visualizing groupings of user stories from a requirements engineering perspective, while this study approaches a similar problem using software architecture features and their grouping through architectural modules, visualized in a functional architecture diagram.

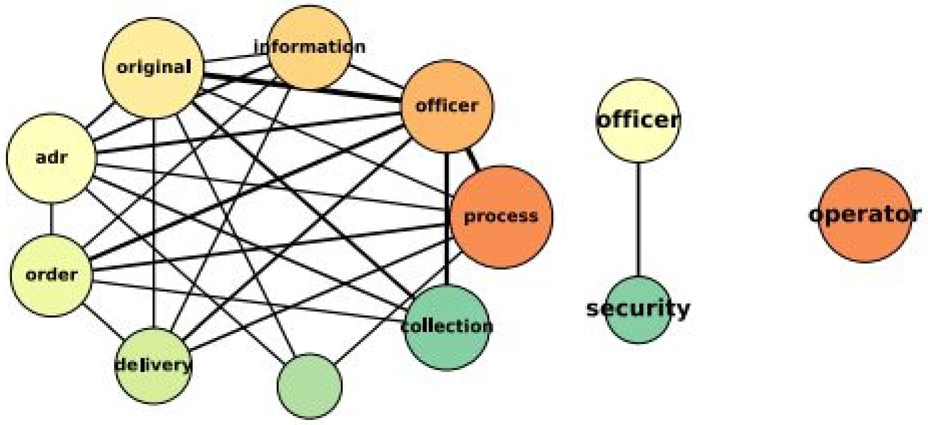


Figure 6 - Visualization of representative user story terms (by Lucassen et al. [25])

Mesquita et al. conducted a study titled US2StarTool: Generating i* Models from User Stories, in which they explore the problem of context visualization of user stories, which reduces understanding of the system as a whole [50]. The researchers attempt to visualize user stories using i* models, placing them in a view similar to that of the contextual architecture diagram. The study concluded that mapping user stories to i* models could be performed successfully. Relevant to this study is that the relation between user stories and software architecture is apparently inherent.
Landhausser et al. present a tool that builds an ontology for code and links completed user stories in natural language with the related code artifacts [53]. The tool also contains links to API components that were used to implement the functional tests. Results showed that reusable test steps for new user stories can be recommended based on these links.

3.1.8 Key Factors In User Story Based Requirements Engineering

In *Requirements Engineering As A Success Factor In Software Projects*, by Hofmann and Lehner, a look is taken at the software specification process from a requirements perspective [49]. An analysis was made of the contributions to project success by the vital factors of knowledge, resources, and process. They conclude that carefully selected teams, an experienced project manager, and consultation with domain experts are vital contributions to a successful requirements engineering process. According to Gomez et al. the order in which user stories are implemented can have a significant influence on the overall development costs [54]. Their paper presents a systematic and lightweight method to identify dependencies between user stories, aiding in the reduction of their impact on the overall project cost. The method proposed does not add extra load to the project and reinforces the value of the architecture. Damian et al. report on the findings of an investigation into industrial practice of requirements management process improvement and its positive effects on downstream software development [55]. A strong relationship between a well-defined requirements process and increased developer productivity, improved project planning, and enhanced ability of stakeholders to negotiate project scope was revealed. The study offers empirical evidence to the importance of sound requirements practice.

3.1.9 User Stories In The Requirements Engineering Domain

In their systematic mapping study on empirical evaluation of software requirements specification techniques, Fernandez et al. identify what aspects of software requirements specification are empirically evaluated, in which context, and by using which research method [51]. Understandability was determined as the most commonly evaluated aspect and experiments the most common research method. In their empirical study, Wang et al. study in depth the role of requirements engineering in agile development [56]. Their survey shows that agile requirements practices play a crucial role in agile development and that they are an important prerequisite for project success though many agile methods advocate coding without waiting for formal requirements and design specifications. Kassab states that although there is ample information available on solid requirements engineering practices, anecdotal evidence still indicates poor practices in industry [57]. Through three surveys between 2003 and 2013 on the state of requirements engineering practice a comparison and analysis is made in order to understand the changing landscape over the years. The emergence of the agile as the dominant paradigm was evident. Overall the study showed that projects employing requirements practices obtained higher satisfaction even though they do not appear to impact the quality of the end product.

Investigating these works has provided valuable insights into the origin and state of both epic and user stories, as such we can partially answer RQ1 as follows.
Beyond the relation between these concepts as presented in agile requirements engineering practices, no direct link between Jobs-to-be-done theory and software architecture was found in the reviewed works on Jobs-to-be-done and epic stories. Epic stories and software architecture are rarely mentioned in the same study, despite the indicated relationship between both as shown in the reciprocal twin peaks model. However, the domain of epic stories has been subject to professional discussion on blogs for some time. Indeed, epics in general are an established concept but lack a greater scientific framework showing what kind of benefits they can provide.

3.2 Software Architecture

To answer RQ1, *What is currently known about requirements engineering in relation to specifying a software architecture?,* an understanding must be obtained of software architecture. Internal documentation from Utrecht University is used as a starting point, this concerns primarily work on the uADL (utrecht Architecture Definition Language). This study of architectural practices mainly employs knowledge from the book Software System Architecture: Working with Stakeholders Using Viewpoints and Perspectives [18] and Software Architecture in Practice by Bass et al [71]. These books were read in their entirety and provide many insights on how requirements and architecture relate, primarily through modeling notations that can be employed to formulate a software architecture. In addition the Utrecht University course on software architecture was a valuable source of information. Most relevant of these findings were that concerns of stakeholders are always the basis for the development of an architecture, as shown in the Twin and Three Peaks models [15,16].

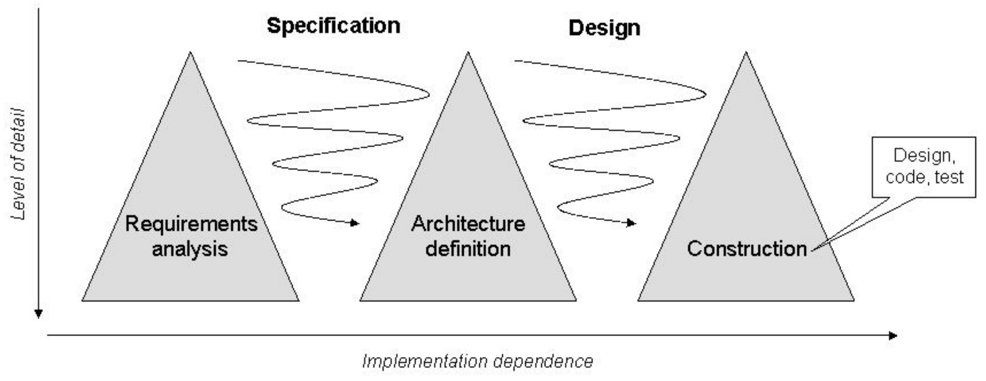
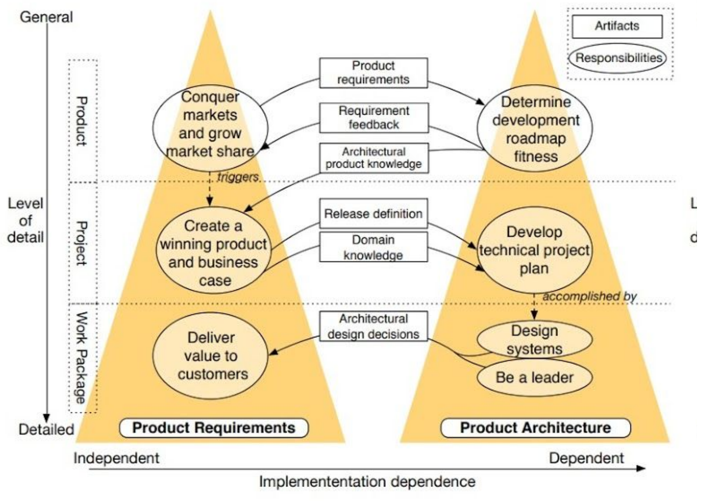


Figure 7 - The Three Peaks Model

These concerns are codified in viewpoints, which offer guidelines, frameworks and templates to construct so called views. Views are abstract representations of a software architecture as seen with a certain set of concerns, such as functionality, deployment or the flow of information. Often these views are represented in a model and the books offer a set of viewpoints that can be used to pick a fitting model and apply it to create a view for a specific concern. Views are part of architecture documentation, also called a definition, which contains explanations on the design decisions made, and the runtime behavior of a system. Another important consideration is quality properties, which are addressed in perspectives and can concern multiple views. Quality properties can be for example security, or maintainability, and are codified in quality scenarios that take into account the context of the property within the system. Finally the book considers software architecture patterns, which are collections of solutions for common problems in a certain context.
A study by Brinkkemper and Pachidi [12] on Functional Architecture Modeling explores in depth the construction of a functional architecture model. The study results in a semi-formal modeling approach of this important viewpoint, allowing for a design approach using specified semantics that can be used as a basis for analyzing the functional aspects of the architecture by employing scenarios.

According to Lucassen et al. bridging the twin peaks model for product software is the definition and realization of a stepwise architectural evolution within the constraints imposed by the product and architecture roadmaps [67]. In a comparative case study they conclude that software architects should not focus on stakeholder communication. The authors devise the theoretical foundation to bridge the twin peaks model, and specialize it in the reciprocal twin peaks model, shown in figure 8. They conclude that this process is possible only through adequate tooling which will have to be used by both product managers and software architects.



(a) The Model

Figure 8 - Reciprocal Twin Peaks Model

3.2.1 Composition Of Software Architectures

Van Hee et al. [17] consider asynchronous communication in software systems, presenting a method to ensure the system is weakly terminating, an important consideration when constructing a concurrency viewpoint as part of the architecture. In a subsequent work, van der Werf [19] considers compositional verification of asynchronously communicating systems, presenting a framework.
Alshuqayran et al. [29] consider a microservices pattern in their systematic mapping study. They state that a microservice architecture can address the maintenance and scalability demands of online service providers. Communication between the microservices and deployment operations were found the be the biggest challenges when employing this kind of pattern.
Namiot and Sneppe [30], consider some advantages of employing a microservice architecture stating that employing this kind of practice will lead to greater flexibility and reduced complexity of the constructed software. Amaral et al. [31] evaluate the performance of microservice architectures using containers. They conclude that from the infrastructure perspective some limitations are still of concern yet successful implementations have been seen with for example server visualization. Krylovskiy et al. [32] look at the implementation of a microservice architecture as an internet of things platform. They conclude that in comparison to a generic service-oriented architecture, microservices provided significant benefits for this type of project.
Further research on software architecture that were investigated did not result in any insights beyond what was already obtained from the papers presented in this section. These works help formulate an answer to RQ1 as follows.

Requirements, often stakeholder concerns codified in a context, as with user stories, are the basis for software architectures. This is represented in the Twin and Three peaks models, which consider iterative cycles between requirements, architecture and software construction, refining definitions and implementation with each cycle between two of the peaks. Without requirements it would be impossible to define what needs to be part of the architecture. A good way to make sure requirements are satisfied is through software architecture, which considers not only the abstract design of a system, but also the people who will have to use it and the context in which it will have to be deployed. This can be done using an Architecture Definition Language.

3.3 Startup Requirements for Software Specification

To determine if the proposed theory is applicable to startups operating in the IT domain, a better understanding of startup needs and requirements has to be obtained. To obtain this understanding various articles by thought leaders are studied, this in addition to the materials from the course ICT Entrepreneurship at Utrecht University. Although RQ2, *What are the requirements of a startup when defining product specification?,* cannot be answered fully by studying related work, a theoretical basis can be laid down for the definition of criteria that will tell us if the problem solution is sufficient for the startup in question.
To this end, papers were selected studying criteria for startup success, risk factors, and startup characteristics as well as articles written by influencers from the startup community. From these perspectives a better understanding can be obtained of how startups operate and might define a product specification.

3.3.1 Startup characteristics

Lowrey uses data from the Kauffman Firm Survey, specifically three year surveys, to better understand the characteristics of modern U.S. startups [60]. The study states that startups drive innovation and economic growth in the US. Even so, ninety-one percent of respondent startups had ten employees or less, indicating small business sizes. Fifty-one percent of the startups, were home based and seventy percent was individually owned.
In a list published on the blog of RocketSpace data is presented from several blogs and reports on what successful startup characteristics are [65]. Focus on a defined product is deemed essential. This is further illustrated by Hall, who states in his Forbes article titled *12 Characteristics Of Wildly Successful Startups* that constant communication and knowing your customers needs is essential [66].
According to Giardino et al [72], Software startups are able to produce cutting-edge software products with a wide impact on the market, significantly contributing to the global economy. Software development, especially in the early-stages, is at the core of the company’s daily activities.

3.3.2 Factors for success and risks

Lussier defines several variables that have high impact on the success rate of a startup [61]. These are respectively capital, financial control, industry experience, management experience, planning, advisors, education, staffing, and product timing. Startups that do not have access to these variables have a higher chance of failing.
Cusumano [73], considers various characteristics for successful startups. A strong management team, attractive market, compelling new product, strong evidence of customer interest and overcoming the credibility gap are deemed essential. In addition, it is considered important that startups demonstrate early growth and profit potential, flexibility in strategy and technology and a potential for large investor payoff.
Van Gelderen et al. look at 517 entrepreneurs to determine success and risk factors in the pre-startup phase [62]. They state that the individual characteristics of the entrepreneur, the environment, and the startup process are of impact on the success rate of the startup. Age, business plan, push motivation, management experience, dummy manufacturing, and market risk were deemed statistically significant factors.
In an article published on their website, Y Combinator partners Ralston and Seibel discuss what they consider to be essential startup advice [63]. They express that building something people want is essential for startups, and suggest that doing things that don't scale will serve a startup well. The founder of Y combinator, Paul Graham, often mentions the same advice on his blog [64]. He suggests that a fast launch of a product, fast growth, and understanding users are essential to startups. In addition, Graham states that startups have to solve problems and need a market big enough to scale in.
In another article by RocketSpace, several reasons for startup failure are discussed [67]. Failure to find market fit and an inability to scale quickly are two of the main reasons.

It is challenging to find adequate scientific sources for this domain, partially because it is young but also due to the constant change in this field. Yet several noteworthy discoveries were made by studying what influencers have to say and what empirical research shows about startups. In answer of RQ2, *What are the requirements of a startup when defining product specification?,* the following findings were most relevant to this study:

- Startups often fail because...
    - They do not understand their customers.
    - They do not communicate properly with stakeholders.
    - They are unable to scale quickly.

- Successful startups…
    - Launch products as soon as possible.
    - Operate in a market where scaling up quickly is possible.
    - Demonstrate flexibility in strategy and technology.
    - Offer a compelling new product.

RQ2 can be answered as follows. These findings illustrate that the topic of this study can be of relevance as it can help scale up more quickly and foster communications. As for criteria, the proposed method resulting from the case study will have to…

- Facilitate communication.
- Not be time-expensive as money and labor time are a concern.
- Foster growth of the startup beyond its initial roots and be scalable.
- Must allow the entrepreneur to better understand its customer and market.

4. RE4SA

In this section, the Requirements Engineering for Software Architecture theory is discussed. This model shows how the envisioned relations between requirements engineering and software architecture are structured. The final subsection will present a process deliverable diagram showing how the theory can be applied as a method.

4.1 RE4SA revisited

The core theory of RE4SA was proposed by Brinkkemper of Utrecht University and is shown in figure 9. In this theory, it is postulated that as software engineers employ user stories to define features, so it should be possible to employ epic stories to define functional architecture modules. The relation stems from the idea that an epic story can be seen as a categorization, or grouping, of user stories, and similarly an architecture module can be seen as the same for features, especially in a functional architecture diagram.

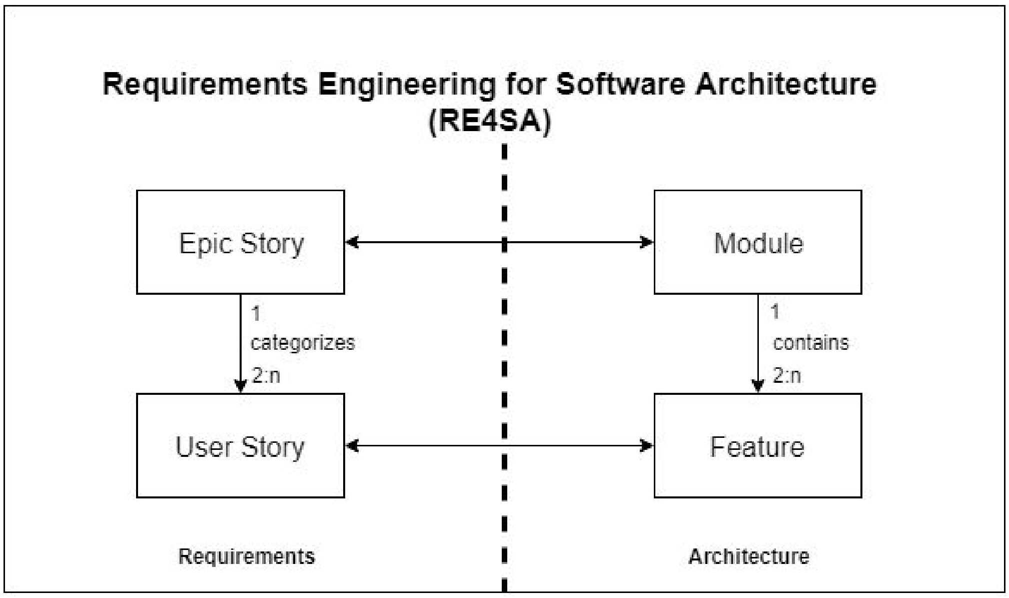


Figure 9 - RE4SA model

This theory is valuable as it will allow for an approach in functional software architecture development that is more closely related to requirements engineering. For example, right now user stories are employed to define features that developers have to construct. This is a very low level approach and for larger projects the requirements will quickly be lost in large backlogs, creating more work for product managers to properly organize the development sprints. By looking at a higher level of requirements, as in RE4SA, and connecting these requirements to a higher level of software architecture, it becomes possible to categorize user stories in a controlled manner by way of epic stories. Currently this is done by some planning tools in the form of epics or themes, indicating a need for this approach. Unlike user stories, the approaches to formulate such categories do not employ a template, and have not been closely investigated by the scientific community.
By introducing the more abstract level requirements of epic stories and linking these to functional software architectures, it becomes possible to quickly sketch out functional architectures without having to flesh out all the details present in user stories and features. This makes it possible to develop a requirements based

roadmapping approach, which can be valuable for organizations with few resources to dedicate to the design process early on, yet still have a need for developing coherent designs for their software.

However, van de Keuken et al [11] employ jobs to categorize and formulate epic stories. For example the job;

*“**Help me** ensure that I deliver high quality work”*

was used to formulate and categorize the epic stories;

*“**When** I am configuring an appliance, **I want** the output power of the appliance to be accurately represented in the flow and return meters, **so that** my model will correctly represent the produced power.”*

*“**When** I am configuring a fabrication specific appliance, **I want** the system to contain accurate information on the value of an appliance (e.g. voltage, output power), that matches the specification from the manufacturer, **so that** I can be sure that my model is correct.”*

This suggests that another meta level is present in the RE4SA model. Jobs are primarily intended to formulate needs of the end user, which can be broken up into motivational driven epic stories. On the architectural side of this idea a group of modules can be said to satisfy this need, forming an application that can be used by the end user. This notion led to the definition of the revisited RE4SA model, presented in figure 10.

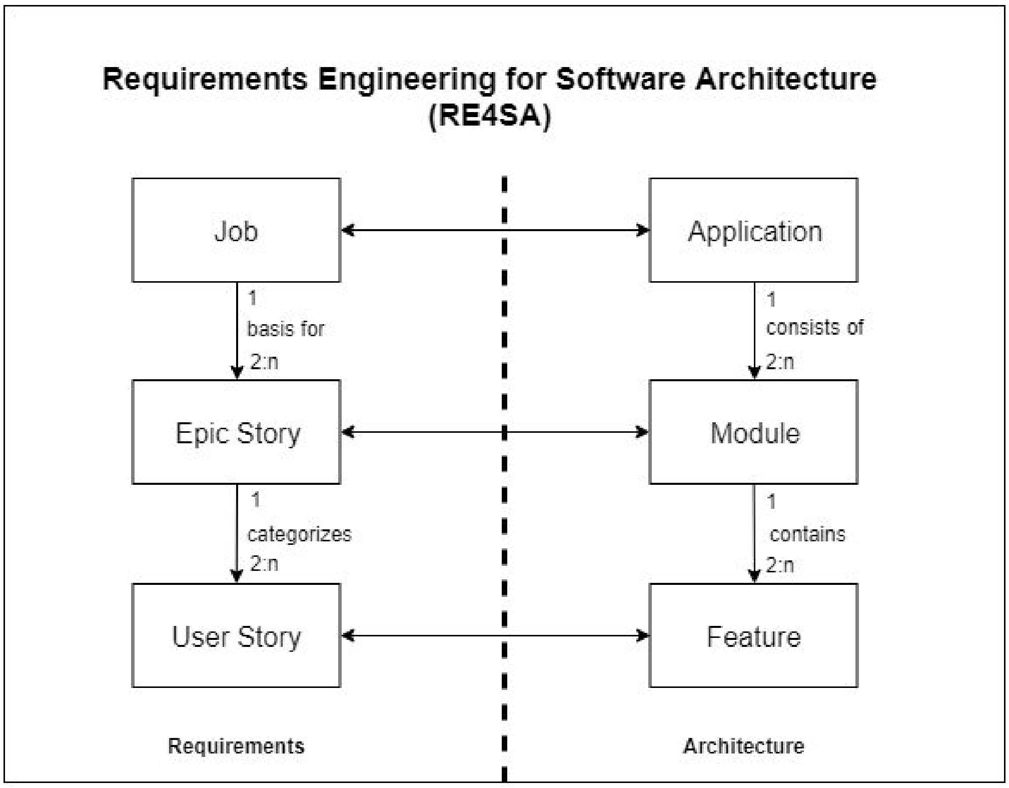


Figure 10 - RE4SA revisited model

The cardinality of this model can be explained as follows. Jobs contain more than one epic, exceptions can be imagined with very simple software products that display just one functional trait, but these are not common enough to justify a 1..n cardinality. Epic stories however, should always deconstruct into multiple user stories, as they might otherwise be misformulated user stories leading to redundant complexity. Similarly, applications consist of multiple modules, and modules always consist of multiple features. The RE4SA revisited model was tested in the case study, where it was shown that the desirable atomic traits of user stories defined the nature of these relations, more on that in chapter 5.

Taking the RE4SA approach it becomes possible to quickly sketch out a new application based on very high level requirements and see what development efforts need to focus on. In such a situation prioritization of efforts becomes based more on what is required from stakeholders without having to search through a long list of detailed user stories that might offer too much information for this phase of development.

4.2 Applying the RE4SA revisited method

Each step of the case study was carefully considered to determine what the right process is for the creation of a list of requirements and how to develop a functional architecture based on this list. The high level Process Deliverable Diagram (PDD) in figure 11 depicts the global process of creating a requirements based architecture.

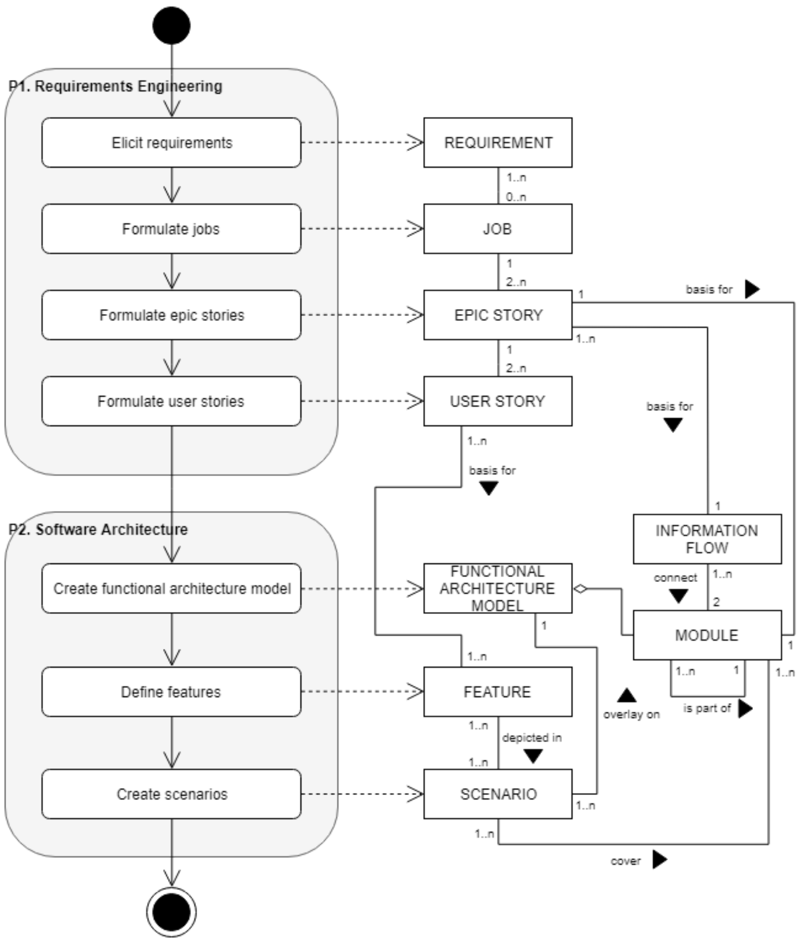


Figure 11 - High level Process Deliverable Diagram of the RE4SA Method

The diagram shows how requirements are first elicited, then written down in the jobs, epic story, and user story templates before formulating an architecture. A more detailed PDD of this process can be found in appendix A. A table explaining each activity of this PDD can be found in appendix B. A table explaining each concept can be found in appendix C. This PDD was created at the end of the case study.

5. Results

5.1 Case Study

In this section, the practical implementation of the RE4SA revisited method in a case study at Aruku B.V. is discussed. To do this, a description of how the RE4SA revisited method was applied to create a functional architecture is given. An overview is given of how jobs were defined based on interviews with startup clients. This is followed by an explanation of how epic stories were written and how user stories were defined. As this is an iterative process, lessons learned about the relations between these concepts are presented. An analysis of the results is given in section 6.

Following the requirements engineering section of the case study, an overview is given of how the architecture is then based on these requirements. Special attention is given to the formulation of the functional architecture model, which bears close relations with epic stories.
The goal of the case study is to create several architectural models for a climate control computer that can operate a greenhouse autonomously. These models will allow the development team to assess what components need to be constructed first and where possible reliancies between components exist. The system will have crop yield predictions at its core. These predictions are already developed and in use by the Aruku startup, one such prediction is shown in figure 12. In this figure the grey line represents real production numbers, the green line the predictions of the farmer and the blue line the Aruku predictions based on historical climate and production data. The X axis shows time in months and the Y axis shows production numbers. With this information software can be written that determines a course of action for climate management in a greenhouse to optimize crop yields by creating the ideal climate.

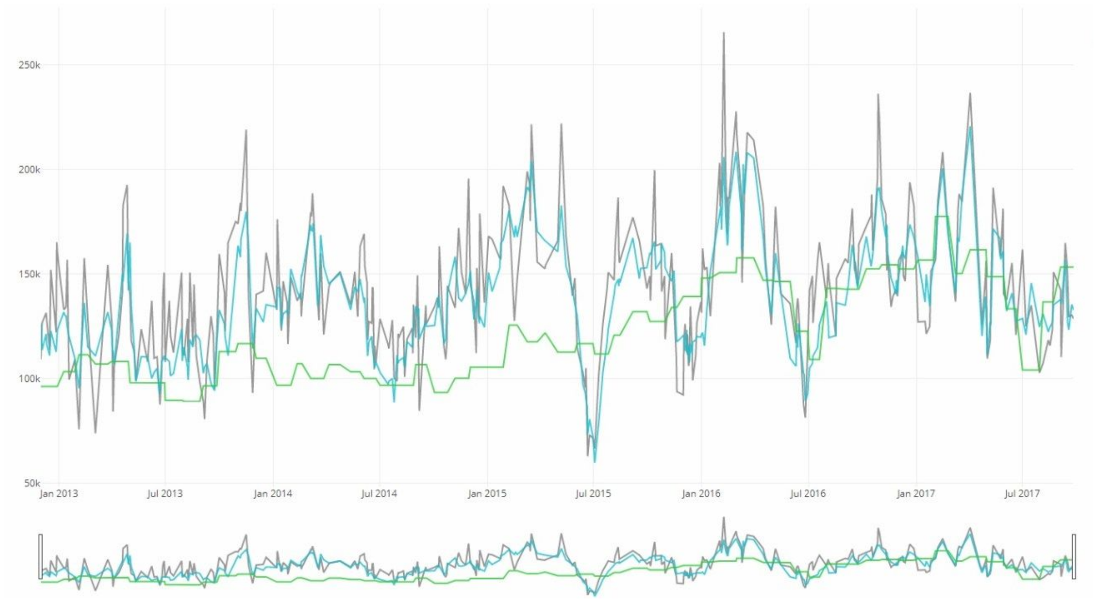


Figure 12 - Crop yield predictions for undisclosed farmer.

Before any modeling can be done, requirements need to be elicited first from the end users. This was done over the course of 5 interviews with potential clients, respectively two chrysanthemum, a rose, a gerbera and a cucumber farmer. These interviews were intended to find the needs of clients and sell them a solution for these needs, as such the interviews are confidential. In addition to the interviews, a wide variety of documentation was gathered on the operation of current greenhouse climate computers, growth manuals of various types of flowers and vegetables, and instructional texts for farmers in training. These documents alone are sufficient to identify the important tasks of a climate computer and the constraints imposed on them.

At the end of the case study the results are presented to the software development team of Aruku and evaluated to see if the developed method is beneficial to the development efforts of the startup and see if continued application is practical. During this evaluation each result is shown separately and in relation with each other to see if the developers can understand the results and apply them in their own work.

5.2 Formulating Jobs-to-be-Done

The first step of applying the RE4SA model means that jobs need to be formulated and prioritized. To ensure no potentially important jobs are overlooked, all tasks identified are included. This resulted in a list of 65 jobs, shown in appendix G, loosely centered around the act of controlling climate in a greenhouse. In the first version of this list, tasks varied from controlling lighting, to moving pots in specific phases of plant growth for specific types of plants. It was expected that approximately 2 to 5 epics would be formulated for each job, and in turn the same number of user stories for each epic. This would have resulted in approximately 1950 stories that had to be written, well outside the scope of this project. Prioritization of jobs was thus deemed necessary. The first step of prioritization was to categorize the jobs under general themes such as prediction, logistics, and plant care so that duplicate jobs could more easily be identified and removed. The general themes of these jobs were gathered from interviews with farmers, who all indicated similar problems that require solutions. This first cut dropped the job count to 45. Some jobs were spotted to be similar but at different levels of granularity, such as *"**help me** move packaged plants"* and *"**help me** transport harvest for packaging"*. These were both kept intact at this stage but marked for evaluation before epics are formulated.

Writing down all these potential jobs for development would not be very useful at all for the startup, as some of these would only become relevant far into the future, if ever, creating unnecessary information overload and work if epics were to be added for each job. One example of such a job is *"**help me** clean the greenhouse."* One way to make a distinction between jobs is by asking customers to prioritize a set of them based on likert scales [68]. One such implementation was employed by van de Keuken et al. [11] to create opportunity scores for jobs. This is recommendable for regular software development projects. However, this method is less suitable for startups where interaction time with a customer is less frequent, and customers are less inclined to put in this extra effort. The risk of project delays is too great to risk taking such an approach.
After grouping the jobs by general theme, such as *"internal logistics"* and *"sapling care,"* each job was inspected to see where the largest need was for the client, and if it fitted the scope of the startup product development when taking into account cost and time constraints. For example, robotics were not within scope, but jobs directly related to greenhouse conditions, such as *"**help me** maintain the correct temperature"* were. For this prioritization the MOSCOW method was used, which is a method to do fast initial prioritization [69]. In MOSCOW, priorities are assigned according to Must Have, Should Have, Could Have, and Won't Have This resulted in a more refined list of jobs that excluded some themes and jobs, but prioritized others resulting in jobs being listed with a letter type: M, S, C, or W.

The MSCoW prioritization, performed by the Aruku product manager on the list of 65 jobs, resulted in 10 jobs being marked with a must have, and 7 with a should have, all other jobs were given a rating below that and were stored for future development.

Lesson Learned 1: Identified jobs that are not included for development should be recorded for later consideration so that future application components can be more easily identified.

With this limited set of jobs granularity of the remaining jobs needed to be assessed to prevent duplicate work. This was a process that took some iteration, as the way jobs are written down can appear to offer different functionality, while resulting in the same execution. Several times work was started on formulating epic stories, only to discover that the jobs were at the wrong level, resulting in duplicates and redundant epics, this was an iterative process that took several weeks to refine. It was found that clearly defining the need of the client, with the attached problem statement, would result in more refined jobs without them becoming either too big, as is the case with high level jobs, or too specific, which might mean they are the problematic situation of an epic story.

Lesson Learned 2: Clearly defining the need of the client, with an attached problem statement, results in more refined jobs that are neither too big or too specific.

Example of a too high level job:

"***Help me** keep my growth models economically viable."*

This job would include the entire economic aspect of the greenhouse, resulting in epics that would have to check for sales prices and exchange market rates. While the jobs intention was to take into account energy prices when heating the greenhouse, accounted for with a user story. User stories according to AQUSA [23], need to be atomic and well defined, this in contrast with jobs, which need to encompass a number of actions.

Example of a job that is actually an epic story:

*"**Help me** record data for analysis."*

The vague formulation of recording data might suggest this is a job, but when formulating epic stories we get epics that map to only one user story, which is a clear giveaway that the granularity is too fine. In reality the data recording functionality was an epic under the job *"**Help me** predict the right course of action"* which is fitting because it states a clear solution to a desire; prediction of what actions are most beneficial.

Lesson Learned 3: The identification of jobs is an iterative, time consuming process highly dependent on the final granularity levels chosen for all stories.

After refining the jobs based on their level of granularity, 8 jobs remained.

5.3 Formulating Epic Stories

Once the appropriate selection of jobs has been made, epic stories can be formulated, this was done with the template shown in figure 13.

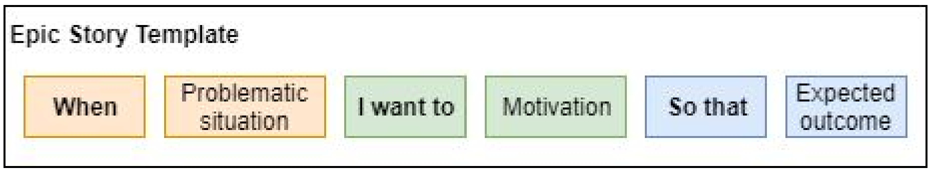


Figure 13 - Epic Story Template

Two to five epics per job are to be expected. To prevent redundancy, an epic should be a theme or category for two or more underlying user stories. Epics that contain only one user story are too detailed and can better be reformulated into a user story. Keeping this in mind made the process of formulating epics easier. Formulating epic stories for the prioritized jobs was a highly iterative process that was revisited at virtually every stage of the study. An initial list was formulated for each job, focused on defining problematic situations. It was interesting to note that the word motivation, part of the epic story template, initially led to epic stories that had two or more components that overlapped. Only when considering a functional motivation did this problem subside and did epics become more distinctive. For example, an epic story with a non-functional motivation:

*"**When** gathering temperature data, **I want** to have artificial fruits in my greenhouse, **so that** I can record realistic fruit temperature data without damaging real fruit."*

In this motivation component a state is described. It does not indicate any sort of functionality or action and this makes it difficult to place it within the functional workings of the system. Instead, this epic should be formulated as follows:

*"**When** gathering temperature data, **I want** to use an artificial fruit to track super local data, **so that** I can record realistic fruit temperature data without damaging real fruit"*

The addition of the word use indicates a type of action, or functionality, the word track too indicated further activities taking place. Instead of statically existing, the artificial fruit, the object of this motivation, now has associated behaviour, which means it will actually do something in the functional architecture.

Lesson Learned 4: Epic stories should always categorize at least two user stories.

Lesson Learned 5: The motivation component of the epic story template should be functional in nature to allow for appropriate mapping to a module.

The outcome as well turned out to better map to a functional architecture when the outcome was functional in nature. Epics that had insubstantial outcomes, such as knowledge or understanding, did not map well to functional architectures, unless a verb was included such as provide or assimilate. Alternatively, outcomes that specifically stated the lack of an action did allow for a mapping.

Example of an epic story where the outcome is non specific: "***When** light absorption drops, **I want** to adjust lighting, **so that** the right light conditions are met."*

When an initial list of epics was established, each component of each epic story was abstracted to discover if there were any functional overlays. This was done to prevent conflicts and redundancy in the system. This approach also allowed to identify several user stories that were formulated as an epic.
The epic *"**When** the temperature rises, **I want** to open overhead windows, **so that** the heat can escape,"* was formulated using the appropriate template, and escaping heat could mistakenly be considered a functional outcome. However, the functional motivation was too specific, namely opening an overhead window. This means that this epic story could only contain one user story, and as such is actually that user story, more on how this difference can accurately be determined in section 6.2. Instead we have to consider the problematic situation in the context of the system, what happens when temperatures are rising? Utilising the jobs as a basis for modeling modules, a very high level FAM was created showing just the module groupings that make up the entire system, shown in figure 14. Note that the individual control jobs such as *"**help me** control lighting"* are collapsed into the higher level *"**help me** operate the greenhouse"* job in this model to prevent redundant modules. When collapsed we have six jobs, mapped to six modules in this model.

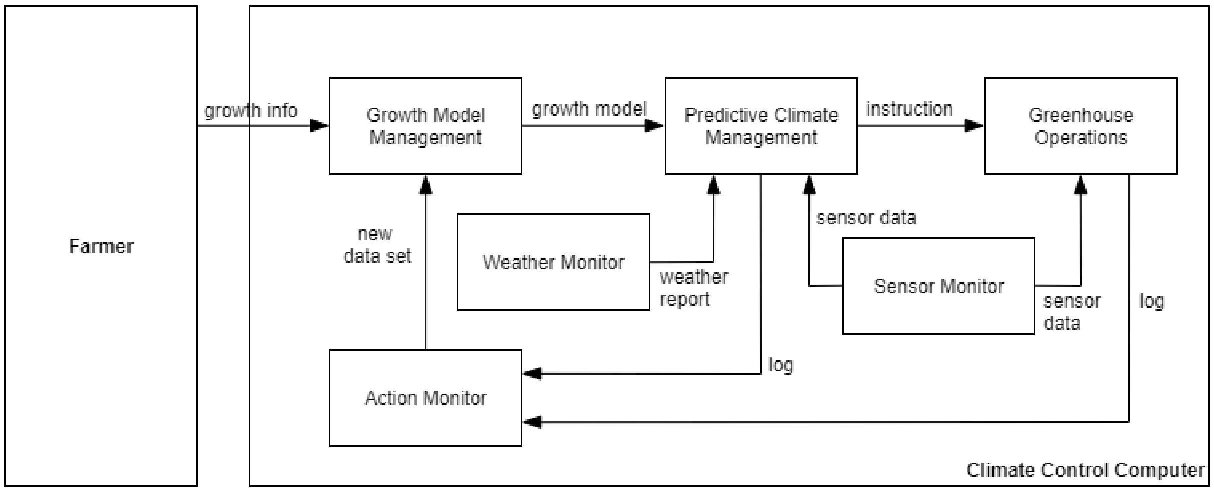


Figure 14 - High level model of the climate control computer based on jobs

This resulted in the dilemma that if it is possible to map jobs to a rudimentary FAM, maybe these jobs were some type of epic. The consideration was made that it was very well possible that there are different types of epics. This avenue was further explored and finally rejected, more on this in chapter 6.2.

Consulting the high level FAM as a baseline for module interaction on the epic level, epics were mapped to modules and some rudimentary interactions modeled, providing more insights in how the system might satisfy user needs when a true functional architecture model will be created.
This resulted in the following epic, encompassing the overall functionality of window control in the greenhouse:

*ES 3-4 "**When** receiving a state request, **I want to** open windows to a certain degree, **so that** the appropriate state is achieved."*

This epic was only established once the initial FAM was formulated, and an overall idea of system interaction was obtained. It merged with several other epics which were too specific, giving clues to what user stories might belong to this epic before even starting to write these down. The functional motivation of this epic allowed for various user stories relating to the opening and closing of the windows. This instead of just the single action of opening a window. Although the outcome is unspecific, it offers a clear outcome in the way of an achieved state that is appropriate, suggesting the system has to adhere to a set of requirements further defined in the user stories. Already this epic would suggest that this is the endpoint of a use case scenario, as no further actions are suggested.

While mapping the epics to modules, it became apparent that the module always tries to satisfy, or execute, the functional motivation of the epic story. Similarly, the when component of an epic story is the input of the module, and in some cases the contextual environment it functions in. As shown in the window epic, it became clear that the outcome of an epic story is the outcome of a module. Alternatively, the outcome of an epic can also be the end state of a scenario.

Lesson Learned 6: The FAM module inherently tries to satisfy, or execute, the motivation of the epic story. Similarly, the problematic situations mirrors the input of the module, and the expected outcome mirrors the output of a module.

The final list considered 32 epic stories. Some of these relate to for example light management:

ES 2-3 "***When*** *lighting needs to be adjusted,* ***I want*** *to adjust lamp brightness,* ***so that*** *lamps are on only when required."*

While others look at handling the prediction system at the core of the climate computer.

ES 1-2 *"**When** there is new prediction data available, **I want** to run it through a trained neural network, **so that** I can make yield predictions"*

The complete list of requirements is added in appendix H.

5.4 Formulating User Stories

Following the first completed version of the FAM that included a mapping for each epic story to a module, user stories were formulated. This was done using the user story template shown in figure 15.

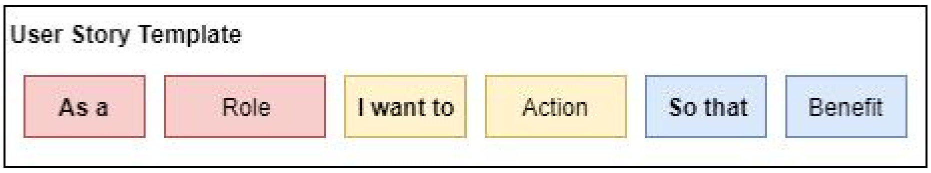


Figure 15 - User Story Template

Considering the iterative nature of the RE4SA process, some user stories were already in place as they had previously been formulated as epics.
According to Lucassen et al. [10], several traits make for more qualitative user stories. For this study it was considered important to have user stories that are atomic, minimal, well-formed, conflict free, problem oriented, full sentence, independent, and unique. As such each user story was continuously evaluated after each iteration to assure adherence to these quality properties.
Despite this consideration, granularity quickly became a difficult issue to overcome. Should user stories be finely grained, specifying precise actions and conditions, or should they leave room for interpretation by the developers. This is still a matter of debate in the scientific community, and most requirements engineers will go with a "what is necessary for your project" approach. As such this study went through several iterations before settling on a level of granularity that fitted with this stage of the project with the base constraint that all user stories should allow for programmers to define tasks, as custom to the agile development methodology.

Lesson Learned 7: The formulation of requirements on the jobs, epic story and user story levels is not a top down one shot approach, instead there will be continuous evaluation between the different levels of granularity and adjustments to stories to place them on the right level.

No strict demands were placed on the number of user stories that should be written for each epic, with the exception of a two user story minimum. Main functionality should be accounted for so that a feature diagram can be defined. This resulted in every epic having at least two, and no more than six user stories, with a total of 92 user stories written. This with the strong side note that as the project enters development phase, or encounters a change in requirements, this number will likely go up. An example of this is the growth model instructor, an epic which contains a user story that is left intentionally rough, instead of being fully exhaustive. *"**As a** growth model instructor, **I want** to account for legal restrictions, **so that** I don't violate the law."* During the elicitation phase, many legal restrictions were encountered, such as that the lights of greenhouses cannot be turned on between two and five at night in certain bird migration zones in certain seasons. These restrictions will need to be accounted for at some point in the list of user stories, and as such the developers need to be aware of them early in the development process. However, it will take the product manager some time to formulate all the user stories related to these requirements, a process which is not important enough to extend this phase of the project, and as such will be executed at a later time.

During this phase, it was discovered that one epic did not have unique user stories, instead overlapping with other epics, showing it was redundant. This epic was then removed from the model, illustrating again the iterative nature of the modeling process.

5.5 Project Briefs

To better explain the connections between the written jobs and epic stories for all stakeholders, project briefs were written. These briefs follow a template defined in the work by Intercom [36]. These briefs allow stakeholders to better understand why a certain epic story can help satisfy a job. The brief template looks as follows.

Project Brief - JOB

(Primary) Epic story

What problem are we solving and why?
Short description of the problem.

What value do we deliver to the customer?

- Related Epic Story 1
- Related Epic Story 2
- Related Epic Story n

How will we measure success?

- Result metric 1
- Result metric 2
- Result metric n

At the top of each brief the job and a central epic story are written. The core problem that the job is solving is described, which on the architecture side of RE4SA will result in an application. This is followed by an overview of the epic stories belonging to this job. Finally, a description is given of the metrics that, when met, will show that the job has been successfully satisfied. These components together will give the development team the required information that is needed to start development in an informed manner.

The briefs were written at the end of the requirements elicitation phase, when a set of jobs and epics had been established. Notable was that two more epics were identified when writing the metrics for success, suggesting that project briefs are a vital component in ensuring an exhaustive design is established.
The briefs have been included in Appendix D.

5.6 Translating Requirements Into a Functional Architecture Model

Translation of the formulated epic stories into a functional architecture model was done following the first iteration of epic stories. At this point user stories had not been formulated so some granularity errors persisted. The reason for this early modeling was to discover if the information available was enough to model a first version of the FAM and to discover where possible errors would occur. This resulted in a collection of modules that had names indicative of the functional motivations of each epic they related to. In many cases the NOUN + ACTION VERB found in the functional motivation component of the epic story gave direct input for the name of the resulting module.

Lesson Learned 8: By identifying the action VERB and NOUN in the functional motivation, the name of the resulting module can often be derived.

Figure 16 shows how an epic story translates into an architectural module, the linguistic mapping is not always perfect, but can help a struggling product manager to find the right names for the functional architecture components.

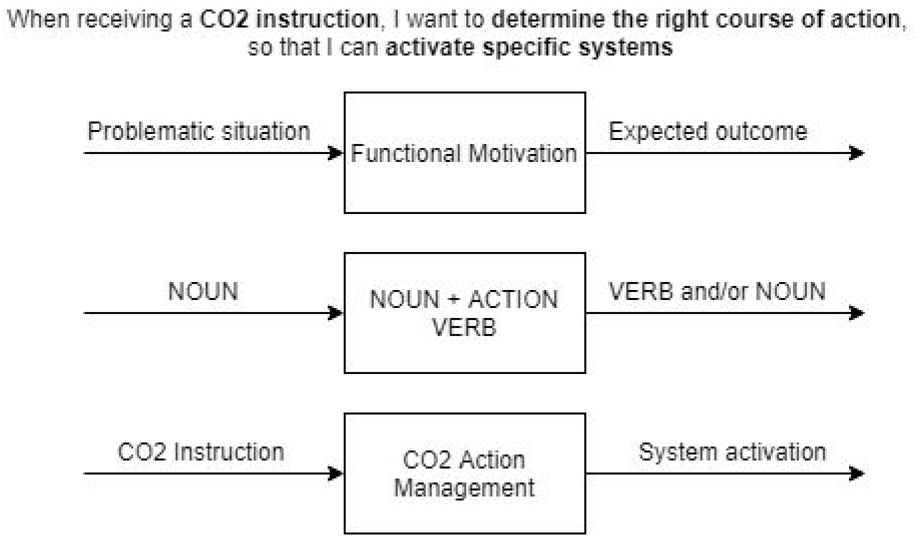


Figure 16 - Epic to module translation

Epic stories are related to each other by either outcomes or problematic situations, as such a rapid grouping of modules according to the jobs their epics belonged emerged, and early relations between them were easy to model. A higher level FAM based on jobs alone was useful to indicate where possible relations between jobs could occur, and outcomes and situations could be roughly matched to further identify dependencies. In some instances this was not possible, and these instances indicated that either an epic was missing, or that an epic existed on the wrong level and needed to be rewritten. This process helped to quickly refine the list of epic stories, making it in turn possible to relate all modules with each other in the model. Special attention needed to be given to modules that had multiple outgoing connections, as generally no indication in the epic story exists as to the number of these outgoing connections. For example, ES 1-3 *"**when** I have a yield prediction, **I want** to plan the right course of action, **so that** I can set the right climate conditions"* indicates rightly that there is only one input relation (a yield prediction), but is vague on the outgoing connections, of which in this case turned out to be 5, one of which was a log event that seems otherwise unrelated to the epic. Some help is offered by the epic stories relating to the different modules connected to this one, these will have something along the lines of *desired climate conditions received* as a problematic situation.

Lesson Learned 9: The number of outgoing information flows of a module cannot easily be derived from an epic story.

Lesson Learned 10: Epic stories with similar problematic situations can hint at similar origins.

However, the log event emerging from the functional motivation of the epic story 1-4 *"**When** I take autonomous action, **I want** to log all actions, conditions, and processes,* **so that** *I can improve my growth models,"* clearly shows that the regular mapping of the template to relations of the module does not always apply when it comes to modules that relate to a multitude of other modules. It is suggested that developers working with this method consider the nature of the problematic situation of these epic stories, in the case of the event logger the unspecific *"autonomous action,"* which describes a set of actions instead of being specific, indicated a multiple relation.

The resulting FAM is shown in appendix E. The entire process of developing this FAM took several weeks, this was mostly due to the intensive process of developing the right requirements upon which an architecture could be based, and figuring out if any mistakes were made. Biggest challenges were determining where the connections between modules should be, primarily by identifying related problematic situations and expected outcomes, and issues with finding the right granularity for both requirements and functional modules, a process further highlighted in section 6.2. The modeling itself took several attempts as different positions for module groupings were considered with the intention of orderly information flows with minimal overlap to improve readability.

When the user stories were written, the model was revised to reflect altered or removed epic stories and possible relations between them. However, this was only necessary in the case of one redundant epic, which was described earlier. Although it was expected that the FAM would have to be reconstructed at least a couple more times due to for example technical considerations not accounted for in the requirements, this method turned out to produce a high quality version early on, resulting in minimal rework costs.

5.7 Context Diagram

The context diagram, shown in figure 17, gives a representation of the system and the corresponding external entities. The climate control computer is centered in the middle as a black box, with associations to external entities. The associations between the system and external entities represent actions of the system either to these systems or vise versa.
As basis for the contextual model for the climate computer, the jobs defined earlier in the case study were considered and modeled as systems. Of these, weather monitoring is the only system that has components that are fully external to the system. Secondly, the jobs were revisited in terms of who the "me" component of the job represented, thus identifying the users of the system, who are by nature external. High level jobs were chosen where practical, for example, specific jobs belonging to greenhouse operations, such as job 4 ***"help me** control humidity"* were grouped together so that redundant information could be combined. As such we get a list of 5 jobs instead of 7, this was done to prevent unnecessary duplication of model elements with the same contextual connections.

ID	Jobs
1	Help me use and improve growth models
2	Help me operate greenhouse climate systems
3	Help me predict the right course of action in the greenhouse
4	Help me monitor external climate conditions
5	Help me monitor internal climate conditions

Table 3 - Jobs depicted in the context diagram

Finally, dependencies of the system on external systems that were not explicitly stated in a job were considered, which resulted in a connection to the let's grow framework, a source of information for the initial growth models that will be included in the system. The only job not depicted in this model is job 8 *"**help me** monitor internal climate conditions"* job, which is an autonomous process that does not interact with external elements or stakeholders.

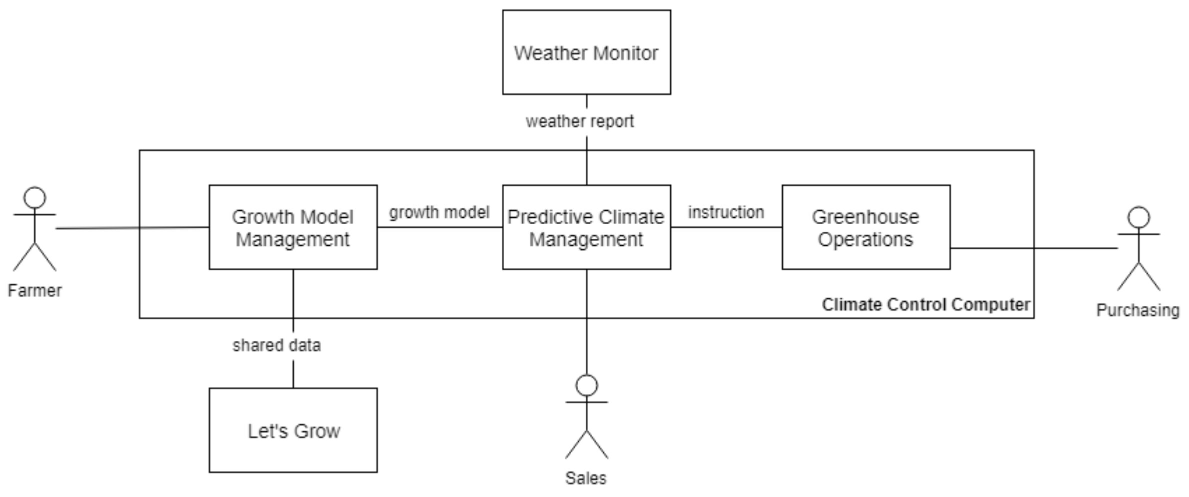


Figure 17 - Context Diagram

Figure 17 also shows various stakeholders. Most obviously related to the system is the farmer, who has ultimate control over his crops and is directly involved in the growth model management. However, the system provides valuable insights to secondary departments present in almost any greenhouse. Sales can obtain valuable insights from the crop yield prediction systems, as it will allow them to prepare more accurate sales prices early on. Purchasing can monitor the logs of the operational system to determine how many resources have been consumed and need to be acquired, such as CO2 canisters and electricity.

5.8 Feature Diagrams

The feature diagram is a standard component of the uADL. It is used to show which features are present within different modules of the FAM. The diagram considers if features should be implemented, allowing to define what features are mandatory or optional.
In the context of RE4SA, this means that the components of the feature diagram are directly related to user stories. This relation can be extracted by considering the goal component of user stories and identifying its verb and possible noun. As a [role], **I want to [VERB] + [NOUN]**, so that [benefit].

Lesson Learned 11: By extracting the verb and nouns from the action component of a user story, it is possible to map user stories to features depicted in a feature diagram.

For example, the user story "*As a data manager,* ***I want to send all input data to the effect monitor****, so that it can be used to improve predictions"* contains the goal verb "send." This verb identifies the feature that will appear in this user stories feature diagram. The "input data" component of the user story indicates what will be sent, resulting in the feature "send input data."

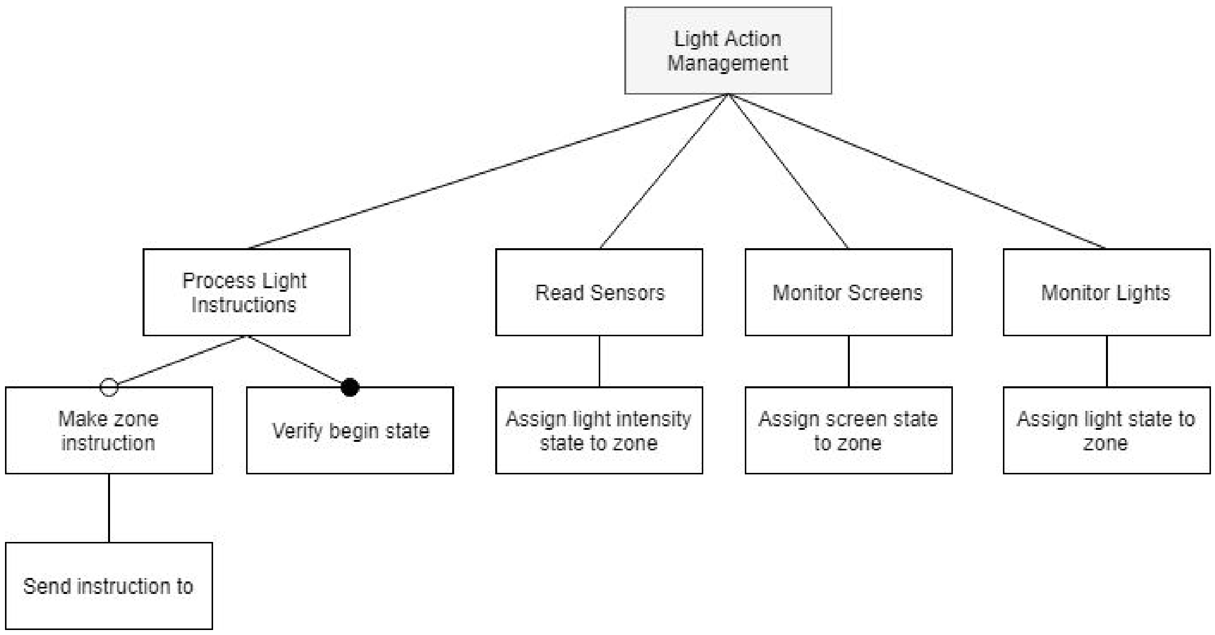


Figure 18 - Feature Diagram for Light Action Management

Figure 18 depicts the basic feature diagram for the Light Action Management module. The epic story this module was based on categorized four user stories. Similarly, the module resulting from that epic story now links to four features. However, more detailed features are depicted. These components often depict parts of the user story that appear in the [benefit] component of the user story template, however, investigating these relations in detail was outside the scope of this study.

User Story	**Feature Names**
As a light manager, **I want to** process a light instruction to determine if a zone needs light or shadow, **so that** the correct lighting is implemented.	Process light instruction
As a light manager, **I want to** use output from light sensors, **so that** I use real world situations and can monitor if my instructions have effect	Read light sensors
As a light manager, **I want to** always check the state of the screens and lamps, **so that** lamps do not turn on when screens are closed	Monitor Screens and Lamps
As a light manager, **I want to** use knowledge about current conditions to determine what should be changed, **so that** conditions can be improved	Interpret current conditions and determine course of action

Table 4 - User Stories and resulting Features

Because these diagrams are not a high priority for this case study, they were only modeled to see if they mapped to the same epic / module, and user story / feature, mapping demonstrated in the FAM. The diagram was initially useful to identify new user stories, but was not significantly related to epic stories, which have the focus of this study.

5.9 Scenarios

Scenario overlays represent the behavior of user scenarios with the system. The climate computer is largely autonomous and as such many of these users are the system itself, triggered by certain external conditions or internal behaviors. In uADL, the behavior originates with the output of user stories. Scenarios are simple overlays on the Functional Architecture Model, with red arrows depicting flow and sequence numbers on top to indicate order. Although a great variety of use cases can be modeled as a scenario, this study considers three varied examples to test the functional model.
Initial difficulty was with determining how to model triggers in the scenarios and concurrent autonomous functionality. For example, scenario 1, triggered by a cold weather front, shows several arrows that occur at the same time, primarily these are sensor output functions, which update automatically every couple of minutes, and log events, which occur after each action of certain functional modules. It was decided that these concurrent events would be given the same flow number as they represent the begin state. Each scenario has been included in Appendix F.

Scenario 1 - heating system activation, depicts the system activating heating as a cold weather front moves in, the initial events are triggered by weather monitor outputs, the predictive climate management group then determines a course of action, which is forwarded for execution to the heating group, which first checks the window state to confirm that the action plan can be executed. This scenario considers the actions of 31 user stories, out of a total of 92 formulated for this system. This scenario covers 33.7% of all features.

Scenario 2 - system initialization, depicts system initialization. The farmer will have been given instructions to set the greenhouse to very basic conditions, for example the overhead screens will be fully opened. The farmer supplies initial growing instructions and greenhouse type, this is then followed by a process in which the greenhouse is brought to conditions ideal for the selected plant type, from which further actions can be determined. This scenario considers the actions of 74 user stories out of 92 formulated for the system. Out of these three scenarios it has the biggest feature coverage. This scenario covers 80.4% of all features.

Scenario 3 - wind direction compensation, shows the flow of system actions if a sudden significant change in wind direction is detected. Wind direction can greatly affect the humidity in a greenhouse. The neural net will determine how systems will need to react and send instructions to the humidity systems accordingly, overhead fans will activate where necessary to ensure equal conditions. This scenario considers the actions of 26 user stories out of 92 formulated for the system. This scenario covers 28.3% of all features.

When considering that a user story maps to a feature in RE4SA, a cover score for each scenario was calculated to see how many user stories, and as such features, are considered in it. In total 79 user stories were depicted in the scenarios, resulting in an overall coverage score of 85.9%.

Scenario	**1 - Heating**	**2 - Initialization**	**3 - Wind direction**
Total user stories	31	74	26
Coverage score	33.7%	80.4%	28.3%

Table 5 - Scenario Cover Scores

5.10 Product Deployment Stack

In cooperation with the development team, the functional model was analysed for dependent technologies so that a product deployment stack could be modeled, depicted in figure 19. This model describes the required and optional software elements of an operational product stack as well as required hardware platforms. After walking through the scenarios to further illustrate the required functionality of the system, several core platforms were settled on, primarily those to drive the machine learning components of the climate computer. In addition, the API of the data sharing platform Let's Grow was included in this model as this API can be a source of information to train the predictive neural network. The RE4SA model did not directly influence the construction of this model, but the creation of the FAM did allow for more informed analysis of the system so that this model could be constructed.

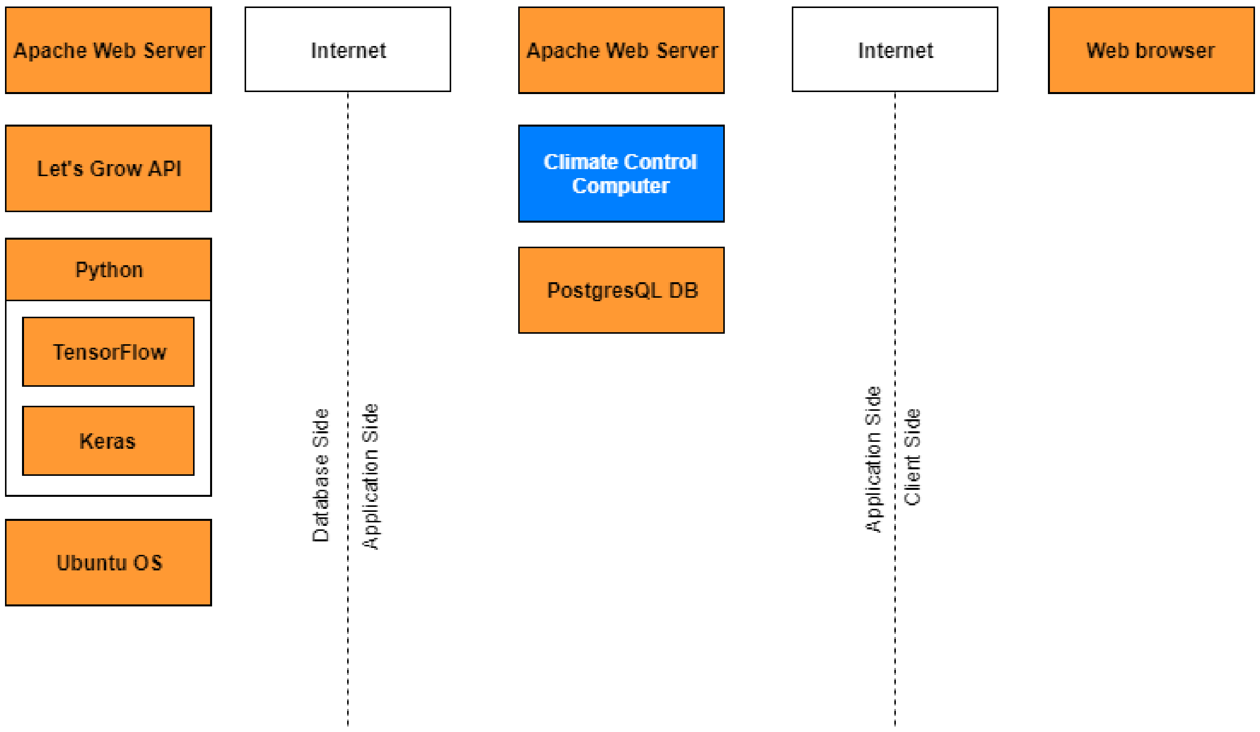


Figure 19 - Product Deployment Stack

6. Analysis

This section analyses the RE4SA method employed in the case study. First, lessons learned while identifying requirements and translating them into an architecture are discussed. Secondly, an expert analysis is conducted with the startup developers to evaluate their perceived usefulness of this technique for requirements engineering and as a roadmapping tool for development. This section answers RQ3 and RQ4; *How can epic stories be employed to create a software architecture using the utrecht Architecture Definition Language?* and *How effective are the results of this study when used for startup product specification?*.

6.1 Applying RE4SA

As discussed, formulating jobs was the first step taken as a basis for identifying the requirements for this case study. The jobs originated primarily from interviews with potential clients and documentation of existing climate computers, greenhouse manuals and greenhouse plant care guides, these sources of information were not exhaustive but sufficient. The jobs were intended as a basis from which to start identifying epic stories, they offered a structured way of identifying required functionality at an abstract level. Jobs can be used to form a very high level FAM and give a rough overview of how the final version of the finished product might look, this type of model had many similarities with the contextual view when considering who the *"me"* part of the job was. In an entrepreneurial context, and very relevant for early stage startups, they can help validate needs of customers, preventing developers from putting too much effort in systems that are not valuable. In a development context they can show dependencies between different system components early on.
When formulating jobs it was discovered to be useful to first create an exhaustive list of all possible help questions that clients might have and then prioritize those jobs that can offer the most value for development. When first formulating jobs it is not important to consider if a job is at a certain granularity level, as the level of granularity has to be consistent with the epic stories and user stories and is better determined at a lower level. Granularity does become relevant when epic stories are being formulated, here it was discovered that higher level jobs would often result in epic stories that were more detailed versions of other jobs in the list, this has to do with granularity issues as demonstrated in the barista problem example. These errors will often become apparent as the list of requirements grows more mature. The final 8 jobs that were calculated categorized 32 epic stories, meaning that on average, each job was host to 4 epic stories. For the formulated epic stories, 92 user stories were written, meaning that on average each epic story categorized 2.9 user stories.

Type	Final Count	Average count of substories
Job	8	4
Epic Story	32	2.9
User Story	92	--

Table 6 - Average substory overview

Lesson Learned 12: For each job, on average 4 epic stories can be formulated, and for each epic story, 2.9 user stories can be formulated.

6.1.1 Problematic Situation

Epics are divided in three parts, problematic situation, motivation and outcome. In the first draft of the epic stories, the problematic aspect of situations was briefly overlooked in favor of general context of epic stories. This quickly resulted in vague epics that were not very informative and significant difficulty in formulating epics. When a situation was made problematic it quickly resulted in an easier to formulate epic story with a more specific purpose. For example, when the situation is *“**When** plants are growing,”* we are dealing with a generic and hard to define epic story. Yet when we add more details and make the situation problematic, for example *“**When** some plants are growing faster than others,”* it becomes easier to develop a solution to the problem. In this case *“**When** some plants are growing faster than others, **I want to** apply growth hormone, **so that** all plants mature at the same time.”*

6.1.2 Functional Motivation

Originally the second part of the epic story is the *motivation*. The motivation was described as follows:

“Capturing the change that needs to occur in order to reach the expected outcome, the motivation generally already implies a solution to alleviate the problematic situation” [11].

As suggested here the motivation will imply a solution to the problematic situation formulated in the first part of the epic story. The template is shown in figure 20.

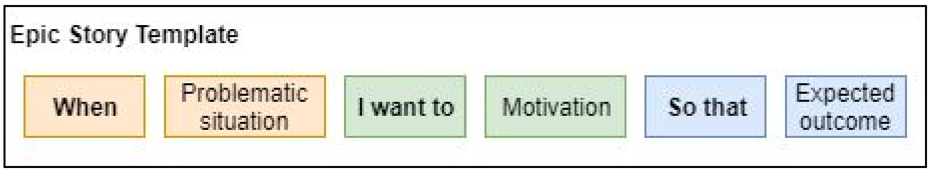


Figure 20 - Epic Story Template

However, this study considers the term motivation on its own to be ambiguous, and while the researcher was mastering the epic story technique it led to some confusion as to the purpose of a motivation in the epic story as it might consider a *why* instead of a *how*. The realization that motivation related to the word *want* in the epic story template cleared up some of the confusion, but was not deemed satisfactory.
The study by van de Keuken et al. also employed the word *generally*, considering exceptions that were not encountered in this study. As such the word motivation was joined by the word functional to clarify this part of the templates’ intended purpose. Adding functional as part of the template indicates that this part of the template should describe a process or change, and turns it from a static why to a more clear how, to prevent further confusion. This means that the epic story template is changed to the one shown in figure 21.

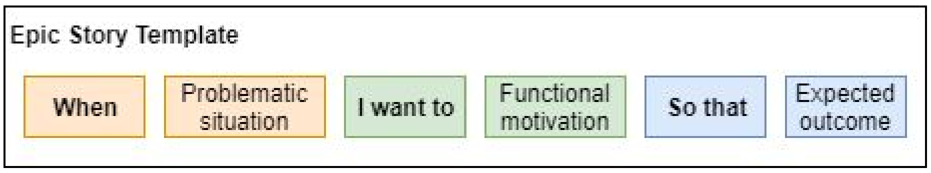


Figure 21 - Updated Epic Story Template

6.1.3 Outcome

The outcome of the epic story template describes the resulting solution to the problematic situation, or at the very least the outcome of the functional motivation. This was continuously the easiest part of the epic story template to formulate, as most requirements are ultimately wishes to a certain outcome. The outcome of an epic story should try to avoid to describe the problematic situation or functional motivation in a negative form or simple nonfunctional states such as knowledge. An example of this is *“When light intensity is high, I want to* *administer more CO2, so that plants have enough CO2."* Rephrasing such outcomes, in this case into; "*so that plant growth speed is given a boost,"* will result in epic stories that are more clearly mappable to architecture modules.

Lesson Learned 13: The outcome of an epic story should try to avoid to describe the problematic situation or functional motivation in a negative form or simple nonfunctional states such as knowledge.

In the case of the example above, the outcome describes a final state, meaning it will not have any relational arrows leading out of the architectural element and form the end state of a use case scenario. However, outcomes can also have direct influence on the problematic situations of other epic stories. In these cases they can offer valuable starting points to identify if any information flows between two modeled epic stories exist. An example of this kind of outcome are found in ES 3-1 and ES 3-2.

*ES 3-1 "**When** the climate computer gives a temperature instruction, **I want** to know how to process it based on current conditions, **so that** I formulate the correct temperature conditions."*

*ES 3-2 "**When** receiving a heating request, **I want** to change the current heating state, **so that** temperature can go up or stay constant."*

In these cases a relation can be found between ES3-1 formulating correct temperature conditions, and ES 3-2 needing these in the form of a heating request. The relation is still difficult to see, which will be the case in most epic stories as natural language is a factor.

6.2 User Story Granularity and the Effect on Jobs (The Barista Problem)

When formulating the epic and user stories, it became apparent that writing the same epic down in different ways could quickly make them either a user story or a job. This problem was primarily related to defining the right granularity for the stories being written. Too broad and they would not contain enough meaning, too narrow and an explosion of irrelevant or duplicate, but differently written, user stories would occur.

Whilst considering the right granularity for epic stories, a less complicated system was considered as part of a brainstorm on how to approach this challenge. What if the system being considered was that of a robot that needs to do what a barista does: make coffee based drinks. A small list of actions was put together, with frothing milk, adding syrup, and preparing an espresso at the core. Initially this resulted in three simple jobs: Help me: Process order, prepare drink, deliver drink.
Diving into the prepare drink job, main functionality was identified, and this resulted in several epic stories. A very simple functional model, shown in figure 22 was created, showing just the functional aspects of making coffee.

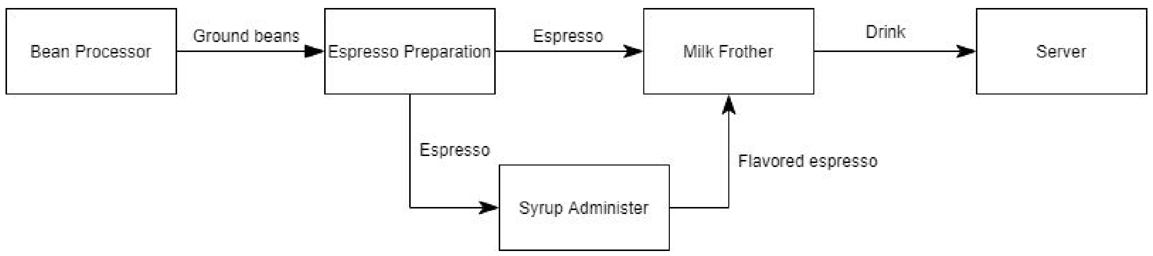


Figure 22 - Coffee preparation FAM

However, it is also possible to shift everything down one level and keep the functional structure intact. This was discovered when starting to formulate user stories. The epic *"**When** I receive ground beans, **I want** to prepare an espresso, **so that** I have the basis for different types of drinks"* could just as well be the user story *"**As a** barista, **I want** to prepare an espresso, **so that** I can make different types of drinks."*
What if there is just one job being considered, *"help me serve drinks."* This would mean that process drink, prepare drink and deliver drink would be the epic stories, resulting in a new model shown in figure 23.

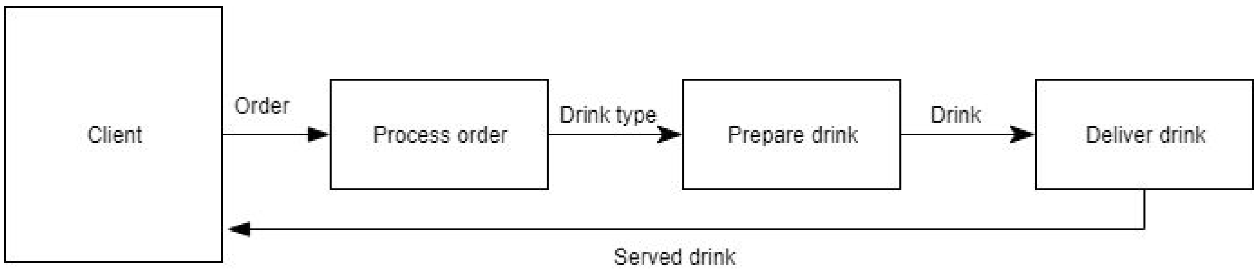


Figure 23 - Higher level coffee preparation FAM

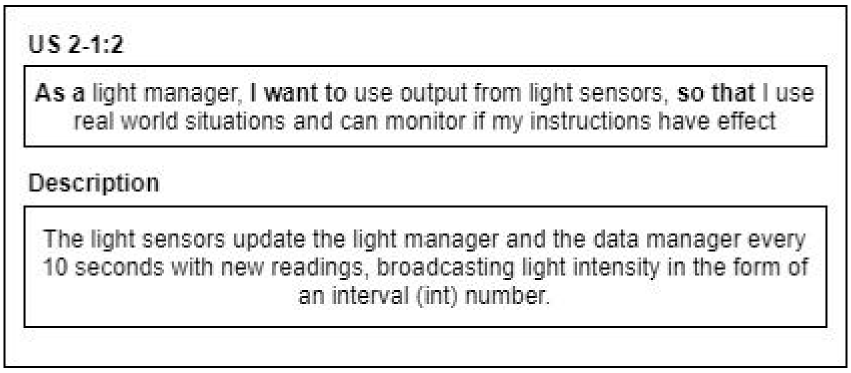
Both models satisfy the requirements in terms of functionality, this led to the exploration of the idea that if this functionality can be written down as both epic story or job, they might actually both be an epic on different levels of epic.
User stories form the bedrock of the RE4SA model. Keeping this in mind, very finely grained user stories can be grouped under epics. Milk frothing can be formulated as the user story *"**As a** barista, **I want** to make frothed milk, **so that** I can prepare mocha, macchiato or cappuccino,"* but this user story could be further divided into for example the user story "***As a** barista, **I want** to keep the frother nozzle at a 15 degree angle, **so that** the milk foams with the appropriate consistency of air bubbles,*" and various others. This means that the first user story is actually the milk frothing epic *"**When** I am preparing a macchiato, cappuccino or mocha, **I want** to froth milk, **so that** the drink can be assembled appropriately."*
This resulted in the idea that the first model considered minor epics, and the second considered major epics. However, employing this idea resulted in a runaway growth of levels that would not necessarily make for a more informative methodology.
The answer came from the Jobs-to-be-done theory [11] that considers different levels of jobs, going up to higher level jobs as more abstraction is required. So instead of inserting multiple levels of epics, a level of granularity for user stories appropriate for the project is established, and this is propagated up until the job level. Considerations can be for example if user stories should include metrics.

Lesson Learned 14: When a level of user story granularity appropriate for the project is established, this change can be propagated to the epic and job, and subsequent higher level job levels until all requirements are consistent with each other.

Although this suggests that projects should first consider user stories, this is not necessarily the case. Jobs and epics can first be defined at a level that is coherent for the product manager, and then refined as the user stories are written. This process is indicative of the iterative nature of this method and was found to help result in more informative user and epic stories during this case study.

6.3 Expert Evaluation With Aruku Development Team

Having developed a functional architecture using the RE4SA methodology, and having analysed the case study from the perspective of the researcher, a final evaluation with the development team of the startup is conducted. To assess the importance of the developed models and supplementative materials, such as lists of requirements, each model was introduced step by step to each developer in a meeting room setting. Developers were kept separate from each other during this evaluation to prevent them from contaminating each others opinion. During the evaluation, each model was presented in both a paper and digital format on a laptop and the developers were left to consider each component without explanation beyond what the already knew: that these documents would functional as a roadmap for future development. This was done to see what their first impression would be without contaminating them with the researchers potential bias. After initial impressions, the developers were allowed to ask questions about the models to clarify their understanding. This was followed by an unstructured discussion of the material during which the full set of components was explained to the developers. This evaluation took approximately four hours for each developer. The explanation of the models first discussed the completed FAM, accompanied by the list of jobs, epic stories, and user stories. Given just these components, the development team indicated that the information available seemed limited in detail. When provided with the project briefs, it was deemed that sufficient information was available to start development. The direct relation between these models had not been immediately apparent to them when left to their own devices but clarified the meaning of each model when connected. However, some core concerns remained. The list of requirements led to additional questions from both developers, primarily about specific engineering choices. Figure 24 shows an example of required information to interpret a user story on a technical level.


US 2-1:2

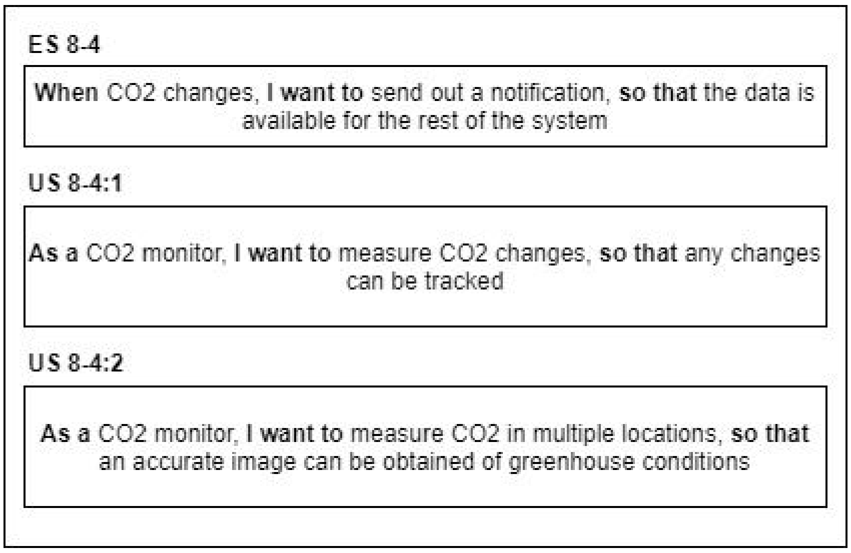
As a light manager, **I want to** use output from light sensors, **so that** I use real world situations and can monitor if my instructions have effect

Description

The light sensors update the light manager and the data manager every 10 seconds with new readings, broadcasting light intensity in the form of an interval (int) number.

Figure 24 - Missing user story description for technical specification

In this example, it was not completely clear what the specific intervals of sensor updates should be. This question, and others like it, are either answerable with a yes or no, or a number (in this case 10 seconds) in most cases. The absence of technical specifics did lead to some confusion when the developers were left to interpret the models on their own, as both developers felt that the word architecture should concern technical details such as design patterns, and not necessarily functionality. It was noted by the senior developer, that an experienced developer would be able to handle the absence of such technical specifics on their own, but that a more comprehensive list of user stories would be desirable for some epic stories. One example of such an epic is epic no. 28, shown in figure 25.


ES 8-4

When CO2 changes, **I want to** send out a notification, **so that** the data is available for the rest of the system

US 8-4:1

As a CO2 monitor, **I want to** measure CO2 changes, **so that** any changes can be tracked

US 8-4:2

As a CO2 monitor, **I want to** measure CO2 in multiple locations, **so that** an accurate image can be obtained of greenhouse conditions

Figure 25 - Epic Story 8-4.

This epic categorizes two user stories that are both unspecific in the sense that they consider *changes* without specifying what kind or at what interval. From a product manager perspective the functionality represented in these requirements did not have a high priority at the start of development and as such only these two user stories were written at the time of presenting the requirements list to the development team. Ambiguities such as what CO2 changes should be measured, like concentration in the atmosphere, its dissipation rate, or the rate by which is was added to the atmosphere, led to request from the development team for clarification during the evaluation of the requirements list. However, when presented with the project briefs for CO2 control, the developers were capable of determining what kind of readings their system would need to satisfy its purpose, mitigating the problem of unspecificity in the epic story of figure 25. One developer stated that the ambiguity actually left room for him to build the software in a way he deemed practical, allowing for a greater degree of freedom during development, which was considered desirable.

To fully understand the system, the context diagram was deemed vital, as it showed where interactions with the systems and external components would occur.

Lesson Learned 15: The list of requirements, functional architecture model, project briefs, and context model on their own do not offer enough functional details to start development, together they form a comprehensive overview of the requirements and their functional satisfaction in the architecture.

Detailed technical architectural decisions are not present in the architecture design, but this was deemed a good thing by the development team as it gives them the freedom to innovate and work in a way that they find comfortable. This space for interpretation means that there are multiple routes towards satisfying the model, depending on what the programmer deems to be the best choice for this design given the available time and resources. This space also means that there is a remote chance that systems will be constructed that do not meet customer requirements, but this is considered to be inevitable in any software project, for if you wanted to avoid this you would have to create the entire software package. An important side note was that the team did consider the chances of these mistakes happening being remote, as the combination of jobs, epics, user stories and briefs makes for a very comprehensive explanation of what needs to be built.

The architecture was deemed to give a good idea of what systems would have to be developed in what order, providing a basis for establishing sprints and a backlog. This roadmap also extends to development beyond version 1.0. For example, what if advanced drones were to be developed that would be able to act as mobile ventilation units, directly affecting airflow in the greenhouse. This system would then have to be included inside the humidity module group. Another example would mean that for example automatic plant watering could be included, which would see a new module group being added to the FAM. The model allows to easily include such groups and extensions, offering a good method of planning future versions of the software product based on changing requirements. An example of the original model next to an expanded version including a drone ventilator is shown in figure 26.

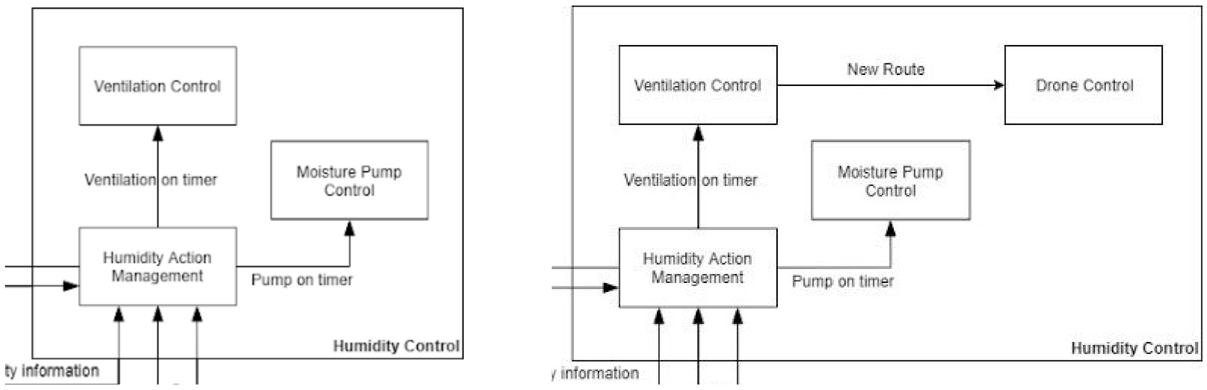


Figure 26 - Original FAM humidity module (left) and an extended version (right).

Overall the developers indicated that they deemed this method comprehensive and informative, and they would be happy to see its continued application for future product specification.

To summarize:

- Some confusion existed about how the models should be interpreted.
- The requirements list, FAM, project briefs, and context diagram should all be present.
- The design offers a clear overview for the developer to determine how to proceed.
- The design is easy to extend.
- The design does not limit developers in their freedom, and instead enables them to make more informed choices.
- Models and accompanying documents are sufficient to base a roadmap on.

7. Discussion & Conclusions

7.1 Conclusions

This thesis has shown that the Requirements Engineering for Software Architecture, or RE4SA revisited, is a valid method for basing a functional software architecture on requirements.
The case study at Aruku B.V. has resulted in a list of 132 requirements upon which the architecture was based.

Type	Count
Jobs	8
Epic Stories	32
User Stories	92

Table 7 - Overview of the number of final stories..

The developed architecture consists of the following models:

- Context Diagram
- Functional Architecture Model
- Product Deployment Stack
- Feature Diagrams
- 3 Scenarios.

The expert evaluation by Aruku developers has shown that these models, supported by project briefs, was considered to be an excellent tool upon which to plan development of the startup's core products. The concept of epic stories itself was expanded upon in this study, with the determination that the motivation component of the template should be functional in nature. The main difficulties with implementing this method were found with determining granularity of the requirements, matching them to get each requirement on the appropriate level for effective communication flow. The analysis of these challenges has resulted in the idea that a final list of requirements on the levels of jobs, epic stories and user stories, can only emerge after an iterative process during which the product manager develops a better understanding of the written requirements.

Armed with the answers on all research questions, the results of the literature review and the case study, combined with its subsequent analysis, it is now possible to formulate an answer to the main research question:
"How can the requirements and software architecture for the software specification of a startup be based on epic stories?"
Early in this report, the design problem was stated as finding a way to construct a software architecture based on epic stories. The solution to this problem was formulated as the Requirements Engineering for Software Architecture Revisited method. This method was subsequently tested in the case study at Aruku B.V.. This case study, and the subsequent analysis, has shown many limitations and benefits to this method. Most importantly this method has shown itself to be a fitting answer to the how component of the main research question. Namely, the formulation of jobs, followed by epic stories and project briefs to link these concepts, allows to assemble a functional software architecture. These components in combination presented themselves to be sufficient to base a functional architecture model and a context diagram, which were later deemed the most important aspects of the architecture in the expert evaluation. The startup component of the question indicated

certain limiters to the study, namely the limited resources available to the startup implementing the method. The case study and expert analysis have shown that the software specification of the startups next product was well achievable using RE4SA revisited, and in practice very beneficial due to its iterative nature, ease of use, and expandability.

RQ1, *"What is currently known about requirements engineering in relation to specifying a software architecture?"* and RQ2 *"What are the requirements of a startup when defining product specification?"* were answered by the comprehensive literature review conducted in the first phase of this study. Important aspects of these answers were the restraints placed on startups by budget, scope and runway, as well as the limited access to clients. In addition, the existing research on for example user story quality, epic stories and Jobs-to-be-done theory allowed to place this study in the context of the requirements engineering domain and formulate the RE4SA method when evaluated in relation to the reviewed works on software architecture.

RQ3, *"How can epic stories be employed to create a software architecture using the utrecht Architecture Definition Language?"* was answered in the case study, during which the entire RE4SA revisited process was tested and evaluated for its limitations and benefits. Most important here was the formulation of the process deliverable diagram, which showed a considered process and it's resulting deliverables for product managers who wish to implement RE4SA revisited.

Finally, RQ4, *"How effective are the results of this study when used for startup product specification?"* was answered in the expert evaluation conducted in conjunction with Aruku developers. This evaluation showed that the results of the case study were very effective as a basis for further product development. An important takeaway from this evaluation was that RE4SA revisited is most effective when presented as a whole, including the list of requirements, project briefs and architecture models, and less when fragmented into for example just a list of requirements and a functional architecture model.

The answering of these research questions provided important findings in relation to the answering of the main research question. The case study has shown that the proposed solution to the formulated design problem in the form of RE4SA revisited works. In addition, there were valuable lessons learned on the implementation of RE4SA revisited, these have been summarized in table 8. This thesis thus concludes that RE4SA is a valid method for developing a requirements based software architecture. Furthermore, the low cost of implementing this method, and its inherent extendibility, makes is perfectly suited for companies that have limited time and resources at their disposal, in this case startups.

Lessons Learned
Lesson Learned 1: Identified jobs that are not included for development should be recorded for later consideration so that future application components can be more easily identified.
Lesson Learned 2: Clearly defining the need of the client, with an attached problem statement, results in more refined jobs that are neither too big or too specific.
Lesson Learned 3: The identification of jobs is an iterative, time consuming process highly dependent on the final granularity levels chosen for all stories.
Lesson Learned 4: Epic stories should always categorize at least two user stories.
Lesson Learned 5: The motivation component of the epic story template should be functional in nature to allow for appropriate mapping to a module.

Lesson Learned 6: The FAM module inherently tries to satisfy, or execute, the motivation of the epic story. Similarly, the problematic situations mirrors the input of the module, and the expected outcome mirrors the output of a module.
Lesson Learned 7: The formulation of requirements on the jobs, epic story and user story levels is not a top down one shot approach, instead there will be continuous evaluation between the different levels of granularity and adjustments to stories to place them on the right level.
Lesson Learned 8: By identifying the action VERB and NOUN in the functional motivation, the name of the resulting module can often be derived.
Lesson Learned 9: The number of outgoing information flows of a module cannot easily be derived from an epic story.
Lesson Learned 10: Epic stories with similar problematic situations can hint at similar origins.
Lesson Learned 11: By extracting the verb and nouns from the action component of a user story, it is possible to map user stories to features depicted in a feature diagram.
Lesson Learned 12: For each job, on average 4 epic stories can be formulated, and for each epic story, 2.9 user stories can be formulated.
Lesson Learned 13: The outcome of an epic story should try to avoid describing the problematic situation or functional motivation in a negative form or simple nonfunctional states such as knowledge.
Lesson Learned 14: When a level of user story granularity appropriate for the project is established, this change can be propagated to the epic and job, and subsequent higher level job levels until all requirements are consistent with each other.
Lesson Learned 15: The list of requirements, functional architecture model, project briefs, and context model on their own do not offer enough functional details to start development, together they form a comprehensive overview of the requirements and their functional satisfaction in the architecture.

Table 8 - Lessons learned overview.

7.2 Discussion

This section contains a discussion of the performed project. The discussion contains an overview of possible shortcomings of the research and suggests possible follow up studies that can be conducted to further strengthen the theory explored in this design study.

It should be kept in mind that this study only considers the RE4SA revisited method as part of the case study at a small Utrecht based start up. Although the method is tested rigorously in real life circumstances, it is highly recommended that it is tested in case studies that consider different conditions, such as larger scale organizations, public institutions, and startups that are moving into expanding their product line. This will allow future researchers to formulate further conclusions on the applicability of RE4SA and identify possible limiters to this methodology. As it is, the results of this study might have been influenced by the extreme stresses present in any early stage startup. The execution of the research was not done under the supervision of experienced

product managers. Although this prevented bias from influencing the study, it should be interesting to further examine if epic stories can further assist experienced product managers. This applies to the construction of the architecture as well.
Originally, the study considered it useful to create various additional architecture models, due to scope constraints the information and concurrency views were not created. As such the possible effect of epic story formulation on these models was not studied.
The case study considered only the initial software specification for the startup, not its actual development. It is recommended that a future study either does both design and development of the software using RE4SA, or uses a design resulting from RE4SA to further explore its usefulness for development.
The case study considered a highly autonomous system relying heavily on opaque systems, for example those driven by deep learning, a technology that exists somewhat in a black box. This is of great influence to the formulated requirements and resulting system design. It is recommended that the same method is applied to a design that is more transparent and note the differences in formulated requirements. This study brings up the applied count of RE4SA projects to sample size 1, the researchers are looking forward to seeing this number grow to further validate the method's applicability. It should be noted that due to the sensitive nature of automating business practices in the highly competitive field of agriculture, some detailed information, including user stories, have been omitted from this report.

The most important findings of this study are as follows.

- Epic stories were considered to be a valid method for documenting requirements.
- Epic stories can be used as a basis for modeling a functional software architecture.
- By matching requirements and architecture through RE4SA, a valid development roadmap can be created in the form of the context model, functional architecture model, and scenarios,.
- Project briefs are essential for communicating the RE4SA results with a development team.
- Finding the right level of granularity to match RE4SA levels takes many iterations, so far no ideal approach has been identified.
- The RE4SA revisited method works in this case study, and it is recommended it is applied in more case studies to further validate its usefulness.

References

[0] W. Lutz, W. Butz and S. KC, *World population and human capital in the twenty-first century*, 1st ed. Oxford: Oxford University Press.

[1] P. Gerland, A. Raftery, H. Sevčíková, N. Li, D. Gu, T. Spoorenberg, L. Alkema, B. Fosdick, J. Chunn, N. Lalic, G. Bay, T. Buettner, G. Heilig and J. Wilmoth, "World population stabilization unlikely this century", *Science*, vol. 346, no. 6206, pp. 234-237, 2014.

[2] P. Gerland, A. Raftery, H. ev ikova, N. Li, D. Gu, T. Spoorenberg, L. Alkema, B. Fosdick, J. Chunn, N. Lalic, G. Bay, T. Buettner, G. Heilig and J. Wilmoth, "World population stabilization unlikely this century", *Science*, vol. 346, no. 6206, pp. 234-237, 2014.

[3] P. Ehrlich and J. Holdren, "Impact of Population Growth", *Science*, vol. 171, no. 3977, pp. 1212-1217, 1971.

[4] M. Sutton and S. Ayyappan, *Our nutrient world*, 1st ed. Edinburgh: Centre for Ecology and Hydrology, 2013.

[5] F. KRAAS, "Megacities and global change: key priorities", *The Geographical Journal*, vol. 173, no. 1, pp. 79-82, 2007.

[6] F. Viviano, "This tiny country feeds the world", *National Geographic Magazine*, no. 2017-09, pp. 82-109, 2017.

[7] K. van Thienen-Visser and J. Breunese, "Induced seismicity of the Groningen gas field: History and recent developments", *The Leading Edge*, vol. 34, no. 6, pp. 664-671, 2015.

[8] "Work - Migration and Home Affairs - European Commission", *Migration and Home Affairs - European Commission*, 2018. [Online]. Available: https://ec.europa.eu/home-affairs/what-we-do/policies/legal-migration/work_en. [Accessed: 26- Oct- 2018].

[9] G. Lucassen, F. Dalpiaz, J. Werf and S. Brinkkemper, "The Use and Effectiveness of User Stories in Practice", *Requirements Engineering: Foundation for Software Quality*, vol. 9619, pp. 205-222, 2016.

[10] G. Lucassen, F. Dalpiaz, J. van der Werf and S. Brinkkemper, "Forging high-quality User Stories: Towards a discipline for Agile Requirements", *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, 2015.

[11] G. Lucassen, M. van de Keuken, F. Dalpiaz, S. Brinkkemper, G. Sloof and J. Schlingmann, "Jobs-to-be-Done Oriented Requirements Engineering: A Method for Defining Job Stories", *Requirements Engineering: Foundation for Software Quality*, vol. 10753, pp. 227-243, 2018.

[12] S. Brinkkemper and S. Pachidi, "Functional Architecture Modeling for the Software Product Industry", *Software Architecture*, vol. 6285, pp. 198-213, 2010.

[13] E. Kamsties, J. Horkoff and F. Dalpiaz, *Requirements engineering*, 1st ed. Utrecht: Springer, pp. 227-241.

[14] Jansen N. & van Rhijn, J, utrecht Architecture Description Language, *internal documentation* GRIMM UU, 2018.

[15] S. Brinkkemper, G. Lucassen, G. Sloof, F. Dalpiaz and M. van de Keuken, "Jobs-to-be-Done Oriented Requirements Engineering: a Method for Defining Job Stories", *Presentation Slides*, Utrecht, Netherlands, 2018.

[16] J. Cleland-Huang, R. Hanmer, S. Supakkul and M. Mirakhorli, "The Twin Peaks of Requirements and Architecture", *IEEE Software*, vol. 30, no. 2, pp. 24-29, 2013.

[17] K. van Hee, N. Sidorova and J. van der Werf, "Construction of Asynchronous Communicating Systems: Weak Termination Guaranteed!", *Software Composition*, vol. 6144, pp. 106-121, 2010.

[18] N. Rozanski and E. Woods, *Software systems architecture*. Upper Saddle River, NJ: Addison-Wesley, 2012.

[19] J. van der Werf, "Compositional Verification of Asynchronously Communicating Systems", *Formal Aspects of Component Software*, vol. 8997, pp. 49-67, 2015.

[20] Brinkkemper S. GRIMM project. *Internal documentation*, 2018.

[21] R. Wieringa, "Design science as nested problem solving", *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology - DESRIST '09*, 2009.

[22] G. Lucassen, F. Dalpiaz, J. Werf and S. Brinkkemper, "The Use and Effectiveness of User Stories in Practice", *Requirements Engineering: Foundation for Software Quality*, vol. 8997, pp. 205-222, 2016.

[23] G. Lucassen, F. Dalpiaz, J. van der Werf and S. Brinkkemper, "Improving agile requirements: the Quality User Story framework and tool", *Requirements Engineering*, vol. 21, no. 3, pp. 383-403, 2016.

[24] G. Lucassen, F. Dalpiaz, J. van der Werf and S. Brinkkemper, "Improving User Story Practice with the Grimm Method: A Multiple Case Study in the Software Industry", *Requirements Engineering: Foundation for Software Quality*, vol. 10153, pp. 235-252, 2017.

[25] G. Lucassen, F. Dalpiaz, J. van der Werf and S. Brinkkemper, "Visualizing User Story Requirements at Multiple Granularity Levels via Semantic Relatedness", *Conceptual Modeling*, vol. 9974, pp. 463-478, 2016.

[26] M. Huisman, H. Bos, S. Brinkkemper, A. van Deursen, J. Groote, P. Lago, J. van de Pol and E. Visser, "Software that Meets Its Intent", *Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications*, vol. 9953, pp. 609-625, 2016.

[27] T. Ormerod and C. Sas, *Proceedings of the 21st British CHI Group annual Conference on HCI 2007*. New York, N.Y.: ACM Press, 2007, pp. 167-175.

[28] N. Bik, G. Lucassen and S. Brinkkemper, "A Reference Method for User Story Requirements in Agile Systems Development", *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, 2017.

[29] N. Alshuqayran, N. Ali and R. Evans, "A Systematic Mapping Study in Microservice Architecture", *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, 2016.

[30] D. Namiot and M. Sneps-Sneppe, "On Micro-services Architecture", *International Journal of Open Information Technologies*, vol. 2, no. 9, 2014.

[31] M. Amaral, J. Polo, D. Carrera, I. Mohomed, M. Unuvar and M. Steinder, "Performance Evaluation of Microservices Architectures Using Containers", *2015 IEEE 14th International Symposium on Network Computing and Applications*, 2015.

[32] A. Krylovskiy, M. Jahn and E. Patti, "Designing a Smart City Internet of Things Platform with Microservice Architecture", *2015 3rd International Conference on Future Internet of Things and Cloud*, 2015.

[33] A. Klement, "Replacing the user story with the job story", *Medium*, 2013. .

[34] M. Cohn, *User stories applied*, 1st ed. Boston: Addison-Wesley, 2013.

[35] X. Wang, L. Zhao, Y. Wang and J. Sun, "The Role of Requirements Engineering Practices in Agile Development: An Empirical Study", *Requirements Engineering*, vol. 432, pp. 195-209, 2014.

[36] D. Traynor, *Intercom on Jobs-To-be-Done*. 2016.

[37] A. Klement, *When coffee and kale compete: Become great at making Products People will buy*, 1st ed. New York, NY: NYC Publishing.

[38] C. Christensen, S. Anthony, G. Berstell and D. Nitterhouse, "Finding the Right Job For Your Product", *MIT Sloan Management Review*, no. 48, p. 38, 2007.

[39] A. Ulwick, "Turn customer input into innovation.", *Harvard Business Review*, vol. 80, no. 1, pp. 91-97, 2002.

[40] A. Ulwick, "Mapping the Job-to-be-Done", *Medium*, 2017.

[41] A. Ulwick and A. Osterwalder, *Jobs to be done*, 1st ed. New York, NY: Idea Bite Press, 2016.

[42] A. Klement, "Your Job Story Needs A Struggling Moment", *Medium*, 2016. .

[43] D. Blair, "Anything Goes, Relevancy and Why I struggle with Christensen's Jobs Theory", *LinkedIn*, 2017.

[44] Strategyn LLC, "The Jobs-to-be-Done growth strategy matrix", Strategyn LLC, 2016.

[45] M. Tessmann, "Job Stories in Practice", *Medium*, 2016.

[46] C. Potts, "Using schematic scenarios to understand user needs", *Proceedings of the conference on Designing interactive systems processes, practices, methods, & techniques - DIS '95*, pp. 247-256, 1995.

[47] A. van Lamsweerde, "Goal-oriented requirements engineering: a guided tour", *Proceedings Fifth IEEE International Symposium on Requirements Engineering*.

[48] T. Bebensee, I. van de Weerd and S. Brinkkemper, "Binary Priority List for Prioritizing Software Requirements", *Requirements Engineering: Foundation for Software Quality*, vol. 6182, pp. 67-78, 2010.

[48] H. Hofmann and F. Lehner, "Requirements engineering as a success factor in software projects", *IEEE Software*, vol. 18, no. 4, pp. 58-66, 2001.

[50] R. Mesquita, A. Jaqueira, M. Lucena and F. Alencar, "US2StarTool: Generating i* Models from User Stories", *Proceedings of the Eighth International i* Worksop (istar 2015)*, vol. 978, 2015.

[51] N. Condori-Fernandez, M. Daneva, K. Sikkel, R. Wieringa, O. Dieste and O. Pastor, "A systematic mapping study on empirical evaluation of software requirements specifications techniques", *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pp. 502-505, 2009.

[52] K. Ryan, "The role of natural language in requirements engineering", *[1993] Proceedings of the IEEE International Symposium on Requirements Engineering*, pp. 240-242, 1993.

[53] M. Landhausser and A. Genaid, "Connecting User Stories and code for test development", *2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE)*, pp. 33-37, 2012.

[54] A. Gomez, G. Rueda and P. Alarcón, "A Systematic and Lightweight Method to Identify Dependencies between User Stories", *Lecture Notes in Business Information Processing*, vol. 48, pp. 190-195, 2010.

[55] D. Damian, J. Chisan, L. Vaidyanathasamy and Y. Pal, "Requirements Engineering and Downstream Software Development: Findings from a Case Study", *Empirical Software Engineering*, vol. 10, no. 3, pp. 255-283, 2005.

[56] X. Wang, L. Zhao, Y. Wang and J. Sun, "The Role of Requirements Engineering Practices in Agile Development: An Empirical Study", *Requirements Engineering*, vol. 432, pp. 195-209, 2014.

[57] M. Kassab, "The changing landscape of requirements engineering practices over the past decade", *2015 IEEE Fifth International Workshop on Empirical Requirements Engineering (EmpiRE)*, 2015.

[58] Y. Wautelet, S. Heng, M. Kolp and I. Mirbel, "Unifying and Extending User Story Models", *Advanced Information Systems Engineering*, vol. 8484, pp. 211-225, 2014.

[59] Y. Lowrey, "Startup Business Characteristics and Dynamics: A Data Analysis of the Kauffman Firm Survey", *RAND Working Paper Series WR*, 2018. [Online]. Available: http://dx.doi.org/10.2139/ssrn.1496545. [Accessed: 26- Oct- 2018].

[60] R. Lussier, "Startup business advice from business owners to would-be entrepreneurs", *Advanced Management Journal*, vol. 60, no. 1, p. 10, 1995.

[61] M. Gelderen, R. Thurik and N. Bosma, "Success and Risk Factors in the Pre-Startup Phase", *Small Business Economics*, vol. 24, no. 4, pp. 365-380, 2005.

[62] G. Ralston, "YC's Essential Startup Advice", *Y Combinator*, 2017.

[63] P. Graham, "Essays", *Paul Graham*.

[64] Rocketspace, "The 7 Characteristics Successful Startups Share", *RocketSpace*, 2017. .

[65] A. Hall, "12 Characteristics Of Wildly Successful Startups", *Forbes*, 2013. .

[66] Rocketspace, "Why Tech Startups Fail and How Founders Can Bounce Back", *RocketSpace*, 2018. .

[67] G. Lucassen, F. Dalpiaz, J. van der Werf and S. Brinkkemper, "Bridging the Twin Peaks: the Case of the Software Industry", *Proceedings of the Fifth International Workshop on Twin Peaks of Requirements and Architecture*, pp. 24-28, 2015.

[68] D. Silverstein, "Technique 1 - Jobs to be Done", *The Innovator's Toolkit*. .

[69] K. Waters, "Prioritization using MoSCoW", *All About Agile*, 2009. [Online]. Available: http://www.allaboutagile.com/prioritization-using-moscow/. [Accessed: 26- Oct- 2018].

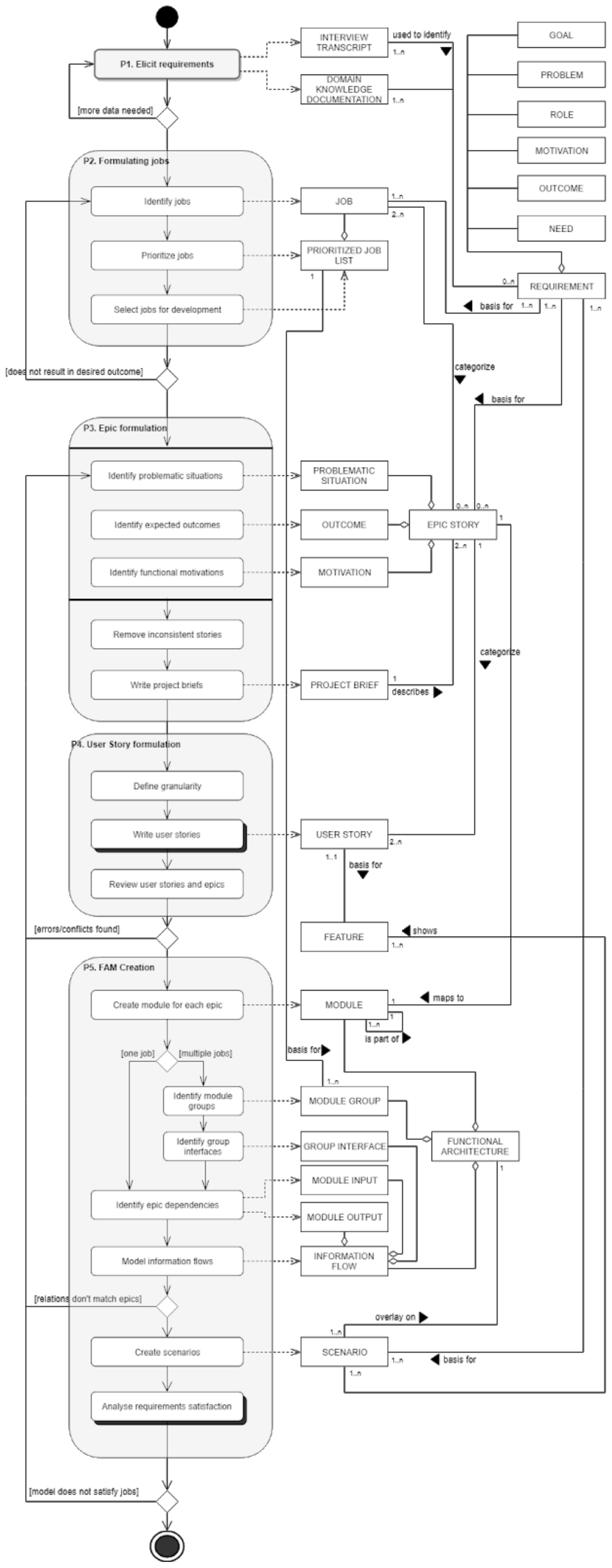
[70] M. Cohn, "User Stories and User Story Examples by Mike Cohn", *Mountain Goat Software*, 2018. [Online]. Available: https://www.mountaingoatsoftware.com/agile/user-stories. [Accessed: 02- Dec- 2018].

[71] L. Bass, P. Clements and R. Kazman, Software architecture in practice, 3rd ed. Upper Saddle River, N.J.: Addison-Wesley, 2013.

[72] C. Giardino, X. Wang and P. Abrahamsson, "Why Early-Stage Software Startups Fail: A Behavioral Framework", *Software Business. Towards Continuous Value Delivery*, pp. 27-41, 2014.

[73] M. Cusumano, "Evaluating a startup venture", *Communications of the ACM*, vol. 56, no. 10, p. 26, 2013.

Appendix A - Detailed Process Deliverable Diagram of RE4S



Appendix B - Activity Table for RE4SA PDD

Activity	Sub-activity	Description
Elicit Requirements		Domain experts are interviewed to create a INTERVIEW TRANSCRIPT and DOMAIN KNOWLEDGE DOCUMENTATION is gathered so that the REQUIREMENTS can be gathered from these sources.
Jobs formulation	Identify jobs	By looking through the REQUIREMENTS high level desired functionality can be uncovered and formulated in a JOB.
	Prioritize jobs	Once the JOBs have been formulated, they can be prioritized by employing a prioritization technique fitting for the project scope, such as calculating opportunity scores or MSCoW. These prioritized jobs are ordered in a PRIORITIZED JOB LIST.
	Select jobs for development	Once the PRIORITIZED JOB LIST has been created, high priority JOBs are selected for modeling. This is done by taking the project scope and feasibility of the project into account and storing JOBs that do not have a high priority for later development.
Epic Story formulation	Identify problematic situations	Keeping the job in mind, a PROBLEMATIC SITUATION is identified to lay the foundation for an EPIC STORY.
	Identify functional motivation	For each PROBLEMATIC SITUATION a functional MOTIVATION is described, this describes the functionality that will give a reason for the satisfaction of the expected OUTCOME.
	Identify expected outcome	For each now partially completed epic story an expected OUTCOME is formulated that will form a solution to the PROBLEMATIC SITUATION.
	Remove inconsistent stories	Each EPIC STORY is compared to see if there are inconsistencies, such as duplicate stories or contradictory stories.
	Write project briefs	For each JOB, a PROJECT BRIEF is written, starting with the name of the job and followed by a important EPIC STORY categorized under this JOB. A small description is written to describe the JOB and its relations to the EPIC STORIES. Finally, a list of success metrics is added that will indicate when the JOB can be considered satisfied.
User story formulation	Define granularity	The minimum required granularity is set for defining USER STORIES, this level determines a minimum level of detail and means that USER STORIES should not be broken up into multiple stories. Stories that can be broken up are EPIC STORIES.
	Write user stories	For each EPIC STORY, USER STORIES are written.

	Review user stories and epics	The consistency between EPIC STORIES and USER STORIES is reviewed, taking special care to make sure the defined granularity levels are obeyed.
FAM Creation	Create module for each epic	For each EPIC STORY a MODULE is created. The MOTIVATION component is used to see what functional name the MODULE should be given.
	Identify module groups	If there are multiple JOBs, loosely group MODULEs together based on the JOB they belong to.
	Identify group interfaces	Identify GROUP INTERFACEs place MODULE GROUPs accordingly.
	Identify epic dependencies	Considering the OUTCOME and PROBLEMATIC SITUATION of each MODULEs EPIC STORY, attempt to identify dependencies between MODULEs in the same MODULE GROUP.
	Model information flows	Considering the OUTCOME and PROBLEMATIC SITUATION of each EPIC STORY, identify and connect MODULEs with INFORMATION FLOWs.
	Create scenarios	With the FUNCTIONAL ARCHITECTURE created, draw SCENARIOs over the model depicting the use cases which show related features in the system. For each SCENARIO draw arrows on the model, with numbers depicting the sequence of the use case.
	Analyse requirements satisfaction	Considering the REQUIREMENTs, the model is analysed to see if all have been considered and satisfied.

Appendix C - Concept Table for RE4SA PDD

Concept	Description
INTERVIEW TRANSCRIPT	A recording or set of notes of an interview with domain experts and / or shareholders from which requirements can be extracted.
DOMAIN KNOWLEDGE DOCUMENTATION	Documents such as manuals, transcripts, informative web pages describing functionality or aspects of existing processes and practices, from which requirements can be extracted.
GOAL	The desired result of a process or action.
PROBLEM	A difficulty encountered by a stakeholder for which a solution can be created.
ROLE	A stakeholder within the domain, such as a person from which processes emerge or the end user of the system. Can be human or autonomous machine.
MOTIVATION	The reason behind a certain need or decision.
OUTCOME	The end result of a process.
NEED	A desire from a stakeholder. a thing that is wanted or required.
REQUIREMENT	A requirement is a desired situation upon which the design of the system can be based.
JOB	A job is an abstract requirement formulated with the template *"Help me..."* It is functional in nature and indicates a need from a user.
PRIORITIZED JOB LIST	The prioritized job list is a filtered list of jobs based on what is needed to satisfy the core requirements of the end user, the scope of the project, and / or, the practicality of their satisfaction within project budget.
PROBLEMATIC SITUATION	The problematic situation is the first component of an epic story. It describes a situation for which a solution needs to be formulated.
OUTCOME	The (expected) outcome is the situation, or a condition, that emerges once the problematic situation is satisfied.
MOTIVATION	The (functional) motivation describes the reason why the problematic situation will be solved in the expected outcome. It consists of a subject, action verb and direct object.
EPIC STORY	An epic story is a formulated requirement consisting

	of a problematic situation, functional motivation, and expected outcome.
PROJECT BRIEF	A project brief is a document that describes a job and the epic stories that can be categorized underneath. It includes a list of success metrics that can help developers determine if the help question of the job has been satisfied.
USER STORY	A user story is a formulated requirement that consists of a role, action, and benefit.
FEATURE	A feature is functionality as it appears in the software system. It can consists of one or multiple behaviors.
FUNCTIONAL MODULE	A functional module is a component of a software architecture that shows a specific functional process.
MODULE GROUP	A module group is a set of modules with a shared theme that together map to a job.
MODULE INPUT	Module input is a part of software architecture that shows an information flow into a module.
MODULE OUTPUT	Module output is a part of software architecture that shows an information flow emerging from a module.
INFORMATION FLOW	An information flow is a relation between two modules, showing a transfer between the finishing of one functional process and the start of another.
FUNCTIONAL ARCHITECTURE	A functional (software) architecture is a model consisting of informations flows and modules that shows the desired functional behavior of a system.
SCENARIOS	Scenarios are overlays upon the functional architecture model that depict a specific use case within the functional behavior of the system. It consists of red arrows with numbers to depict sequence of events.

Appendix D - Project Briefs

Appendix D.1 - Brief Template

Project Brief - JOB HERE

(Primary) Epic story here

What problem are we solving and why?
Short description here

What value do we deliver to the customer?
- Related Epic Story 1
- Related Epic Story n

How will we measure success?
- result metric 1
- result metric n

Appendix D.2 - Autonomous Climate Computer Briefs

Project Brief - Help me use and improve growth models

When *a plant is selected to be grown in a greenhouse,* ***I want*** *to determine plant requirements and environment,* ***so that*** *the initial growing conditions can be determined.*

What problem are we solving and why?
When planting a new crop, a farmer will mostly adhere to the growing conditions specified by the producer of the seeds or seedlings he is planting. Over time the farmer will instinctively or actively make changed based on the specific configuration of his greenhouse to improve his yields. However, such process can be made much more efficient by allowing an A.I. to start out with the defined growth model and then rapidly improving it over time as more data becomes available. It will identify subtle patterns that are difficult to discover for humans, such as the effect of a sudden drop in external temperatures outside on temperatures in a specific zone, and the subsequent effect on harvest of the plants there 4 months later, and suggest improvements to the growth model based on this information.

What value do we deliver to the customer?
- **When** a plant is selected to be grown in a greenhouse, I want to determine plant requirements and environment, so that the initial growing conditions can be determined.
- **When** new requirements are known, I want to combine all available information, so that a growth model can be constructed
- **When** an improved dataset is made available, I want to compare this to actual harvest yield data, so that I can suggest changes to the growth model

How will we measure success?

- The outcome helps to continuously improve growth methods employed by the autonomous system.
- The solution will offer a basis for the initial setup of the greenhouse climate conditions according to the specifications of the farmer or seed manufacturer.
- A database of improvement suggestions will be created, allowing for future comparison by other greenhouses.

Project Brief - Help me predict the right course of action

***When** I have a yield prediction, **I want** to plan the right course of action, **so that** I can set the right climate conditions.*

What problem are we solving and why?

In traditional farming practices, farmers work based on their gut feeling or by using simple knowledge rules. In greenhouse conditions everything is about maximizing efficiency and reducing operational costs to obtain the biggest crop yield possible. Employing a neural network, we can effectively use growth models and climate data to analyse subtle patterns in greenhouse conditions and their effects on plant production. Based on these predictions a simpler algorithm suggests changes such as heat up or humidity down and sends them back to the neural net for analysis, refining with each iteration until finally an optimal course of action is determined. This course of action is then executed by the climate control systems.
Our (improved) growth models, and internal and external climate data has to be formatted in such a way our default prediction neural network can use it to forecast crop yields. In addition, we want to log all iterations and action plans so that developers can obtain insights on the neural network efficiency, but also so that we can send the most successful plans on to the growth model improver.

What value do we deliver to the customer?

- **When** using a neural network, **I want** to gather and format data continuously, **so that** an A.I. can interpret the data.
- **When** there is new prediction data available, **I want** to run it through a trained neural network, **so that** I can make yield predictions.
- **When** I have a yield prediction, **I want** to plan the right course of action, **so that** i can set the right climate conditions.
- **When** I take autonomous action, **I want** to log all actions, conditions, and processes, **so that** I can improve my growth models.

How will we measure success?

- The greenhouse will autonomously assign tasks to control systems to improve climate conditions.
- The greenhouse climate will remain more stable as preemptive measures are calculated into the action plan.
- A log will be created with all historical action plans.

Project Brief - Help me control lighting

__When__ lighting needs to be adjusted, __I want__ to adjust lamp brightness, __so that__ lamps are on only when required.

What problem are we solving and why?
Traditionally, lighting is turned on and off on a simple timer, with farmers in control of the light switch should they determine a crop needs more light. This results in many plants receiving too little or too much light. Based on an instruction of the action planner, and the readings from light sensors in the greenhouse to determine if it is day or night, lights will be autonomously turned on and off for each zone. This means that if during the day a specific section of plants has received too little light due to cloud positions, this section might receive more lamplight in the evening.

What value do we deliver to the customer?

- **When** lighting needs to be adjusted, **I want** to adjust lamp brightness, **so that** lamps are on only when required.
- **When** more or less shadow is needed, **I want** to open or close the screen, **so that** the screen is in the correct state
- **When** the climate computer gives a light instruction, **I want** to process it based on current conditions, **so that** I maintain correct lighting conditions

How will we measure success?

- Automatic lighting will turn on and off as required.
- Automatic screening will turn on and off as required.
- There will be no errors that are damaging to plants.

Project Brief - Help me control temperature

__When__ receiving a heating request, __I want__ to change the current heating state, __so that__ temperature can go up or stay constant.

What problem are we solving and why?
Heating is turned on and off using basic reactionary principles, the thermometer reaches 20 degrees celcius, and the heating pumps are powered up. This results in highly fluctuating temperatures instead of ideal stable temperatures. By preemptively defining how much heat is required and when, heating can be made more efficient and effective. The same goes for cooling, which is rarely done in most greenhouses as the process is too difficult to control, but is seen as necessary in a world with a changing climate. Currently cooling is done mostly by opening windows, and this is one of the heaviest automated processes in modern climate computers as it also impact humidity and airflow. Current systems use simple formulas based on sensor readings without a predictive element, which results in optimal, but not ideal growing conditions.

What value do we deliver to the customer?

- **When** the climate computer gives a temperature instruction, **I want** to know how to process it based on current conditions, **so that** I maintain the correct temperature conditions
- **When** receiving a heating request, **I want** to change the current heating state, **so that** temperature can go up or stay constant
- **When** receiving a timer, **I want** to turn on the window sprays, **so that** the greenhouse cools down
- **When** receiving a state request, **I want** to open windows to a certain degree, **so that** the appropriate state is achieved

How will we measure success?

- Plants will suffer less heat damage.
- Temperatures will remain more constant throughout the day.
- There will be fewer and less severe temperature fluctuations.

Project Brief - Help me control humidity

***When** receiving a ventilation timer, **I want** to activate fans, **so that** the desired airflow is achieved.*

What problem are we solving and why?
Temperature and humidity control are closely intertwined, and humidity is mostly influenced by how high the temperature of a greenhouse is. However, farmers have some artificial means of increasing humidity in a greenhouse, by for example adding additional moisture to the atmosphere, opening windows to release some moisture, or activating a ventilator to spread our moisture. The subtle interactions are difficult for humans to predict, but easy for an AI to see coming. The humidity instruction will result in a course of actions that effectively allow the system to influence humidity conditions. For example, if temperature goes up in 1 part of the greenhouse (meaning humidity will also go up), a ventilator could be activated to spread that moisture to a part of the greenhouse where windows are open or more moisture is required.

What value do we deliver to the customer?

- **When** receiving a humidity instruction, **I want** to determine a course of action, **so that** I can control humidity systems
- **When** receiving a moisture pump timer, **I want** to activate moisture pumps, **so that** humidity in the greenhouse increases
- **When** receiving a ventilation timer, **I want** to activate fans, **so that** the desired airflow is achieved

How will we measure success?

- This solution will result in a more constant humidity distribution in the greenhouse throughout the day.

- Temperature and humidity will autonomously inform each other of possible changes or interactions.
- Plants will suffer less from overheating or cold damage.

Project Brief - Help me control co2

When *I have determined to pump CO2,* ***I want*** *to activate a co2 pump,* ***so that*** *CO2 is added to the atmosphere.*

What problem are we solving and why?
CO2 is a very difficult and new thing for farmers to actively control in the greenhouse. This is often due to sensors not being able to adequately measure levels in the atmosphere and its effect on growth models being unknown. However, AI can account for both these shortcomings, not caring about the actual sensor output but caring about its effect on crop yields. Autonomously adding CO2 to the atmosphere will increase plant growth speed.

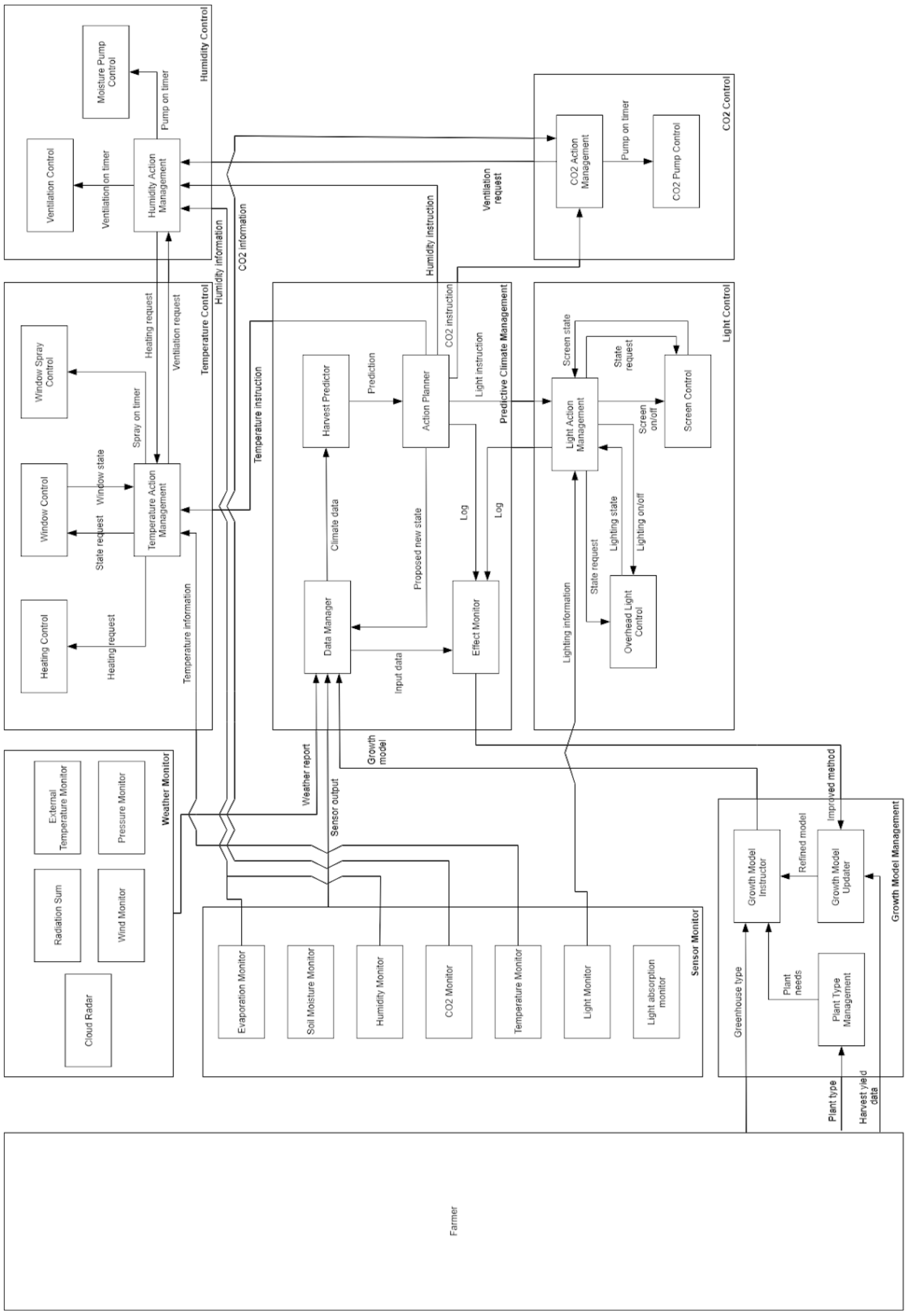
What value do we deliver to the customer?

- **When** receiving a CO2 instruction, **I want** to determine how to achieve the desired values, **so that** I can activate specific systems
- **When** I have determined to pump CO2, **I want** to activate a co2 pump, **so that** CO2 is added to the atmosphere

How will we measure success?

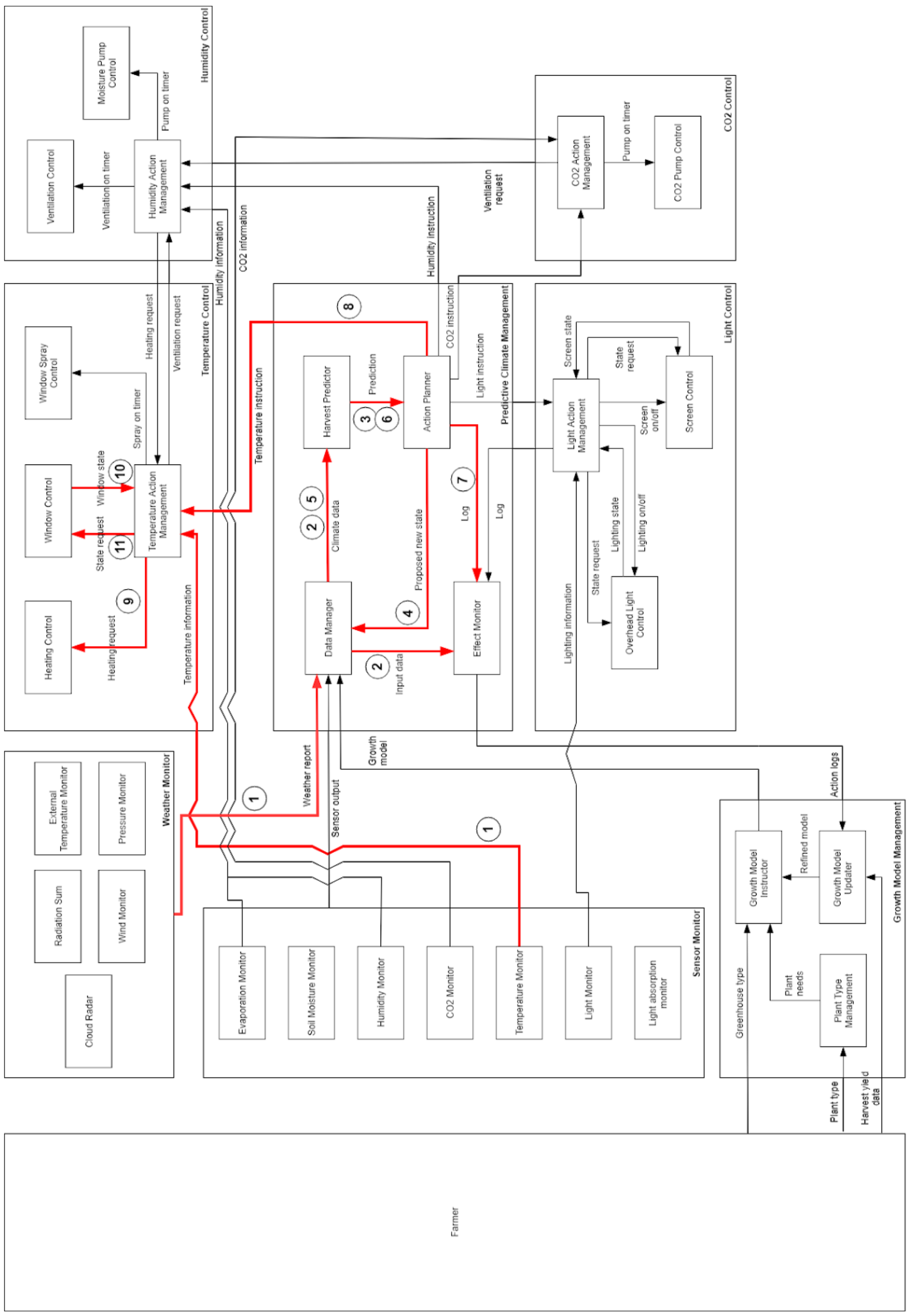
- More optimal CO2 levels to increase plant health.
- Levels should never pose a danger to humans.
- The system will autonomously interact with humidity control to distribute co2 evenly.

Appendix E - Functional Architecture Model

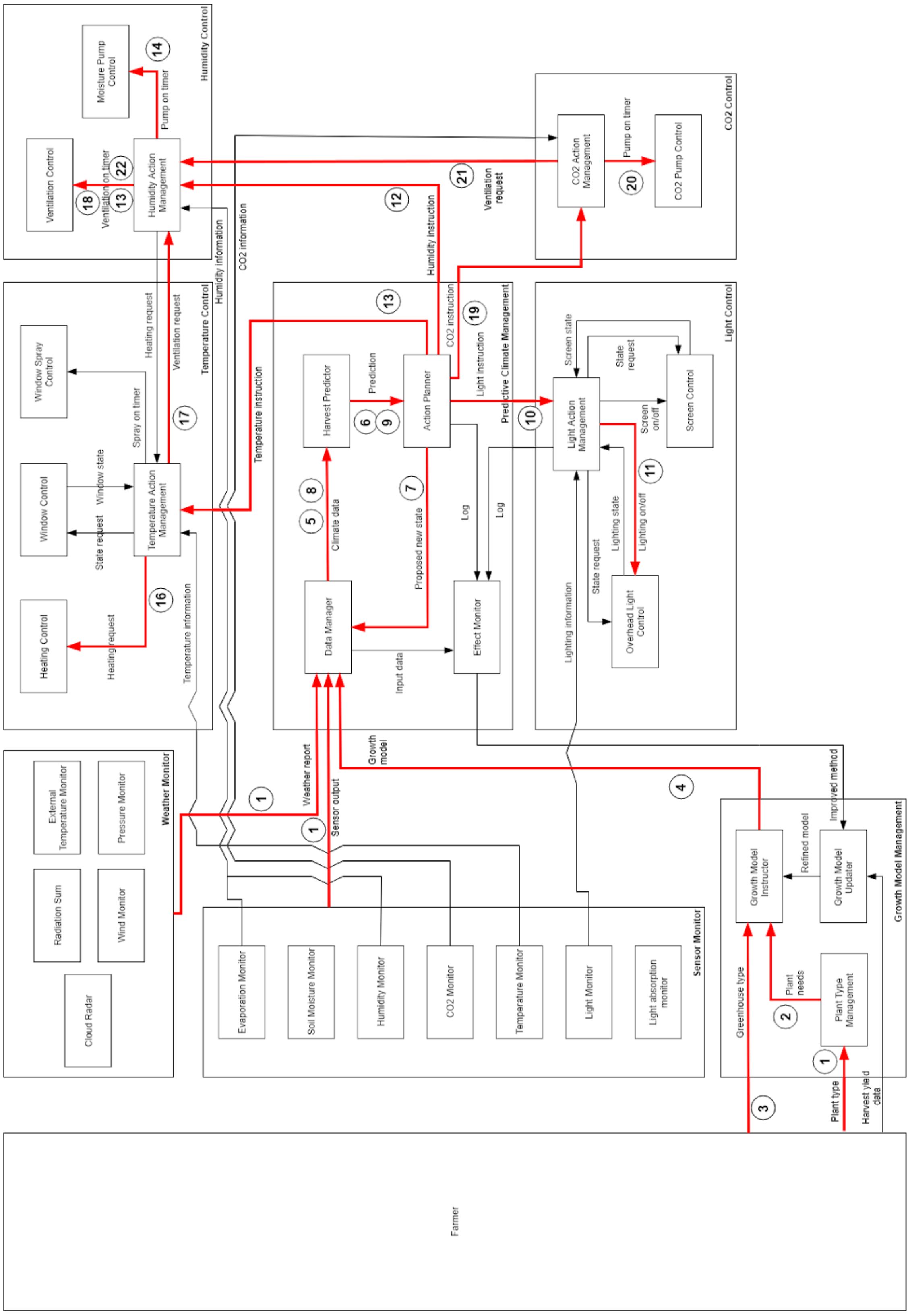


Appendix F - Scenarios

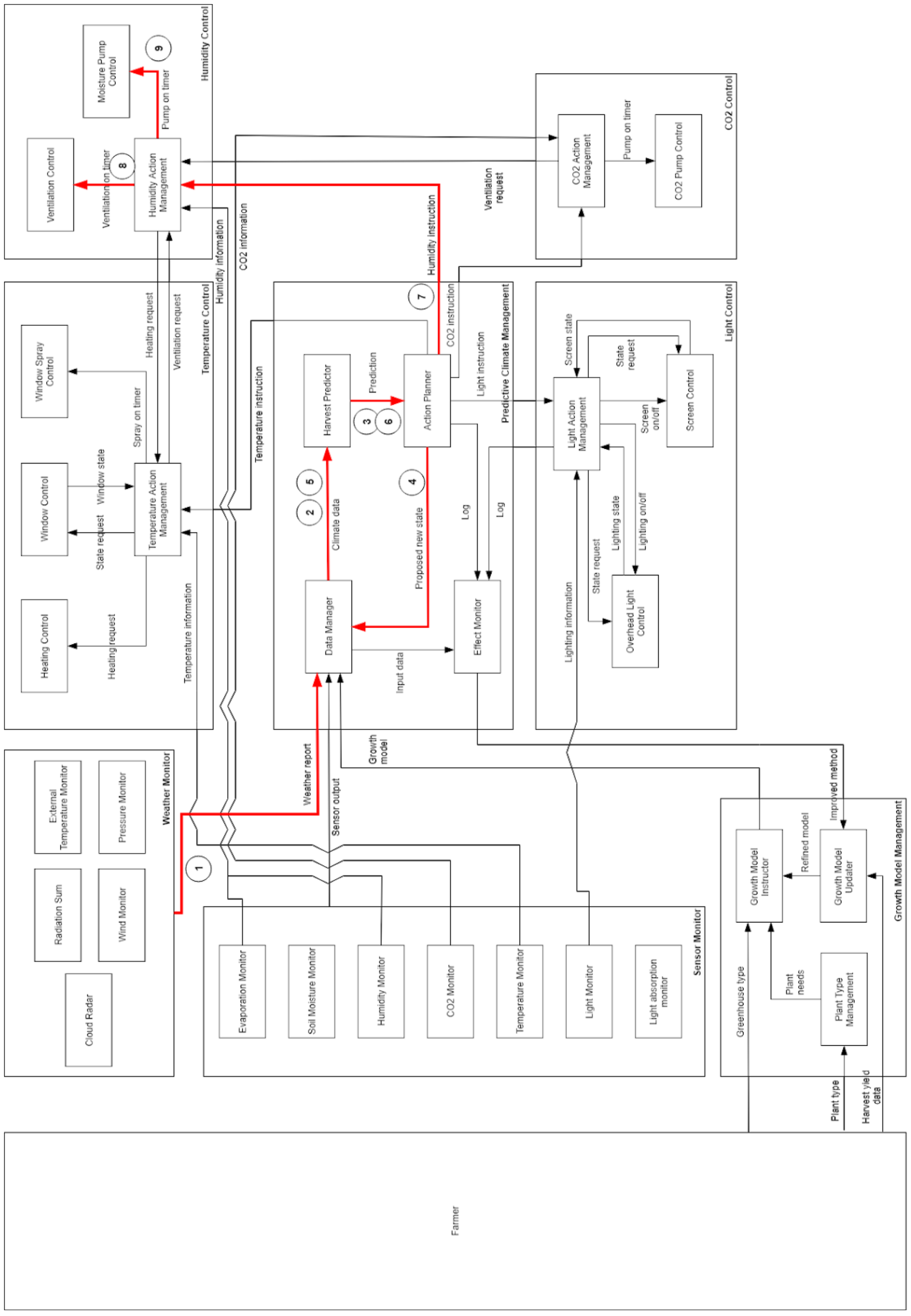
Scenario 1: Heating System Activation



Scenario 2 - System Initialization



Scenario 3 - Wind Direction Compensation



Appendix G - Unfiltered jobs list

1	**Help me predict a course of action**	**Type**
2	Help me predict production numbers	M
3	Help me determine sales prices	W
4	Help me record data for analysis	M
5	Help me understand the effects of my actions	M
6	Help me predict sales	W
7	Help me predict harvest	M
8	Help me predict the effect of environmental conditions	M
9	**Help me control internal climate**	N
10	Help me maintain the correct temperature	M
11	Help me maintain correct lighting	M
12	Help me maintain correct humidity	M
13	Help me administer CO2	S
14	Help me maintain correct ventilation	M
15	Help me monitor weather	S
16	Help me monitor moisture deficit	S
17	Help me water plants	S
18	**Help me take care of plants**	M
19	Help me monitor plant growth	M
20	Help me fertilize plants	S
21	Help me shape growth	C
22	Help me cut leafs	W
23	Help me slow growth	W
24	Help me harvest	W
25	**Help me manage diseases**	W
26	Help me fight pests	W
27	Help me detect diseases	C
28	Help me fight diseases	W
29	Help me monitor fungi	W
30	Help me identify a disease	W
31	Help me take preventative measures	W
32	Help me detect dying plants	W
33	**Help me take care of seedlings**	W
34	Help me plant seedlings	W

35	Help me plant seeds	W
36	Help me prepare growth medium for seedlings	W
37	Help me filter unfit seedlings	W
38	Help me grow seeds into seedlings	W
39	**Help me manage inbound logistics**	W
40	Help me store inventory	W
41	Help me maintain supplies	W
42	Help me order supplies	W
43	**Help me manage internal logistics**	W
44	Help me move pots	W
45	Help me move plants	W
46	Help me clean the greenhouse	W
47	Help me cull dead plants	W
48	**Help me manage outbound logistics**	W
49	Help me package plants	W
50	Help me transport harvest for packaging	W
51	Help me transport packaged plants	W
52	Help me keep track of transport	W
53	**Help me control infrastructure**	
54	Help me control power generation	W
55	Help me charge robot batteries	W
56	Help me monitor machine status	W
57	Help me maintain systems	W
58	**Help me manage knowledge tasks**	S
59	Help me catalogue correct growing conditions for various plants	M
60	Help me catalogue correct growing procedure for various plants	W
61	Help me adhere to growth models	S
62	Help me determine/optimize/make growth models	S
63	Help me keep my growth models economically viable	M
64	Help me adhere to the law	W
65	Help me stay up to date with new research	W

Appendix H - Numbered Requirements List

1	**Help me predict the right course of action**
1-1	When using a neural network, I want to gather and format data continuously, so that the A.I. can interpret the data
1-1:1	As a data manager, I want to provide an id with my dataset, so that the predictor knows if it is a hypothetical or current state prediction
1-1:2	As a data manager, I want to use sensor output data, so that my data set always contains the most actual information on climate conditions
1-1:3	As a data manager, I want to use recorded weather data, so that the predictor is alerted when a new prediction is needed
1-1:4	As a data manager, I want to format data in a certain way, so that it can be processed by a trained neural network
1-1:5	As a data manager, I want to send all input data to the effect monitor, so that it can be used to improve predictions
1-1:6	As a data manager, I want to store all input data in a database, so that a farmer can later access it
1-2	When there is new prediction data available, I want to run it through a trained neural network, so that I can make yield predictions
1-2:1	As a harvest predictor, I want to use prediction data to determine what the effects of current conditions will be, so that I can predict yields
1-2:2	As a harvest predictor, I want to let the action planner know what kind of prediction I am providing, so that it knows if it is based on its own action plan, or on current conditions
1-2:3	As a harvest predictor, I want to always pass along a copy of the original data, so that the action planner can determine what needs to be changed
1-2:4	As a harvest predictor, I want to tag a file for execution when confidence is above 95%, so that the analysis spiral stops
1-3	When I have a yield prediction, I want to plan the right course of action, so that i can set the right climate conditions
1-3:1	As an action planner, I want to determine the desired climate conditions for every greenhouse zone based on predictions, so that a new state can be proposed
1-3:2	As an action planner, I want to formulate instructions for specific systems based on predictions, so that climate conditions improve or remain stable
1-3:3	As an action planner, I want to use yield prediction and climate information on current conditions, so that I can formulate the correct course of action
1-3:4	As an action planner, I want to test a plan before I run it, so that I don't make mistakes
1-3:5	As an action planner, I want to determine the ideal sensor output, so that individual system action management can determine how to get there
1-3:6	As an action planner, I want to let systems know if the ideal situation has been reached, so that they know when to interrupt a process if needed

1-4	When I take autonomous action, I want to log all actions, conditions, and processes, so that I can improve my growth models
1-4:1	As an effect monitor, I want to store proposed actions and executed actions seperately, so that I can later compare planner accuracy
1-4:2	As an effect monitor I want to use executed plans that have resulted in higher yields to formulate better growth methods, so that growth model management can improve its performance
2	**Help me control lighting**
2-1	When the climate computer gives a light instruction, I want to process it based on current conditions, so that I can formulate the right lighting instructions
2-1:1	As a light manager, I want to use a light instruction to determine if a zone needs light or shadow, so that the correct lighting is implemented
2-1:2	As a light manager, I want to use output from light sensors, so that I use real world situations and can monitor if my instructions have effect
2-1:3	As a light manager, I want to always check the state of the screens and lamps, so that lamps do not turn on when screens are closed
2-1:4	As a light manager, I want to use knowledge about current conditions and desired conditions to determine what should be changed, so that conditions can be improved
2-2	When more or less shadow is needed, I want to open or close the screen, so that the screen is in the correct state
2-2:1	As the screen controller, I want to open a screen, so that light intensity rises
2-2:2	As the screen controller, I want to close a screen, so that light intensity drops
2-2:3	As the screen controller, I want to record the state of the screens, so that screen states are known when requested
2-3	When lighting needs to be adjusted, I want to adjust lamp brightness, so that lamps are on only when required.
2-3:1	As overhead light control, I want to record the intensity of each lamp, so that light states are known when requested
2-3:2	As overhead light control, I want to turn on a lamp, so that a group of plants gets the right amount of light
2-3:3	As overhead light control, i want to turn off a lamp, so that a group of plants gets the right amount of light
2-3:4	As overhead light control, I want to increase light when absorption drops below 0.8, so that plants get the right amount of light
3	**Help me control temperature**
3-1	When the climate computer gives a temperature instruction, I want to know how to process it based on current conditions, so that I formulate the correct temperature conditions
3-1:1	As a temperature manager, I want to calculate how hot the heating pipes have to be, so that heating control knows how much warm and cold water to spread to what zone
3-1:2	As a temperature manager, I want to open or close the overhead windows, so that hot air can exit the

	greenhouse
3-1:3	As a temperature manager, I want to open or close the overhead windows, so that I can abide a humidty manager request
3-1:4	As a temperature manager, I want to turn on window sprays when window state is insufficient, so that the greenhouse cools down
3-1:5	As a temperature manager, I want to use output from temperature sensors, so that I use real world situations and can monitor if my instructions have effect
3-1:6	As a temperature manager, I want to coordinate between my own goals and those of humidity, so that no conflicts emerge
3-2	When receiving a heating request, I want to change the current heating state, so that temperature can go up or stay constant
3-2:1	As a heating controller, I want to mix hot and cold water, so that I get pipes up to the right temperature
3-2:2	As a heating controller, I want to activate the right pumps, so that heated water spreads to the right pipes
3-3	When receiving a timer, I want to turn on the window sprays, so that the greenhouse cools down
3-3:1	As a window spray controller, I want to turn on sprays when I receive a timer, so that the air cools down
3-3:2	As a window spray controller, I want to turn off sprays when the timer reaches 0, so that I don't waste water
3-4	When receiving a state request, I want to open windows to a certain degree, so that the appropriate state is achieved
3-4:1	As window control, I want determine what windows to open, so that I don't accidentally open the wrong zone
3-4:2	As window control, I want to notify temperature management of a succesful state change, so that I can account for opening or closing windows and prevent conflicts
3-4:3	As window control, I want to open a window based on the percentage I receive, so that the correct window state is achieved
4	**Help me control humidity**
4-1	When receiving a humidity instruction, I want to determine a course of action, so that I can control humidity systems
4-1:1	As a humidity controller, I want to coordinate between my own goals and those of temperature and co2, so that no conflicts emerge
4-1:2	As a humidity controller, I want to determine if and how ventilation should be activated, so that the right airflow is maintained
4-1:3	As a humidity controller, I want to turn on the moisture pumps, so that I can increase humidity
4-1:4	As a humidity controller, I want to determine a course of action based on external climate conditions, so that I dont get unexpected results
4-1:5	As a humidity controller, I want to know how much extra moisture I can expect to enter the atmosphere based on sensor output, so that I can adjust my course of action

4-2	When receiving a moisture pump timer, I want to activate moisture pumps, so that humidity in the greenhouse increases
4-2:1	As a moisture pump controller, I want to switch on specific pumps, so that humidity rises in a certain zone
4-2:2	As a moisture pump controller, I want to have access to water supplies, so that I can vent moisture
4-2:3	As a moisture pump controller, I want to switch off pumps when the timer reaches 0, so that I don't increase humidity too much
4-3	When receiving a ventilation timer, I want to activate fans, so that the desired airflow is achieved
4-3:1	As a ventilation controller, I want to activate fans in a certain zone, pattern or height, so that the desired airflow is reached
4-3:2	As a ventilation controller, I want to deactivate fans when the timer reaches 0, so that I dont disrupt airflow
5	**Help me control CO2**
5-1	When receiving a CO2 instruction, I want to determine how to achieve the desired values, so that I can activate specific systems
5-1:1	As a CO2 action manager, I want to simulate ventilation, so that I know how CO2 will spread through the greenhouse
5-1:2	As a CO2 action manager, I want to calculate a pump timer in a certain zone, so that CO2 gets added to the atmosphere
5-1:3	As a CO2 action manager, I want to activate ventilation, so that CO2 spreads to the right zone
5-2	When I have determined to pump CO2, I want to activate a CO2 pump, so that CO2 is added to the atmosphere
5-2:1	As a CO2 pump controller, I want to activate a pump when I receive the instruction to do so, so that I add the right quantity of CO2
5-2:2	As a CO2 pump controller, I want to deactivate a pump when the timer is finished, so that I don't add too much CO2
6	**Help me use and improve growth models**
6-1	When a plant is selected to be grown in a greenhouse, I want to determine plant requirements and environment, so that the initial growing conditions can be determined
6-1:1	As an initializor, I want to retreive a list of known plant requirements, so that I can make known the initial growing conditions
6-1:2	As an initializor, I want to retreive an initial growth phase schedule, so that this information is known for the basis of a growth model
6-1:3	As an initializor, I want to know what type of greenhouse has been set up, so that this is known for the growth model
6-1:4	As an initializor, I want to have access to the let's grow framework, so that I can integrate known growing methods and data.

6-2	When new requirements are known, I want to combine all available information, so that a growth model can be constructed
6-2:1	As a growth model instructor, I want to use growing conditions and templates from my database, so that I can assemble the ideal growth model
6-2:2	As a growth model instructor, I want to use continuously refined models, so that I can improve my recommendations
6-2:3	As a growth model instructor, I want to use received plant phase information, so that I can send the appropriate part of the growth model
6-2:4	As a growth model instructor, I want to use templates, so that I never miss any information in my instructions
6-2:5	As a growth model instructor, I want to account for legal restrictions, so that I don't violate the law
6-3	When an improved dataset is available, I want to compare this to actual harvest yield data, so that I can suggest changes to the growth model
6-3:1	As a growth model updater, I want to receive harvest yield data from an independent source, so that I can run a comparative analysis with data from the control computer
6-3:2	As a growth model updater, I want to receive improved method suggestions, so that I can compare them and discover new patterns
6-3:3	As a growth model updater, I want to offer refined models, so that the instructor can improve its work
6-3:4	As a growth model updater, I want to store all improvements in a cloud side database, so that future software will have more data to analyse for improvements
7	**Help me monitor weather**
7-1	When wind direction changes, I want to send out a notification, so that the data is available for the rest of the system
7-1:1	As a wind monitor, I want to continuously send out information on wind direction, so that any change is tracked
7-1:2	As a wind monitor, I want to access a wide variety of sensors over a large region, so that even distant changes can be tracked
7-2	When air pressure changes, I want to send out a notification, so that the data is available for the rest of the system
7-2:1	As a pressure monitor, I want to continuously send out information on air pressure, so that any change is tracked
7-2:2	As a pressure monitor, I want to access regional air pressure sensors, so that I can accurately track changes
7-3	When monitoring sunshine, I want to send out a notification when conditions change, so that the data is available for the rest of the system
7-3:1	As a radiance monitor, I want to continuously track external radiation, so that any change is tracked

7-3:2	As a radiance monitor, I want to calculate external radiation sum, so that total sum radiation is known
7-4	When outside temperature changes, I want to send out a notification, so that the data is available for the rest of the system
7-4:1	As a temperature monitor, I want to track outside temperature, so that any change is tracked
7-4:2	As a temperature monitor, I want to use the data of sensors across the region, so that changes caused by weather can be tracked
7-5	When cloud cover changes, I want to send out a notification, so that the data is available for the rest of the system
7-5:1	As a cloud monitor, I want to track overhead cloud cover, so that any change can be tracked
7-5:2	As a cloud monitor, I want access to radar images, so that I can accurately track cloud cover
8	**Help me monitor internal sensors**
8-1	When evaporation changes, I want to send out a notification, so that the data is available for the rest of the system
8-1:1	As an evaporation monitor, I want to track moisture coming from leafs, so that I can track changes
8-1:2	As an evaporation monitor, I want to use infrared cameras, so that I can currately track evaporation changes
8-2	When soil moisture changes, I want to send out a notification, so that the data is available for the rest of the system
8-2:1	As a soil moisture monitor, I want to use sensors attached to soil, so that I can accurately track changes
8-2:2	As a soil moisture monitor, I want to track the water content of soil, so that plants can be given water accordingly
8-3	When greenhouse humidity changes, I want to send out a notification, so that the data is available for the rest of the system
8-3:1	As a humidity monitor, I want to measure water content in the atmosphere, so that any changes can be tracked
8-3:2	As a humidity monitor, I want to track humidity in multiple locations, so that an accurate image can be obtained of greenhouse conditions
8-4	When CO2 changes, I want to send out a notification, so that the data is available for the rest of the system
8-4:1	As a CO2 monitor, I want to measure CO2 changes, so that any changes can be tracked
8-4:2	As a CO2 monitor, I want to measure CO2 in multiple locations, so that an accurate image can be obtained of greenhouse conditions
8-5	When temperature changes, I want to send out a notification, so that the data is available for the rest of

	the system
8-5:1	As a temperature monitor, I want to measure temperature, so that any changes can be tracked
8-5:2	As a temperature monitor, I want to measure temperature in multiple locations, so that an accurate image can be obtained of greenhouse conditions
8-5:3	As a temperature monitor, I want to track temperature on multiple heights, so that an accurate image can be obtained of greenhouse conditions
8-6	When light changes, I want to send out a notification, so that the data is available to the rest of the system
8-6:1	As a light monitor, I want to measure light in lumen, so that any changes can be tracked
8-6:2	As a light monitor, I want to track lighting per zone, so that every zone can get the correct lighting
8-7	When light absorption changes, I want to send out a notification, so that the data is available to the rest of the system
8-7:1	As a light absorption monitor, I want to use a sensor attached to a leaf, so that I can track conditions of the plant
8-7:2	As a light absorption monitor, I want to use infrared cameras to measure light absorption, so that changes can be tracked
8-7:3	As a light absorption monitor, I want to measure conditions from various plants, so that a representative set is tracked