UNIVERSITY OF VAASA

THE SCHOOL OF TECHNOLOGY AND INNOVATIONS

AUTOMATION AND INFORMATION TECHNOLOGY

Alexander Tupeli

A REQUIREMENTS SPECIFICATION TEMPLATE FOR TEMPERATURE CALIBRATION SYSTEM

Master's thesis for the degree of Master of Science in Technology submitted for assessment, Vaasa, 13th March 2020.

Supervisor Taija Maunumäki

Instructor Prof. Jouni Lampinen

FOREWORDS

First, I would like to give a thanks to the company where this project took place for allowing me to make my master's thesis as a part of this project and helping me with learning a lot of skills. Additionally, I want to send a special thanks to Taija Maunumäki, who has taught me a lot of documenting, reporting and management skills. I would also like to thank the University of Vaasa and all the professors with whom I have collaborated with during these years of studies. Last but not least I would like to thank my lovely wife and daughters that have been supported throughout this whole stage.

I have learned a lot during the project that this thesis part of. This is the first time in my carrier that I've had the opportunity to work on a major project and additionally to get to manage it. I have noted more than once that I am pretty much fresh out of school and do not have much experience in project management, but these types of skills are learned through experience and time will teach. But I am very thankful for my supporting team that is ready to back me up anytime, thank you.

TABLE OF CONTENT

ABBREVIATIONS

BA	Business Analyst
SI	Système International d'unités
FDS	Functional Design Specification
SRS	Software Requirements Specification
URS	User Requirements Specification
R&D	Research and Development

UNIVERSITY OF VAASA
The School of Technology and Innovations
Author: Alexander Tupeli
Topic of the Thesis: **A Requirements Specification Template for Temperature Calibration Software**
Supervisor: Taija Maunumäki
Instructor: Professor Jouni Lampinen
Degree: Master of Science in Technology
Major of Subject: Automation and Information Technology
Year of Entering the University: 2013
Year of Completing the Thesis: 2020 **Pages:** 68

ABSTRACT

Calibration laboratories follow a set of standards and these standards change all the time. The ISO17025 standard concerns calibration laboratories and have recently got a new version. This new version mainly adds the requirement for traceability in the calibration process. The company in question is creating a new temperature calibration software, due to the adaptation of a new version of the ISO17025. This thesis concentrates on the creation of the software requirements specification for this new software system. Calibration theory and software requirements theory are topics that are researched in this thesis. The goal of this thesis is to create a new template for internal software products software requirements specification and utilize it in the project.

In the first part of the thesis, we will learn about the different key elements in this project, information that the reader must know to understand the subject. We will learn about calibration theory and temperature calibration in particular. Then we will jump straight into the theory of software requirements, a part of good practices in software requirements specification that can be helpful in the different stages. The development process plays also a part in the creation and management of the requirements, so this aspect is also looked at. Then we will utilize the gathered information in the creation of a new internal software requirements specification and functional design specification.

The new software requirements specification template for internal software systems is more suitable for this type of software system than the already existing template. Additionally, the functional design specification is included in the same document, which results in a smaller number of project-related documents. This results in a reduction of the amount of work required by the process. By reducing the number of documents into one larger document, makes requirement engineering tasks easier and allows for a deeper integration between requirements and implementation.

KEYWORDS: Software Requirement Specification, Temperature Calibration, Software Development, SCRUM

1 INTRODUCTION

When thinking about buying something, people always have a set of requirements in mind. These requirements are a description of what the new product must and must not fulfil. This can be as simple as what brand of phone you want to buy. The same principle applies when a requirements specification is created for a software product. With the exception of some cases where an external customer wants their product developed and it is upon their requirements that the software product will be developed. This thesis will try to determine some of the problems that usually occur during this stage, but also provide a new software requirements specification template for internal software products, or systems. At the end of the thesis, the new template will be utilized in a real-world case. After this, the result will be compared to the existing template.

At this company, where this research is taking place, there is an accredited temperature laboratory. This laboratory uses a temperature calibration software that helps the calibration technicians with the gathering of the temperature calibration measurement data and the analysing of the calibration results. The company is adopting the new ISO17025:2017 standard and the old temperature calibration software is not, in its current state, fulfilling this new standard, on all points. It has been decided it is more beneficial to create a new temperature calibration software than fixing the old one.

This thesis will be restrained to these two topics; temperature calibration theory and the software requirements specification writing process. The main material for temperature calibration theory comes from the book Traceable Temperature by Nicholas and White (2001) and the main material form software requirements theory comes from the book Software Requirements by Wiegers, Karl and Beatty (2013). The questions that this thesis will try to answer is: What work goes into a requirement, where do they come from, how can dependencies between different requirement be handled, what is a well-suited software requirements specification document?

To fully understand the subject and the problems that this thesis will assess, the layout of this thesis is accordingly: First, we will get acquainted with the theory about

measurement, calibration and temperature measurement. Followed by the theory of software requirements and the different techniques and processes used in software requirements engineering activities. Finally, the theory will be put to use in the development of a new template for software requirements specification, for the development of in-house software systems. Then the new template will be utilized in the documentation of the requirements for the new temperature calibration system.

2 TEMPERATURE MEASUREMENT AND CALIBRATION THEORY

Let's start with a simple everyday example. One day during the summer vacation, you decide to build a sandbox for your kids, and you call the local timber merchant, to buy the timber required for the project. The local timber merchant asks how much timber you need, and you answer that you need about 16 meters of planks. The local timber merchant then asks: "What's a meter?". You and the local timber merchant does not use the same measure of length and this makes the task of buying the planks for the sandbox a quite difficult task.

This example shows the importance of standardizing units of measurement. Over the past millennia, humans have tended to use different body parts to communicate measures of length; feet, inch, palms, hands, etc. As we all know, our body proportions are not exactly like another. This raises the need for a single standard for each measurement. Here comes traceability in the picture. Traceability is the task of producing the same measurement result at a different time, but with the same settings. Traceability is a key factor in calibration. (Nicholas & White 2001.)

2.1 Temperature Measurement

In this chapter, we will go through the basics of temperature measurement. Because most of the company's internal software products are calibration automation systems, and this new one, in particular, will calibrate temperature, temperature measurement is an important part of the software requirements specification document.

What is measurement?

Measurement is one of the most fundamental human tasks. Our curiosity, intelligence, and self-awareness fuel the urge for exploration and understanding of our surroundings. All knowledge in the world is gained through the human senses, and the decisions we make are based upon these measurements. The knowledge we have gained prepares us

for unexpected situations and handling these situations. The sensors and measurement instruments that humans have created are an extension of the limitations of the human senses. Temperature can be for example felt, but different people feel temperature different and it is not standardized with just the use of feel. (Nicholas & White 2001.)

The official definition of a measurement reads: "The set of operation having the object of determining a value of quantity." and the official definition of the result of a measurement reads: "The value attributed to a measurand obtained by measurement." (Nicholas & White 2001:2). While these definitions are technically correct, they do not provide a very good explanation of the difference between a measurement and a meaningless assignment of numbers. The book Traceable Temperatures by Nicholas and White (2001:3) gives an alternative definition, their definition of a measurement reads: "The symbolic representation of a state, event or attribute to aid in the making of a decision.". This alternative definition highlights some aspects that were not apparent in the official definition. (Nicholas & White 2001.)

- The result of measurement does not need to be numeric: a colour, a name and or mood can be adequate results of a measurement, in the right context.

- Every measurement serves a purpose. This is the difference between a measurement and a meaningless set of numbers.

- Decisions have both risks and rewards. It is important to be knowledgeable about the risk within a measurement, i.e. the uncertainties that come with the measurement.

What is temperature measurement?

For most materials, the temperature can be seen as a measure of the density of heat in a body. Temperature measurement is the study of how the physiological properties of a medium are affected by temperature. All temperature measurement instruments are based on this phenomenon. Heat is transported by conduction, convection, and radiation. The

temperature measurement can happen either when the temperature is changing or in a stable phase, such as the triple-point of water. To make an accurate temperature measurement, the system needs to be in a stable phase, called thermal equilibrium. In thermal equilibrium, there is no net flow of heat between any of the components in the system. Different temperature measurement instrument gets different measurement results when there is a net flow of heat in the system. (Nicholas & White 2001.)

There are two main categories of temperature measurement instruments used in temperature measurements: liquid-in-glass and resistance thermometry. Liquid-in-glass thermometers use the expansion of a medium to make an indication of the temperature of the medium. This method of thermometry is the oldest, it has its origins in ancient Greek (ca. 150 AD.). The more modern method in thermometry is using the resistive properties of a material. The resistance of metals changes in proportion to its absolute temperature. (Nicholas & White 2001.)

Errors and uncertainties are always a part of the result when taking measurements. Nicholas and White (2001: 39) gives this general definition of the uncertainty of measurement: "The parameter associated with the result of a measurement that characterises the dispersion of the values that could reasonably be attributed to the measurement.". So, an error or uncertainty fulfils the criteria of being an error or uncertainty when it is so major that it could reasonably alter the result. The simplest way of eliminating the majority of errors and uncertainties is to perform multiple measurements and estimate the range of possible values. But this can prove to be a time-consuming procedure, depending on the nature of the measurement. The other way is to utilize physical theory, information from handbooks, or various types of experience of similar situations. The error in measurements can be divided into two different categories: Random errors and Systematic errors. Random errors cause unpredictable scattering of readings over a period of time. Systematic errors cause all of the readings to be biased away to one side, from the true readings. Systematic errors can be eliminated or corrected for in the result. (Nicholas & White 2001.)

2.1.1 ITS-90

ITS-90 is an abbreviation for the International Temperature Scale of 1990 and is a temperature scale with a wide temperature range, to make sure it covers most implications. ITS-90 approximates the thermodynamic scale using several defined temperatures, as fixed points and is highly reproducible in a simple laboratory environment. It approximates the thermodynamic scale with a number of fixed temperature points. Highly reproducible thermometers are then interpolated between the defined temperature points. The main goal for ITS-90 is for different calibration laboratories to be able to replicate the result from other laboratories, i.e. replication. ITS-90 uses the triple point of water and the ice point as calibration points. These calibration points are highly replicable and quite easy to accomplish. (Nicholas & White 2001.)

2.1.2 Callendar-van Dusen

All metals are good electrical conductors. This is because of the atomic structure of all metals. The electrons are not bound to atoms but can move freely throughout the metal. When a voltage is applied to a metal wire, the electrons rush to the positive terminal and the moving electrons constitute an electric current. This flow of electric current is being restricted by electrical resistance. This electrical resistance is due to two mechanisms; temperature and impurities and lattice defects. For metals, the electric current is proportional to the voltage and can be summarized in this equation, called Ohm's law. (Nicholas & White 2001.)

$$R = \frac{U}{I}, \tag{1}$$

where R = resistance, U = voltage, and I = current.

This equation depends on the resistance to be stable. But as the temperature of the metal lattice increases, the electrical resistance decreases, and the flow of electrons increase in the proportion to the absolute temperature, i.e. the electric current increases. This

resistance-temperature relationship can be displayed in this equation. (Nicholas & White 2001.)

$$R(t) = R(0°C)\ (1 + \alpha t), \quad (2)$$

where α is the given temperature coefficient of resistance, approximately equal to 1/273,15K and t = the temperature in ℃.

All metals behave accordingly, but some metals are more suited than others as resistance thermometers. Platinum has a linear relationship between resistance and temperature and is, therefore, a well-suited metal for this application. Callendar-van Dusen found a simple quadratic equation, with constants A and B, that worked well at describing the resistance of platinum at a given temperature. (Nicholas & White 2001.)

$$R(t) = R(0°C)\ (1 + At + Bt^2), \quad (3)$$

The Callendar-van Dusen equation was the basis for the temperature scales of 1927, 1948, and 1968, and as of today is used to define the resistance-temperature relationship for industrial resistance thermometers. (Nicholas & White 2001.)

2.2 Temperature Calibration

In this chapter, we will get acquainted with the calibration theory and the calibration process.

What is calibration?

"A calibration refers to the set of operations carried out by an instrument manufacturer in order to ensure that the equipment has a useful measurement scale." (Nicholas & White 2001: 161). Often when making multiple measurements of the same target you get different measurements and wonder which one of them is the closest to the truth. This experience is common because most of the measuring devices are not as accurate as the

manufacture’s specifications. Most calibration laboratories have experienced that one in five instruments are outside the manufacturer's specification. This error rate is independent of different manufactures and tends to decrease with higher cost instruments. You could say that calibration is where a measuring instrument is compared to a known reference, to find out how much wrong the measuring instrument shows. (Nicholas & White 2001.)

A calibration process can be divided into 8 steps. (Nicholas & White 2001: 185-186).

1. Start record keeping
2. General visual inspection
3. Conditioning and adjustment
4. General checks
5. Comparison
6. Analysis
7. Uncertainties
8. Complete records

2.2.1 Traceability

The ISO definition of traceability is: “The property of the result of a measurement or the value of a standard whereby it can be related to stated references, usually national or international standards, through an unbroken chain of comparisons all having stated uncertainties.” (Nicholas & White 2001: 21.)

This definition can be interpreted with many points of view. The use of the word traceability in temperature measurement and calibration is referred to as that different users can meaningfully compare measurement results with each other, non-depending on the geographical location. If a laboratory is involved in the calibration of instruments and final measurements, it is required to have all produced calibration certificates stored and in addition, it is required that an independent and expert audit of the entire measurement process. This way it is not possible to have corrupt measurement results assumed that the accrediting body is legit. To achieve traceability, it is required to have a source of primary physical standards (the SI-scale is often used in thermometry), a source of documentary standards and a source of independent third-party assessors. (Nicholas & White 2001.)

2.2.2 ISO/IEC 17025

ISO/IEC 17025 General requirements for the competence of testing and calibration laboratories, specifies the general requirements on the quality system of testing and calibration laboratories. ISO/IEC 17025 is the main ISO standard that emphasizes on the use of an independent accreditation body. The aim is to improve the ability to consistently produce valid results. A valid result is a result can be trusted. The version ISO/IEC 17025:2017 was published in 2017 and the new automation calibration system will fulfil this standard. (ISO/IEC 2017.)

ISO/IEC 17025 has a structure. The structure is Scope, Normative References, Terms and Definitions, General Requirements, Structural Requirements, Recourse Requirements, Process Requirements, and Management System Requirements. General Requirements and Structural Requirements are related to the organization of the laboratory itself. Structure Requirements cite those issues related to the people, plant and other organizations used by the laboratory. Process Requirements are the heart of this version of the standard in describing the activities to ensure that results are based on accepted science and aimed at technical validity. Management System Requirements are those steps taken by the organization to give itself tools quality management system in supporting the work of its people in the production of technically valid results. (ISO/IEC 2017.)

Data integrity is a new requirement in the use of automation systems at the company in question and therefore it sets a whole new aspect in the software development for automation systems. Data integrity requires that the initial calibration data must remain untainted. This means that the calibration result can be traced back to the time of the calibration run. If some case requires that changes are needed to the calibration data, it must leave a trace of the person and comment on the decision. One additional requirement is the regulation of personnel permitted to create, edit or delete calibration data. This means that a person without the right authority cannot perform a calibration or alter the calibration data. (ISO/IEC 2017.)

3 SOFTWARE REQUIREMENT THEORY

In this chapter, we will learn more about what a software requirement is and what work goes into the Software Requirements Specification document.

What is a requirement?

Requirements for a software system can be viewed as a list of properties that the final software product will and should not have, so it as precisely as possible meets the expectations that the creator had envisioned. In many product development teams, there is one or a few personnel responsible for the visioning of the product and multiple developers that implement the requirements that the personnel had envisioned. (Wiegers & Beatty 2013.)

3.1 Requirements Specification

Technology and development methods have evolved a lot during the last decade. Major requirements trends during the past decade include:

- The recognition of business analysis as a professional discipline and the rise of professional certifications and organizations, such as the International Institute of Business Analysis and the International Requirements Engineering Board.

- The maturing of tools both for managing requirements in a database and for assisting with requirements development activities such as prototyping, modelling, and simulation.

- The increased use of agile development methods in R&D and the evolution of techniques for handling requirements on agile projects.

- The increased use of visual models to represent requirements knowledge.

Requirement elicitation in software development is, in many cases, a human interaction challenge, instead of a technically and process difficult task. Even though many various tools have been made to visualize the task. (Wiegers & Beatty 2013.)

“Software development involves at least as much communication as it does implementation, yet both education curricula and project activities often emphasize the computing over the communication aspect.” (Wiegers & Beatty 2013: xxvi)

Software requirements are important. Errors introduced during requirements activities account for 40 to 50 percent of all defects found in software products (Davis 2005: 4). Inadequate user input and shortcomings in specifying and managing customer requirements are major contributors to unsuccessful projects. Because requirements are the foundation for both the software development and the project management activities, all stakeholders should commit to applying requirements practice. Stakeholders include customers, users, business analysts, developers, and many others. (Wiegers & Beatty 2013.)

3.1.1 Requirements Engineering

“Requirements are a specification of what should be implemented. They are descriptions of how the system should behave, or of a system property or attribute. They may be a constraint on the development process of the system.” (Wiegers & Beatty 2013: 6)

Requirement Engineering Activities

Software requirements can be divided into three different levels (see Figure 1). The layers are business requirements, user requirements and functional requirements. Every system also includes a set of non-functional requirements. Functional requirements describe what functions the system must fulfil. On the other hand, the non-functional requirements do not specify what the system must do, but rather how well the system performs certain tasks. (Wiegers & Beatty 2013.)

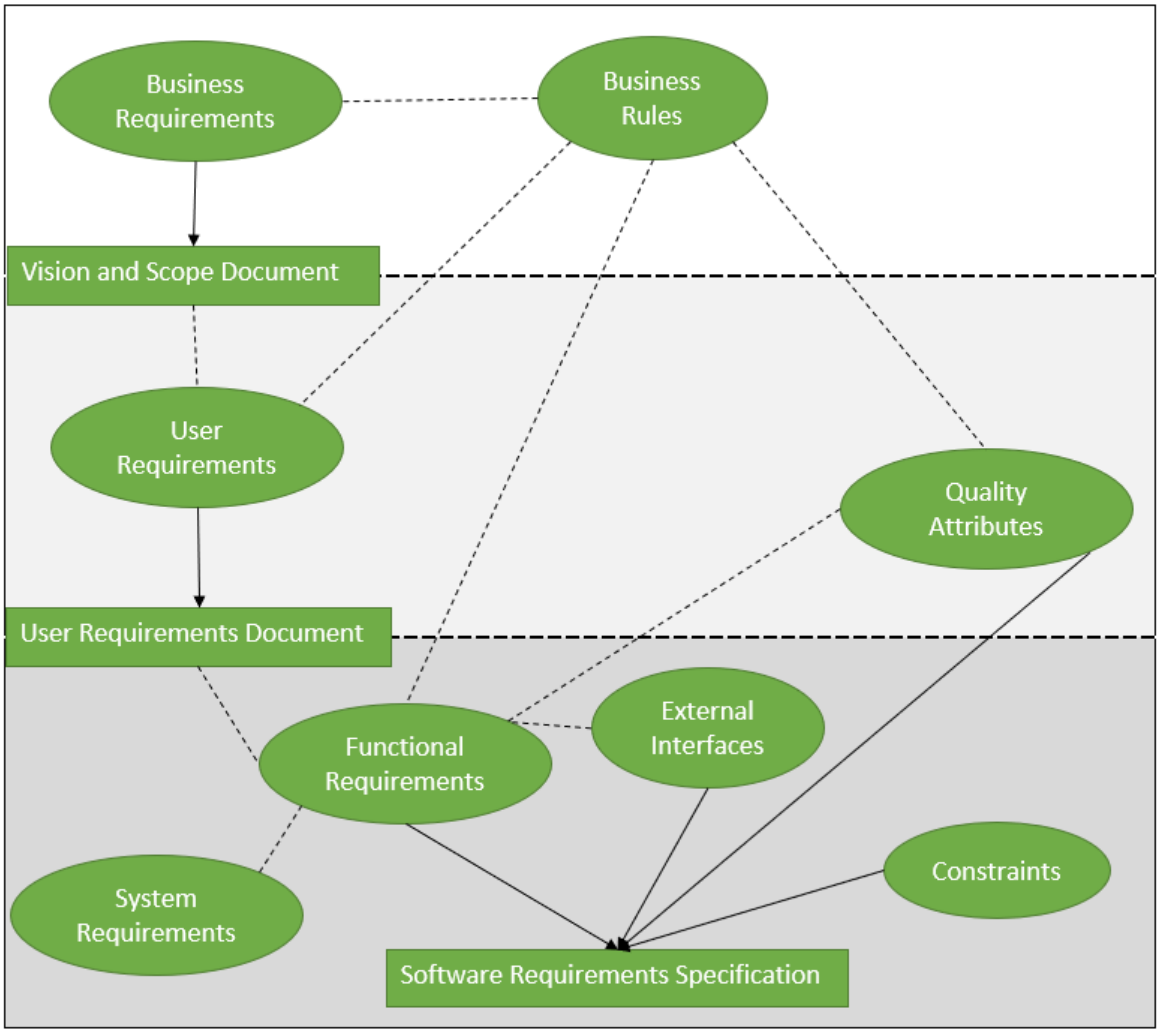

Figure 1. Relationship between several types of requirements information. Solid arrows mean "are stored in" and dotted lines mean "are affected by" (Recreated from Wiegers & Beatty 2013:8).

Business requirements specify why the system will be implemented and the business benefit the system will hopefully bring. Business rules include corporate policies, government regulations, standards, and computational algorithms. The focus of business requirements is how the system will support both the business objective of the customer and the company selling the system. The Business requirements are usually included in the vision and scope document. (Wiegers & Beatty 2013.)

Business rules include corporate policies, government regulations, industry standards, and computation algorithms. Business rules are not by themselves viewed as software

requirements. This is due to the existence beyond the boundaries of any single software application, that they pose. But they often impose some functionality that the system must fulfil to comply with the pertinent rules. Business rules are sometimes the origin of specific quality attributes. Therefore, some functional requirements can be traced back to a particular business rule. (Wiegers & Beatty 2013.)

User requirements describe what tasks the users must be able to perform with the product so that the product will add some value. Most user requirements consist of descriptions and characteristics of product attributes that are important to user satisfaction. There are different ways to document user requirements. These include use cases (Kulak & Guiney 2003), user stories (Cohn 2004) or event-response tables. User representatives are ideally used, to provide this information. In short, user requirements describe what actions the user will be able to perform with the finished product. (Wiegers & Beatty 2013.)

Functional requirements specify the behaviour of the system so that the end-users are able to accomplish their objectives and also fulfil the business requirement. The functional requirements are documented in the user requirements document or URS. Their alignment among the three levels of requirements is essential for project success. Functional requirements often are written in the form of the traditional “shall” statement. (Wiegers & Beatty 2013.)

System requirements describe the requirements for the collaboration and interface between different components or subsystems in a product. A “system” is not in all cases just an information system. A system can consist entirely of a software product, or it can include both software and hardware subsystems. People and processes are also often part of a system. Some system functions might be just allocated to human beings. Often the term “system requirements” are used to describe the requirements of a software system in more detail, but that is not adequate in all cases. (Wiegers & Beatty 2013.)

The task of the business analyst is to document functional requirements in a software requirements specification, or SRS. The SRS describes the expected behaviour of the software system in such a detail that the developer can implement the behaviour. The SRS

document is used throughout the development, testing, quality assurance, project management, and related project functions. The name of the document can vary from company to company, for example, business requirements document, functional spec, requirements document, and others. There are requirement management tools that generate a report of the requirements based on the stored information, that can be used as the SRS document. (Wiegers & Beatty 2013.)

As previously mentioned, non-functional requirements or other-than-functional requirements are requirements that do not specify what the system must accomplish, but rather how well the system works. Quality attributes, also known as quality factors, quality of service requirements, constraints, and the "-ilities", are part of the non-functional requirements. Quality attributes describe the characteristics of a product in various dimensions. These characteristics are important either to users or to developers and maintainers, such as performance, safety, availability, and portability. There are other types of non-functional requirements describing the external interfaces between the system and the outside world. These interfaces can be, for instance, connections to other software systems, hardware components, and users, as well as communication interfaces. Design and implementation constraints impose restrictions on the developer during the construction of the product. These restrictions affect the technical options the developers can utilize. (Wiegers & Beatty 2013.)

Software requirements have been classified as functional or non-functional requirements for many years. The functional requirements are self-explanatory; they all describe the expected behaviour of the system under various conditions. The term "non-functional requirements" is disliked among many people. Non-functional requirements say what the requirements are not, but not what they are, and this is quite confusing. The Software Requirements book by Karl Wiegers and Roy Beatty (2013) propose a different term that is more self-explanatory: "Other-than-functional requirements". This term is better at specifying, not what the system does, but rather how well it does those things. Other-than-functional requirements are describing important characteristics or properties of the system. Some examples of non-functional requirements are the system's availability, usability, security, and performance. Non-functional requirements can easily be thought of

being synonymous with quality attributes, but non-functional requirements cover much more than just quality attributes. For example, design and implementation constraints are also non-functional requirements, as are also external interface requirements. (Wiegers & Beatty 2013.)

Other non-functional requirements still address the environment the system works in, such as platform, portability, combinability, and limitations. Compliance, regulatory and certification requirements also affect many goods. Localization conditions may exist for products that must take into consideration cultures, languages, currencies, terminology, spelling and others. Although such specifications are defined in non-functional terms, the business analyst will typically obtain countless pieces of functionality to guarantee that all required behaviours and characteristics are present in the scheme. (Wiegers & Beatty 2013.)

A system feature is one or more logically related system capabilities that add value to an end-user and are described by a set of functional requirements. A client's rundown of wanted item highlights isn't proportional to a depiction of a user's undertaking related needs. Internet browser bookmarks, spelling checkers, the capacity to characterize a custom exercise program for a bit of activity hardware, and programmed infection signature refreshing in an enemy of malware items are instances of highlights. An element can encompass various client necessities, every one of which suggests that specific user requirements must be actualized to enable the client to play out the errand depicted by every client prerequisite. (Wiegers & Beatty 2013.)

3.1.2 Requirements Development

Requirements engineering can be partitioned into Requirement Development and Requirement Management. Requirements Development can be separated into Elicitation, Analysis, Specification, and validation. These subdisciplines, Figure 2, envelop every one of the exercises required with investigating, documenting and confirming the requirements for a product. The following are the fundamental activities in each subdiscipline

described in much detail in the book Software Requirements by Karl Wiegers and Joy Beatty (2013).

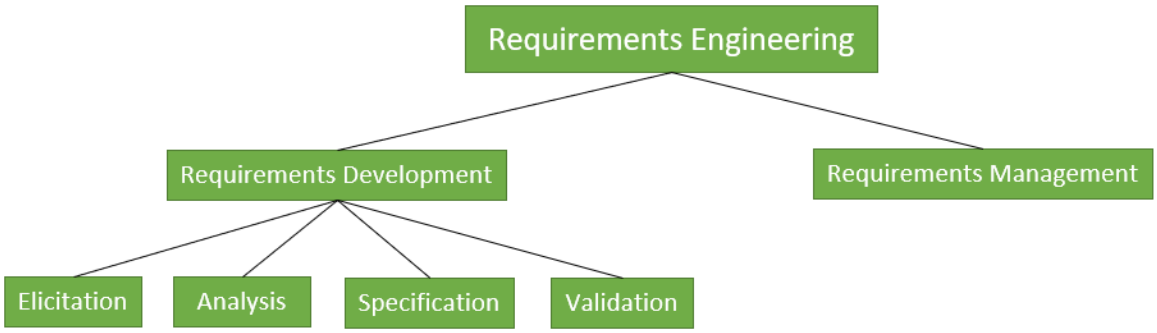

Figure 2. Subdisciplines of software requirements engineering (Recreated from Wiegers & Beatty 2013:15).

Elicitation includes all discovery operations in requirements engineering, such as interviews, workshops, document analysis, prototyping, and others.

- Identifying the anticipated customer classes of the product and other stakeholders.
- Understanding the user's tasks and objectives and the company objectives to which those tasks are aligned.
- Learning how the product will be used in the setting.
- Working with people representing each user class to know their requirements in terms of functionality and quality.

A stakeholder is a person, group or organization that is actively involved in a project, is affected by its process or outcome, or can influence its process or outcome. Stakeholders can be internal or external to the project team and the developing organization. Stakeholder analysis is an important part of the development of requirements (Smith 2000, Wiegers 2007, IIBA 2009). When searching for potential stakeholders for a particular project, it is good to cast a wide net to avoid overlooking some important community.

Then one single candidate's stakeholder list can be broken down to the core set whose input is needed, to make sure all of the project's requirements and constants are understood, so the development team can deliver the right solution. (Wiegers & Beatty 2013.)

Analysis, analysing requirements includes coming to a more extravagant and increasingly exact comprehension of every prerequisite and speaking to sets of prerequisites in various ways. Following are the foremost activities:

- Analysing the information received from users to recognize their task objectives from functional requirements, quality expectations, business rules, suggested solutions, and other information.
- Decomposing high-level requirements into a suitable degree of detail.
- Deriving functional requirements from other requirements information.
- Understanding the relative significance of quality attributes.
- Allocating requirements to software components defined in the system architecture.
- Negotiating implementation priorities.
- Identifying gaps in requirements or unnecessary requirements as they relate to the defined scope.

Specification, requirements specification involves representing and documenting the gathered requirements information in a tireless and efficient fashion. The principal activity is:

- Translating the collected user needs into written requirements and diagrams suitable for comprehension, review, and use by their intended audiences.

Validation, requirements validation affirms that the right arrangement of requirements information has been gathered, which will enable developers to build a solution that fulfils the business objective. The central activities are:

- Reviewing the documented requirements and correct any problems before the development group accepts them.

- Developing acceptance tests and criteria to confirm that a product based on the requirements would meet customer needs and achieve the business objectives.

Iteration is a key to success in requirements development. Plan for multiple cycles of exploring requirements, progressively refining high-level requirements into more precision and detail and confirming correctness with users. This takes time and it can be frustrating. Nonetheless, it's an intrinsic aspect of dealing with the fuzzy uncertainty of defining a new software system. (Wiegers & Beatty 2013.)

3.1.3 Requirements management

Requirements management activities include the following:

- Defining the requirements baseline, a snapshot in time that represents an agreed-upon, reviewed, and approved set of functional and non-functional requirements, often for a specific product release or development iteration.

- Evaluating the impact of proposed requirements changes and incorporating approved changes into the project in a controlled way.

- Keeping project plans current with the requirements as they evolve.

- Negotiating new commitments based on the estimated impact of requirements changes.

- Defining the relationships and dependencies that exist between requirements.

- Tracing individual requirements to their corresponding designs, source code, and tests.

- Tracking requirements status and change activity throughout the project.

The object of requirements management is not to stifle change or to make it troublesome. It is to foresee and accommodate the genuine changes that can always be expected to minimize their disruptive impact on the project. (Wiegers & Beatty 2013.)

Often, it is impossible or unnecessary to fully specify the functional requirements before commencing design and implementation. In those cases, an iterative or incremental approach can be beneficial, implementing one portion of the requirements at a time and obtaining customer feedback before moving on to the next cycle. This is the essence of agile development, learning just enough about requirements to do thoughtful prioritization and release planning so the team can begin delivering valuable software as quickly as possible. This is not an excuse to write code before contemplating requirements for that next increment, though. Iterating on code is more expensive than iterating on concepts. (Wiegers & Beatty 2013.)

People sometimes balk at spending the time that it takes to write software requirements. But writing the requirements is not the hard part. The hard part is determining the requirements. Writing requirements is a matter of clarifying, elaborating and recording what has been learned. A solid understanding of a product's requirements ensures that the development team works on the right problem and devises the best solution to that problem. Without knowing the requirements, it is difficult to tell when the project is done, determine whether it has met its goals, or make trade-off decisions when scope adjustments are necessary. Instead of balking at spending time on requirements, people should instead balk at the money wasted when the project does not pay enough attention to requirements. (Wiegers & Beatty 2013.)

3.2 Good Practices in Requirements Engineering

This is the general approach to the requirements elicitation process for a software product:

1. Identify the stakeholders
2. Perform Requirements Elicitation. Focus on quality attributes.
3. Requirements Analysis and Validation
    a. Reaching Agreement on Requirements
4. Requirements Documentation

Before the process can start, we will take part in some good practices in the process of requirements engineering, found in the Software Requirements book written by Karl Wiegers and Joy Beatty (2013).

3.2.1 Requirements elicitation

It is good to start the requirements elicitation process by creating user personas and product champions who will represent each class and document them. Followed by identifying the events and responses in the system that are being developed. A list is made including all the different external events that the system can experience and the expected response to each event is included. External events can be divided into three classes: signal events, control signals, and data received from external hardware devices. Temporal, or time-based, events trigger a response. In business applications, there are also business events, that triggers use cases. (Wiegers & Beatty 2013.)

Current systems should always be examined for requirement ideas, if such a system or similar is available. Try to reuse the existing requirement documentation as much as possible. It is unnecessary to reinvent the wheel. (Wiegers & Beatty 2013.)

3.2.2 Requirements analysis

The requirement analysis procedure involves processing and analysing the requirements to ensure that all stakeholders understand them and check them for errors, omissions and other deficiencies. Requirements analysis includes braking down high-level requirements into a more appropriate format, building prototypes, evaluation feasibility, and negotiating priorities. The goal is to develop requirements of sufficient quality and precision that managers can construct realistic project estimates and technical staff can proceed with design, construction, and testing. (Wiegers & Beatty 2013.)

It is truly valuable to represent some of the requirements in various ways, for instance in both textual and visual forms, or in the form of both requirements and tests (Wiegers 2005). Display the requirements in various diagrams, for easier understanding. Diagrams such as data flow diagrams, entity-relationship diagrams, state-transition diagrams, state tables, dialog maps, decision trees, and others. Context diagrams are a simple analytical model for displaying how the new system fits into its environment. It defines the interfaces between the new system and external entities such as users, hardware devices and other systems. An ecosystem map shows the various systems in the solution space that interact with each other and how they interact with each other. (Beatty & Chen 2012.)

Prototypes can be to great aid in the development stage. Having prototypes can help when developers or users are not entirely certain about the requirements. A prototype is a partial, possible or preliminary implementation, to make concepts and possibilities more tangible. It is, after all, a hands-on experience for all parties. Prototypes allows developers and users to achieve a mutual understanding of the problem being solved, as well as helping to validate requirements. (Wiegers & Beatty 2013.)

The Business Analyst, or simply short BA, should work with developers to evaluate the feasibility of implementing each requirement at an acceptable cost and performance in the intended operating environment. This allows stakeholders to get a better understanding of the risks associated with implementing each requirement, including conflicts and dependencies with other requirements, dependencies on external factors, and technical

obstacles. Requirements that are technically infeasible or overly expensive to implement can perhaps be simplified and still contribute to achieving the project's business objective. (Wiegers & Beatty 2013.)

It's important to prioritize the requirements to make sure that the developers implement the functionality with the highest value or most urgent, first. It is good to apply an analytical approach to determine the relative implementation priority of product features, use cases, user stories or functional requirements. Determine which release or increment will contain each feature or set of requirements, based on priority. The priorities are adjusted throughout the project as new requirements are proposed and as customer needs, market conditions, and business goals change. (Wiegers & Beatty 2013.)

An analysis model can be in the form of a diagram that represents requirements visually. The other option, the more common one, is to use textual representation of a list of functional requirements. Models have the benefit that it can reveal incorrect, inconsistent, missing and superfluous requirements. Examples of analysis models include data flow diagrams, entity-relationship diagrams, state-transition diagrams, state tables, dialog maps, decision trees, and others. (Beatty and Chen 2012.)

Interfaces between the system and the outside world should be analysed thoroughly. All software systems have connections to the outside world through external interfaces. Information systems have user interfaces and often exchange data with other software systems. Embedded systems involve interconnections between software and hardware components. Network-connected applications use communication interfaces. Analysing these interfaces helps with making sure that the application will fit smoothly into its environment. (Wiegers & Beatty 2013.)

3.2.3 Requirements specification

The core of requirements specification is to document requirements of different types in a consistent, accessible and reviewable way that is readily understandable by the intended audiences. The business requirements can be documented in a vision and scope document.

User requirements typically are represented in the form of use cases or user stories. Detailed software functional and non-functional requirements are documented in a software requirement specification (SRS) or in an alternative repository, such as a requirements management tool. (Wiegers & Beatty 2013.)

It is recommended to adopt standard templates for the documentation of requirements. The templates have the benefit of providing a consistent structure for recording various groups of requirement-related information. Even if the requirements are not stored in traditional documented form, the template will be a reminder of the various kinds of requirements information to analyse and record. (Wiegers & Beatty 2013.)

To ensure that all stakeholders know why every requirement is needed, each one is traced back to its origin. This might be in the form of a use case or some other customer input, a high-level system requirement, or a business rule. By linking the stakeholders that are affected by each requirement tells whom to contact when a change is requested. (Wiegers & Beatty 2013.)

Try to document the business rules separately from a project's requirements, because they typically exist beyond the scope of a specific project. That is, treat business rules as an enterprise-level asset, not a project-level asset. Some business rules will end up in a functional requirement, so define traceability links between those requirements and the corresponding rules. (Wiegers & Beatty 2013.)

It is possible to implement a software solution that does exactly what it is supposed to do but does not satisfy the users' quality expectations. To avoid this problem, it is needed to go beyond the functionality discussion to understand the quality characteristics that are important to success. These characteristics include performance, reliability, usability, modifiability, and many others. Customer input on the relative importance of these quality attributes lets the developer make appropriate design decisions. Also, document external interface requirements, design and implementation constraints, internationalization issues, and other non-functional requirements. (Wiegers & Beatty 2013.)

3.2.4 Requirements Validation

The validation process ensures that the requirements are correct, to test the desired quality characteristics, and that customer needs will be satisfied. Requirements that seem fine at the first glance might turn out to have ambiguities and gaps when they are implemented. The requirements must be correct if the foundation for design and for final system testing and user acceptance testing are solely based on the requirements. (Wiegers & Beatty 2013.)

Peer review of requirements, particularly the type of rigorous review called inspection, is one of the highest-value software quality practices available (Wiegers 2001). A small team of reviewers is assembled, who each represent a different perspective (such as analyst, customer, developer, and tester), and the written requirements, analysis models, and related information are carefully examined for defects. Informal preliminary reviews during requirements development can also prove to be a valuable addition. It is important to train the team members on how to perform effective requirements reviews and to adopt a review process for the organization. (Wiegers & Beatty 2013.)

Testing the product can give an alternative view of the requirements. While composing the tests, it must be agreed beforehand on how to decide if the expected functionality was correctly implemented. Tests are created with the user requirements in mind, to document the expected behaviour of the product under specified conditions. The tests are examined together with customers to ensure that they reflect user expectations. The test should also be mapped to the functional requirements to ensure that no requirements have been overlooked and that all have corresponding tests. The tests can be used to verify the correctness of analysis models and prototypes. Agile projects often create acceptance tests, to see if the product is in line with the functional requirements. (Wiegers & Beatty 2013.)

It is important to have the acceptance criteria defined in an early stage of the development. The users are asked to describe how they will determine whether the solution meets their needs and is fit for use. Acceptance criteria include a combination of the software passing a defined set of acceptance tests based on user requirements, demonstrating satisfaction

of specific non-functional requirements, tracking open defects and issues, having infrastructure and hand-on training for a successful release, and more. (Wiegers & Beatty 2013.)

Today there are commercial tools that allow a project team to simulate a mock-up system either in place of or to augment written requirements specifications. Simulation is the next level of prototyping, by letting the Business Analyst to work with users to rapidly build executable mock-ups of a system. Users can interact with the simulated system to validate requirements and make design choices, to test the requirements before they are set in stone. Simulations does not eliminate a thoughtful requirements elicitation and analysis process, but it provides a powerful supplement. (Wiegers & Beatty 2013.)

3.2.5 Requirements management

After the initial list of requirements has been created, there is an unavoidable supply of changes that customers, managers, marketing, the development team, and others request during the development stage. To make change management effective, a process for the proposing of changes, evaluation of their potential, cost and impact on the project, can be implemented. In addition, to making sure that appropriate stakeholders make sensible business decisions on which proposed changes to incorporate. (Wiegers & Beatty 2013.)

Well-established configuration management practices are essential for effective requirements management. The version control tools that are used for the codebase, can also be used to manage the requirements documents. There are also specific requirements management tools available. The core purpose of requirements management tools is to perform requirements management activities and are therefore well-suited for this application. (Wiegers & Beatty 2013.)

Rather than stifling change or counting on that changes does not come, accept that changes will come along the way. Establish a good requirement change control process, that prevents unforeseen changes to cause chaos in the project. The change process should define how requirements changes are proposed, analysed and resolved. Every proposed

change is managed through this process. Defect-tracking tools can be at help in the change control process. A change control board (CCB) can be created, consisting of a small group of project stakeholders, to evaluate proposed requirements changes and to decide which ones to accept, and to set implementation priorities. (Wiegers & Beatty 2013.)

Perform impact analysis on requirements changes. Impact analysis is an important element of the change process that helps the CCB make informed business decisions. Each proposed requirement change is evaluated to assess the effect it will have on the project. The requirements traceability matrix can be used to identify the other requirements, design elements, source code, and tests the proposed changes beforehand. The tasks required to implement the change needs to be identified and the time effort needed to perform those tasks. (Wiegers & Beatty 2013.)

A baseline defines a set of agreed-upon requirements, typically for a specific release or iteration. When the requirements for a specific release or iteration has been agreed on, they are being baselined. Possible changes to these requirements are made only through the project's change control process. It is beneficial to have a unique identifier for each version of the requirements specification, to prevent confusion between different versions of the baseline. It is good to also retain also a history of the changes made to individual requirements. Sometimes requirements have to be reverted to an earlier version and in those cases, it is practical to know how the requirement came to be in its current form. The dates, who made each change, and why changes were made to the requirements are recorded. A version control tool or requirements managements management tool can help with these tasks. (Wiegers & Beatty 2013.)

When busy people are working on a complex project, it is easy to lose sight of the many issues that arise, including questions about requirements that need resolution, gaps to eradicate, and issues arising from requirements reviews. A tool for issue-tracking can keep these items from falling through the cracks. A responsible person can be assigned for each issue and the status of requirement issues is monitored to determine the overall state of the requirements. (Wiegers & Beatty 2013.)

It is favourable to use links that connect each functional requirement to the design and code elements that implement it and the tests to verify it. Such a requirements traceability matrix is helpful for confirming that all requirements are implemented and verified. It is also useful during maintenance when a requirement has to be modified. The requirements traceability matrix can also connect functional requirements to the higher-level requirements from which they were derived and other related requirements. This matrix should be populated rather during the development stage, not at the end. (Wiegers & Beatty 2013.)

3.2.6 Knowledge

Various team members might perform the role of a business analyst on a given project, but few software developers receive formal training in requirements engineering. Business analysis is a specialized and challenging role, with its own body of knowledge (IIBA 2009). But as with all technical disciplines, there is no substitute for experience. It is not reasonable to expect all people to be instinctively competent at the communication-intensive tasks of requirements engineering. Training can increase the proficiency and comfort level of those who serve as analysts, but it cannot compensate for absent interpersonal skills or lack of interest in the role. (Wiegers & Beatty 2013.)

The steps that the organization follows to elicit, analyse, specify, validate and manage requirements, should be documented. By providing a guide or guidance on how to perform the key steps, will help analysts do a consistently good job. It will also make it easier to plan each project's requirements development and management tasks, schedule and required resources. The project manager should intercorporate requirements activities as discrete tasks in the project plan. (Wiegers & Beatty 2013.)

A glossary could be created that defines specialized terms from the application domain will minimize misunderstandings. Synonyms, acronyms and abbreviations, terms that can have multiple meanings, and terms that have both domain-specific and everyday meanings is documented in this glossary. A glossary could be a reusable enterprise-level asset. Developing a glossary could be an activity for new team members, because they will most

likely be the ones most confused by the unfamiliar terminology, and they would learn them at the same time. (Wiegers & Beatty 2013.)

3.2.7 Project management

Software project management approaches are tightly coupled to a project's requirements processes. The project manager should base project schedules, resources, and commitments on the requirements that are to be implemented. An alternative strategy is to timebox development cycles, such that the team estimates the scope of the work they can fit into an iteration of fixed duration. This is the approach taken by agile development projects. Scope is regarded as negotiable within the schedule. This transforms scope creep into "scope choice"-the product owner can ask for anything and as much as he wants, but he must prioritize it, and the team quits developing when they run out of time. Then the team plans a subsequent release for the remaining requirements. (Wiegers & Beatty 2013.)

It is beneficial to have several development life cycles defined, that are appropriate for various types of projects and different degrees of requirements uncertainty (Boehm & Turner 2003). Each project manager should select and adapt the life cycle that best suits the project. Requirements activities should be included in the life cycle definitions. When possible, the sets of functionalities implemental are specified and implemented so that a useful software can be delivered to the customer as early as possible. (Larman 2003, Schwabler 2004, Leffingwell 2011.)

Each project team should plan how it will handle its requirements development and management activities. An elicitation plan helps to ensure that input is identified and obtained from appropriate stakeholders at the right stages of the project using the most appropriate techniques. The BA and project manager should work together to ensure that tasks and deliverables related to requirements engineering appear in the project management plan. (Wiegers & Beatty 2013.)

Stakeholders often want to know how long it is going to take to implement the requirements for a project and what percentages of their total effort should be devoted to requirements development and management. Naturally, this depends on many factors. The factors that would indicate that more or less time than average is spent to ensure the requirements lay a solid foundation for development, should be considered in an early stage. (Wiegers & Beatty 2013.)

Plans and schedules for the project is developed iteratively as the scope and detailed requirements become clear. It good to begin by estimating the effort needed to develop the user requirements from the initial product vision and project scope. Early cost and schedule estimates based on fuzzy requirements will be highly uncertain, but they can be improved as the understanding of the requirements improves. On agile projects, the timeboxed nature of iterations means that planning involves adjusting the scope to fit within the fixed schedule and resource constraints. (Wiegers & Beatty 2013.)

Software development involves making many decisions. Conflicting user inputs must be resolved, commercial package components must be selected, change requests must be evaluated, and on and on. Because so many decisions involve requirements issues, it is essential for the project team to identify its requirements decision-makers, preferably before they confront their significant decision. (Wiegers & Beatty 2013.)

Unanticipated events and conditions can wreak havoc on an unprepared project. It is beneficial to identify, and document risks related to requirements as part of the project's risk-management activities. There are multiple approaches that should be considered for migrating or preventing these risks, implementing the migration actions, and for tracking their progress and effectiveness. (Wiegers & Beatty 2013.)

One way to improve the ability to estimate the resources needed for requirements work on future projects, is to record the effort put in by the team on requirements development and management activities (Wiegers 2005). By monitoring the effect that the requirements activities have on the project can help judge the return on the investment in requirements engineering. (Wiegers & Beatty 2013.)

A learning organization conducts periodic retrospectives to collect lessons learned from completed projects or from earlier iterations of the current project (Kerth 2001, Derby & Larsen 2006, Wiegers 2007). Studying the lessons learned from previous requirements experiences can help project managers and business analysts steer a more confident course in the future. (Wiegers & Beatty 2013.)

3.2.8 The business analyst's tasks

The business analyst must first fully understand the business objectives for the project and then define user, functional, and quality requirements, which will allow the development team to estimate and plan the project, and to design, build, and verify the product. The BA is also a leader and a communicator, translating vague customer notions into clear specifications, that will guide the software team's work. This section describes some of the typical activities that a business analyst will perform, according to the book Software Requirements, written by Karl Wiegers and Joy Beatty (2013).

- **Define business requirements**. The work of a BA begins with helping the business or funding sponsor, product manager, or marketing manager to define the project's business requirements. The BA can have a saying in what template to use for a vision and scope document since it is an essential tool for the BA.

- **Plan the requirements approach**. The BA should develop plans to elicit, analyse, document, validate and manage requirements throughout the project. The BA should work closely with the project manager to ensure these plans align with the overall project plans and will help achieve the project goals.

- **Identify project stakeholders and user classes**. The BA should work with the business sponsors to select appropriate representatives for each user class, engage their participation, and negotiate their responsibilities. The BA explains what the expectations from the customer collaborators are and agrees on an appropriate level of engagement from each one.

- **Elicit requirements**. A proactive analyst helps users articulate the system capabilities they need to meet their business objectives by using a variety of information-gathering techniques.

- **Analyse requirements**. A good approach is to look for derived requirements that are a logical consequence of what the customers requested and for implicit requirements that the customers seem to expect without saying so. Requirements models can be used to recognize patterns, identify gaps in the requirements, reveal conflicting requirements, and confirm that all requirements specified are within the scope. Stakeholders and the BA should determine the necessary level of detail for specifying the user and functional requirements.

- **Document requirements**. The analyst is responsible for documenting requirements in a well-organized and well-written manner that clearly describes the solution that will address the customer's problem. By using standard templates, requirements development can be accelerated by reminding the BA of topics to discuss with the user representatives.

- **Communicate requirements**. The requirements must be communicated effectively and efficiently with all parties. The BA should determine when it is helpful to represent requirements by using models other than text, including various types of visual analysis models, tables, mathematical equations, and prototypes. Communication is not simply a matter of putting requirements on paper and tossing them over a wall. It involves ongoing collaboration with the team to ensure that they understand the information that is being discussed.

- **Lead requirements validation**. The BA must ensure that requirement statements possess the desired characteristics and that a solution based on the requirements will satisfy stakeholder needs. Business analysts are the central participants in reviews of requirements. They should also review designs and tests that were derived from the requirements to ensure that the requirements were interpreted correctly. If they are creating from the requirements to ensure that the requirements

were interpreted correctly. If they are creating acceptance tests in place of detailed requirements on an agile project, those should also be reviewed.

- **Facilitate requirements prioritization**. The analyst brokers collaboration and negotiation among the various stakeholders and the developers to ensure that they make sensible priority decisions. In alignment with achieving business objectives.

A business analyst is involved throughout the entire software development life cycle, so he or she should aid in creating, reviewing, and executing a requirements management plan for a project. After the establishment of a requirement baseline for a given product release or development iteration, the BA's focus shifts to tracking the status of those requirements, verifying their satisfaction in the product, and managing changes to the requirements baseline. With input from various colleagues, the BA collects traceability information that connects individual requirements to other system elements. (Wiegers & Beatty 2013.)

4 THE DEVELOPMENT METHOD

How will the development method, used during the development, play a role in the documentation of requirements? We will take a closer look at Waterfall versus Agile, and especially the SCRUM development methods and how the documentation of the requirements is created and maintained throughout the project.

4.1 Waterfall

The waterfall development method is the most commonly known. If someone wants to create something, this is probably the development method he or she will use. The name waterfall comes from the approach of development used in the method. The development process in waterfall flows generally only in one direction, downwards like a waterfall. You could state that the waterfall development method is closest to our humans' way of thinking. A project begins with some collection of requirements and these requirements are then implemented into the general design of the product. The implementation is then based on the design and in the end, is some sort of testing or verification that ensures the finished product meets the requirements. After the development of the product is completed, follows the maintenance. (Parody 2018.)

All steps are more or less based on the requirements, as we can see in Figure 3, specified at the beginning of the project. This means that it is vital for the outcome and success of the project that the requirements are created with the highest precision. If there are some misunderstanding or mistake included in the initial requirements, all the following steps are based on this faulty requirement and can compromise the outcome of the whole project. (Parody 2018.)

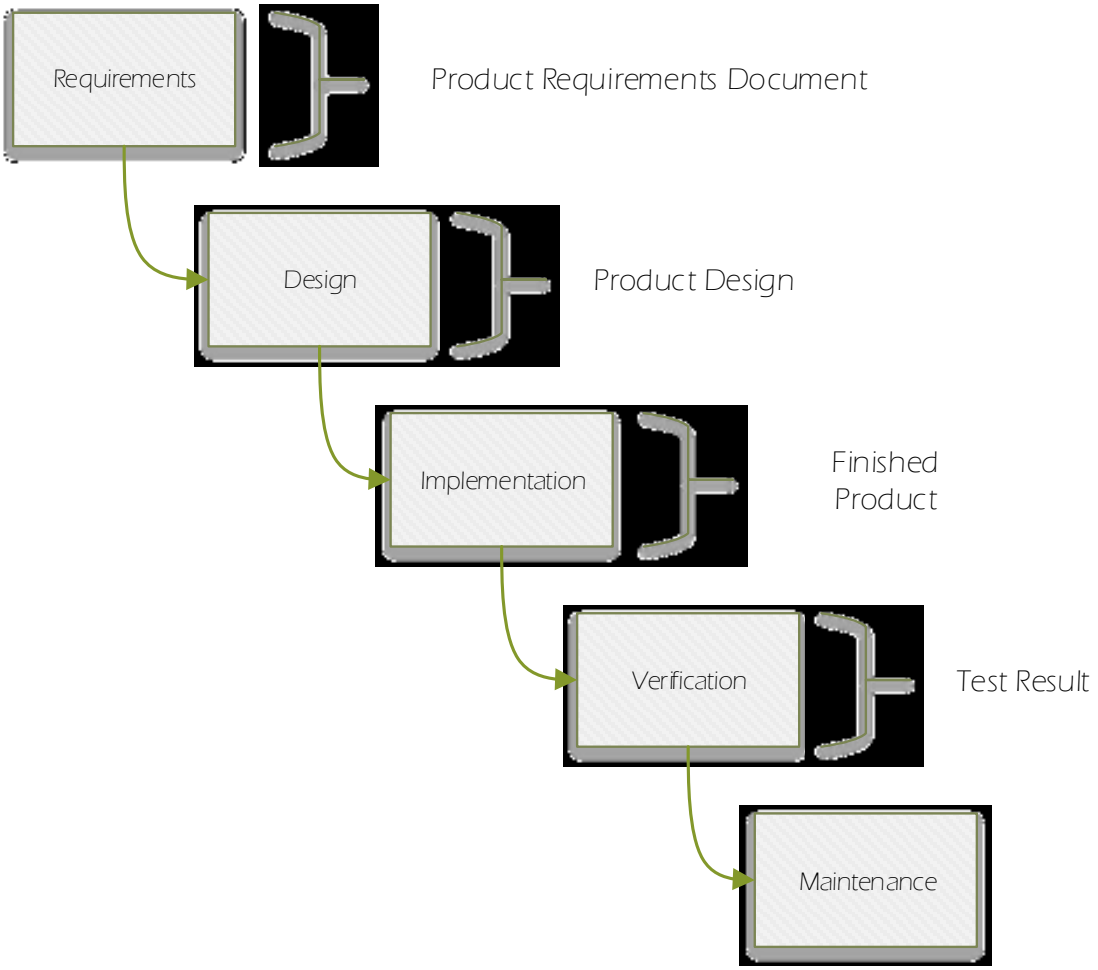

Figure 3. Waterfall Development Model.

4.2 Agile

The key aspect of the agile development method is to split the development process into smaller stages, called sprints, as seen in Figure 6. One stage, called sprint, consists of similar steps as in the waterfall but it is circular, as can be seen in Figure 4. One sprint usually consists of a planning, implementation, testing and review stage. In the planning stage is a suitable number of tasks chosen to be developed during the sprint. The tasks undergoing development are locked during this time. At the end of a sprint, there will be a sprint review, where the progress during the sprint is analysed and documented. If some problem has occurred during the sprint that results in the need for a reassessment, it will be taken into account in the next sprint planning meeting. This results in that the SRS and

FDS are not locked until the official release and can freely evolve during the project. The agile development method will better allow for sudden changes and reduce the overall risk of the project. (Rubin 2012.)

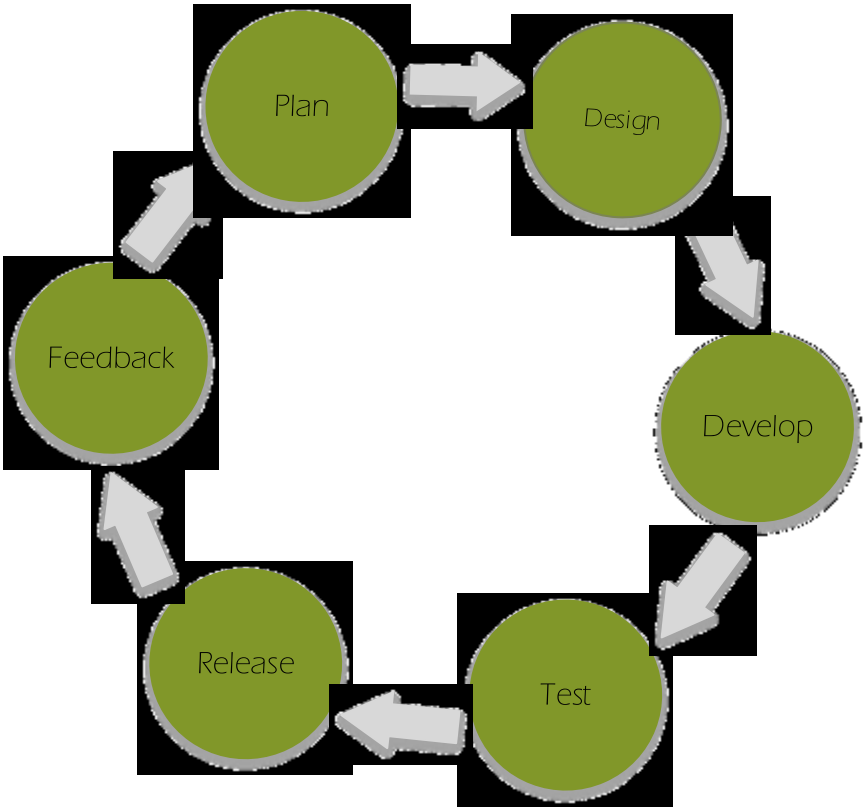

Figure 4. The Agile Development Model.

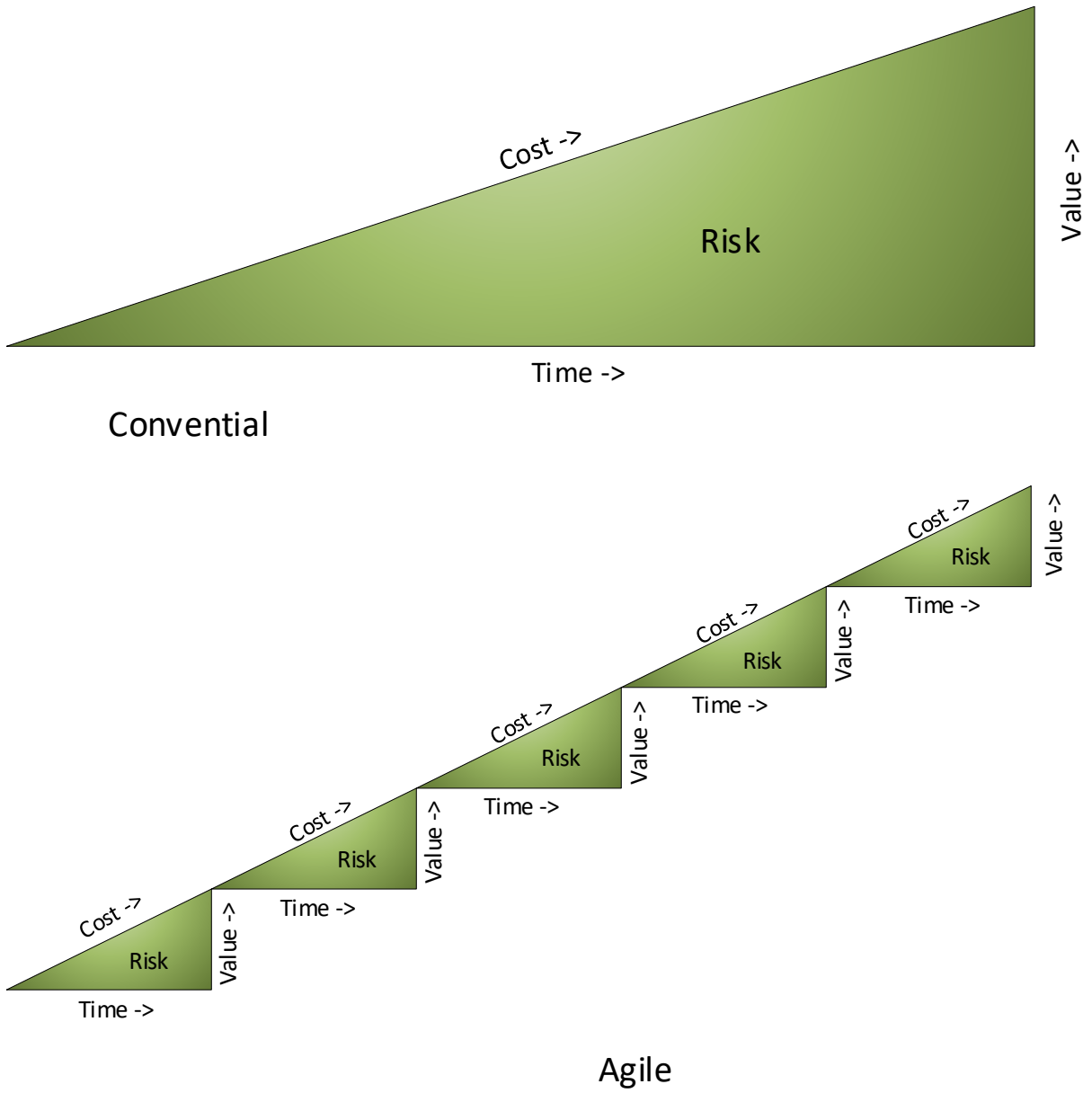

Figure 5. The risk ideology of the Agile Development Methods. (Rubin 2012).

One pyramid in Figure 5 means one release of a product. The Agile development method relies on multiple smaller releases, instead of one big release. This results in a lower risk in the project and the ability to better adapt to sudden changes during the development. Meanwhile, the overall cost of the project stays the same.

In an empirical study about the productivity trends in incremental and iterative software development projects (Tan, Li, Boehm, Yang, He & Moazeni 2009), it was seen that the

breaking down of the whole project into smaller pieces, allows for the engineers to gain experience and knowledge as the project progresses. This can reduce the effort in future increments while maintaining a stable staffing level. This learning advantage from build to build can help the engineers to learn more about the system requirements and architecture. But also, it was pointed out the lowering of the risks early on in the project, due to the use of incremental or iterative software development processes. There are also risks with an incremental or iterative software development process, that can lead to more needed effort: deficiency in design for integration can lead to additional effort needed for the modules to fit properly together, but also code breakage and the necessity of more testing and fixing in every level to ensure a good fit between the core and the new module.

4.2.1 SCRUM

The SCRUM development method is part of agile development methods. There are multiple roles in scrum for handling certain tasks, so the project can advance without minimal hiccups, as can be read in the book Essential Scrum by Kenneth Rubin (2012).

- Product Owner is responsible for the vision and business side of the project. The product owner collaborates with the team and other stakeholders to create and manage the product backlog.

- Scrum Master is responsible for organizing meetings, dealing with challenges and bottlenecks. The scrum master interacts with the Product Owner to make sure that the product backlog is ready for the next sprint.

- The scrum team is a dynamic team without the usual roles of developers, designers, and testers, rather everyone has a set of tasks that they complete together. The scrum team plans the amount of work they can complete in each iteration.

The scrum development method contains a set of steps, as can be read in the book Essential Scrum by Kenneth Rubin (2012).

- Product backlog consists of a list of all the features to be implemented in the product. Each feature can be split into user stories and the user stories can be split into tasks.
- Sprint planning
- Backlog management
- Daily scrum
- Sprint review meeting
- Sprint retrospective meeting

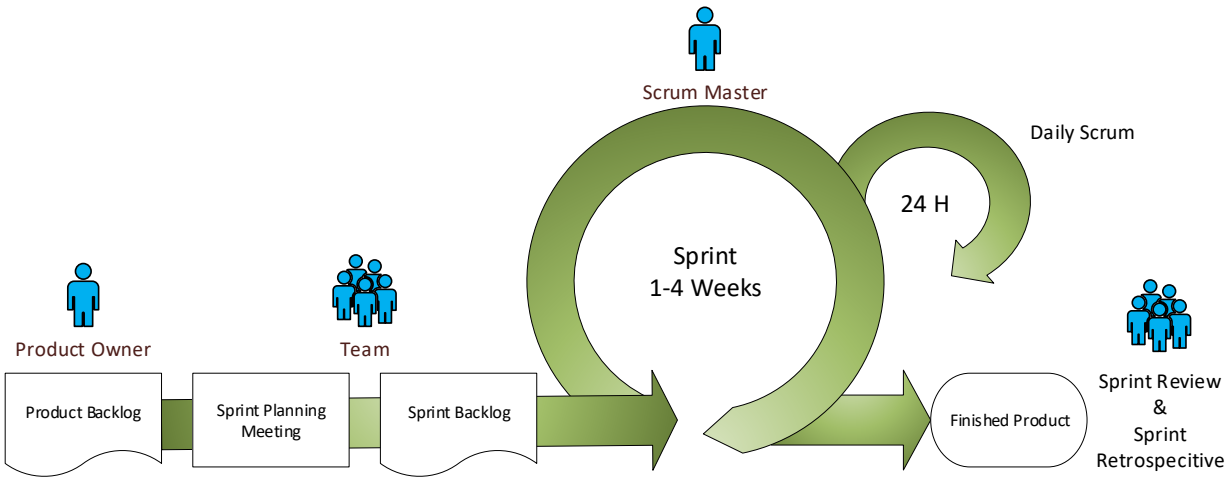

Figure 6. The SCRUM Development Model.

4.3 The development method's effect on the SRS

One thing to keep in mind in the creation of the SRS and FDS, while following any of the agile development methods, is that the document(s) will most likely change throughout the project. The SCRUM development method also utilizes more people reviewing other's work. This essentially lowers the number of errors and misunderstandings. By making it a group exercise, you essentially start the discussions about different topics and handles the problems at the right time. (Rubin 2012.)

When following the waterfall development method, the SRS and FDS are locked throughout the project and are not as adaptive to changes as in the agile development methods. This can lead to one mistake at the beginning of the development process follows and affects the whole project and in the worst case, it can have an effect on the outcome of the project. (Rubin 2012.)

The SCRUM development method solves this by locking only the features taken under development, in the sprint planning meeting. If some changes to the project happens under the sprint, the content of the sprint does not change. The sprint can be cancelled at worst, but often the sprint continues, and the changes effects the planning of the next sprint. (Rubin 2012.)

5 THE PLAN OF THE STUDY

In order to understand the point of the study, we need to first understand the main goal of the study. The main goal of the study is a new SRS template for internal calibration automation software systems and put the new template to use in the documentation of the requirements for the new temperature calibration software system.

5.1 Research method

The research method that is most suitable for this type of study is the waterfall model, where the activities in the project can be set up in linear sequential phases. The waterfall model is described in more detail in chapter 4.1. The different phases in the study will be described in a numbered list that follows the chronological order in the progression of the study. Before a software requirements specification is born, a new SRS document template will be developed and then put into use in this study.

1. Create a suitable template for the SRS document.

2. Possibly make some changes to the template that it will be better suited for the needs.

3. Identify the stakeholders.

4. Perform the requirements elicitation activities.

5. Requirements analysis and validation.

6. Requirements documentation.

7. Requirements split into features.

8. The final version of SRS and FDS accepted.

5.2 Research plan

The company where this project took place has already templates for the Requirements Specification and Functional Design Specification, but they are more suited for commercial products, not for a production system that will only be used internally. Therefore, I will create a new template that will be the foundation for the documentation of the requirements. This new template will be more suitable for internal software products and tools. The plan is to include FDS into the SRS document, to minimize the number of different documents that is part of the project's documentation. This will not only result in the number of documents is reduced, but also all the necessary information, for the developers, can be found in one document. The features in the FDS document provides a view of what functionalities the software solution must support that the software fulfils the requirements. Since the requirements and the features are so closely linked it is beneficial to have both in the same document.

After a new template for SRS and FDS has been created, it will be put into use in the documentation of the requirements of the new temperature calibration software system. This will most likely result in some ideas of what needs to be tweaked or changed in the template. A template for the SRS document will change some from project to project, but a good template is dynamic enough to suit most projects.

6 IMPLEMENTATION OF THE STUDY

The implementation of the study will follow the same layout as the plan of the study.

1. The internet is full of all kinds of templates for SRS documents. But I ended up looking just at a few SRS templates to see which one would best suit the need for a new SRS template for internal software products. The book Software Requirements by Wiegers and Beatty (2013) is one example of an SRS document, but I found this template to be much more complex and wanted to scale it down a bit. Some of the sections would be a nice addition to the existing template used at the company in question. Then I looked at the SRS template found in the IEEE's paper on Recommended Practice for Software Requirements Specifications (1998: 21-26), but this was also too extensive for this application. Lastly, I decided to use the already existing template, the template's table of content can be found in Appendix 1, as a foundation for the new SRS template and to make changes to it. The layout and general feel are familiar to the developers and it has some nice features that the others not necessarily have.

2. The changes made to the existing SRS template is mostly imported from the suggested SRS and FDS templates found in the Software Requirements book, written by Karl Wiegers and Joy Beatty (2013: 191). These changes include sections about Features, Data requirements and External interface requirements.

3. A meeting was held where all the possible persons of interest were gathered. The ones that said that they do not play a role in the project was free to leave. The ones that stayed were considered as stakeholders.

4. Before I got to perform any requirements elicitation activities, I read some theory on the subject of temperature calibration and looked at the old program in more detail, since the new program shares some of the basic functionality with the old one. After I got to know the theory, terminology and required functionality, we held an initial meeting with all the stakeholders. In the meeting, we discussed the

requirements and listed them down. We went also through the functionality of the old program to see what could be reused in the new program, to save some development time. Many more meetings and workshops were held throughout the development of the new software, where different views upon the requirements were discussed.

5. All the requirements were split into different categories: functional and non-functional requirements and quality attributes. The requirements were documented at this stage in the new template. The document was then sent to the stakeholders for review and validation.

6. The documentation of requirements was done simultaneously as the previous stage, to get the validation done on the final requirements.

7. After we have agreed on the final requirements, the requirements were split into features. The features were then documented in the same document as the requirements.

8. The acceptance of the SRS happened as late as possible in the project's first release, in order to have the document to be acceptable to change throughout the progress of the project.

7 RESULT OF THE STUDY

7.1 Software Requirements Specification Template

The software requirements specification document states the functions and capabilities that a software system must provide, its characteristics, and the constraints that it must respect. It should describe as completely as necessary the system's behaviours under various conditions, as well as desired system qualities such as performance, security, and usability. The SRS is the basis for subsequent project planning, design, and coding, as well as the foundation for system testing and user documentation. However, it should not contain design, construction, testing or project management details other than known design and implementation constraints. The vision and scope document usually contain the business requirements, and user requirements can be captured in the form of use cases or user stories. The product's functional and non-functional requirements are often stored in a software requirements specification, or SRS, which is delivered to those who have the task of designing, building and verifying the software solution.

The template that I have been created for the intent to be used in projects regarding internal software automation systems, can be found in the next chapter, chapter 7.2. When the template is oriented towards internal software systems, some aspects can be neglected from the original requirements, for example, some aspects regarding customer needs. Often the documents are strongly linked together and contain the same information and can, therefore, be combined in the same document.

Here we will go through the different chapters in the new template, to give a brief explanation as to why I have chosen to add or keep them. The first two chapters of the new SRS & FDS template includes the purpose of the document and some necessary abbreviations and acronyms, that the reader might know before reading the document.

The third chapter, product overview, gives the reader a brief understanding of why this system is needed. The different stakeholders or user classes and a list of the different

requirements, that the system must fulfil, are located in the fourth chapter. The fifth chapter contains all of the different features that are expected of the system. Features are a more-in depth explanation of how the system will be implemented so that the requirements are met. A feature consists of a short description, usually why and what the feature will do, and a functional requirement, which is a more technical explanation of how it will be implemented.

A list of priorities, possible trade-offs, and the planned lifecycle subsections can be found in chapter 6. A numbered list of priorities can be very helpful in the development stage of a system, this allows the scrum master to more easily plan the content of the sprints according to the project plan. If for some reason the deadline starts to creep too close, the scrum master can take a look at the trade-offs subsection and see what features can be postponed to a later release. The planned lifecycle of a calibration system is usually linked to some product's lifecycle. This is because the system is often used to perform the calibration and adjustment of the calibrator. This subsection contains in many cases a link to a document where the planned lifecycle of the product(s).

I have decided to include a whole subsection of Data Requirements, this chapter cannot be found in the older template. This displays the synergy and interfaces between the different parts of the whole system. The first subsection includes Logical Data Models, such as UML diagrams, flowcharts, and other diagrams for presenting the handling of data in the system. The second subsection asks about the different interfaces throughout the system. In these sections, is the saying "A picture is worth more than thousands of words" good to keep in mind. Subsection 7.3, data acquisition, integrity, retention, and disposal, is especially crucial with the new ISO 17025:2017 version of the ISO17025 standard, how the data will be acquired and handled that the system fulfils the data integrity requirement. Especially, what data is required for the system to obtain during usage, to make it possible to fulfil the requirement?

The eight-chapter refers to all the quality attributes of the software system. These quality attributes include usability, maintainability, performance, security, and others. At the end of the document, in chapter 9, is the business context explained, aspects such as resources,

cost estimation, and a risk assessment. The last two chapters, chapters 10 and 11, are a conclusion and references to other documents.

7.2 The Content of the new SRS template

7.2.1 Purpose

What is the purpose of this document?

7.2.2 Definitions, Acronyms, and Abbreviations

List all definitions, acronyms, and abbreviations used in this document. See example below:

- SRS Software Requirements Specification
- FDS Functional Design Specification
- UML Unified Modelling Language
- GUI Graphical User Interface
- SW Software
- HW Hardware
- DUT Device Under Test
- DB Database

7.2.3 Product Overview

This chapter gives an overview of the system, as well as giving a deeper view of the project.

7.2.4 Scope and Background

Scope of the project.

7.2.5 Vision

Vision of the project.

7.2.6 Relation to Strategy

The system’s relation to the strategy of the company in question. Is it a part of the development of a new product?

7.2.7 Product Requirements

Product Requirements.

7.2.7.1 User Classes

List the different user classes and stakeholders of the system.

7.2.7.2 R1

Requirement 1.

7.2.7.3 R2

Requirement 2.

7.2.7.4 R3

Requirement 3.

7.2.8 System Features

System Features.

7.2.8.1 F1

Feature 1.

Description

Explain the feature so it is more easily understandable.

Functional Requirement

Give a brief explanation of the functional requirements, so that the feature will be implemented correctly.

7.2.8.2 F2

Feature 2.

Description

Explain the feature so it is more easily understandable.

Functional Requirement

Give a brief explanation of the functional requirements, so that the feature will be implemented correctly.

7.2.8.3 F3

Feature 3.

Description

Explain the feature so it is more easily understandable.

Functional Requirement

Give a brief explanation of the functional requirements, so that the feature will be implemented correctly.

7.2.9 Data Requirements

This chapter will give a better understanding of the data that the system is using and producing. This way it will be easier to prevent any misunderstandings during the design and development stage.

7.2.9.1 Logical Data Model

Insert any UML diagrams, flowcharts or other diagrams. Remember that “One picture explains more than a thousand words”!

7.2.9.2 Data Dictionary

This is a more in-depth view of the kind of data that the software in the system will use. The information will be of great use to the software developers.

User Interface

A mock-up or picture of the user interface and preferably some explanation about the functionality.

DUT Interface

What data will be read and written from/to the DUT? What drivers will be used?

HW Interface

What interfaces are needed in the system to measurement, generator or some other devices used in the system?

DB Interface

What databases will the system use and what data will be read or stored in the databases?

Communication Interface

Are there some other interfaces needed in the system's software? Some domain, for verifying the users' credentials, as an example.

7.2.9.3 Data Acquisition, integrity, retention, and disposal

What are the channels for the required data to be acquired? How will it be processed; will it follow some data integrity constraints? How will the data be disposed of after it has fulfilled its purpose?

7.2.10 Quality Attributes

The following attributes are not necessarily requirements, but rather nice to have characteristics of the system. These attributes are often about giving more value to the system and its users.

7.2.10.1 Usability

Explain briefly the usability of the system.

7.2.10.2 Maintainability

How easy it will be maintained in the future, both SW and HW.

7.2.10.3 Performance

What is the expected performance or precision of the system?

7.2.10.4 Security

Give some brief aspects of the security in the system.

7.2.10.5 Others

Possibly some other attributes can exist.

7.2.11 Business Context

7.2.11.1 Resources

What are the resources that can be put on the project?

7.2.11.2 Cost Estimation

Give a cost estimation of the whole project.

7.2.11.3 Risk Assessment

What risks are associated with the project?

7.2.12 Conclusions

Final discussion and conclusions.

7.2.13 References

List all the needed references.

7.3 Final Software Requirements Specification Document

The new temperature calibration automation system that came out of this project is called CATS, short for Calibration & Adjustment Temperature System. When I got to put the new template to use in this project, I appreciated two things, first, the template follows a chronological order and secondly it is comprised of the SRS and FDS in the same document. This reduces the hassle of repeating the same content throughout many documents, and all the vital information stored in one place.

7.4 Identification of the stakeholders

This step was easy when all the stakeholders and I work quite close to each other and I know who is responsible for what. The different stakeholders for this new software system are the Product Owner of the new temperature calibrator, the Product Owner of all Product Automation systems, the end-users, and the laboratory.

7.5 Perform the requirements elicitation activities

This process was a bit more painstaking than the previous when there were a lot of requirements that were in quite a detail, for example how the system must perform temperature measurements. Luckily the company in question is following an agile development process, which allows for the requirements to be more supple and not set in stone.

7.6 Requirements analysis and validation

The requirements analysis part was important to me. In this step, I got to go much deeper into the meaning of the requirements and the origin of them. My thought at this stage was: If there is no reason to have it, then why have it? When I understood the meaning and origin of the requirements it was much easier to later put them into features. The requirements were also validated when the stakeholders were questioned more about the requirements and their origins.

7.7 Requirements documentation

The documentation of the requirements was an undergoing process during the previous three steps. This was helpful to me when I tend to forget things that have already been handled.

7.8 Requirements split into features

To put the requirements into features should have not been that difficult, but when I wrote the features, I only had myself as a developer in mind. This resulted in, when a different developer read the features and were to implement them, it got wrong at some points and required much more information, than was documented. This mistake I will try to keep in mind when I do it next time and surely it also comes more naturally with experience.

7.9 The final version of SRS and FDS accepted

The accepting of the SRS and FDS was quite late in the project, but rightfully so because the document has seen many updates and corrections during the time since it has been created.

8 CONCLUSION

The existing User Requirements Specification template, found in Appendix 1, was not seen to be 100% compatible with an internal software system and more suited towards the external customer market. Additionally, a separate document, Functional Design Specification, which is more in-depth is used. By combining these two and also some other subjects as well, we get an easier maintained documentation template. The value of a well suited/designed requirements specification template is beneficial from every aspect.

The different standards that the calibration process must follow in an accredited laboratory cannot be negotiated and therefore plays a big role in the requirements elicitation process. It is one point that needs to be put much thought into so it cannot be misunderstood during the development. It can easily lead to a lot of rework. The person responsible for the requirements elicitation must put his or her mind into it and understand what they mean.

The specification process of a new software system is considerably more straightforward when using a process that is well fitted for just that type of project. If a development process is followed that is intended for a different type of product is used, it can be more destructive than helping. The software requirements documentation in a scrum development environment has the strengths that it is agile and live throughout the whole project. One more content dense document is then better than multiple documents that have a strong linkage. This can result in a large effort in the maintenance of requirements when changes need to be implemented in multiple documents.

The thing that I have noted time after time throughout the implementation phase of this project, is that “change happens”. Even if I tried to take all aspects into consideration during the specification elicitation and analysation phase, there is always some aspect that has been misunderstood, forget or simply does not work that way that I had planned. So that is why the agile development method has its benefits in the development process. Nothing is locked and can be changed between development iterations. But I also noticed

on multiple occasions, as an unexperienced technical coordinator and developer, it is hard to describe the features in such a detail that it gets developed just the way that was intended. The human aspect in the requirements elicitation stage can easily lead to misunderstandings. In future projects, I would like to utilize some sort of tool for this application. A tool that can visually present all the stakeholder's requirements and their dependencies. This would possibly eliminate some of the loss of details and misunderstandings.

Prioritization among the requirement should be a prioritization among itself, especially when following the SCRUM development method. By prioritizing among the requirements there is a significant rise in success-rate and better risk management. This has been proven in a case study by Nancy Mead and analyzed by Daniel Port, Barry Behm and David Klappholz (2008). The participants in the study got to use a cost-value method, that visually presents the cost of implementing and the value of a requirement, set by the stakeholders. This approach could be used in prioritizing the requirements.

One aspect to take into consideration in future SRS templates is improving and balancing software qualities. Nowadays software qualities, SQs or non-functional requirements, are put more and more in focus in software development and probably will be even more in the future. The trend in software development points towards more software-intense systems, with greater complexity, autonomy, speed of change, and need for interoperability between systems. The key aspects of the SQOTA, System Qualities Ontology, Trade space, and Affordability, framework is if the higher levels of SQ are implemented correctly, then the lower level SQ's are automatically, or more easily, supported, see Appendix 2. (Barry 2016).

REFERENCES

Beatty, Joy & Anthony Chen (2012). *Visual Models for Software Requirements*. WA. Microsoft Press. ISBN 0-735-66772-1.

Boehm, Barry (2016). *Improving and Balancing Software Qualities. 2016 IEEE/ACM 38th IEEE International Conference on Software Engineering Companion, 890-891.* A Technical briefing on Software Qualities.

Boehm, Barry & Richard Turner (2003). *Balancing Agility and Discipline: A Guide for the Perplexed.* Boston. Addison-Wesley. ISBN 0-321-18612-5.

Cohn, Mike (2004). *User Stories Applied: For Agile Software Development*. Boston. Addison-Wesley. ISBN 0-321-20568-5

Davis, Alan M. (2005). *Just Enough Requirements Management: Where Software Development Meets Marketing*. New York. Dorset House Publishing. ISBN 0-932-63364-1.

Derby, Esther & Diana Larsen (2006). *Agile Retrospectives: Making Good Teams Great*. Raleigh, NC. The Pragmatic Bookshelf. ISBN 978-0-977616-64-0.

IEEE. (1998). *Standard 830, IEEE Recommended Practice for Software Requirements Specifications*. A standard for practices in the software requirements specification, in which a suggested template for software requirements specification is presented.

IIBA. (2009). *A Guide to the Business Analyst Body of Knowledge (BABOK Guide)*, *2nd edition*. Toronto. International Institute of Business Analysis. ISBN 0-981-12921-8.

ISO/IEC. (2017). 17025:2017, *General Requirements for the Competence of Testing and Calibration Laboratories.* Geneva, Switzerland. International Organization for Standardization. An ISO/IEC standard that concerns testing and calibration laboratories.

Kerth, Norman L. (2001). *Project Retrospectives: A Handbook for Team Reviews*. New York. Dorset House Publishing. ISBN 978-0-932633-44-6.

Kulak, Daryl & Eamonn Guiney (2003). *Use Cases: Requirements in Context, 2nd edition*. Boston. Addison-Wesley. ISBN 0-321-15498-3.

Larman, Craig (2003). *Agile and Iterative Development: A Manager's Guide.* Boston. Addison-Wesley. ISBN 0-131-11155-8.

Leffingwell, Dean (2011). *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise.* Upper Saddle River, NJ. Addison-Wesley. ISBN 978-0-321635-84-6.

Nicholas, J. V. & D. R. White (2001). *Traceable Temperatures, An Introduction to Temperature Measurement and Calibration, 2nd edition*. West Sussex, England. John Wiley & Sons. ISBN 0-471-49291-4.

Parody, Liz (2018). How to Manage Modern Software Projects: *Waterfall vs. Agile*. [online]. [cited at: 20.8.2019]. Located at: https://medium.com/@lizparody/waterfall-vs-agile-methodology-in-software-development-1e19ef168cf6. A web platform for people to learn and share on different topics.

PMI. (2013). *A Guide to the Project Management Body of Knowledge: PMBOK Guide, 5th edition.* Newton Square, PA. Project Management Institute. ISBN 8925598620.

Port, Boehm & David Klappholz (2008). *Nancy R. Mead: Making Requirements Prioritization a Priority.* Published in: *2008 21st Conference on Software Engineering Education and Training, 250-261.* IEEE. ISBN: 1093-0175.

Rubin, Kenneth S. (2012). *Essential Scrum: A Practical Guide to the Most Popular Agile Process.* Upper Saddle River, NJ. Addison-Wesley. ISBN 0-137-04329-5.

Schwaber, Ken (2004). *Agile Project Management with Scrum.* Redmond, WA. Microsoft Press. ISBN 0-735-61993-X.

Smith, Larry W. (2000). *"Project Clarity Through Stakeholder Analysis"* CrossTalk 13(12):4-9. Technical research paper.

Sommerville, Ian & Pete Sawyer (1997). *Requirements Engineering: A Good Practice Guide*. Chichester, England. John Wiley & Sons Ltd. ISBN 0-471-97444-7.

Tan T., Li Q., Boehm B., Yang Y., He M. & Ramin Moazeni. (2009). *Productivity trends in incremental and iterative software development.* In: *2009 3rd International Symposium on Empirical Software Engineering and Measurement, 7.* IEEE. ISBN 978-1-4244-4842-5.

Wiegers, Karl E. (2001). *Peer Reviews in Software: A Practical Guide*. Boston. Addison-Wesley. ISBN 0-201-73485-0.

Wiegers, Karl E. (2005). *More About Software Requirements: Thorny Issues and Practical Advice*. Redmond, WA. Microsoft Press. ISBN 0-735-62267-1.

Wiegers, Karl E. (2007). *Practical Project Initiation: A Handbook with Tools*. Redmond, WA. Microsoft Press. ISBN 0-735-62521-2.

Wiegers, Karl E. & Joy Beatty (2013). *Software Requirements: Developer Best Practices*, *3rd edition*. Redmond, WA. Microsoft Press. ISBN 0-735-67966-5.

APPENDICES

Appendix 1. The existing SRS template.

The table of content of the existing SRS template at the company in question.

1. Purpose
2. Definitions, Acronyms, and Abbreviations
3. Relation to strategy
4. Product Overview
5. Customer needs
6. Composing requirements
7. Product requirements
8. Development framework
9. Financial and commercial aspects
10. Risk assessment
11. References

Appendix 2. The table of the System Qualities Ontology, Tradespace, and Affordability framework hierarchy.

Stakeholder Value-Based SQ Ends	Contributing SQ Means
Mission Effectiveness	Stakeholders-satisfactory balance of Physical Capability, Cyber Capability, Human Usability, Speed, Endurability, Maneuverability, Accuracy, Impact, Scalability, Versatility, Interoperability, Domain-Specific Objectives
Life Cycle Effectiveness	Development and Maintenance Cost, Duration, Key Personnel, Other Scarce Resources; Manufacturability, Sustainability
Dependability	Reliability, Maintainability, Availability, Survivability, Robustness, Graceful Degradation, Security, Safety
Changability	Maintainability, Modifiability, Repairability, Adaptability
Composite SQs	
Affordability	Mission Effectiveness, Life Cycle Efficiency
Resilience	Dependability, Changeability