Defining maturity of agile requirements engineering practices

Leon de Reeder

Business Informatics, Faculty of Science
Department of Information and Computing Sciences
Utrecht University
Domplein 29, 3512 JE Utrecht

Info Support BV
Kruisboog 42, 3905 TG Veenendaal

Supervised by
Dr. Sjaak Brinkkemper and Dr. Fabiano Dalpiaz

December 11th 2019

Abstract - Scrum provides a framework for project management that has become very popular over the past few years in the context of software development. However, Scrum has many gaps and holes in which practitioners are free to adjust the framework to their individual needs or apply best practices. This study investigates what common practices are in agile requirements engineering and defines a maturity matrix that can be used to assess Scrum teams. The maturity model is developed based on a case study and validated using interviews and focus groups in order to assess teams and guide toward good agile RE practices.

Utrecht University

Acknowledgements

Foremost, I would like to express my gratitude for all the supervision and guidance received from my supervisor dr. Sjaak Brinkkemper. His expertise on the subject of maturity models helped me immensely for those moments where I needed some directions.

Secondly, I would like to thank dr. Fabiano Dalpiaz for being my 2nd supervisor and, most of all, supplying me with an incredibly valuable dataset that I could use for my research and contribute to.

Thirdly, my sincere thanks go out to my daily supervisor at Info Support, Beau Verdiesen. His help with contacting relevant experts and participants in my study was invaluable, as was his help proofreading my work.

Finally, I thank all experts and colleagues that were willing to participate or contribute to my study. Alex van Assem, Cas Jongerius, Erik Sackman, Ivo Diepstraten, Marc Wils, Marieke Keurntjes, Mark Jousma, Mark van Schaik, Patricia van Eijk & Remi Kok.

List of Abbreviations

RE	Requirements Engineering
PBI	Product Backlog Item
PB	Product Backlog
BDD	Behavior-Driven Development
RQ	Research Question
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration

1. Introduction

1.1. Problem statement

One of the great characteristics of agile methodologies is that they, as their name implies, are agile. They can easily be adapted to operate in a great number of different scenarios and contexts and be flexible in volatile environments. While the adaptability of these methodologies is one of their biggest advantages, it can also become one of their disadvantages. The flexibility of these methodologies means that usually, no two scenarios in which they are applied will be identical, making it hard to perform scientific research on an experimental basis due to the many confounding variables introduced. Currently, one of the most popular agile software development methods in use is Scrum [1]. Scrum is defined as *"A framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value"* [2]. Even though many teams claim they use Scrum, many of them do not use 'pure Scrum' as described by the original creators [3]. As stated by the Scrum guide written by the original authors, *"Scrums' roles, events, artifacts, and rules are immutable and although implementing only parts of Scrum is possible, the result is not Scrum. Scrum exists only in its entirety and functions well as a container for other techniques, methodologies, and practices"* [2]. As many teams adapt or extend Scrum to fit their own needs, it becomes easy for organizations to lose track of the way of working of teams, as a uniform process no longer exists.

"*I estimate that 75% of those organizations using Scrum will not succeed in getting the benefits that they hope for from it … Scrum is a very simple framework within which the "game" of complex product development is played. Scrum exposes every inadequacy or dysfunction within an organization's product and system development practices. The intention of Scrum is to make them transparent so the organization can fix them. Unfortunately, many organizations change Scrum to accommodate the inadequacies or dysfunctions instead of solving them … Scrum purposefully has many gaps, holes, and bare spots where you are required to use best practices … "* - Ken Schwaber on Scrum [4]

As the above quote from Ken Schwaber indicates, many teams change Scrum to accommodate inadequacies or dysfunctions. For those areas that Scrum leaves room for improvisation, teams should use best practices. However, it is unclear what these best practices are. Therefore, this study investigates software development teams and the processes they use within the Scrum framework to deliver value to customers. The focus is on requirements engineering (RE), as this is a critical process in the software development lifecycle with a big impact on the success of a project [5]. Requirements engineering is the subset of systems engineering concerned with discovering, developing, tracing, analyzing, qualifying, communicating and managing requirements that define the system at successive levels of abstraction [6]. The study by Hall, Beecham & Rainer suggests that mature companies have fewer requirements

problems [7]. We are curious if and how teams have altered their RE practices in Scrum and if it is possible to define 'best' or 'mature' practices that can be generalized across industries.

1.2. Research context

The goal of this research is to develop a model to assess the maturity of requirements engineering practices in Scrum. This allows for the following benefits;

- to compare the processes used by different software development teams.
- to provide an easy assessment for teams to determine their maturity.
- to guide immature teams toward maturity, speeding the adoption of good practices.

Translating the goal into the main research question (MRQ) gives us;

MRQ: How to develop a maturity model for RE practices in Scrum?

In order to answer the MRQ, several research questions are chosen to steer the research.

RQ1: What are common RE practices in Scrum used by software development teams?
In order to determine maturity practices, we need to get an understanding of RE practices that are currently being used by software development teams. This question will be answered by using scientific literature and case studies.

RQ2: How can RE maturity in Scrum be defined?
Once an understanding of common RE practices in Scrum is defined, we must discover how to define maturity. What does it *mean* to be mature in RE in Scrum? Hopefully, the answer to RQ1 provides us an overview of common practices and characteristics of agile RE and Scrum, which can be used to identify the meaning of maturity. This needs to be done based on expert opinions and best practices according to literature and case studies.

RQ3: How can RE maturity be measured and compared among different software development teams?
RQ4: Is the constructed artifact valid?
Answering RQ3 will result in a model that can be applied to software teams that are working in different contexts. As all software teams are unique due to size, environmental influences, differences in skills among others, it is important to develop a model that allows for comparison of their RE practices and maturity. This model also needs to be validated.

1.3. Contributions of this study

Contribution to science

This study provides a scientific contribution by performing a study on commonly used RE practices in Scrum. While there have been studies done on processes used in Scrum and how they differ among practitioners (e.g. Diebold et al. [3]) there are no studies that explicitly focus on RE practices in Scrum, nor is there any measurement tool to assess and compare different software development teams' RE practices. This study contributes to the body of knowledge by introducing both knowledge on common agile RE practices and variations in RE and a validated assessment model that can be applied on teams. This study introduces a maturity model for RE practices in Scrum. While there have been previous maturity models that are based on requirements engineering these have been based on traditional development techniques like the waterfall model. These cannot simply be converted to an agile development technique like Scrum. While the purpose of traditional based RE and agile RE is the same, they have key different characteristics.

Contribution to Info Support

As organizations grow and become larger their organizational structure adjusts. Business units grow in size and become more independent. These units start forming their own internal policies and guidelines. At Info Support, software development teams have always been given lots of freedom to approach their way of working. After years of growth, upper management has obtained new interest in their software development teams' way of working. While it is not required that all teams follow a uniform process, insight in processes/techniques used by teams can provide valuable knowledge. The development of a maturity model can allow for easy assessments of teams and might lead to improvements among teams.

1.4. Outline and milestones

This thesis work is spread out over a period of 8 months from May until December 2019. There are two phases. The first phase takes three months and answers RQ1 and RQ2 by performing an exploratory literature study, expert interviews and case studies. The second phase takes 5 months and is used to answer RQ3, RQ4 and the writing of the thesis.

2. Research approach

2.1. Design science

This study is a study of design science, as the main goal is to design and investigate an artifact in a certain context [8]. The goal of this particular study is to design a maturity model for RE practices in Scrum. The artifact in question is the maturity model. From the design of this artifact, we also want to investigate its problem context and the validation of the artifact. Wieringa's design cycle provides a framework for this study to follow; a problem investigation, treatment design, treatment validation and treatment implementation [8].

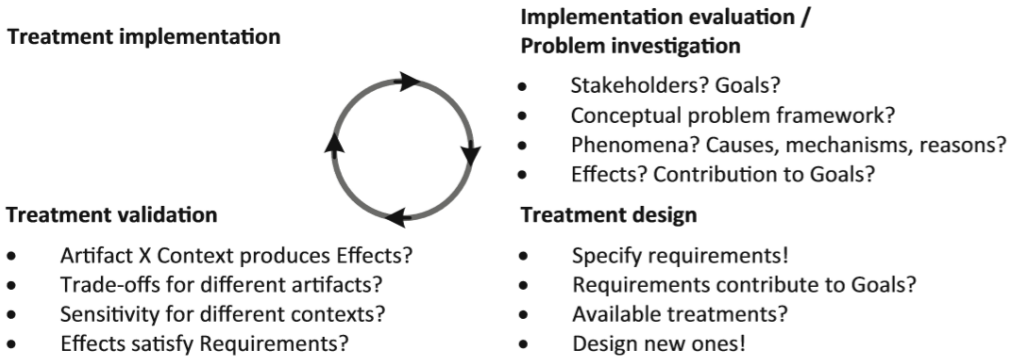


Figure 1: The design cycle [8]

2.2. Solution design & validation

Before the maturity model can be designed, we need to get an understanding of the common RE practices in Scrum currently used by teams. This is done by answering RQ1 ("*What are common RE practices in Scrum used by software development teams?"*), based on a literature review and case studies of development teams.

Next, an understanding of maturity in the context of these practices must be defined. This is done by answering RQ2 "*How can RE maturity in Scrum be defined?*". Here we try to define best practices by studying literature and case studies. RQ1 and RQ2 fit in the problem investigation of the design cycle.

Once it is clear what constitutes as RE maturity, we need to answer RQ3, *"How can RE maturity be measured and compared among different software development teams?"*. The answers to RQ1, RQ2 and RQ3 will be used to create a maturity model, which relates to the second phase of the design cycle *Treatment design*. The model will be designed based on the literature review and case studies. In RQ4 (*"Is the constructed artifact valid?"*) we aim to validate the designed model. The research method used to validate

the model will be expert opinions and a focus group. A team of experts will be selected, consisting of product owners or other practitioners who extensively encounter requirements engineering practices in their daily work, e.g. business/information analysts. Experts are used to check whether the designed artifact conforms to the desired requirements. Feedback from the experts will be used to redesign the artifact, if necessary. Table 1 shows what sources will be used to answer each sub research question. The research questions have been plotted against the design cycle in figure 2.

Question	Literature	Case studies	Experts
RQ1	X	X	
RQ2	X		
RQ3	X	X	
RQ4			X

Table 1: What sources will be used to answer the research questions.

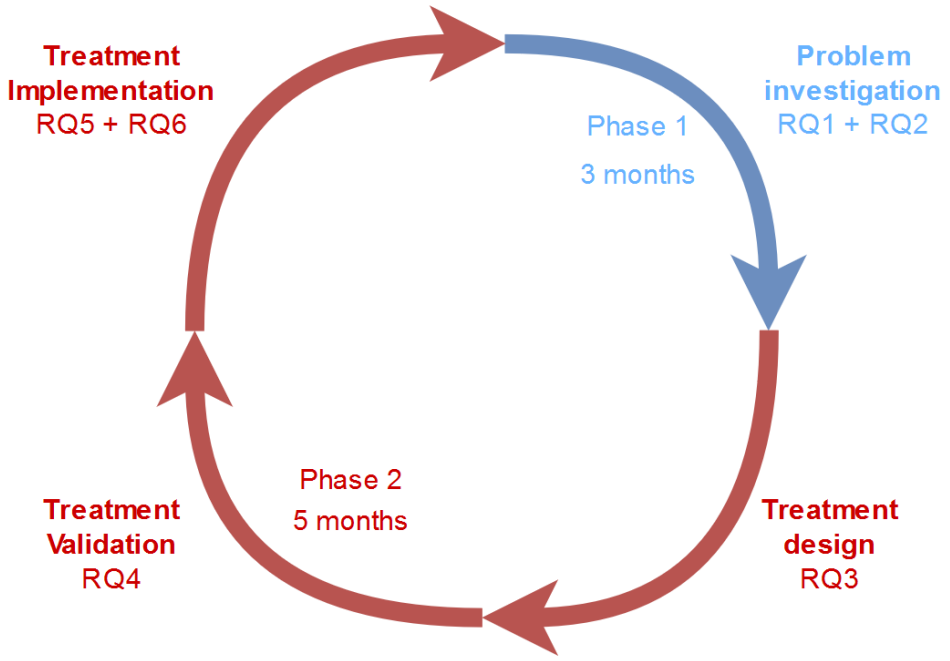


Figure 2: The research questions of this study mapped on the design cycle

As can be seen in figure 2 there originally were two other research questions. RQ5 (*In which ways does high maturity provide benefits?*) and RQ6 (*Should software development teams strive for high maturity in RE?*) did not serve to directly answer the main research question but instead wanted to answer knowledge questions about the designed artifact to determine whether the artifact can be useful in practice. The main idea was that once the model can be applied on software development teams we can investigate

whether and to what extent high maturity provides benefits over low maturity. If that were the case, then it would sound logical that high maturity in RE practices in Scrum is something these teams should strive for. To check for this, RQ6 wanted to discover in which scenarios development teams should actually invest effort in achieving certain levels of high maturity. It is entirely possible that raising the level of maturity requires a big investment of resources for relatively low returns. Sometimes, a convenient practice might be the best practice, being more beneficial than some expansive practice. Unfortunately, these knowledge questions about the artifact had to be cut due to time constraints.

2.3. Theory Building

Not only is this a study of design science, but it is also a study of theory building. We try to construct a theory on what mature practices in agile RE are to be able to construct an artifact that assesses these practices. Following the flowchart (figure 3) by Dul & Hak, we perform theory-building research [9]. We start with an exploration of theory, which is done by performing a literature review for RQ1 and RQ2. Now, an exploration of practice is done, which in this study consists of case studies, to also answer RQ1. These two explorations are used to build a theory of RE maturity in Scrum (the maturity model, RQ3) which is then followed by initial theory-testing (validating the model, RQ4).

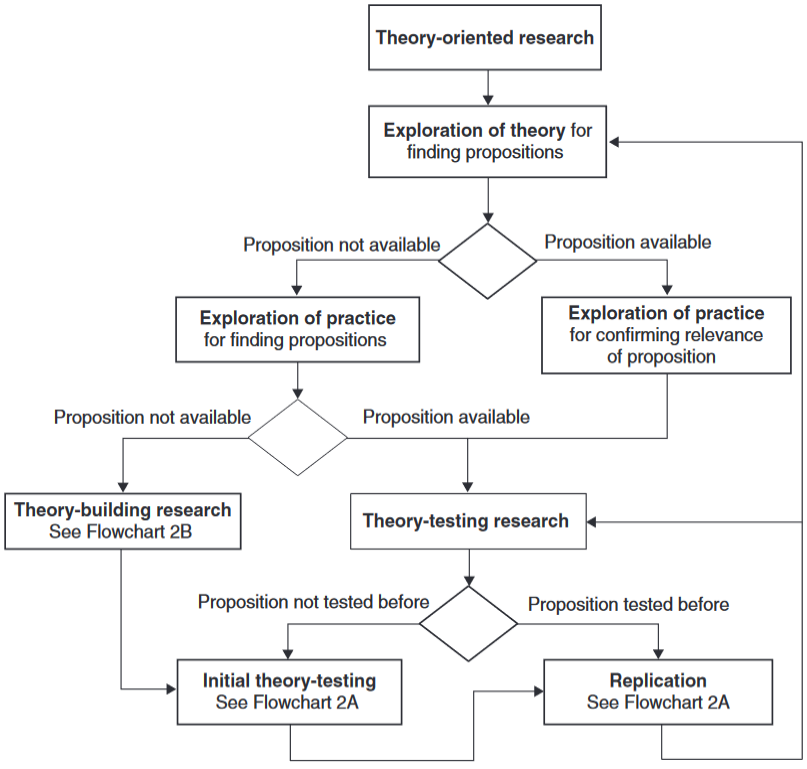


Figure 3: Deciding on the type of research can be done by following this flowchart [9]

2.4. Literature review

A literature review is used as a starting point to answer the first three research questions. It is *not* a systematic literature review but focuses on the specific research questions. As for the procedure, a search engine is first used (primarily Google Scholar). Relevant work is selected, from which backward and forward snowballing is used to find other relevant literature. Journal and conference publications were also scanned. Grey literature is also included if it's relevant to the question at hand and publicly available. Finally, previous related literature performed by Utrecht University was also considered. The only inclusion criteria for scientific articles is that the article must be accessible through the available portals without a paywall.

2.5. Case studies

In order to confirm and expand findings in the literature review (the exploration of theory) case studies are used. Case studies are used as an exploration of practice. A case study is defined by Dul & Hak as 'a study in which (a) one case (single case study) or a small number of cases (comparative case study) in their real-life context are selected, and (b) scores obtained from these cases are analyzed in a qualitative manner [9]'. In this study, we combine a multiple-case study performed by the author of this study at the host company of Info Support with a multiple-case study performed by students at other companies. These case studies have been described in reports and are encoded and analyzed in NVIVO, which is computer software for qualitative analysis.

2.6. Interviews

Expert interviews are used at several points during the study to collect qualitative data. Expert interviews are used to:

- gather data on RE practices in agile software development teams (chapter 4 & appendix A).
- validate the designed maturity model (chapter 5 & appendix C).
- apply the maturity model to a team (chapter 6).

The sampling method used for all interviews is convenience sampling. One concern with interviews discussed in the work by Englander is that the sampled interviewees belong to the required population (agile software development teams) and that the interviewee has the right expertise [10]. Due to the nature of the host research company (software development) selecting interviewees belonging to the correct population was guaranteed. Experts having the right expertise is more difficult to guarantee. Experts were selected based on their role which had to be related to requirements engineering but the amount of experience the experts had was out of the control of the researcher.

2.7. Metamodeling for situational analysis and Design Methods

In this study, there will be several figures in which the metamodeling technique described by van de Weerd & Brinkkemper is applied [11]. This UML-based modeling technique is used to model and compare the processes used in the context of Scrum by software development teams. It is a useful technique to visualize the processes in a project, the order in which they occur, who executes the process, what kind of deliverable or concept is generated by the process and how these deliverables relate to each other.

3. Literature review

This section executes the literature review to answer RQ1 *"What are common RE practices in Scrum used by software development teams?"* and RQ2 *"How can RE maturity in Scrum be defined?"*. We briefly introduce the topics requirements engineering and Scrum to provide an understanding of the concepts. Section 3.1 provides a literature review on the common practices of agile RE. Section 3.2 provides a literature review on maturity research in Scrum and RE.

3.1. Requirements engineering

In this study, the concept of requirements engineering takes the definition provided by Dick, Hull & Jackson [6]. Requirements engineering is the subset of systems engineering concerned with discovering, developing, tracing, analyzing, qualifying, communicating and managing requirements that define the system at successive levels of abstraction [6]. In order to achieve these goals, many techniques and tools have been proposed. A requirement is a 'statement that identifies a product or process operational, functional, or design characteristic or constraint, which is unambiguous, testable or measurable, and necessary for product or process acceptability' [6]. Requirements engineering has been found to be a critical process in software development with a big impact on the end success of a project [5].

While traditionally RE has been a phase that (mostly) took place prior to the actual development in the waterfall model some of the activities that belong to RE take place during the entire development lifecycle, such as elicitation, documenting and managing of requirements. Traditional RE consists of sub-sequential activities [12]. First, requirements elicitation is performed where initial information regarding requirements and context is gathered. Secondly, during requirements analysis, this information gathered is analyzed to understand the requirements. During requirements specification information is turned into precise system specifications. Requirements validation identifies and corrects errors in the requirements. Requirements prioritization supports the elicitation by specifying the most valuable requirements. Finally, system validation validates whether the output of the software matches the requirements. There is some discussion whether system validation is part of requirements engineering, as some studies state it does ([12]), and some state that it does not [13]. In general however, the traditional phases of RE are quite clear and well defined.

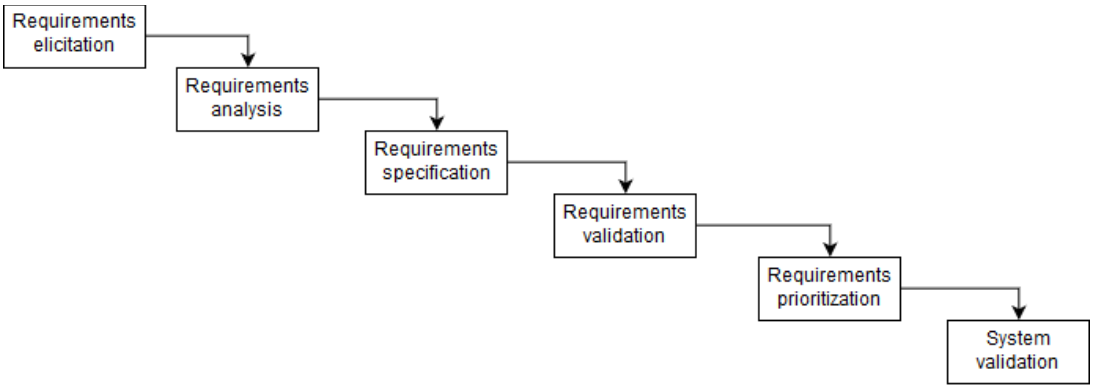


Figure 4: RE phases in the waterfall model [12]

3.1.1. Agile software development

Agile software development is an approach to software development that has seen a significant rise in popularity during the last two decades [14]. This rise followed the release of the agile manifesto, a manifesto that called for principles such as 'Responding to change over following a plan' and 'Working software over comprehensive documentation' [15]. Agile development was an answer to the tendency of requirements to change during processes, leading to an increase in costs [16]. Agile methodologies aim to provide flexibility that traditional methods, such as the waterfall model, lack.

Agile requirements engineering

Agile RE differs from traditional RE because it assumes that RE is practiced during the entire lifetime of a product/system, whereas traditional RE is performed before the development of the system is performed. One of the twelve principles of the agile manifesto relates directly to this; 'Changes in requirements are welcome, even late in development' [15]. The requirements are initially defined with the customer, after which they become more refined, reprioritized and discussed over several iterations of work to fit the scope of the next iteration [17].

3.1.2. Scrum

According to the official guide written by Scrums' original creators, Scrum is 'a framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value' [2]. The annual report on the State of Agile found that Scrum is by far the most used framework for agile project management, with 72% (n=1319) of the respondents using Scrum or an adapted variant of Scrum [1]. Scrum is characterized by its short, iterative feedback loop and Scrum Teams. All that the Scrum framework defines are some roles, events and artifacts. Therefore, many other aspects can be adapted to a teams' preferences. The following description on Scrums' Roles, Events and Artifacts is directly based on

the official guide by Schwaber and Sutherland and is, therefore, a description on Scrum in its 'purest' form [2].

Scrum team & Roles

Scrum is performed by a Scrum team of 3 to 9 members, excluding the product owner and Scrum Master (unless they also execute work of the sprint backlog). There are three roles in the Scrum team; the Scrum Master, the Development Team and the Product Owner.
The Product Owner is responsible for maximizing the value of the product resulting from the work of the Development Team. The Product Owner has the responsibility of managing the Product Backlog. This contains activities like expressing backlog items, ordering the items in the backlog, optimizing the value of work for the Development Team, ensuring visibility and transparency of the Product Backlog and ensuring that the Development Team understands items in the Product Backlog on the level needed.
The Development Team consists of professionals who work on the delivery of a releasable increment of the product at the end of each sprint. The Development Team is self-organizing, cross-functional and shares accountability for the release.
The Scrum Master promotes and supports Scrum by helping everyone understand Scrum theory, practices, rules and values.

Scrum events

Scrum contains one all-encompassing event that serves as a container for other events; the Sprint. The sprint is a phase of less than one month during which a releasable product increment is created. The sprint contains four other events. Firstly, Sprint Planning serves to identify the goal and work to be done for the current Sprint. Secondly, there is the Daily Scrum. These are short, standup meetings with the Development Team which discuss the work to be done for the day and to inspect the progress toward the sprint goal.
Thirdly, after there is no more work to be done for the current Sprint (or the time limit is reached) there is a Sprint review, which reflects on the sprint and work left to be done. Finally, the Sprint Retrospective aims at identifying areas of improvement which can be enacted during the next sprint.

Scrum artifacts

Scrum consists of three artifacts; the Product Backlog (PB), the Sprint Backlog (SB) and Increments.
The Product Backlog is an ordered list of everything required in the product. An item in the product backlog is often called a PBI (product backlog item). It contains requirements for any change necessary to be made. The PB evolves over time as new requirements emerge. The PB not only contains a list of requirements but also features, enhancements and fixes that are required to be made. Often, they contain even more concepts, such as testing definitions. The PB is never done and

continuously evolves. Backlog refining (sometimes also called Backlog grooming or cleaning) involves the adding of detail, estimates and order to backlog items.
The Sprint Backlog contains all backlog items selected for the current Sprint, together with a plan for delivering the product Increment and realizing the Sprint Goal. The purpose of the Sprint Backlog is to make all work visible that the development team identified as necessary to meet the Sprint Goal.
The Increment is the sum of all the Product Backlog items completed during a Sprint and the value of the increments of all previous Sprints. An increment must be 'Done' (what the definition of 'done' means is defined by the development team) at the end of a sprint. As teams mature, the definition of 'done' is likely to expand and include stricter criteria to ensure a higher quality of the increment. According to its official guide, it is very important that all team members have a 'shared understanding' of the definition of done.

Meta Modeling Scrum

The events, artifacts and roles present in Scrum are meta modeled in Figure 5 below. Many activities and concepts that occur in traditional RE are still undefined. As mentioned before, Scrum is a framework with many gaps in which users can choose what to apply, like techniques, processes or best practices [4]. Therefore, the meta-model of Scrum looks like Figure 5, but in real-world cases will look different depending on how development teams choose to alter/extend Scrum. Scrum technically only consists of iterations of Sprints, which consists of four activities that have their own sub-activities. The actual sub-activities are undefined by Scrum itself. The Sprint Planning results in a Sprint Backlog, which consists of Backlog Items. Following the Sprint Planning, Daily Scrum is performed iteratively for as long as the Sprint Backlog is not empty, contributing to the Increment. Finally, the Sprint is ended with the Sprint Review followed by the Sprint Retrospective, after which a new Sprint is started.

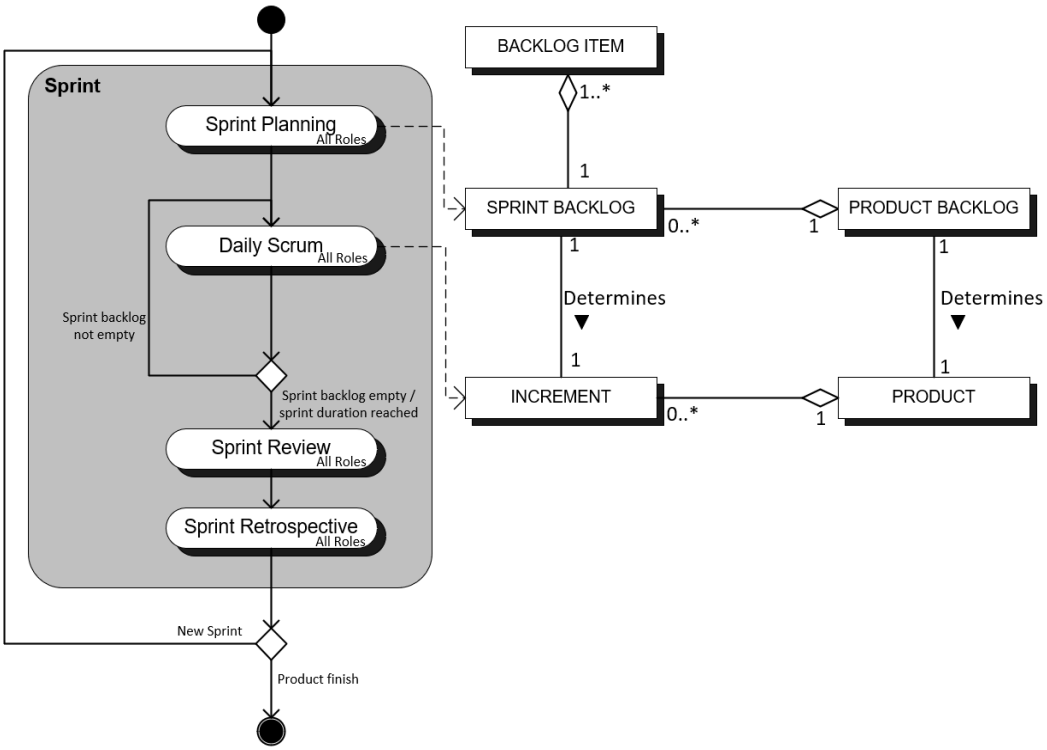


Figure 5: Metamodel of the Scrum framework with activities, roles and concepts

3.1.3. Requirements engineering in Scrum

In Scrum, the Product Owner is responsible for requirements elicitation and prioritization. The requirements reside in the Product Backlog, which is a prioritized list for the work still to be done for the software [12]. The Product Owner and development team perform requirements analysis, specification and validation in collaboration. At the end of an iteration, the Product Owner should perform the system validation [12]. Some studies report that agile RE resolves issues like communication, over scoping, requirements validation, requirements documentation and rare customer contact [18].

Although the official Scrum guide describes events, artifacts and roles many things are still unknown or open to interpretation. This is also the case for requirements engineering. While some of the responsibilities and events of requirements engineering are clear, details are missing. The Scrum guide, for example, states that the Product Owner is responsible for the elicitation of the requirements, but not which elicitation technique should/can be used. The guide is clear that the Product Owner should prioritize the requirements, but not which technique should be used (e.g. MoSCoW, Stack ranking, Kano model). The guide is clear that requirements are contained in the backlog, but not how they should be documented (e.g. format, acceptance criteria, quality criteria).

The mapping study by Curcio et al. on requirements engineering in agile software development identified 9 systematic literature or mapping studies on the topic of

requirements engineering in agile practices [17]. This study is used as a starting point to discover agile RE practices discussed in the literature.
The first study discussed by Curcio et al. is the systematic literature review by Inayat et al., which aimed to answer three research questions [18];

- RQ1. What are the adopted practices of agile RE according to published empirical studies?
- RQ2. What are the challenges of traditional RE that may get alleviated by agile RE?
- RQ3. What are the challenges of agile RE?

Twenty-one studies were included in the review. The study identified 17 different RE practices of which none had a higher occurring frequency than 5. This indicates that there is no single standard practice. The practices and their reported frequency can be found in table 2 below. While this table demonstrates some practices, it is unclear how they are performed. For example, many different techniques can be used for prototyping (e.g. wireframe, storyboard, full-fledged prototype) or prioritization (e.g. MoSCoW, gut feeling, business value, story points). It is also unclear if teams used several techniques for the same purpose (e.g. use both wireframes and storyboards).

Practice	**Reported frequency**
Requirements prioritization	5
Testing before coding	4
Face-to-Face communication	3
Customer involvement	3
Iterative requirements	3
Retrospectives	3
User stories	2
Change management	2
Prototyping	2
Requirements modeling	2
Review meeting and acceptance tests	2
Requirements management	2

Table 2: Identified requirements engineering practices with a frequency > 1 [18]

The study concludes that agile RE practices differ from traditional RE with practices such as customer involvement, change in requirements management, cross-functional teams, review meetings and sessions. These are features missing in the traditional way of RE; thus, they can outperform traditional RE practices [18].

The study by Medeiros et al. [19] focused on two specific RE activities in agile development;

1. What RE techniques are used to elicit requirements?
2. What RE techniques are used to specify requirements?

The study included 21 studies of which 50% used Scrum and others used methods like Kanban or Extreme Programming. Nine of these studies reported the requirements elicitation technique used. Almost all (8 out of 9) of these studies used interviewing as a technique to elicit requirements, joint application design (JAD) and focal groups were used three times, brainstorm was used twice and questionnaires, workshops and trawling were only mentioned once.
On the specifying of requirements, the study found that user stories were by far the most used format for defining requirements (19 out of 24). The popularity of this format is confirmed in other works, although the template of user stories can be different [20,21]. Most of the studies included in the review also reported the use of more than one specification format (21 out of 24) [19].

The literature study by Schön et al. was aimed at identifying practices that involve stakeholders in the process of RE and are compatible with agile software development and what common ways are for requirements management [13]. They found that 77% of participating teams directly involved stakeholders in the development process and 56% directly involved the end-user as well (n=27). On the topic of requirements management, 93% used an artifact, 63% had documentation that was understandable without prior knowledge and 50% differentiated between functional and technical requirements (n=27). On the types of artifacts, the user story was once again the most popular with 56% of studies reporting the use of user stories (n=27). Forty-one percent used some form of prototyping, followed by use cases (26%), scenarios (22%), story cards (22%), personas (15%) and other artifacts (<15%).

The study by Cao & Ramesh researched differences in practices between traditional and agile RE [22]. They found that in agile RE requirements are continuously reprioritized, unlike traditional RE where requirements are only prioritized at the start of the project. Six practices were identified; face-to-face communication over written requirements, extreme prioritization, prototyping, reviews and tests, iterative RE and constant planning. Agile RE processes were found to blend together, instead of the traditional RE processes which are clearly defined separate steps. Cao et al. published another study as a follow-up on the study on agile versus traditional RE practices [23]. A case study was conducted on sixteen organizations. The same six common practices were identified, as well as seven common occurring challenges (Figure 6).

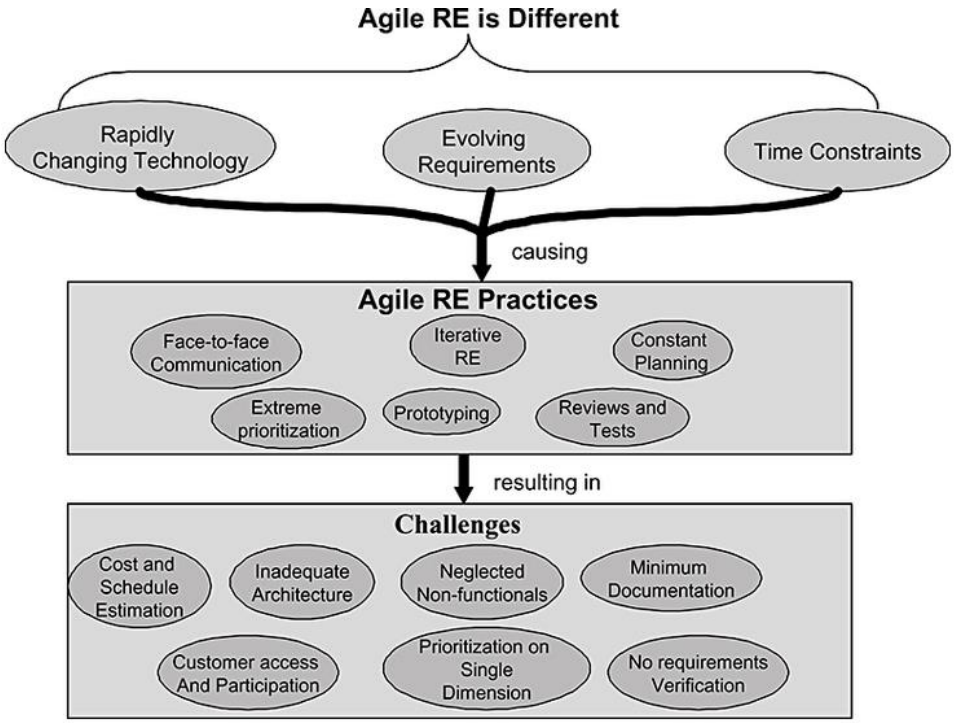


Figure 6: Common agile RE practices and resulting challenges identified by Cao et al. [23]

Heck & Zaidman performed a systematic literature review on quality criteria for agile requirements specifications [24]. The quality of the requirements is typically considered as an important factor for the quality of the end product, yet agile RE has not specified any form of standard to guarantee quality [24]. In an earlier study on quality frameworks, Heck & Zaidman found that experts in the agile domain agree on the importance of quality frameworks [25]. In their study, Heck & Zaidman identified 16 papers that discuss a total of 28 different quality criteria for agile requirements specifications. Based on this, they recommend three best practices;

- *Use a list of quality criteria*. Teams should create a list of quality criteria. There is no recommendation for a single list of criteria; teams should decide for themselves what criteria are useful in their context. Some example criteria that are mentioned are SMART (tasks should be specific, measurable, achievable, relevant and time-boxed) and INVEST (user stories should be Independent, Negotiable, Valuable, Estimable, Small and Testable).
- *Have a definition of ready*, which is a checklist to determine whether a user story is ready for developers to start implementing it.
- *Use a tool to manage requirements*. A tool should be used to store requirements, store traceability information and apply quality checks.

However, it is unclear what Heck & Zaidman mean with an agile requirements specification. In traditional RE, the RE specification would be the end product of the RE process, used as input to start developing the product. An agile method such as Scrum does not recognize an agile requirements specification artifact except for the

product and sprint backlog. It is also unclear how many teams actually invest time and effort into this agile requirements specification.

The work by Paetsch, Eberlein & Maurer compares traditional RE approaches and agile software development to determine possible ways how agile software development can benefit from RE methods [26]. Paetsch et al. recognize traditional RE in five categories; elicitation, documentation, validation, management and analysis. RE techniques that are important or synergize well with agile approaches are;

- *Customer involvement.* As encountered in other works, customer involvement is vital in agile development. For RE, direct customer involvement is important to elicit requirements with the right stakeholders.
- *Interviews*. These are the most commonly used technique for elicitation and stimulate face-to-face communication. Direct interaction is important for trust between the customer and the developer.
- *Prioritization is highly important* in agile development. The requirements with the most business value must be developed first. Whichever requirement provides the most business value is determined with prioritizations.
- *Joint-application development* is used to increase customer involvement and to gain an understanding of the system to be built.
- *Modeling* is used in agile development but usually does not become part of the documentation. Instead, models are quickly drawn on whiteboards or paper and then discarded after fulfilling their purpose. In agile development, requirements are often not modeled and instead quickly translated into running code.
- *Documentation is usually underperformed* in agile development. The scope often focuses on core aspects of the system and not all requirements will be included in the documentation.
- *Validation* is often done with review meetings and acceptance tests. By delivering working software, customers can immediately validate whether the requirement has been correctly implemented. Review meetings are useful for discussion about strengths and weaknesses of the software.
- Management of requirements should (traditionally) enable the tracing of changes made to requirements and rationale behind the changes. However, ensuring traceability costs time and money and it is not proven whether any economic benefit exists. This is why agile projects often do not base themselves on a fixed scope/price but on time and expenses.
- Observation and social analysis can be used for the elicitation of requirements, especially as a highly qualified user is not always the best person to elicit requirements from.
- *Non-functional requirements are often ill-defined* in agile development. Agile methods need to include more explicitly the handling of non-functional requirements. The authors do not have any recommendations to solve this issue.

Medeiros et al. proposed a model that improves the quality of software requirements specifications [27]. The most referenced models identified in their literature study were based on ISO-IEEE 830, which contains eight quality characteristics for a good specification; correctness, unambiguousness, completeness, consistency, prioritization, verifiability, modifiability and traceability. Other findings were that (1) software requirements specifications should not be used as a mechanism to validate requirements, (2) developers should prevent a fragmented software requirements specification, (3) experience of the developers with agile software development is crucial for success and companies need to promote best practices, (4) smaller requirements and sprints make for a simpler, clearer and readable software requirements specification and (5) specific tools can (and should) be used to support traceability of requirements and thereby facilitate the impact analysis of requirements change [27]. It remains unclear from this study how often teams produce a software requirements specification, as one of the more often identified challenges in agile RE is a lack of documentation.

There have also been extensive studies on the quality of the most common type of requirement used in Scrum; the user story. The quality user story framework (QUS) by Lucassen et al. attempts to identify quality violations in user stories based on 14 criteria [28].

Criteria	Description
Syntactic	
- Atomic	A user story expresses a requirement for exactly one feature
- Minimal	A user story contains nothing more than role, means and ends
- Well-formed	A user story includes at least a role and a means
Semantic	
- Conflict-free	A user story should not be inconsistent with any other user story
- Conceptually sound	The means expresses a feature and the ends expresses a rationale, not something else
- Problem-oriented	A user story only specifies the problem, not the solution to it
- Unambiguous	A user story avoids terms or abstractions that may lead to multiple interpretations
Pragmatic	
- Complete	Implementing a set of user stories creates a feature-complete application, no steps are missing
- Explicit dependencies	Link all unavoidable, non-obvious dependencies on user stories
- Full sentence	A user story is a well-formed full sentence
- Independent	The user story is self-contained, avoiding inherent dependencies on other user stories
- Scalable	User stories do not denote too coarse-grained requirements that are difficult to plan and prioritize
- Uniform	All user stories follow roughly the same template
- Unique	Every user story is unique, duplicates are avoided

Figure 7: The quality criteria influencing the quality of a user story [28]

Lucassen et al. [29] created a tool, AQUSA (Automatic Quality User Story Artisan), that is capable of analyzing a set of user stories and determine whether they are in accordance with 5 out of the 14 quality criteria;

- Well-formed
- Atomic
- Minimal
- Uniform
- Unique

The study by Lucassen et al. solely focuses on the quality of the actual user story sentence "as a … I want … so that …". The work by Heck & Zaidman focuses on the entire quality of the backlog item that contains the user story [25]. It proposes a quality assessment framework for just-in-time requirements. In this framework, a user story has three categories of quality criteria (QC1 = completeness, QC2 = uniformity, QC3 = consistency/correctness) in two dimensions. The dimensions are C (for '*on creation*') and J (for '*Just in time*') indicating at which point in time the criteria need to be met. For example, when a user story is first elicited and documented it is not necessary to have acceptance criteria yet. The quality criteria are;

- QC1.C: On creation, a user story needs to consist of the basic elements role, functionality and business value.
- QC1.J: Just in time, a user story needs to have additional elements which are acceptance criteria (and the team can decide on more elements they want to add)
- QC2. C: Stories are uniform by using the same template
- QC2.C: Any attachments to a user story use uniform and standardized modeling languages.
- QC3.J: A user story should conform to the INVEST standard

When evaluating the framework, practitioners preferred over using a 'gut-feeling'. A checklist aids in not forgetting criteria, basing opinions on facts, to educate other team members and to standardize the review process [25].

What remains unclear about the quality of requirements is whether there is a benefit to ensuring a certain quality standard. None of the studies discussed about quality frameworks provided any evidence or significant results on the importance of quality.

Requirements specification

As mentioned before, user stories are by far the most popular format in which requirements are documented. A user story is a description of a feature written from the perspective of the person that requires it [13]. A user story is usually part of an epic story. User stories are often written in a certain format/template, for example the Connextra template; "As a [type of user], I want [goal], (so that [some reason])" [20]. In this way, user stories answer a Who (who is it for), What (does he/she want) and Why (why does he/she want it). Regardless of the template, a user story always includes a role (the who), a means (the what), an (optional) end (the why), and a format (figure 8) [28].

In a study conducted on user story practices, 59% (N=182) of practitioners were found to use the Connextra format [20]. For quality frameworks, 23% of respondents used the INVEST framework, while 33% had their own quality criteria and 40% did not check their stories on quality. On templates (such as the Connextra template) practitioners noted that it does not really matter which template is used, as long as the team uses a single one that everyone agrees on. The alignment of a team relying on a standardized structure improves work productivity and quality [20]. Also, while templates can allow

for the removal of the benefit part (or Why part) practitioners state that it is an essential part of the user story.

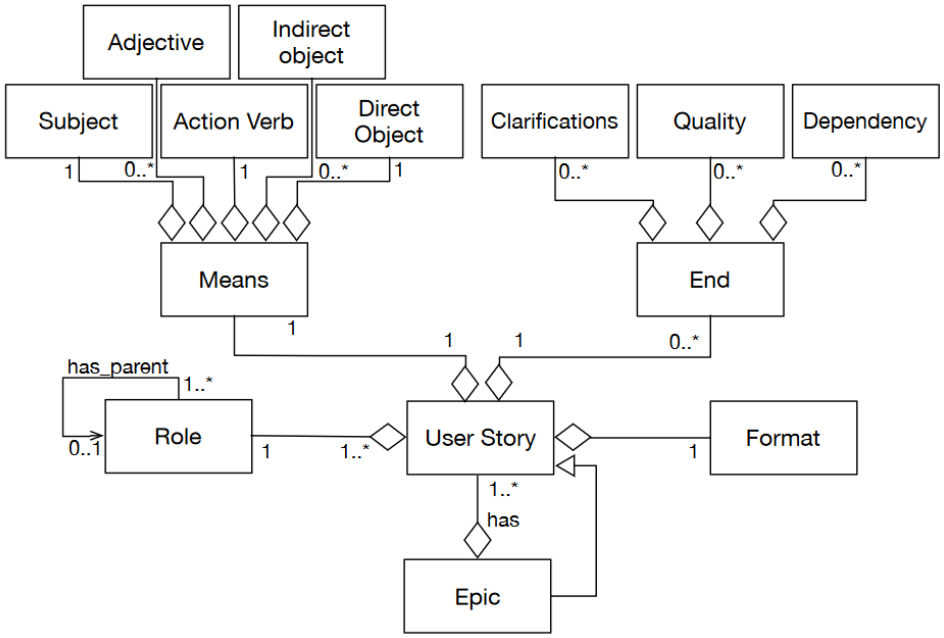


Figure 8: Conceptual model of a user story [28]

Challenges in Agile RE

Several studies investigated challenges that are present in agile RE. Inayat et al. identified several challenges of agile RE and practices to solve them [18]. Firstly, minimal documentation (one of the principles of the agile manifesto) often causes problems. Without altering Scrum, only user stories and backlog items are documentation of the product, causing traceability issues. Secondly, customer availability is advocated by agile methods but in reality can be scarce. Thirdly, budget and schedule estimation can be difficult due to the volatility of the requirements; some may be added, some may be dropped, and nobody knows when either will occur. While effort estimation techniques are used in Scrum, such as planning poker, the volatility of requirements still makes it hard to make correct estimations. The team discusses user stories and everyone estimates the amount of work a story is likely to be. Fourthly, inappropriate architectures created by teams in earlier iterations can become problematic in later iterations due to new requirements. Fifthly, non-functional requirements are considered to be a major challenge for agile RE. Finally, customer inability and agreement can occur when the customer has trouble with decision making and is incompetent in the domain knowledge.

The study by Medeiros et al. [19] also discusses challenges in agile RE. They found two main challenges; interaction with the client and change management. The agile principle of 'teams quickly adapt to change' is something not experienced by the companies included in the study. Also, the principle of 'continuous interaction with

the customer' is hard to achieve. This challenge was also mentioned by Inayat et al [18].

The mapping study performed by Heikkilä et al. found that the user story format is often insufficient [12]. Only simple, customer visible functional requirements can be described with user stories, and the lack of additional documentation causes issues with traceability. Furthermore, requirements prioritization is often based on the requirements that offer the most business value for the customer, which might cause system architecture/improvements/non-functional requirements to be ignored [12].

Another challenge was identified in the work of Cao & Ramesh (2008). They found that non-functional requirements (NFR) are often neglected over functional requirements, which often led to redevelopment and bottlenecks [22]. In a follow-up study, Cao et al. identified another six issues; cost and schedule estimation, customer access & participation, inadequate architecture, minimum documentation, lack of requirements verification and prioritization on single dimensions (focus on business value) [23].

The literature study by Elghariani & Kama confirms most of the aforementioned challenges [30]. Lack of documentation, client availability, incorrect budget and time estimations, inappropriate software architectures and ignoring non-functional requirements are once again mentioned as challenges in agile RE.

Hall, Beecham & Rainer performed an empirical analysis on twelve companies to discover requirements engineering problems [7]. The authors found that the RE process is often hindered by organizational issues like lack of skill and poor staff retention. The authors conclude that most requirements problems are organizational rather than technical and that there is a relationship between companies' maturity and patterns of requirements problems. Process assessment to achieve higher maturity seems to benefit companies even when they are very immature.

In 2017, the Naming the Pain in Requirements Engineering (NaPiRE) initiative published a qualitative analysis of RE problems for 228 companies in over 10 countries [31]. The ten most-cited problems can be found in table 3 below.

RE problem	Total	Cause for project failure
Incomplete and/or hidden requirements	109 (48 %)	43
Communication flaws between project team and customer	93 (41 %)	45
Moving targets (changing goals, business processes and/or requirements)	76 (33 %)	39
Underspecified requirements that are too abstract	76 (33 %)	28
Time boxing/Not enough time in general	72 (32 %)	24
Communication flaws within the project team	62 (27 %)	25
Stakeholders with difficulties in separating requirements from known solution designs	56 (25 %)	10
Insufficient support by customer	45 (20 %)	24
Inconsistent requirements	44 (19 %)	15
Weak access to customer needs and/or business information	42 (18 %)	16

Table 3: The 10 most cited problems in agile RE [31]

Fernández et al. performed a cause-effect analysis of the three most critical RE problems that they identified. The ten most reported problems and their causes can be seen in figure 9 below. Figure 9 shows that the biggest cause of problems is the lack of RE related experience. Therefore, a maturity model might be a good way to suggest improvements to those who are unaware of techniques to implement or use.

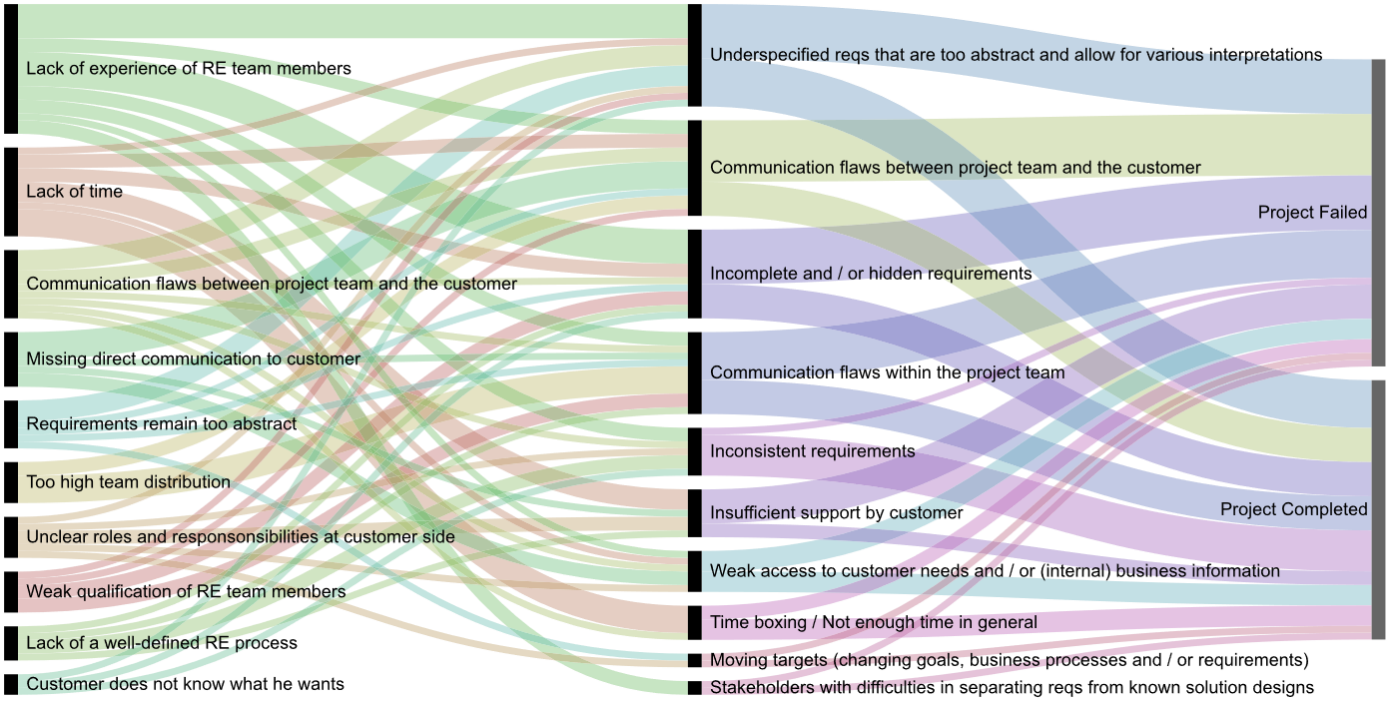


Figure 9: The relation of the top 10 causes, top 10 problems and the project impact [31]

3.1.4. Conclusion

This section discussed common RE practices in Scrum, as well as the benefits and challenges of applying RE in Scrum. Traditional RE used to be performed prior to software development, but in Scrum RE and development are practiced during the entire lifetime of the product/system. Agile RE is characterized by evolving requirements and constant reprioritization. A key concept of Scrum is that teams are self-organizing and should apply best-practices where needed. The literature clearly shows that it is unclear which RE practices are 'best-practice', which is reflected by the fact that many studies looking into RE practices used by Scrum teams had low frequencies and a high variance of different practices. In agile RE, several studies discuss the quality of agile requirements and techniques/frameworks to ensure quality. Quality frameworks such as INVEST are used in some form in a majority of the teams analyzed in studies. The user story is by far the most commonly used technique to document functional requirements, although the templates used differs. While agile RE has benefits, such as lower times to market, better customer involvement and better suited for dealing with volatile requirements, there are also significant challenges noted by several studies. These studies often reported similar problems such as a lack of customer involvement (or no onsite customer), negligence of non-functional requirements, lack of documentation, insufficiency of the user story format to capture all relevant information and estimation issues. As one study noted, there is a relationship between a team's maturity and patterns of requirements problems, making maturity a beneficial quality to strive for [7]. Because the literature is at many

times vague on how certain processes and techniques are performed, the literature review will be combined with a case study, which is discussed in chapter 4.

3.2. Defining maturity

This section is concerned with research question two; '*How can RE maturity in Scrum be defined?*'. Now that an understanding of common RE practices in Scrum is defined, we must discover what maturity in RE practices actually means. This needs to be done based on expert opinions and best practices according to literature. In section 3.2.1. a brief introduction of maturity modeling is discussed. Section 3.2.2. discusses maturity models that were designed for agile environments. Section 3.2.3. discusses maturity modeling for requirements engineering.

According to the mapping study by Wendler, the purpose and definition of maturity models can often differ among publications. Wendler gives the following definition and purpose of a maturity model;

'The purpose of models dealing with maturity is to outline the conditions when certain examined objects reach the best (perfect) state for their intended purpose. In addition, there has to be a "final" state of maturity (fullness of growth) in which no further development is possible … maturity models describe and determine the state of perfection or completeness (maturity) of certain capabilities' [32].

As maturity models describe and determine the state of perfection or completeness of certain capabilities, the final states in these models can be considered best practices. Best practices are defined by the Oxford dictionary as;

Commercial or professional procedures that are accepted or prescribed as being correct or most effective.[1]

However, this would be an incorrect definition of the maturity we aim to achieve in our model. There exist many practices that are generally acknowledged as good or best practices, but they can still be useless for a team in a certain context. In the context of this study, a mature practice can be *any* practice as long as it is justified in the context in which it is applied. As one study noted, there is a relationship between a team's maturity and patterns of requirements problems, making maturity a beneficial quality to strive for [7]. Yet the maturity to strive for can be different depending on the project context.

[1] https://en.oxforddictionaries.com/definition/best_practice

3.2.1. Maturity modeling

Maturity modeling has its roots in the field of quality management [33]. The most well-known maturity model is probably the capability maturity model (CMM) [34]. The CMM was published in 1993 and its goal was to help organizations improve their software process by assessing processes in order to find possible areas for improvement. The CMM focuses on the maturity of organizational processes. As most developed maturity models are based on the CMM, this section explains the concept and the way a maturity model works.

In an immature software organization, software processes are generally improvised by practitioners and their management during the course of the project [34]. Processes are usually not specified and if they are, not rigorously adhered to. Contrasting, a mature organization has clear standards for processes and they are accurately followed. Communication, roles and responsibilities are clear throughout the organization and resources are spent on continuous improvement [34].

The CMM provides a framework for organizing steps towards maturity into five levels, creating an ordinal scale for measuring a process's maturity. A maturity level is a '*well-defined evolutionary plateau toward achieving a mature software process*' [34]. According to the CMM, a higher process capability level results in better performance. Although an organization can have different processes on a different maturity level, a process usually needs to be improved on a level-by-level basis for the improvement to be successful. This means that 'skipping' a level should be avoided. A maturity level consists of several key process areas. A key process area is a cluster of related activities that are considered important to perform in order to reach a new level of maturity.

The CMM publication provides several reasons for its usefulness but for this study the main reason is that a maturity model can be used to assess teams to identify strengths and weaknesses. It can then guide a team on what steps need to be taken to improve on these identified weaknesses.

Since the publication of the CMM, the research and development of other maturity models has reached over 20 different domains [32]. The mapping study by Wendler found 237 articles studying maturity models [32]. Of these, 56% (108) were concerned with the development of new maturity models. According to Wendler, the high amount of new maturity model development is an issue, as authors often do not consider previous work. Authors intending to develop new maturity models should carefully check if there are other solutions available. Wendler notes that oftentimes authors do not review existing models that might serve their purpose. In other cases, Wendler states that authors sometimes 'referred to other models (mostly CMMI) and transferred their structure and content without checking if this makes sense for their intended purpose'. Wendler gives the advice that authors of new maturity models first have to analyze if there are existing models and then carefully check their applicability. This is important for the quality and relevance of the new model and to

prevent development expenditure in case suitable models already existed. Furthermore, maturity modeling studies often have issues regarding the validation of their designed maturity model; over half of the identified studies (52%) did not validate their models. Wendler states that there is no clear or ideal method to validate a maturity model. In most studies case studies and interviews were used. In about a third of the studies, qualitative surveys were included to validate the designed artifact.

Maturity model types

In this study, we consider two different types of maturity models; *fixed-level* maturity models and *focus area* maturity models. Most maturity models are fixed-level maturity models, e.g. the CMM. Fixed-level maturity models often consist of several levels of maturity and each level is associated with several processes that need to be implemented/conformed to. A limitation of these models is that dependencies between processes are unclear and therefore provide little guidance on the order in which processes should be implemented [35]. Additionally, fixed-level maturity models do not recognize that different levels of maturity can occur for different processes (e.g. some activity might be associated with maturity level 1, yet another activity that is also performed is associated with maturity level 3).

The focus area maturity model is also referred to as a maturity grid in other studies [33,36]. A maturity grid consists of activities and a maturity level for each activity. The difference between the two models is that fixed-level models have global descriptions of maturity are described for each level, there are no individual descriptions for each activity at each maturity level, whereas a maturity grid identifies process areas and activities with their individual maturity level.

As stated earlier, many maturity models have already been created. It is important to analyze previously created models before creating new models as they may already serve the purpose of assessing RE practices in Scrum. As stated by Wendler, it is important to check if there are existing suitable maturity models which can be (partially) re-used, and whether these models have been validated [32]. For the purpose of analyzing previously created models, this chapter discusses maturity models and related studies found in scientific literature that are related either to Scrum/agile (section 3.2.2.) or requirements engineering (section 3.2.3.).

3.2.2. Maturity and Scrum/Agile

Maturity models like the CMM and agile can, according to some, be described as oil and water; they do not like to mix [37]. This dilemma is well demonstrated by the following quote of Schweigert et al.

"From the agile camp you can hear someone say that CMMI is the big American waterfall model monster, and is outright contra-productive to agile methods. From the CMMI camp you can hear someone to say that agile methods are hackers from hell that uses the agile paradigm to enjoy anarchy with no rules. You can also hear some say that agile works best in CMMI level 5 companies. The context of the dilemma, however, is slightly awkward. CMMI describes characteristics of good development practices, and agile is a lifecycle concept. So from a meta point of view they can easily co-exist" [38].

Despite the fact that, on paper, maturity modeling and agile seem to contradict each other to some extent, there has been plenty of research on the topic [37,39–42]. The study by Marçal et al. mapped CMMI project management process areas to Scrum to check whether Scrum satisfies any of the CMMI areas [40]. It was concluded that not all processes of CMMI are satisfied in Scrum and that organizations looking to achieve high CMMI maturity should look for other methods for project management. A trend that coincides with this finding was found in the literature review of agile and maturity model research by Henriques & Tanner [43]. Figure 10 below demonstrates how at first, the majority of research on the topic of CMMI and Agile was focused on combining agile and CMMI, whereas this trend has turned around and now studies focus more on studying agile maturity without the use of CMMI.

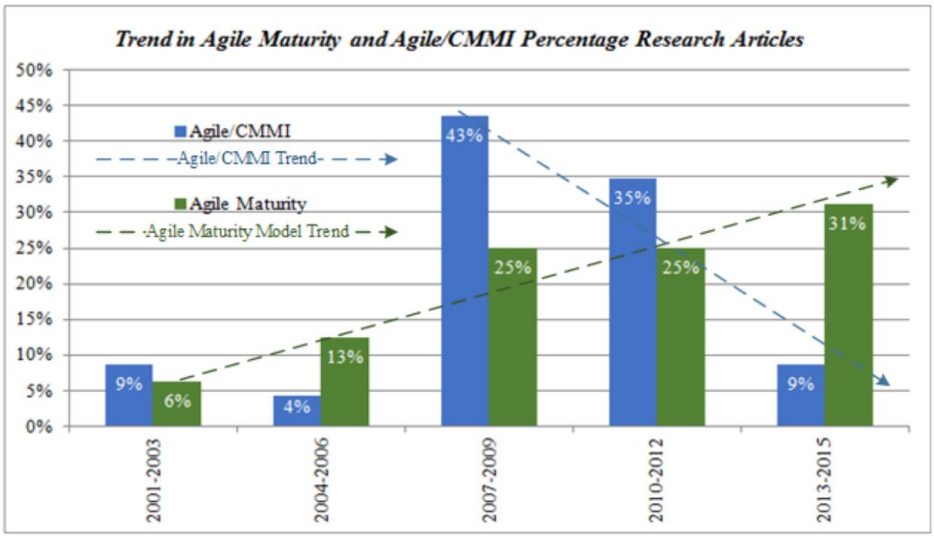


Figure 10: Trend in agile Maturity and Agile/CMMI percentage research articles [43]

Despite the decline in popularity of CMMI/agile research articles, the CMMI Institute a report on improving agile performance with CMMI [41]. According to the report, there is value in both paradigms and they are not incompatible at all. The incorporation of CMMI goals into the project activities of agile teams can help make these teams more mature and/or capable at handling the continuity of projects [42].

The agile maturity model by Petal & Ramachandram (figure 11) is a fixed-level maturity model and consists of five different maturity levels, each with its own goals [44]. Similar to the CMM, at the initial level (1) agile processes are (barely) defined and the development environment is unstable. The second level (2) explored indicates a more structured and complete software development practice. One of the goals of this level is 'improved agile requirements'. According to the authors, at this level organizations should have specified policies and resources for requirements engineering. Development should follow story cards (cards with user stories written on them) driven development, which is a very common practice in Scrum. For each maturity level, the activities of the associated level that were RE related were analyzed. The results for this can be found in table 3 below.

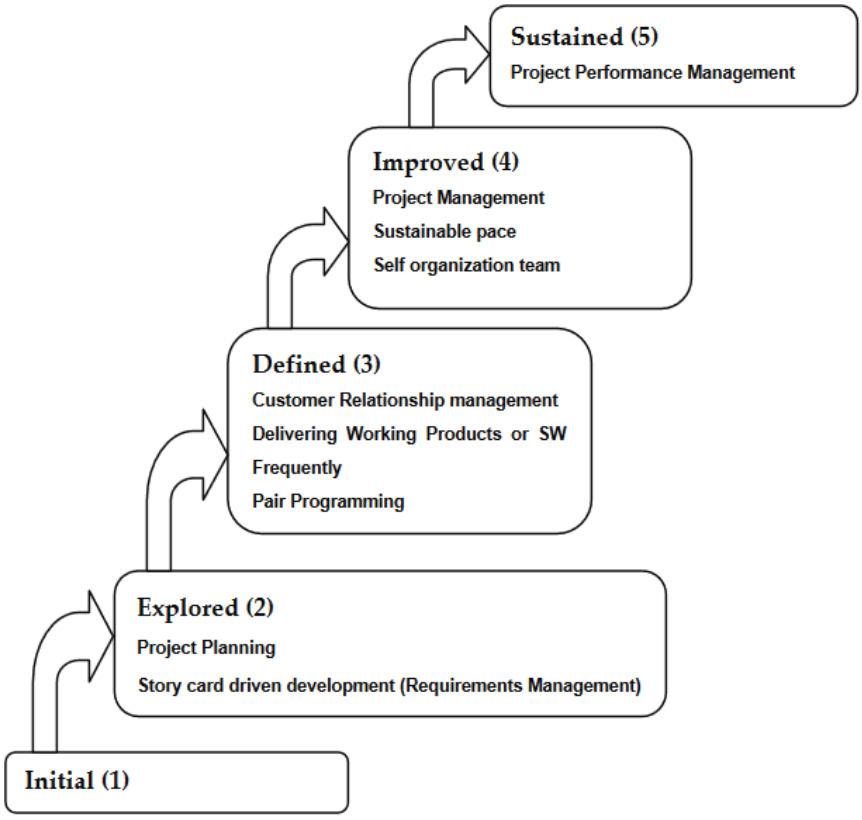


Figure 11: The agile maturity model by Petal & Ramachandram [44]

Although Petal & Ramachandram defined the tasks/processes in table 3 to measure whether the goal is achieved, their model has issues. The activities (and measures for

whether they are achieved) are rather ambiguous. For example, ‘there is a plan to manage story cards’ is very unclear on what a plan is, how detailed it should be, et cetera. Another example would be the criterium ‘there is commitment to each story card’. The authors do not specify any definition or metrics for commitment. This issue was also mentioned in the work by Yin; the model lacks measurement goals to monitor and evaluate the performance of the practices listed in each level [45]. As most RE related activities (15 out of 20) are associated with maturity level 2, this seems to suggest that high maturity in Scrum can only be achieved by having a solid, well-defined and repeatable set of RE related activities as a building point. Another limitation of this model is that it has not been validated in an industrial setting and therefore is only based on literature.

Maturity level	**Key Process area**	**RE related criteria**
2	Project planning	The planning game is used to create project plans
		Planning work is based on the business value
		Estimation is done by the developers
	Story cards driven development	User stories are written to solve requirements engineering
		Unique story card structure is defined
		A plan exists to manage story cards
		Obtain an understanding of the story cards
		Obtain commitment to story cards
		Identify inconsistency between story cards and customer’s original requirements
		Elicit customer needs from onsite customer
		Manage requirements change
		Story cards are written from multiple viewpoints
	On-site customer availability	Onsite customer is a domain expert
		Obtain commitment to story cards from onsite customer

		Customer is always available
3	Mutual interaction	Story cards are written by the collaboration of onsite customer and developers
		Communicate results through acceptance testing
	Implementation and interaction	The list of user stories is reprioritised based on an updated evaluation of the project at each iteration boundary
4	-	No RE related activities
5	Project planning	Each story card comes with acceptance tests
	Story cards driven development	Implementation of acceptance tests with story cards

Table 3: RE related activities, their associated focus area and maturity level based on Patel & Ramachandran [44]

In their study, Ozcan & Demirörs assessed several agile maturity methods [46]. They analyzed five models, of which we already discussed and the model by Patel & Ramachandran [44]. The study found that most models were inconsistent with their abstraction levels, having some concepts at process level and others at practice level. Furthermore, most models lack metrics that can be used to determine whether a goal is achieved. Out of the five models, the best-assessed model was the agile adoption model by Sidky, Arthur & Bohner [47].

The model by Sidkey et al. is not only a fixed-level maturity model but expands on it using a measurement index, the Sidkey Agile Measurement Index (SAMI) [47]. This measurement index consists of four components;

- Agile levels; these are the standard 1-5 fixed levels.
- Agile principles; these are principles advocated by agile methods to ensure that processes are agile.
- Agile practices; this is a list of 40 activities and techniques, distributed over the agile levels and principles.
- Indicators; these are questions an assessor can use to determine whether an organization is ready to introduce a new practice.

Of these 40 activities, three are RE related. The activity ‘evolutionary requirements’ is associated with agile level 2 and the agile principle of *Embracing change to deliver customer value*. At agile level 4 and the agile principle of *technical excellence* is the practice of using user stories.

The work by Fontana et al. discusses what defines maturity in agile software development [48]. By abstracting the details of 14 analyzed agile maturity models they

concluded that agile maturity can be characterized as;

- Practices adoption
- Continuous improvement
- Sustaining agility
- Project performance
- Highly productive teams

Additionally, the authors discuss which agile practices can be considered mature. The authors conclude that their results uncover the diversity of possibilities to reach agile maturity. There are different structures, assessment modes, underlying concepts of maturity and mature practices.

3.2.3. Maturity and RE

RE has traditionally been a well-documented process resulting in an expansive specification and there are several works discussing maturity modeling for RE. Niazi, Cox & Verner published an adapted measurement framework (REMMF) for assessing the maturity of the requirements engineering process [49]. The structure of REMMF can be seen in figure 12 below.

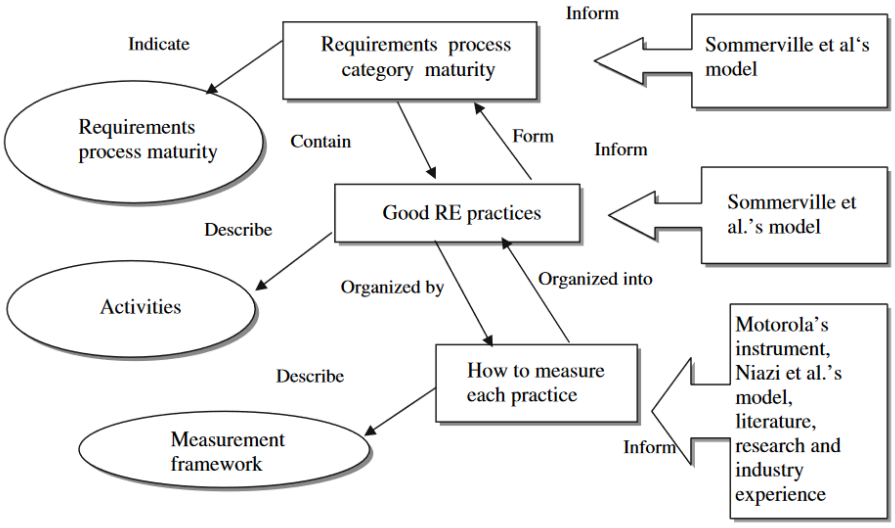


Figure 12: Structure of REMMF includes reasoning for the categories (focus areas), practices and ways to measure each practice [50]

This assessment model categorized 66 RE practices over 7 categories. Niazi et al. based their RE process maturity categories and good RE practices on the work by Sommerville [51];

1. Requirements documentation: practices related to structuring and organizing the requirements documents.
2. Requirements elicitation: practices to help discover the requirements from stakeholders, application domain and organizational environments.

3. Requirements analysis and negotiation: practices to help identify and resolve incompatibilities and missing information problems.
4. Describing requirements: practices for effectively writing requirements.
5. System modeling: practices for the development of models in order to better understand requirements
6. Requirements validation: practices to help establish formal validation procedures relating to incompleteness, inconsistency or incompatibility problems.
7. Requirements management: practices for requirements management.

Based on our findings of section 3.1, these categories have overlap between them or do not exist in agile RE. For example, requirements documentation does not comprise many activities in agile RE, as the only place requirements are usually documented is the product backlog. System modeling is also frequently omitted by development teams as teams tend to not commit resources to such activities.

Niazi et al. tries to measure the RE processes by adopting a measurement framework originally developed by Motorola, yet tailored towards RE [52]. This measurement works by awarding points for a process on three dimensions; approach, deployment and results. For every dimension, a process can receive a score of poor (0 points), weak (2 points), fair (4 points), marginally qualified (6 points), qualified (8 points) and outstanding (10 points). Figure 13 below shows an example of scoring the category 'requirements documents practices', which contains 8 processes. Each process is awarded a number of points, the total number of points is cumulated and divided by the number of practices in the category.

ID	Type	Requirements documents practices	0	1	2	3	4	5	6	7	8	9	10	NA
RD1	Basic	Define a standard document structure						√						
RD2	Basic	Explain how to use the document						√						
RD3	Basic	Include a summary of the requirements		√										
RD4	Basic	Make a business case for the system				√								
RD5	Basic	Define specialized terms						√						
RD6	Basic	Make document layout readable						√						
RD7	Basic	Help readers find information						√						
RD8	Basic	Make the document easy to change				√								

The 3-dimensional overall score: (5 + 5 + 1 + 3 + 5 + 5 + 5 + 3/No of practices) → 32/8 = 4

Figure 13: Grading example in the REMMF model [49]

A practice in the model of Niazi et al. can be one of three categories; basic, intermediate and advanced. These categories were based on the work of Sawyer, Sommerville & Viller [53]. A basic practice is often relatively simple and cheap to introduce and use. They provide a foundation for repeatable RE processes. An intermediate practice is more complex and leads to a more defined RE process, costing more and taking more time to implement. Finally, advanced practices are intended to support the continuous improvement of an RE process which consists of advanced

methods and requires specialist expertise. Some of these practices are very costly to introduce, others are relatively cheap but may require organizational change.

The Requirements Engineering Process Maturity (REPM) model designed by Gorschek et al. [54]. The model is a fixed-level maturity model with 5 levels and three key process areas; requirements management, requirements elicitation and analysis/negotiation. The model was applied on 4 different projects at different companies. It was found that none of the teams were able to achieve even maturity level 1, since they did not meet all criteria. However, some teams did achieve activities that belonged to level 2, 3 or even higher. This indicates that the fixed-level maturity model does not paint a true picture on the maturity of a team and a focus-area maturity model might be more appropriate.

The problem with the maturity models discussed above is that they are relatively old (all studies were published before 2010) and focus on the more traditional RE process instead of the agile RE process this study focuses on. As discussed earlier in the introduction of this chapter, agile activities can appear to be done in an ad hoc, chaotic fashion while in fact they are very repeatable or appropriate, albeit with a lack of extensive documentation. Applying maturity models that were not specifically designed with agile practices in mind will likely result in low maturity scores, even though the development effort or performance of the team might be going very well. For example, applying the model of Niazi et al. on a scrum project team will likely result in a lower maturity score than a team that performs traditional RE as the model was not constructed with agile principles and practices in mind.

In section 3.2.2. the process improvement framework by Patel & Ramachandran was discussed. This framework *is* aimed at agile maturity. Another model published by these authors which is specifically aimed at agile RE practices is the story card maturity model [55]. A story card is very similar to a backlog item, as it stores some requirement and other information like business value or acceptance criteria. This fixed-level maturity model consists of four levels; initial (1), repeatable (2), defined (3) and managed (4). There are 19 key process areas and 71 areas for improvement. While this makes for an extensive model, a lack of validation and evaluation of the model tells us little on the accuracy/usefulness of the model. The key process areas can be seen in figure 14 below.

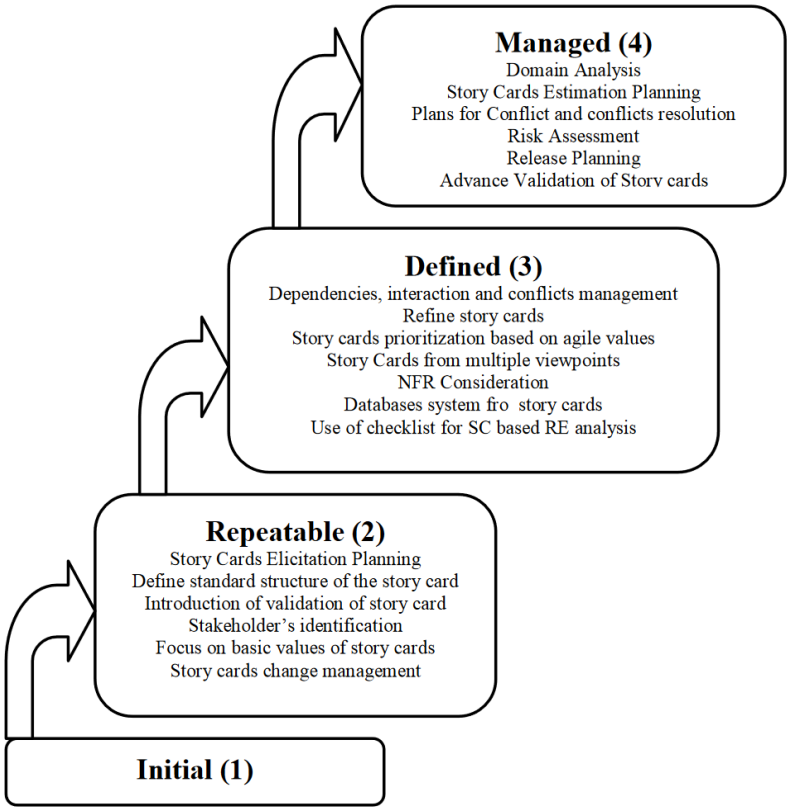


Figure 14: The story card maturity model levels and key process areas [55]

Searching for *best* practices in agile RE resulted in surprisingly little literature on this topic, and when there is it is often written for traditional RE practices. One study by Patel & Ramachandran discusses best practices for agile RE [56]. The guidelines are only for story cards, which is an artifact mostly used in extreme programming. However, a story card shares a lot of similarities with a backlog item. They often both contain a user story describing the functionality, a workload estimation, some priority, acceptance and test criteria and additional details. Therefore, the guidelines proposed for story cards should also be applicable for (functional) backlog items. Patel & Ramachandran propose a total of 31 unique guidelines that should be applied. Unfortunately, this article is hidden behind a paywall and was inaccessible.

The work of Hofmann & Lehner investigates how traditional RE contributes to the success of software projects [5]. They make a recommendation on 10 best practices which can be seen in the table below. Although these are for traditional RE they may still hold relevance for agile RE. The recommended practices, cost of introduction, cost of application and key benefit are described in table 4.

Best practice	Cost of introduction	Cost of application	Key benefit
Involve customers and users throughout RE	Low	Moderate	Better understanding of real needs
Identify and consult all likely sources of requirements	Low/Moderate	Moderate	Improved requirements coverage
Assign skilled project managers and team members to RE activities	Moderate/ High	Moderate	More predictable performance
Allocate 15 to 30 percent of total project effort to RE activities	Low	Moderate/High	Maintain high-quality specification throughout the project
Provide specification templates and examples	Low/Moderate	Low	Improved quality of specification
Maintain good relationships among stakeholders	Low	Low	Better satisfy customer needs
Prioritize requirements	Low	Low/Moderate	Focus attention on the most important customer needs
Develop complementary models together with prototypes	Low/Moderate	Moderate	Eliminate specification ambiguities and inconsistencies
Maintain a traceability matrix	Moderate	Moderate	Explicit link between requirements and work products
Use peer reviews, scenarios and walk-throughs to validate and verify requirements	Low	Moderate	More accurate specification and higher customer satisfaction

Table 4: Best practice suggestions for traditional RE [5]

Although not a maturity model, the International Requirements Engineering Board (IREB) published a handbook on their standards for agile RE [57]. The handbook presents five categories of topics that are important in agile RE;

- Prerequisites for successful system development, such as balancing vision and/or goals, stakeholders and scope of the system.
- Functional requirements
- Quality requirements
- Estimating, ordering and prioritizing of requirements
- Scaling RE for distributed teams

This handbook is a very useful source for the to-be-developed maturity model, as it not only explains what processes are important for effective agile RE but also discusses several different techniques that can be used. For example, documenting the system boundary can all be done with either a context diagram, natural language

documentation, use case diagrams, story mapping or a combination of these. The handbook does not specify which of these is better, but that one of these should be used to achieve a correctly documented system boundary.

3.2.4. Concluding maturity models

Maturity models are a useful tool to assess the current practices in a team, to compare teams with one another and to guide teams toward best practices. There are several existing maturity models that focus on agile or on requirements engineering, but none combine the two topics. Some models that focus on agile do incorporate some topics/techniques that relate to requirements engineering. The maturity models that focus on requirements engineering are all (except for one) focused on traditional RE and do not take into account how agile changed RE, making them inapplicable to agile RE. Furthermore, all models we encountered (both agile and RE focused) are fixed-level models. These are limited due to the fact that growth is not as linear as they describe and dependencies between capabilities are unclear. Nevertheless, the fixed-level models likely include practices and criteria that will be applicable to the matrix as well.

4. Case studies to discover practices

To discover which and how the RE methods mentioned in the previous chapter are used/adapted in real-world software development teams several case studies are performed. This chapter discusses two different case studies that were performed:

- A multiple case study at software development teams from Info Support
- A multiple case study at other companies performed by students for a project at the requirements engineering course at Utrecht University.

4.1. Case study Info Support

The case studies are performed at teams from Info Support. From these teams, data collection is performed. In order to gain an understanding of the processes performed, semi-structured interviews are performed on team members of software developments who are responsible for or involved with the RE process. These can be business analysts, scrum masters and product owners. Finally, we want to discover if the teams face the same challenge as described in the literature and how they mitigate these challenges. The interview template that was used to guide the interviews can be found in appendix A. This template was based on the template that was used to guide students of the RE course during their case study (section 4.2) so the findings can be merged and compared.
The case study serves to gain an understanding of several topics;

1. The processes that are applied in a team and the rationale behind them.
2. The operational context of the team
3. The artifacts produced by and used in (1).

Data was collected using expert interviews. Expert interviews are used to understand the RE processes used by Scrum teams. Using a semi-structured template, we want to gain a detailed understanding of the techniques they apply and the rationale behind the choices that are made. This template was based on the template used by the students who performed the case study in section 4.2 and can be found in appendix A.

The case study was performed at software development teams from Info Support BV which is the company that the author is doing his research internship at. Info Support develops tailor-made software for customers. Often, employees of Info Support are mixed with employees of the customer to form a project team at the customer (onsite). In other cases, teams develop from the head office using product owners by proxy (surrogate customers). Info Support has around 500 employees divided over 5 business units; healthcare, finance, industry, mobility & agriculture/food/retail. For this case study, we looked at a total of 4 teams that were sampled based on convenience sampling. The teams had to adhere to the same requirements as the teams in section 4.2. Table 4 below shows the participating experts and their experience.

ID	Function(s) / Roles	Experience in current organization	Experience in current function
CaseExpertA	Agile Coach	13	13
CaseExpertB	Business analyst	0.5	0.5
CaseExpertC	Product owner/business analyst/ lead dev	1.5	7
CaseExpertD	Product owner/ Business analyst	5	5

Table 5: Participating experts in the case study

4.2. Case studies at other companies

Similar to the case study at Info Support, several other case studies were performed by students of the requirements engineering course at Utrecht University. A group of students investigated user story practices. Divided into groups of 2, they analyzed 28 software development teams at 27 different companies. Participating companies adhered to the following requirements;

- The company uses user stories to document functional requirements.
- The company creates one or more software products or provides software development services.
- The company applies an agile software development method, preferably Scrum, in a team-based fashion.

The students interviewed 2 persons per team, after which they generated a report based on their findings. This report included three categories;

- Pre-user story practices. This contains practices that are used to create a goal, vision, epic stories, roadmap generation.
- Specification practices. This consists of practices that are used to elicit and document user stories, as well as tools that are used, templates used for stories, quality criteria and acceptance criteria.
- Post-user story practices, consisting of practices such as validation of the user stories, prioritization techniques, traceability of user stories.

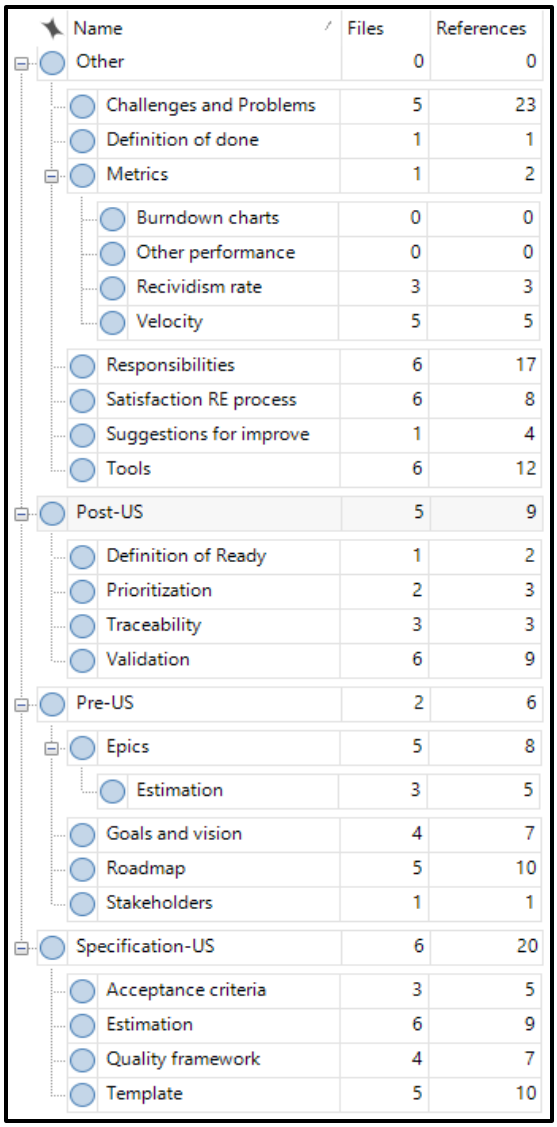
Additionally, three kinds of metrics were collected from the teams (if possible);

- Release burndown charts. A common way to measure the work still to be done in a sprint. A burndown chart contrasts the planned work and work still to be done.
- Recidivism rate, which is the number of stories that remain unfinished in a sprint and which go back into the sprint backlog.
- Velocity, which is the sum of story points completed in a sprint. This is often used as an indicator of the amount of work a team does.

4.3. Method of analysis

The case studies both at Info Support and other companies have been combined into a dataset. This dataset consists of 32 teams in total. The data was analyzed using Nvivo 12, which is computer software for qualitative data analysis.

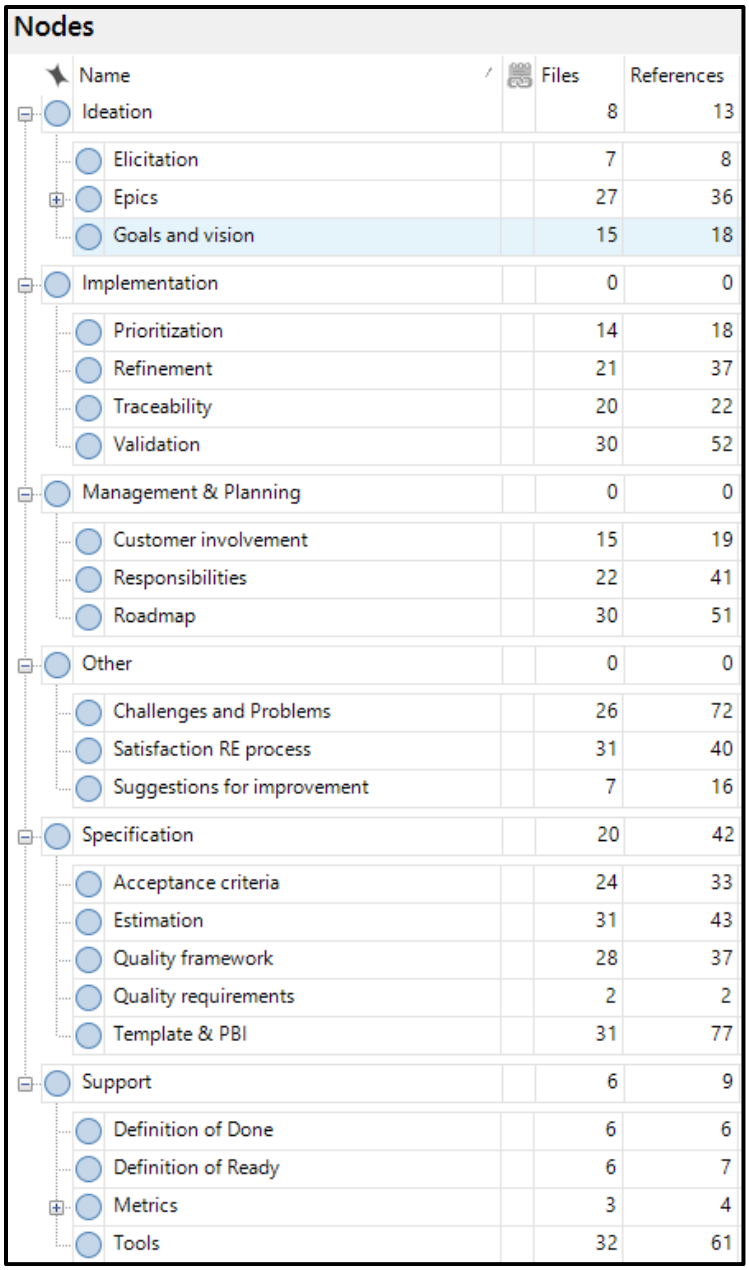
In order to make sense of all unsorted data in the reports, Nvivo provides the node system. A node is a collection of references about a specific theme. Using nodes, text (or other information elements) can be encoded in an appropriate node. Nodes can also be ordered in a certain hierarchy. The set of nodes started with four categories which were based on the interview template (Pre-US, Specification-US, Post-US and Other). During the analysis of the reports new nodes can be created under one of these category nodes. For example, information about elicitation techniques was entered in the *elicitation* node and categorized under the *Pre-US* node, as elicitation takes place before User stories are documented. The figure below shows an example of the node hierarchy after the analysis of several reports. Eventually, no new topics will emerge and all relevant information can be stored in one of the nodes in the node structure.



Name	Files	References
Other	0	0
Challenges and Problems	5	23
Definition of done	1	1
Metrics	1	2
Burndown charts	0	0
Other performance	0	0
Recividism rate	3	3
Velocity	5	5
Responsibilities	6	17
Satisfaction RE process	6	8
Suggestions for improve	1	4
Tools	6	12
Post-US	5	9
Definition of Ready	1	2
Prioritization	2	3
Traceability	3	3
Validation	6	9
Pre-US	2	6
Epics	5	8
Estimation	3	5
Goals and vision	4	7
Roadmap	5	10
Stakeholders	1	1
Specification-US	6	20
Acceptance criteria	3	5
Estimation	6	9
Quality framework	4	7
Template	5	10

Figure 15: An example of nodes encoded in Nvivo divided among several categories

After all reports have had their relevant information encoded in nodes the resulting node structure, the number of reports referenced per node and the number of references per node. The overall categories of Pre-Us, Spec-US, Post-US and Other have been changed to Ideation, Implementation, Management & Planning, Support & Other. This can be seen in figure 16 below.



Nodes

Name	Files	References
Ideation	8	13
Elicitation	7	8
Epics	27	36
Goals and vision	15	18
Implementation	0	0
Prioritization	14	18
Refinement	21	37
Traceability	20	22
Validation	30	52
Management & Planning	0	0
Customer involvement	15	19
Responsibilities	22	41
Roadmap	30	51
Other	0	0
Challenges and Problems	26	72
Satisfaction RE process	31	40
Suggestions for improvement	7	16
Specification	20	42
Acceptance criteria	24	33
Estimation	31	43
Quality framework	28	37
Quality requirements	2	2
Template & PBI	31	77
Support	6	9
Definition of Done	6	6
Definition of Ready	6	7
Metrics	3	4
Tools	32	61

Figure 16: The final node structure, number of files referenced per node and number of references per node from 32 teams

Figure 17 below shows how nodes are encoded. Figure 18 demonstrates what a node looks like in Nvivo. The analysis of a node was done manually and was done by;

- Identifying common patterns of practices
- Creating a logical order of process improvement

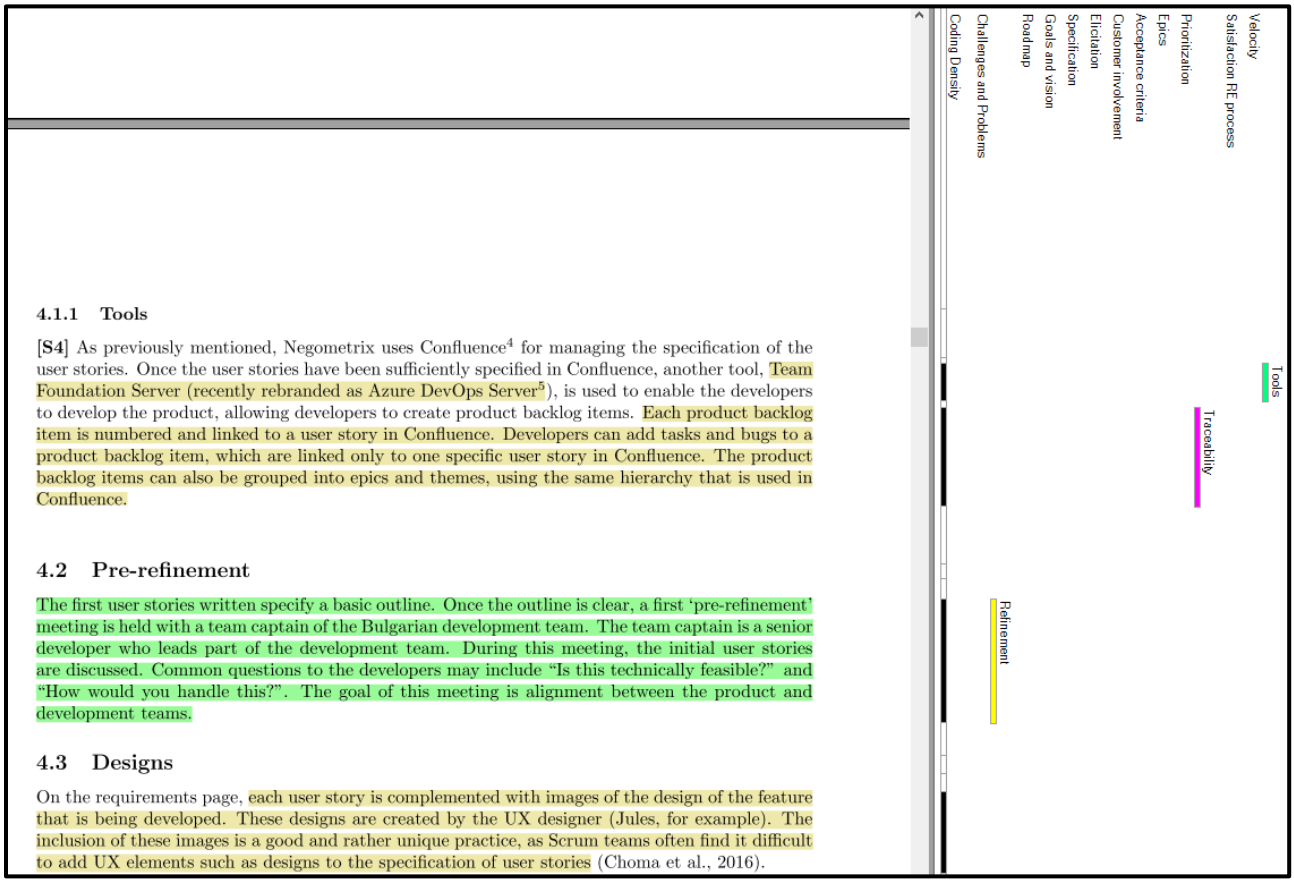


Figure 17: Sample screenshot on the encoding of a report. The highlighted text is text that is encoded in a node. The corresponding node can be seen on the right.

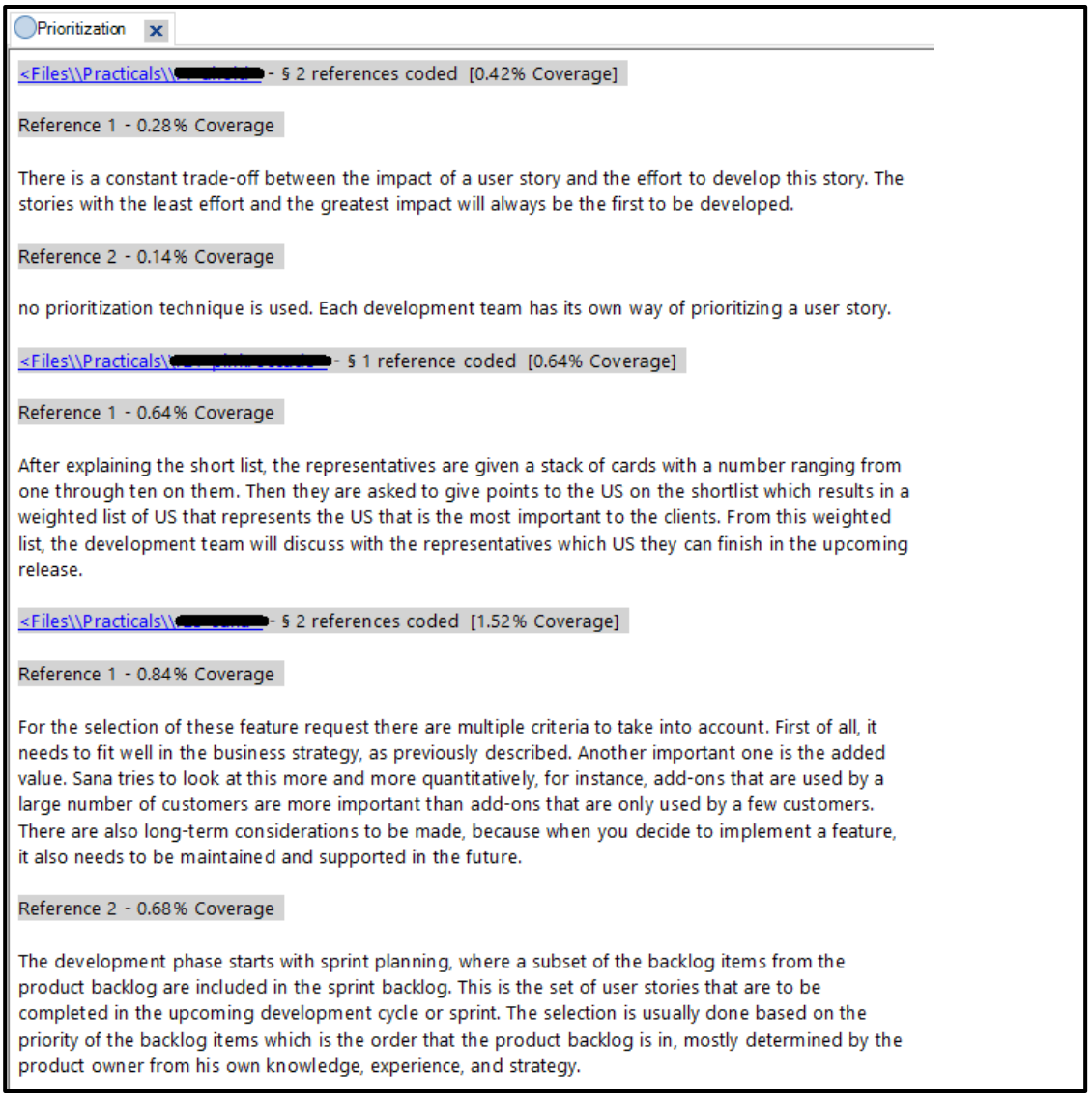


Figure 18: Example screenshot of a node. In this case, the prioritization node

4.4 Results

In total, we analyzed 32 teams located at 29 different companies. The companies range from multinationals to startups. From these teams, 18 work for software service companies; companies who sell their services to customers in order to create a specific software solution/service for that customer. The other 14 work for software product companies; companies who sell a software product or build a product for their own business needs.

Based on 32 teams there were 33 nodes containing a total of 817 references, or 25.5 references per team. The findings of the analysis are discussed together with the maturity matrix, focus areas and capabilities in chapter 5. A link to the fully analyzed dataset can be found in appendix B.

5. Designing the maturity model

Out of all the models and studies discussed on agile and RE maturity in chapter 3, none of them are focus area maturity models. Yet a focus-area maturity model has several advantages over a fixed-level model. As mentioned before, focus-area maturity models can demonstrate dependencies between different capabilities and determine maturity for individual focus areas, giving more insight into the maturity of an individual focus area [35]. Therefore, the aim is to design a focus-area maturity model.

Before getting into the actual design of the model, we quickly explain how these models work and can be read. Figure 19 below shows a focus area maturity model for software product management [58]. This model consists of 15 focus areas divided among four categories; requirements management, release planning, product planning and portfolio management. Every focus area has some capabilities (the capital letters) on a specific maturity level (0 to 10).

	0	1	2	3	4	5	6	7	8	9	10
Requirements management											
Requirements gathering		A		B	C		D	E	F		
Requirements identification			A			B		C			D
Requirements organizing				A		B		C			
Release planning											
Requirements prioritization			A		B	C	D			E	
Release definition			A	B	C				D		E
Release definition validation					A			B		C	
Scope change management				A		B		C		D	
Build validation					A			B		C	
Launch preparation		A		B		C	D		E		F
Product planning											
Roadmap intelligence				A		B	C		D	E	
Core asset roadmapping					A		B		C		D
Product roadmapping			A	B			C	D		E	
Portfolio management											
Market analysis					A		B	C	D		E
Partnering & contracting						A	B		C	D	E
Product lifecycle management					A	B			C	D	E

Figure 19: A maturity model for software product management [58]

As mentioned before, maturity grids contain dependencies between capabilities. There are two types of dependencies [59];

- Intra-process capability dependency
- Inter-process capability dependency

Capabilities within focus areas are always dependent on each other, i.e. capability B from focus area *x* is always dependent on capability A, C is dependent on B and so on. This is known as intra-process capability dependency and demonstrated in the figure below.

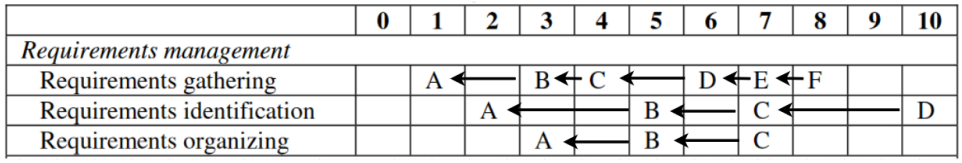


Figure 20: Dependencies between capabilities within a focus area

Capabilities from different focus areas can also be dependent on each other. In this case, the order in which they appear on the maturity scale decides the dependency, i.e. a capability that appears on maturity level 2 can be dependent on some capabilities of maturity level 1. This is known as an inter-process dependency and demonstrated in figure 21 below. Note that inter-process dependencies *can* exist, but don't need to. That is to say, it is entirely possible (and actually much more likely) that a capability from one focus area is *not* dependent on a capability from a different focus area. In a maturity matrix, the actual dependencies are not visible, as there are many intra-process dependencies and often several inter-process dependencies. The dependencies are therefore described in the table describing a capability.

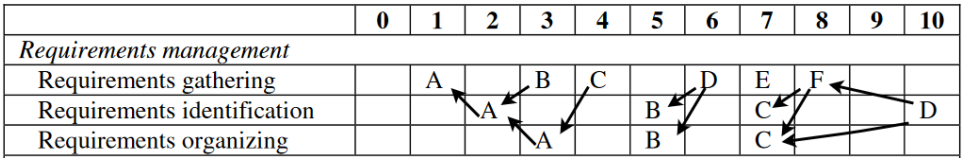


Figure 21: Possible inter-process dependencies between capabilities between focus areas

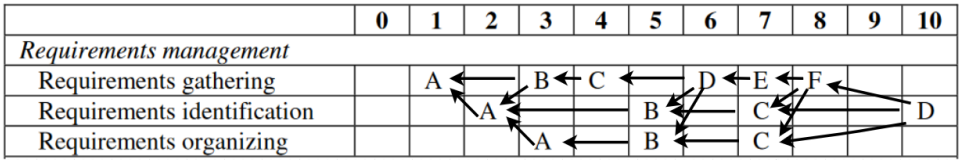


Figure 22: All possible dependencies between capabilities between/within focus areas

5.1. Method

Methods, techniques and comments on the creation and design of maturity models have been described in some studies. While there are many studies on the design of fixed-level maturity models (e.g. [60]) there are few methods specifically made for focus area maturity models. The study by van Steenbergen et al. reviews several methods proposed by other literature to propose a new method, which can be seen in figure 23 below [35].

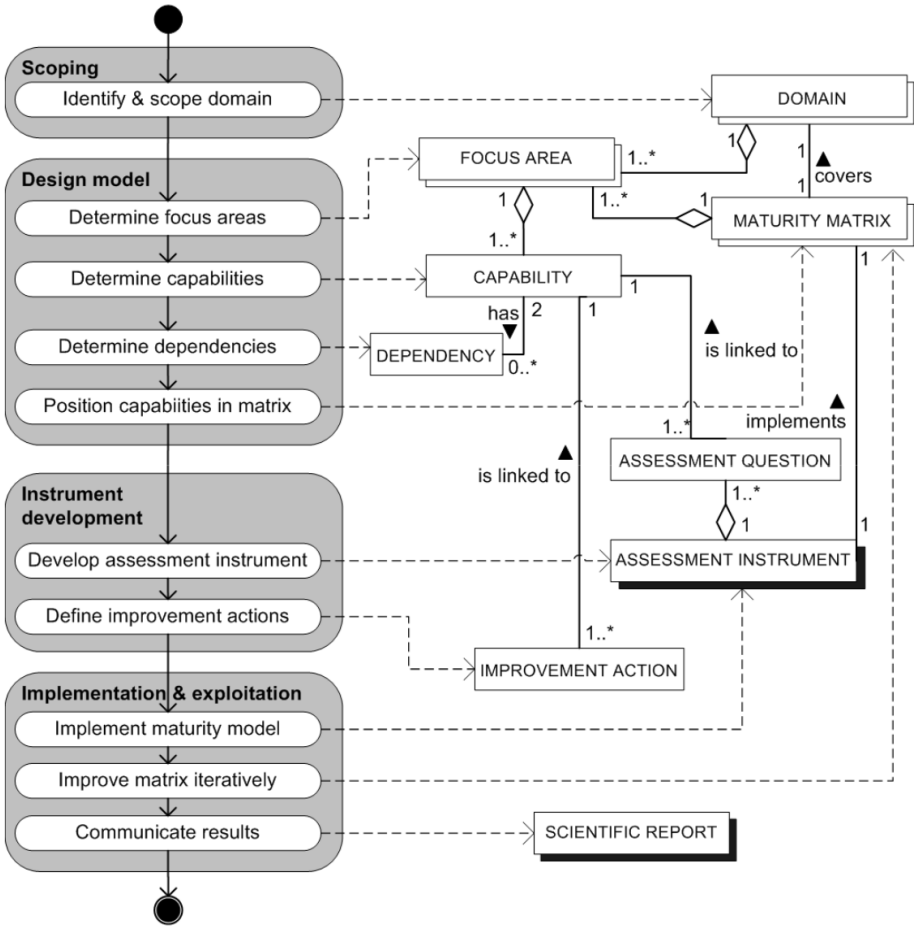


Figure 23: The development method of a focus area maturity model [35]

An altered method of the development method proposed by van Steenbergen et al. is used for the development of the agile RE maturity matrix (figure 24). In this study, the Scoping activity is done by means of a literature study and case study. Additionally, this study validates the matrix after the initial design, as a lack of validated models is a key issue in maturity model design [32]. The validation of the model is done by combining two different approaches:

1. Expert interviews
2. Focus group

During the validation sessions, *improvement actions* and *issues* are produced. The addition of the validation activity is visualized in the process-deliverable diagram based on the diagram by van Steenbergen et al in figure 24. In this study, the final two activities *Instrument development* and *Implementation & Exploitation* which are part of the original development method are not performed.

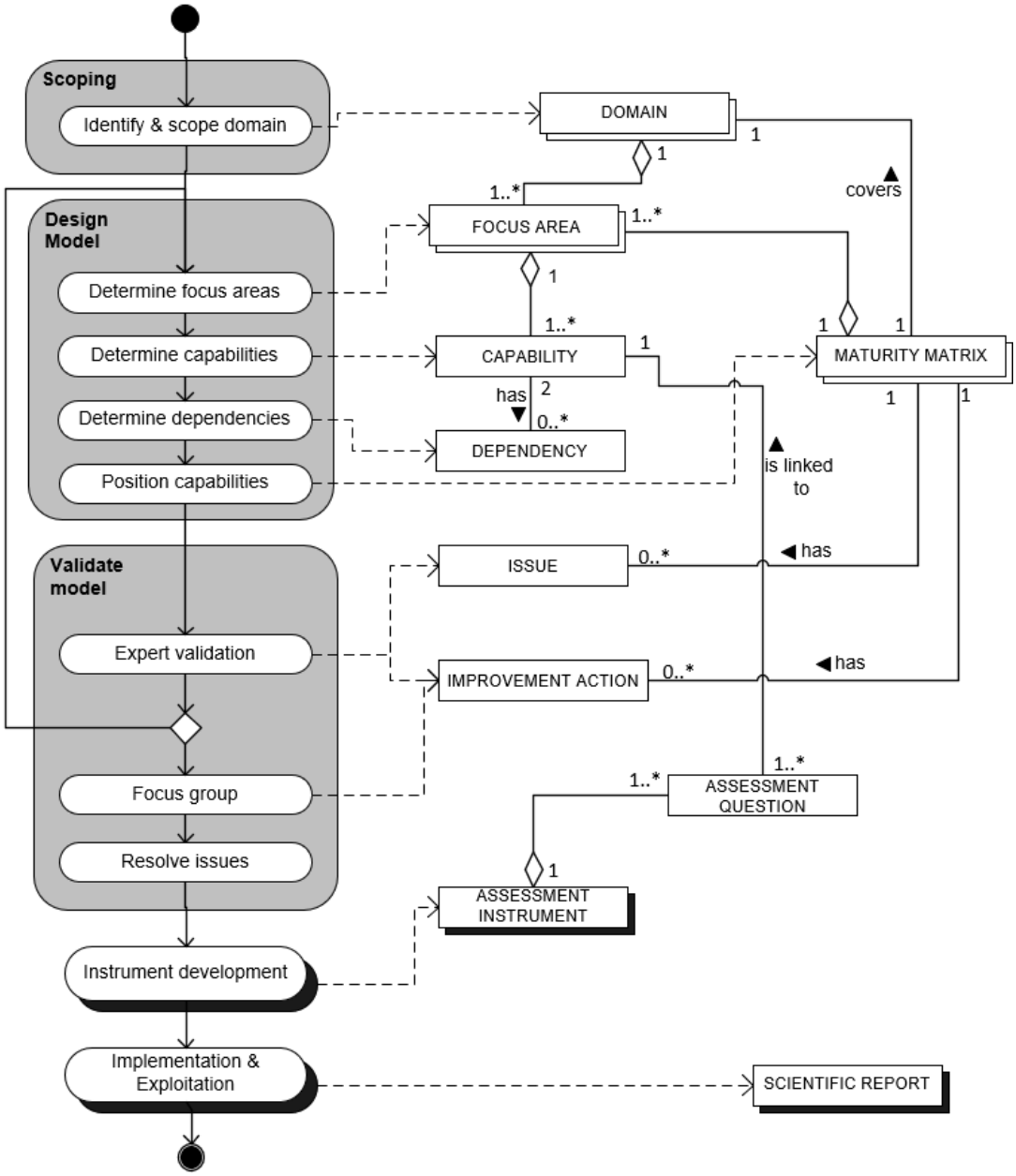


Figure 24: The altered development method used in this study. After an expert validation is performed the design model activity is performed again to process improvements suggested by the expert. Remaining issues were discussed during the focus group activity.

5.1.1. Expert interviews

Firstly, expert interviews are conducted to improve the model, focus areas and capabilities iteratively. As the first version of the model was created based on identified capabilities in the literature review and case study there might still be capabilities missing, unclear or incorrect. Some focus areas might be too big or too small. Therefore, the first validation round consists of semi-structured expert

interviews. This validation method is partially based on the validation method used by de Feijter in his validation of a DevOps maturity model [61]. Firstly, the sessions started with the explanation of the maturity model, focus areas and capabilities. The following questions were posed:

- [Understandability] Do you think the focus areas and its capabilities are understandable?
- [Relevance] Do you think the focus areas and capabilities are relevant for agile RE?
- [Completeness] Do you think the focus areas and its capabilities are complete? If not, what should be added to make them complete?
- [Maturity] Do you think the order in which the capabilities should be implemented is correct? (capabilities should be achieved in the presented order)

When experts suggested improvements or issues with the model, the model was improved before the next expert validation session. If there were issues that could not be resolved, these issues were discussed with the focus group.

Name	**Function(s) / Roles**	**Organization**	**Experience in current organization**	**Experience in current function**
ExpertA	Agile Coach	InfoSupport	13	13
ExpertB	Product Owner	Create	2	5
ExpertC	Product owner/business analyst	InfoSupport	3	5
ExpertD	Product owner/ Business analyst	InfoSupport	5	5

Table 6: Participants in the expert validation, experience is in years

5.1.2. Focus group

The expert validation sessions resulted in many suggestions and improvements. However, in some cases the expert came up with an issue of the model but did not have a clear idea on how to resolve the issue. Therefore, these issues were presented and explained to the focus group, after which the participants discussed on a solution for the issue.

Name	**Function(s) / Roles**	**Experience in current organization**	**Experience in current function**
FocusGroupExpertA	Service delivery manager	14	2
FocusGroupExpertB	Product owner / Business analyst	4	4
FocusGroupExpertC	Agile coach / Project manager	18	6
FocusGroupExpertD	Arealead Way of Working / Manager / Senior Consultant	11	1.5
FocusGroupExpertE	Senior Consultant	13	6

Table 7: Participants in the focus group, experience is in years. All participants are employees of Info Support

5.1.3 Results of the validation

The expert validation resulted in 37 revisions. These revisions can be any of the following actions:

- Addition, removal, merging or splitting of focus areas.
- Addition, removal, merging or splitting of capabilities.
- Addition or removal of a dependency between capabilities.
- Altering the content of a capability in a focus area.

In addition, there were 5 issues that remained unsolved. These issues were resolved by the focus group, resulting in another 6 revisions. The initial matrix consisted of 17 focus areas with 59 capabilities and 5 dependencies. The final version of the matrix has 13 focus areas with 56 capabilities and 9 dependencies. Table 8 below shows how the density of the matrix evolved over the iterations. The first validation session immediately resulted in the removal of a great many focus areas (from 17 to 13) and capabilities (from 59 to 49). Over the next validations and focus group the density of the model slowly increased. The full changelog and evolution of the model can be found in appendix C.

Matrix version	Nr of focus areas	Nr of capabilities	Nr of dependencies	Capabilities per focus area (average)	Capabilities per maturity level (average)	Nr of revisions
Initial (V0.2)	17	59	5	3.47	5.90	N/a
1st validation (V0.3)	13	49	5	3.77	4.90	14
2nd validation (V0.4)	13	53	6	4.07	5.30	14
3rd validation (V0.5)	13	54	7	4.15	5.40	5
4th validation (V0.6)	13	54	7	4.15	5.40	4
Focus group (V1)	13	56	8	4.31	5.60	6

Table 8: Results of the validation sessions

5.2. Defining the scope of the SBRE model

As mentioned before, the scope of the model is focused on all activities, processes or artifacts that are related to RE in Scrum. Additionally, the model should focus on assessing a team and not their entire organization. It addresses processes, activities and problems that occur at this level. For example, many problems identified in the work of Fernández cannot be addressed by team's themselves [31]. Policy restrictions, mistakes/incompetence at the customer side and insufficient resources granted from higher management are some of the causes of RE problems that teams cannot easily address themselves [31]. The model should focus on aspects of agile RE that the team *can* influence and improve. Examples are the use of certain techniques, responsibilities and roles within a team, communication with end-users & stakeholders, et cetera.

5.3. Defining focus areas & capabilities

The original results of the case study resulted in 19 focus areas distributed over 5 categories. During the validation of the model focus areas were merged with each other, changed or removed. This resulted in 13 focus areas distributed over 4 categories. Some of the focus areas have a clear inspiration from literature, like eliciting requirements, documenting requirements, validating requirements and managing requirements [50,51]. Others originate from the prevalence encountered in the case study, like estimating effort.

The categories are;

- Management & Planning. This category contains focus areas that describe capabilities which influence the management and planning activities within

agile RE. The focus areas are *roadmapping* (long term planning), *stakeholder involvement* and *sprint planning* (short term planning).

- Ideation. This category contains focus areas that describe capabilities which influence the process of the ideation (a creative process which forms new ideas or concepts) of new requirements. The corresponding focus areas are *Goals & Vision* and *Eliciting requirements*.
- Specification. This category contains focus areas which are involved in the process of specifying requirements before they can be built. The corresponding focus areas are *Documenting requirements*, *Templates*, *Refining requirements* and *Acceptance criteria*.
- Implementation. This category contains focus areas that are involved in the process of implementing requirements. The corresponding focus areas are *Prioritizing requirements, Estimating effort, Validating requirements* and *Traceability & Documentation*.

In traditional RE the main and sub-activities could be easily identified (figure 4). In agile RE, the boundaries between processes blur significantly. This can be seen by the great variety in the focus areas. There are some that are related to high-level project management (*roadmapping*, *goals & vision*), some are clearly based on Scrum activities (*sprint planning, refining requirements*) and others become relevant when the requirement has been implemented (*acceptance criteria, validating requirements*). Yet they all have in common that there is a significant relationship with RE. In the following sections, each focus area and its corresponding capabilities will be explained. Sometimes the capabilities refer to literature work, a report or transcription in the case study (indicated with r (report) and t (transcript) or the changelog of the matrix which can be found in appendix C). Reports and transcripts can be found by accessing the NVIVO file in appendix B.

5.3.1 Roadmapping

Roadmapping is not only the act of creating a roadmap. It can be seen as an act of high-level planning the development effort. Roadmaps are commonly used to establish the order of software releases. Usually, they are established for a year or less. Only 2 companies made longer roadmaps, for 2 and 5 years respectively. They are often based on an epic-level. As most epics are usually estimated on a t-shirt based sizing it is hard to estimate the duration of several epics, and it is therefore not recommended to try to estimate as accurate as possible. Product software companies should establish both long-term and short-term strategies for their product.

The length of sprints is also important. When sprints are too short, this creates too much overhead with other activities. When sprints are too long, the agile principles (such as frequent delivery of working software) are ignored. All teams in the case study had either a 2- or 3 week duration for sprints. Sprints should not be shorter, as the overhead of some activities will take up too much team to complete actual backlog items. They also shouldn't be longer, as this conflicts with agile values such as

continuous delivery of working software and hinders the ability of the team to quickly respond to changes. According to the first expert validation (*changelog v0.3*), roadmapping should also be seen as a prioritization activity, as stakeholders actively plan and prioritize high-level requirements. The first capability is the existence of a roadmap that describes for at least a multitude of sprints which high-level requirements (like epics or features) will be built. Sprints should have a consistent length and 2-3 weeks are commonly used as a default sprint length. Roadmaps are often based on a feature or epic level.

A. Existing roadmap	
Description	A roadmap exists that considers what epics/features will be built in the upcoming sprints. Sprints are of consistent length of 2 to 3 weeks. Epics or features have some rough estimation, for example using T-shirt sizing.
Prerequisite(s)	None
Reference(s)	[57]

The second capability is defined by teams continuously updating their roadmap. Agile RE is characterized by continuous reprioritization. This does not mean that the roadmap should change every sprint, but that there are regular moments at which the team considers the current roadmap and whether there need to be any changes based on recent performance, changing needs, et cetera. Expert C mentioned during the validation that having a living roadmap increases the agility of a team, i.e. it is easier to adapt to changes.

B. Living roadmap	
Description	The roadmap is 'living' by being continuously updated and reprioritized in accordance with stakeholders and the team.
Prerequisite(s)	None
Reference(s)	*changelog v0.5*

Capability C states that roadmaps should be established for a longer period. Product software companies are often more dependent on the success of their software product and probably feel the need to plan ahead further in time as a strategic need. Considering the high dependency of the success of the company on the success of the software product, these teams should ensure to plan ahead further yet still ensure capability B of continuously reconsidering the roadmap. In case of project based teams, they should have some outline mapped out for the duration of the project.

C. Long term roadmaps	
Description	The team has a roadmap established for releases of the product for the upcoming year or duration of the project.
Prerequisite(s)	None
Reference(s)	[57]

Capability D is based on teams that operate in multi-team projects and is therefore situational for these teams. Working in multi-team projects can cause significant challenges; one of these is dependencies between teams. Sometimes, teams will be dependent on software updates that are built by other teams. For many reasons (e.g. underperforming, being delayed) a single team can cause a chain reaction impacting other teams. For this reason, the RE@agile handbook advices to create multi-team roadmaps which also consider dependencies between teams [57]. Some examples are proposed in the Scaled Agile Framework (SAFe).

D. Multi-team roadmaps	
Description	The teams use multi-team roadmaps that take into account explicit dependencies between teams
Prerequisite(s)	Situational for multi-team projects only.
Reference(s)	[57]

5.3.2 Stakeholder involvement

Customer involvement and stakeholder management are key aspects for successful releases. By involving the customer and other stakeholders during certain activities it is more likely that the release will be satisfactory for said customer/stakeholders. Note that customer and stakeholder can have different meanings. The customer is often a member of the client company who is assigned the role of product owner, but there are often more members of the client company who also have an interest or stake in the product. Although these can also be considered customers, we call them stakeholders. The work by Schön et al. was aimed at identifying practices that involve stakeholders in the process of RE and are compatible with agile software development [13]. The first capability states that some stakeholders are occasionally involved in RE activities such as elicitation or validation. Involving customers and users throughout RE is recommended as a best practice in the work of Hofmann & Lehner, as it results in better understanding of the real needs [5].

A. Occasional involvement	
Description	Stakeholders are involved during some activities such as elicitation, validation, decision-making, prioritization, but not on a consistent basis.
Prerequisite(s)	None
Reference(s)	[5,13,26]

The second capability is a simple improvement upon capability A, stating that stakeholders need to be involved *consistently*. That means stakeholders will participate in certain activities (like refinement sessions) on a consistent basis. Consistent participation lowers the communication barrier between the team and stakeholders.

B. Consistent involvement	
Description	Some stakeholders are involved in some activities such as elicitation, validation, decision-making and prioritization on a consistent basis.
Prerequisite(s)	None
Reference(s)	[13,26]

The involvement of stakeholders is very dependent on the capability of a team to identify all relevant stakeholders. Ignoring stakeholders can result in missing important sources for requirements. Failing to take into account some sources stakeholders, such as compliance or legal, might have severe consequences for the success of the project. Therefore, capability C states that teams should ensure to perform a systematic identification of relevant stakeholders at the start of a project and keep it up-to-date during the project, as the importance of certain stakeholders changes during the course of a project.

C. Stakeholder identification	
Description	The team has a clear overview of stakeholders of the product/project. Systematic identification of stakeholders is done at the beginning of a project and is kept up-to-date during the project.
Prerequisite(s)	None
Reference(s)	[57]

As a key concept of agile is personal communication with stakeholders it is important that stakeholders are onsite. Having onsite stakeholders makes it easier for the team to have continuous communication with these stakeholders. It is not necessary that *all* stakeholders are *always* onsite, but it should be a goal to ensure easy communication lines within the team and between stakeholders. The availability of stakeholders was also a criteria in the maturity model of Patel & Ramachandran (table 3) [44].

D. Onsite stakeholders	
Description	Customers and (important) stakeholders are onsite, facilitating their ability to be consistently present during elicitation, verification and validation activities.
Prerequisite(s)	None
Reference(s)	[44]

Finally, very few of the reports included elicitation and verification with end-users. For some teams, this is easier to achieve than for others. For example, the end-user can be onsite in case of internal software systems. For external end-users, involving these users is more difficult but still possible, for example by requesting feedback, organizing consumer panels, et cetera.

E. End-user involvement	
Description	The team involves the end-user in elicitation and verification activities
Prerequisite(s)	Situational, as the end-user does not always exist or is not always available
Reference(s)	r4, r15 (feature requests), r11 (customer journeys)

5.3.3 Sprint planning

Sprint planning is an activity in which backlog items are selected to be implemented in the upcoming sprint. Naturally, even performing sprint planning is the first capability in this focus area. To be able to perform sprint planning, teams should ensure that there are enough backlog items to enter the upcoming sprint. Having too few items to work on is a waste of development resources. Sprint planning should be a small activity at the start of the sprint where it is quickly discussed which items will be done in the upcoming sprint. Some teams in the case study also use the sprint planning to refine items (e.g. r9), but according to expert C this is way too late and all items should be refined prior to the sprint planning.

A. Performing basic sprint planning	
Description	Sprint planning is done prior to the start of a sprint. The team ensures there are enough backlog items to enter the upcoming sprint.
Prerequisite(s)	None
Reference(s)	r10, r12, [2]

The second capability is based on prioritization. Sprints should be planned based on a certain prioritization. Note that sprint planning itself is *not* a prioritization activity. That is to say, the backlog should be prioritized prior to the sprint planning activity. Because of this, the capability has a dependency on *prioritizing requirements : A*.

B. Prioritized sprint planning	
Description	Backlog items are prioritized and enter a sprint backlog based on their prioritization.
Prerequisite(s)	Dependency on Prioritizing requirements : A
Reference(s)	*changelog v0.5*

As a common problem identified was a lack of time/scheduling issues it is important that teams are able to estimate the capacity and workload for their sprints correctly [62]. As with prioritization, this should be done prior to the actual sprint planning is performed. Therefore, this capability has a dependency on *Estimating effort : A*. Teams should also take into account their expected capacity for the upcoming sprint, for example by using Yesterday's weather[2].

C. Capacity and estimations	
Description	Team takes into account the expected capacity of the team for the next sprint and the expected workload of items.
Prerequisite(s)	Dependency on Estimating effort : A
Reference(s)	*changelog v0.5*

The fourth capability is based on the definition of ready. Having a definition of ready was one of the three practices recommended by Heck & Zaidman [24]. This definition of ready acts as a checklist whether backlog items are ready to be worked on, the team understands what needs to be done, the item is estimated and prioritized. The actual act of establishing a definition of ready is done in the refinement focus area, and therefore this capability has a dependency on *Refining requirements : B*. This capability only states that the definition of ready is used to ensure that items that are not defined as 'ready' will not enter the upcoming sprint.

D. Definition of ready	
Description	The team has an established definition of ready. Backlog items that are not defined as 'ready' will not enter a sprint.
Prerequisite(s)	Dependency on Refining requirements : B
Reference(s)	e1, [25,63]

The final capability in sprint planning is the verification of items prior to the development. Verification also does not explicitly need to happen during the sprint planning activity itself, but can be done by including a stakeholder in the planning activity or publishing the sprint backlog to the stakeholders and have them confirm their approval. This capability is partially based on a maturity criteria from the agile

[2] https://www.scruminc.com/yesterdays-weather/

maturity model by Patel & Ramachandran (table 3, '*obtain commitment to story cards from onsite customer*').

	E. Verification of the sprint backlog
Description	Before a PBI enters a sprint backlog, its verified with the customer/stakeholder whether the item still conveys the correct wishes of the customer/stakeholder.
Prerequisite(s)	None
Reference(s)	[44]

5.3.4 Goals & Vision

Most companies have strategic goals and visions established for their products or business. Most reports from the case study do not specify exactly how these goals and visions are established, but two of the companies used the OGSM framework to specify their goals. OGSM stands for Objectives, Goals, Strategies and Measures and is a strategic planning process. It is clear that software product companies need to establish goals and visions for their products to continue developing and improving them. While software service companies do not need to establish these goals/visions themselves (as their customer has a certain goal/vision in mind), they *do* need to make sure they correctly understand the goals/visions of their customers. This is often done with simple interviews but some companies pay extensive attention to stakeholder management. Thus, the first capability states that the team must have a shared understanding of the goal and vision of the product/project. High-level requirements are requirements that are *elicited* from stakeholders, based on their goals and visions [57]. Therefore, correctly understanding this goal and vision is an important first step for the correct identification of high level requirements.

	A. Shared understanding of the goal and vision
Description	The team has established a shared understanding of the goal and vision of the product or project.
Prerequisite(s)	None
Reference(s)	*changelog v0.3*

Relevant stakeholders formulate the vision and the goals. Therefore, the identification of a new stakeholder may have an impact on the vision or the goals (and this is also why the identification of stakeholders is a capability in the *Stakeholder involvement* focus area). Additionally, different stakeholders can have different goals and visions for a product/project. Teams should not be aware of only the goal & vision of one stakeholder, but to be consider other stakeholders as well. These vision and goals can be used to define an initial scope [57].

B. Stakeholder Goals & Visions	
Description	The team extracts the goals and vision of individual stakeholders for the product to define an initial scope.
Prerequisite(s)	Dependency on Stakeholder involvement : C
Reference(s)	[57]

Understanding the goal and vision of the product/project is one thing. According to expert A during the validation the establishment of a teams' own goals and vision for the product/project is a clear sign of a mature team. The construction of their own goal & vision allows a team to determine the best ways to contribute toward this goal/vision, and involves the team into the product/project better than merely following some stakeholders' goal/vision. In the case studies, software teams usually do not define the vision & goal for themselves. There are some exceptions in the reports. Startups for example, are often so small that the development team represents the entire company (r6, r13). In other teams, the team generates a vision & goal by themselves that *align* with the vision & goal of the company (r4, r5). Expert C states that she believes this is a clear sign of a more mature team, as a team that aligns their own vision & goal to think and work pro-active. An example was one of the reports (r5), in which teams are given the freedom to formulate their own goal/vision, as long as it aligns/contributes to the department goal/vision.

C. Team Goal & Vision	
Description	Teams establish a goal & vision for themselves in which they state how they will achieve the company goals & vision.
Prerequisite(s)	None
Reference(s)	r5, *changelog v0.3*

There are techniques that can be used to establish goals & vision. Having team knowledge of such techniques and applying them where applicable is seen as the final capability for this focus area. Examples of techniques to define the goal & vision of their product are proposed by the RE@agile handbook, the works of Pichler or the OGSM framework we encountered at some teams (r5, r2) [57,64]. These techniques can have different purposes. For example, a technique that can be used to generate high-level requirements from goals and vision is impact mapping, with other techniques like the OGSM serve for creating strategies and metrics to achieve a goal. Capability D states that teams should know and apply dedicated techniques when relevant.

D. Dedicated techniques	
Description	The team uses appropriate techniques to establish goals & visions
Prerequisite(s)	None
Reference(s)	[57,64], e1, r5, r2

5.3.5 Eliciting requirements

Elicitation is the process of seeking, uncovering, acquiring, and elaborating requirements from stakeholders [65]. The most popular elicitation method in both literature and the case studies was interviews with stakeholders [19]. Regardless whether these interviews were structured, semi-structured or free format the ease of use that interviews offer appear to make them very popular. Several reports mention solely interviews as the way elicitation is performed. Therefore, the first capability states that requirements are elicited from stakeholders using interviews. Interviews can be of any type, and we also consider just a face to face discussion an interview. The work by Paetsch, Eberlein & Maurer state the value of interviews is that direct interaction is important for establishing trust between the customer and developer [26].

A. Elicitation by interview	
Description	Requirements are elicited from stakeholders using interviews.
Prerequisite(s)	None
Reference(s)	[19,26]

As several reports mention only interviews as the way elicitation is performed, applying different techniques in different contexts is a sign of more mature teams. Some reports mention more expansive elicitation techniques such as collaborative workshops with multiple stakeholders together (e.g. t1, t2 & r2). Very rarely, the end-user is also used to elicit new requirements. This can be done by using feature requests (r15), or some teams apply customer journeys (r11). This happens more often when the end-user is a user within the company, as it is easy to contact this type of end-user, and is much more rare in companies who do not have easy ways to contact their end-users. Often, elicitation results in the extraction of high-level requirements and some low-level requirements. The stakeholders will not be involved again until verification and validation activities, or when something remains unclear.

B. Elicitation techniques	
Description	The team uses dedicated elicitation techniques to generate low-level requirements from high-level requirements. Techniques like story mapping or other workshop/collaboration sessions
Prerequisite(s)	None
Reference(s)	t1, t2, r22, r11, r15, [57,66]

Capability C is somewhat similar to capability B as they both imply the use of dedicated workshops or meetings. The difference is that capability C assumes the combination

of different disciplines during certain workshops. For example, 3 amigo sessions[3] are a commonly used approach in Scrum to involve the disciplines of testing, business and development. Involving several disciplines such as business, technical and testing in a meeting allows for a different dynamic. Such depth is often not necessary for most requirements, but can be useful when discussing challenging backlog items or stakeholder wishes. Note that combining disciplines is also possible for the refinement of items, and that these two activities can be done in the same meeting/workshop. Expert C noted during the validation that many teams likely apply a technique like 3 amigo sessions but are unaware that this is a specific technique.

C. Combining disciplines	
Description	The team combines disciplines (such as testing, technical & business) during elicitation activities.
Prerequisite(s)	None
Reference(s)	t2

Capability D states that mature teams know their stakeholders' goals and vision so well there is elicitation meetings are highly efficient. This capability arose from the validation with expert D, who stated "*Very mature teams no longer need to elicit small details from stakeholders. If they are mature enough, they can think for themselves well enough to fill in these details by themselves. I consider this to be a final step of maturity*". The team will be able to determine for themselves how to interpret the wishes of the stakeholder and fill in missing details. This capability was discussed by the focus group. The focus group agreed this was an indication of high maturity. However, there does need to be a dependency to the validation focus area. If elicitation with the stakeholder is omitted (or done very rarely) the corresponding requirements need to be validated extra thoroughly with the stakeholder after implementation, even more so than if elicitation were not omitted. As a result of this discussion capability D has a dependency to the validating requirements focus area capability E, as the experts stated that when omitting elicitation the validation needs to be *thorough* and not on a lower level validation capability.

D. Efficient elicitation	
Description	The team no longer needs to elicit low-level requirements and can implement low-level requirements without prior consultation with a stakeholder.
Prerequisite(s)	Dependency on Validating requirements : E
Reference(s)	*Changelog v0.6 , changelog v1*

[3] https://www.agilealliance.org/glossary/three-amigos/

5.3.6 Documenting requirements

The documentation of requirements can be done when a requirement is first elicited, and stays relevant all the way throughout the remaining development process. Tools can be used to support RE activities, and the main use of tools in teams is the documentation of requirements. Almost all teams use either JIRA (17 teams) or Azure Devops (11 teams) as a centralized place to collect and refine on requirements. This is also the place where additional information is stored, such as estimates, prioritization, designs, notes, acceptance criteria. Confluence (5 teams) is used by some as a collaboration tool and Excel also still sees some use (3 teams). This results in Capability A, the use of a tool to document, store and manage requirements. At this level, the tool can be as simple as an Excel file that contains backlog items. However, capability C will demonstrate why this is not a good way to store requirements. Some teams also use a physical Kanban or scrum board (10 teams). While the functioning of a physical board is similar, one expert (v1) states '*A physical board is useful during standup meetings, allowing for a quick overview and discussion on backlog items*'. Usually, teams who use a physical backlog also use a digital backlog. The use of a tool was also recommended as a best practice by Heck & Zaidman [25].

A. Documenting with tools	
Description	The team uses a tool to manage requirements. This can be a physical board, such as a Scrum board, or an electronic tool, such as JIRA or Azure devops.
Prerequisite(s)	None
Reference(s)	[25]

The second capability is that teams should regularly refine and reprioritize their backlog, which is an ongoing activity [2]. This is often done in refinement sessions, and therefore has a dependency on *Refining requirements : A*. This ensures the regular production of items that can enter into a new sprint and also prevents a cluttered backlog with items that are no longer relevant. This concept is also known as backlog grooming.

B. Backlog refinement	
Description	Product backlog and backlog items are regularly refined and reprioritized
Prerequisite(s)	Dependency on Refining requirements : A
Reference(s)	*Changelog v0.3*

The third capability was created during a validation session with expert B (changelog v0.4). The expert states that it is very important (and also stated by the scrum guide) that backlog items are transparent and can be accessed by all stakeholders. The expert states that this is also the value of a tool like azure devops or Jira as it allows for easy transparency over a file like excel to keep a backlog. Another advantage of having a

transparent backlog is that, as mentioned in *sprint planning : E*, stakeholders can verify a sprint planning by themselves.

C. Transparent backlog	
Description	The documented requirements are transparent and accessible by all relevant stakeholders. Requirements are documented in a template.
Prerequisite(s)	Dependency on Templates : A
Reference(s)	*Changelog v0.4*

Capability D and E are concerned with the use of quality frameworks. Requirements should be documented in such a manner that they adhere to a quality framework, such as INVEST or QUS. INVEST criteria are similar to the definition of ready. It provides criteria to assess whether a story or backlog item adheres to a certain quality, which in turn means it is ready to enter a sprint. Lucassen et al. found that practitioners who use quality frameworks such as INVEST believe they improve productivity and product quality [20]. However, as so few teams in the case study actually used any form of quality criteria (6 out of 32 teams used some quality framework, such as INVEST) it has been defined as capability D. Finally, there have been more extensive frameworks to ensure the quality of stories, such as QUS [29]. As such a framework does not see any use by the teams in the case study and is even more extensive than INVEST criteria, it is defined at a high maturity level in capability E. The applicability of such an extensive framework was confirmed by expert A during the validation, but only in certain contexts like safety-critical software where the quality of software and requirements must be extensively proven. The use of quality criteria was recommended as a best practice in the work by Heck & Zaidman [63]. A checklist also aids in not forgetting criteria, basing opinions on facts, to educate other team members and to standardize the review process [25].

D. Quality frameworks	
Description	The team has quality criteria (such as INVEST) to ensure the quality of stories.
Prerequisite(s)	None
Reference(s)	[24,25,57,63], r15, r14, r17

E. Extensive quality frameworks	
Description	The team uses an even more expansive quality framework to test for user story quality, such as QUS.
Prerequisite(s)	None
Reference(s)	[29]

5.3.7 Templates

Templates dictate the format in which a requirement is stored. As was found in literature, it does not really matter what template is used as long as a template is used consistently [20,25]. Hence capability A, the consistent use of a template, does not recommend a certain template. Although user stories and the Connextra template are highly popular, we do not want to enforce the use of them via a capability. The important thing is that team members know which template to use and what template they can expect. Specification templates are a recommended best practice in the work of Hofmann & Lehner [5].

A. Established template	
Description	The team has an established template for requirements which is used consistently.
Prerequisite(s)	None
Reference(s)	[5,20,25]

An issue in agile RE is the lack of attention toward non-functional requirements [22]. Capability B states that teams should make a clear distinction between functional and non-functional requirements, and that non-functional requirements should not be specified in the user story format as it is considered an unsuitable format (e1). According to an expert (e1) it is important that teams distinguish between functional and nonfunctional requirements. *"There is a reason we call backlog items backlog items, they do not all need to be functional requirements. These are just items that in the end should be small enough to be done in a sprint, and whether it is a user story, technical requirement or test case is irrelevant. The argument that NFRs do not have business value is not true at all. If the customer says that is the case, then the product owner is probably talking to the wrong stakeholder (e1)."* We try to enforce this with capability B. The issue of neglecting non-functional requirements was also mentioned in the work by Paetsch et al. [26]. The authors noted that agile methods need to include more explicitly the handling of non-functional requirements. Some suggestions for handling non-functional requirements are discussed in the RE@agile handbook [57].

B. Distinction functional/non-functional	
Description	There is a clear distinction between functional and non-functional requirements. Non-functional requirements are not specified in a format that is intended for functional requirements (such as a user story).
Prerequisite(s)	None
Reference(s)	e1, t1, [22,26]

Some templates, such as the user story, contain a description of the benefit of a requirement. The use of a benefit part is considered vital by many teams (but not all,

e.g. r13, r14, r16) and this was also confirmed in literature and during the validation (*changelog v0.3*, *changelog v0.4*) [20]. Capability C therefore states that the team should always describe the benefit of implementing the requirement.

C. Benefit argumentation	
Description	The team always describes the benefit in their requirements.
Prerequisite(s)	None
Reference(s)	r10, r21, r22, *changelog v0.3*, *changelog v0.4,* [20]

Backlog items can also consist of a certain 'template', or structure of information elements. From the four teams at Info Support it became clear that only a user story is not enough. The teams often find themselves adding additional items to backlog items, such as acceptance criteria, additional natural language to further explain the item, technical analyses, mockups/diagrams, business value of the item, unanswered questions, tasks for the developers, relevant stakeholders for the item, et cetera. This is also mentioned in the mapping study by Heikkilä et al., who found that the user story format is often insufficient [12]. There is no hard limit on what elements should be included; this is for the teams to determine which elements should be included. A mature team will not set a hard demand for information elements that should be included but is able to determine relevant elements on an individual backlog item (e.g. some teams might always want to include some technical description, while others may want to omit it). The agile maturity model by Patel & Ramachandran also state that teams should create their own structure for backlog items [44].

D. Additional information elements	
Description	Backlog items consist of additional relevant information aside from the user story, such as acceptance criteria & business value of the item. The team creates and alters this template according to their needs.
Prerequisite(s)	None
Reference(s)	r22, [12,25,44], *changelog v1*

5.3.8 Refining requirements

Refining requirements is the addition of details and expansion of requirements after the initial elicitation. Although not an official Scrum activity, requirements are often refined in a *refinement meeting*. Refinement sessions are vital to ensure quality of requirements and the shared understanding of the team. The goal of these sessions is to add detail to backlog items. The case study showed that teams often have 1-3 refinement meetings per sprint. Therefore, the first capability states that a team should have at least one refinement meeting per sprint.

A. Regular refinement meetings	
Description	Refinement meetings occur at least once per sprint. Backlog items that are not yet ready are refined by adding detail/missing information, divide a PBI into tasks, re-prioritization, et cetera.
Prerequisite(s)	None
Reference(s)	r10, r13, r15

The second capability involves the definition of ready. The use of a definition of ready allows a team to easily identify work items that are not yet ready to be worked on and hence, need to be further refined. What the definition of ready should define is something for teams themselves to determine according to their needs fitting their context. Having a definition of ready was recommended as a best practice in the work of Heck & Zaidman [25].

B. Definition of ready	
Description	The team uses a definition of ready to identify backlog items that are ready to enter a sprint and items that need to be further refined.
Prerequisite(s)	None
Reference(s)	[25,63]

The third capability was based on similar principles as estimating using a baseline. It is desirable that work items that are too big are refined into smaller tasks. If an item is estimated at a certain amount (e.g. >13 story points) of work, the team should decide whether the item is too big to work on in a sprint. This item should then be divided into smaller items by the team. The use of a baseline was suggested by expert B during the validation of the model. Naturally, this can only be done if items are actually estimated, and the capability therefore has a dependency on *Estimating effort : A*. Another advantage of keeping requirements small is that it makes for a simpler, clearer and more readable requirements specification/backlog [27].

C. Refining with a reference size	
Description	The team uses some reference point at which items are considered to be too ‘big’ to enter a sprint. The team tries to refine these items into smaller individual items.
Prerequisite(s)	Dependency on Estimating effort : A
Reference(s)	[27]

The final capability is based on the agile principle of ‘just enough, just in time’. It is desirable that a team has enough backlog items ‘ready’ to enter the upcoming sprint. If there are enough items, the team should ensure to not keep blindly refining items

simply because not *all* items in the backlog are ready. If for example, the team refines enough items for the upcoming four sprints and then the prioritization or wishes of stakeholders changes all this refinement effort could be wasted. This capability was suggested by expert C during the validation.

D. Refining just enough, just in time	
Description	The team ensures to refine enough items for the upcoming sprint.
Prerequisite(s)	None
Reference(s)	*Changelog v0.5*

5.3.9 Acceptance criteria

Acceptance criteria document criteria which a PBI must adhere to. Many teams use these to validate their PBI's. Not all teams use these, 16 out of 32 teams explicitly stated they use acceptance criteria for each PBI. Acceptance criteria are usually added during refinement sessions. Acceptance criteria are unique for a given PBI. The case study showed that acceptance criteria are used in some manner by half of the teams (16 out of 32). Some teams describe acceptance criteria as vital (e.g. r11). In some cases, acceptance criteria are defined by the customer (or product owner) and in some cases by the development team itself. There is no clear consensus which is better. However, the maturity model by Patel & Ramachandran also includes a criteria specifically stating the use of acceptance criteria [44]. Acceptance criteria are also used to validate with the customer that a requirement is correctly implemented.

A. Acceptance criteria	
Description	Acceptance criteria are defined for PBI's.
Prerequisite(s)	None
Reference(s)	r11, [44]

There are several ways acceptance criteria can be added. Specification by example is a popular method to perform acceptance criteria using examples as a source of truth, instead of having different kinds of documentation created by business analysts and developers separately. Specification by example was suggested by the focus group as a great practice that fit in between simple acceptance criteria and the step toward automated acceptance testing.

B. Specification by example	
Description	Acceptance criteria are defined using specification by example.
Prerequisite(s)	None
Reference(s)	e1, t2, t3, t4, *changelog v1*

Some teams apply acceptance criteria in simple natural language, while others go as far as automatic testing. An automatic testing format used for acceptance tests is based on the BDD template 'given *[a condition]*, when *[an event occurs]*, then *[outcome]*'. This can be tested automatically by writing tests and attaching these to the code. Some teams go beyond textual acceptance criteria and use the formal language Gherkin (e.g. r1), which is a domain specific language that should also be understandable for non-technical stakeholders. The main idea is to allow the binding of acceptance criteria to the code, allowing for automating acceptance tests. Starting with the creation of formal acceptance criteria such as feature files is the first step toward acceptance test-driven development.

C. Formal acceptance criteria	
Description	Acceptance criteria are defined using a formal syntax, such as Gherkin
Prerequisite(s)	None
Reference(s)	[67], r1

The focus group had some issue with the distinction between capability C and D. The experts were rather surprised that there are teams who perform capability C without also performing D, as a team should create formal acceptance criteria because they want to do automate their acceptance testing. Only performing C seems like a waste of effort. However, after some discussion the experts agreed that only performing C does provide some other benefits aside from allowing to progress to capability D. Therefore, these capabilities have not been merged, but the experts highly recommend to achieve level D when the team achieves level C.

D. Acceptance test-driven development	
Description	Acceptance criteria are automatically tested whenever possible by using acceptance test-driven development
Prerequisite(s)	None
Reference(s)	[67]

5.3.10 Prioritizing requirements

Prioritizing requirements is the process of identifying the most valuable requirements from a set of requirements [68]. The literature already established that a key aspect of agile RE is its continuous reprioritization [22]. During the interview with the agile expert (e1) the expert mentioned that prioritization should be done based on business value, but teams should keep in mind that non-functional requirements provide business value as well. Other reports mention prioritization is always a tradeoff between the effort and value of a story. In technical projects prioritization is often less of an important consideration as technical dependencies often make it clear which work needs to be done first (t4). This contrasts with functional requirements, in which a stakeholder will often have a preference which is implemented as soon as possible.

Dedicated prioritization methods such as MoSCoW are rarely used (only in r18), with teams usually relying on experience, gut-feeling and expert opinions. One report stated that the team prioritizes based on the story points estimated for PBI's (r15). The first capability only requires that backlog items are prioritized, regardless of the method. The product owner is responsible for the prioritization and does not need to include the team or built a proper argument for his/her prioritization. Prioritizing requirements was also a best practice recommended in the work of Hofmann & Lehner [5].

A. Basic prioritization	
Description	PBI's are prioritized by the product owner without proper argument to the team.
Prerequisite(s)	None
Reference(s)	*Changelog v1,* [5]

While literature showed that prioritization in agile RE is one of its key concepts it does not specify which prioritization techniques are used [12]. Some literature did however mention prioritization should not be overly focused on only delivering 'business value', as this may result in a lack of attention toward architectural or technical quality [12,30]. Other literature claims that requirements with the most business value must be developed first [26,44]. The focus group determined that requirements should be prioritized based on a proper argumentation, regardless of the value a team gives to business value (this can differ, is it functionality? little risk high reward? low costs?). An example mentioned in literature is involving the Cost of Delay. This concept assumes that most requirements also have some sort of cost when they are delayed. For example; delaying the adoption of a new architecture, which might not provide a lot of business value straight away, can be increasingly costly as adoption is delayed. The new architecture might provide new opportunities which are not directly measurable in business value.

B. Prioritization based on some concept	
Description	The prioritization of PBI's is based on a proper argument, which can include concepts such as business value, the cost of delay & the results from previous sprint
Prerequisite(s)	None
Reference(s)	[44,69], c*hangelog v1*

The focus group discussed that the team should be included as well in the prioritization. A product owner ignoring advice or feedback from their team is a clear sign of immaturity according to the focus group. While the product owner has the final responsibility over the backlog and its prioritization the PO should include the team in the prioritization as well. The focus group did not specify how this should be achieved, but it can be done during refinement meetings.

C. Team-based prioritization	
Description	Prioritization is a co-production with the development team.
Prerequisite(s)	None
Reference(s)	*changelog v1*

The final capability was determined by the focus group as well. The group stated that a good prioritization should be justifiable with facts and numbers. For example, costs have been calculated, risks are predicted. Naturally, not every backlog item needs to have such a justification. However, for large items, features or epics these calculations can and should be done.

D. Substantiated prioritization	
Description	The prioritization is made justifiable/measurable with facts and numbers, like the costs, value, risks and impact of the items.
Prerequisite(s)	None
Reference(s)	*changelog v1*

5.3.11 Estimating effort

Estimating effort allows teams to judge the complexity of backlog items. Additionally, it aids the team with sprint planning. Therefore, a backlog item should always be estimated in some manner. Capability A assumes that estimation is always done, regardless of the method (e.g. planning poker, expert opinion), unit of estimation (e.g. work hours, story points) and actors involved (e.g. the entire team, just the product owner).

A. Estimating of backlog items	
Description	The required effort for backlog items is always estimated.
Prerequisite(s)	None
Reference(s)	r10, r11, r28

The estimation of a PBI is often done during refinement sessions. As with prioritization, it should be a team activity in which discussion about the solution/implementation of the PBI takes place, as this gives members insight into how much work will go into it and better understanding of the requirement. Having team-based estimation is a good way to ensure the estimations are more accurate as team members get to discuss their rationale behind a certain estimation. The scrum guide also specifically states that the development team is responsible for estimates [2].

B. Team-based estimations	
Description	All team members participate in estimations of backlog items, as the resulting discussions increase shared understanding.
Prerequisite(s)	None
Reference(s)	[2]

There are different methods used to estimate the workload of a PBI. The most common method is by far planning poker (29 teams), which is valued for its accuracy, light weight (short time/effort) and contributing to the shared understanding of the team on the implementation. Other methods are expert opinion (4 teams) or t-shirt estimation (1 team). There was one team that always assigned the same amount of work to a PBI, so therefore had no estimation process. Capability C values the use of light-weight estimation techniques, as teams value the shared understanding a technique like scrum poker creates and not whether every items' estimation is 100% correct.

C. Light-weight estimation	
Description	Teams use a consistent, light-weight technique such as planning poker to estimate backlog items.
Prerequisite(s)	None
Reference(s)	

The final capability D was added during the validation by expert B. The expert stated that when estimating it is very important that the team uses some reference backlog item or baseline which estimations are based on. Otherwise, everyone can have their own interpretation of what a 'work hour' or 'story point' means. This is not a difficult capability to achieve, but according to the expert, is something that teams often don't do.

D. Baseline estimations	
Description	Teams use a baseline to estimate. Using a baseline (e.g. always estimate based on a specific item) aids in shared understanding.
Prerequisite(s)	None
Reference(s)	*Changelog v0.4*

5.3.12 Validating requirements

After the development team has implemented a requirement, it needs to be validated. Is the requirement implemented in a way that satisfies the stakeholders' wish?

Reality shows that some teams do not take validation as serious. Some teams in the case study do not validate their implemented requirements at all (e.g. r26). Naturally, not all requirements are as easy to validate as others. Functional requirements can be validated in cooperation with stakeholders using a simple demo or prototype. Technical requirements which cause 'invisible' changes are not so simply verified, and teams validate these based on expert opinions, using acceptance criteria, unit tests and code coverage.

The first capability states that the product owner validates the implemented requirements. As the person responsible for the backlog, the PO can check whether the implemented requirements meet the wishes of the customer/stakeholders.

A. Product owner validation	
Description	Validation is done by the product owner by checking whether the described functionality has been established.
Prerequisite(s)	None
Reference(s)	r18, r21, r2

The second capability concerns using a definition of done. Having a uniform definition of done for all backlog items is important according to the official Scrum guide [2]. The definition of done defines when a PBI is finished and requires no more work. It is simply a list of criteria which must be met before a product increment is considered to be completed, just as how a definition of ready is a list of criteria which must be met before a PBI can enter a sprint. Typically, the Definition of Done is created by the development team.

B. Validating with a Definition of Done	
Description	The team has established is a definition of done which is identical for all PBI's. An item is validated when it satisfies to the criteria in the definition.
Prerequisite(s)	None
Reference(s)	r3, r19, r7

A mature team should use acceptance criteria tailored to individual items to decide whether they have been implemented correctly. Because acceptance criteria are a focus area on their own, this capability has a dependency on *Acceptance criteria : A*. Hence, this capability only serves to assert the maturity of the validation process and not the maturity of the use of acceptance criteria.

C. Validating with acceptance criteria	
Description	Validation is supplemented by using acceptance criteria for PBI's.
Prerequisite(s)	Dependency on Acceptance criteria : A
Reference(s)	r2, t1, t4, r10, r14

According to literature, validation is often done with review meetings and acceptance tests [26]. By delivering working software, stakeholders can often easily verify whether the implemented requirement is in accordance with their expectations. Review meetings are an official Scrum event and should be present at the end of each sprint [2]. Capability D states that teams should include the customer and other relevant stakeholders to check their satisfaction with the release. A common way to achieve this are review meetings, in which some sort of demo or prototype can be presented to the stakeholder. This type of validation allows for immediate feedback. According to the focus group the capability should not specifically demand 'review meetings' as they could see scenarios in which review meetings would be turned down. For example; in projects with fast release cycles (e.g. every day) review meetings might be done only once every sprint while there are 14 releases in the same sprint. Reviewing all work released in 14 releases in a single review meeting is difficult, and should therefore be validated by the customer/stakeholder in live production. Because of this, there is a capability E which states that customers/stakeholders validate items live or in a test environment without having to schedule team meetings. Validating with stakeholders is a recommended best practice in the work of Hofmann & Lehner [5].

D. Validating with customer/stakeholder	
Description	Validation is done by including the customer or other stakeholders to check their satisfaction with the release/sprint. There are review meetings in which the team and important stakeholders discusses which items are done.
Prerequisite(s)	None
Reference(s)	[26], r15, r21, r22, *changelog v1*

E. Validating in production	
Description	The customer/stakeholder checks all PBI items in the production environment or test environment.
Prerequisite(s)	None
Reference(s)	*changelog v1*

It should be noted that many teams in the case study also test for the quality of the written code or code coverage, but this is not considered to be within the scope of requirements engineering.

5.3.13 Traceability & Documentation

As was found in literature and confirmed by the case studies, traceability is a topic that gains little attention from teams [70]. The majority of teams ignores the topic, some teams trace code back to the original backlog item (e.g. r6), and some have a wiki or other format in which they document decisions (e.g. t3). The lowest level of traceability that originated from the case study was the tracing of code back to the original backlog item/requirement it was written for. Expert A stated during the validation that dependencies between stories should be traced as well, and one team was found to actually do so (r20). For example, when several stories combine toward one specific requirement this should be made visible. In that case, the team can quickly trace possible issues that occur with stories when something needs to be changed.

A. Code to PBI tracing and inter-story dependencies	
Description	Code can be traced back to its original backlog item. Dependencies between stories are traced.
Prerequisite(s)	None
Reference(s)	r6, r20, *changelog v0.3,*

The second capability is based on how teams document information such as architectural designs, functionalities of the software and release changelogs. An example is the use of a wiki with some kind of format, like Arc42. The team must be aware of different types of documentation they may have to create. The agile@re handbook identifies different kinds of documentation [57]. Most documentation, like the backlog is merely documentation for communication purposes, to communicate what work needs to be done. As mentioned by Paetsch et al. models are often still used in agile RE, but are quickly drawn for some discussion and then discarded [26]. This documentation is known as documentation for thinking purposes. However, two types of documentation warrant more formal documentation; legal documentation and preservation documentation. Some preservation documentation techniques were encountered in the case study like the use of a wiki that the team contributes to during development (t4, r7).

B. Basic documentation	
Description	The team is aware what kinds of documentation needs to be created. has a wiki or other format of documentation containing information on releases, functionality & definitions.
Prerequisite(s)	None
Reference(s)	t4, r7

The use of a traceability matrix was recommended by Hofmann & Lehner as a best practice [5]. Having a medium cost of introduction, such a matrix allows a team to track

a requirement from the moment it originates through its specification and eventual implementation.

C. Traceability matrix	
Description	The team uses a traceability matrix
Prerequisite(s)	None
Reference(s)	[5]

The final capability is the use of living documentation. Living documentation is documentation based on automatic testing and can therefore update itself with each release. In doing so, it provides both traceability as it will automatically show if tests have failed and documentation that requires little work once established. Feature files, for example, are files that contain Gherkin acceptance criteria and scripts to automatically test the criteria.

D. Living documentation	
Description	Teams create living documentation by following BDD practices, for example with the use of feature files.
Prerequisite(s)	None
Reference(s)	*changelog v0.3, changelog v0.4*

5.4. The maturity matrix

The validated maturity matrix can be seen in table 9 below. A detailed description of the evolution of the model during the validations and focus group can be found in appendix 9.3. The 56 capabilities are distributed over 13 focus areas (horizontal rows) and 10 maturity levels (vertical columns). The amount of maturity levels (10) is somewhat arbitrary; it ensures a relatively good spread of the capabilities and allows the dependencies to be correctly reflected in the model. The first three maturity levels contain mostly level A capabilities; 13 out of 15 are A capabilities, 2 out of 15 are B capabilities. Levels four and five are mostly B capabilities; 10 out of 12 are B capabilities, 2 out of 12 are C capabilities. Maturity levels six and seven are mostly C capabilities; 9 out of 11 are C capabilities, 1 out of 11 is capability B, 1 out of 10 is a D capability. The final maturity levels eight to ten are mostly D and E capabilities; 12 out of 18 are D, 4 out of 18 are E, 2 out of 18 are C. Table 10 shows the maturity matrix with some interesting statistics.

Scrum based Requirements Engineering maturity matrix										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Management & Planning										
Roadmapping			A		B		C		D	
Stakeholder involvement	A		B		C			D		E
Sprint planning	A			B			C		D	E
Ideation										
Goals & Vision	A					B		C		D
Eliciting requirements		A		B		C				D
Specification										
Documenting requirements	A			B		C		D		E
Templates			A		B		C		D	
Refining requirements		A			B		C			D
Acceptance criteria		A		B				C	D	
Implementation										
Prioritizing requirements	A			B		C		D		
Estimating effort			A	B		C				D
Validating requirements	A		B		C		D		E	
Traceability & Documentation			A		B		C		D	

Table 9: The validated maturity model

Matrix with stats v1											
Focus area	**Maturity level**										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Capabilities per focus area**
Management & Planning											
Roadmapping			A		B		*C*		*D*		4
Stakeholder involvement	A		B		C			D		*E*	5
Sprint planning	A			**B**			**C**		**D**	E	5
Ideation											
Goals & Vision	A					**B**		C		D	4
Eliciting requirements		A		B		C				**D**	4
Specification											
Documenting requirements	A			**B**		**C**		D		E	5
Templates			A		B		C		D		4
Refining requirements		A			B		**C**			D	4
Acceptance criteria		A		B				C	D		4
Implementation											
Prioritizing requirements	A			B		C		D			4
Estimating effort			A	B		C				D	4
Validating requirements	A		B		**C**		D		E		5
Traceability & Documentation			A		B		C		D		4
Number of capabilities per maturity level	6	3	6	6	6	5	6	5	6	7	**Total = 56 capabilities**
											Average capabilities per focus area = 4.30 **Average capabilities per maturity level = 5.6**

Table 10: The maturity matrix with statistics and dependencies visualized

6. Applying the model

Applying the model is another way to highlight any missed problems with the model and to discover any insights a filled in matrix can offer. Due to time constraints the model is applied to only two teams, making it impossible to discover any trends or patterns for our dataset. Also, due to the lack of an assessment model and established metrics for individual capabilities applying the model was done in a semi-structured interview with the researcher present to explain the capabilities if necessary.

6.1. Team A

Team A achieved 44 out of all 56 capabilities. The team achieves all capabilities in 5 out of 13 focus areas; Stakeholder involvement, refining requirements, prioritizing requirements, estimating effort and validating requirements. Interesting is the fact that there are three capabilities achieved which should be dependent on capabilities occurring earlier in the focus area. These capabilities have been indicated with yellow, indicating that they are achieved but strictly adhering to the rules of a maturity matrix shouldn't be achieved. For sprint planning, the team achieves capability E (*Verification*

of the sprint backlog) and not capability D (*Having a definition of ready*). For eliciting requirements, the team achieves both capability C (*Combining disciplines*) and D (*Efficient elicitation*) while not achieving capability B (*Elicitation techniques*). The highest maturities achieved (10) are capability E in *stakeholder involvement,* capability D in *refining requirements*, capability D in *Prioritizing requirements,* capability D in *estimating effort* and capability E in *validating requirements*. The lowest maturity achieved (3) is capability A in *eliciting requirements*. The average maturity level of team A (when ignoring 'yellow' capabilities) is 8.

Matrix v1										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Management & Planning										
Roadmapping			A		B		C		D	
Stakeholder involvement	A		B		C			D		E
Sprint planning	A			B			C		D	E
Ideation										
Goals & Vision	A					B		C		D
Eliciting requirements		A		B		C				D
Specification										
Documenting requirements	A			B		C		D		E
Templates			A		B		C		D	
Refining requirements		A			B		C			D
Acceptance criteria		A		B				C	D	
Implementation										
Prioritizing requirements	A			B		C		D		
Estimating effort			A	B		C				D
Validating requirements	A		B		C		D		E	
Traceability & Documentation			A		B		C		D	
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

Table 11: The filled in matrix for team A

6.2. Team B

Team B achieves 38 out of the 56 capabilities. The team achieves all capabilities in only one out of 13 focus areas; estimating effort. Similar as with team A, there are several instances where a capability in a focus area is achieved without achieving a prior one. As with team A, for sprint planning, the team achieves capability E (*Verification of the sprint backlog*) and not capability D (*Using a definition of ready*). For templates, the team achieves capability C (*Benefit argumentation*) and D (*Additional information elements*) without achieving capability B (*Distinction between functional/non-functional requirements*). For refining requirements, the team achieves capability C (*Refining with a reference size*) and D (*Refining just enough, just in time*) without achieving capability B (*Having a definition of ready*). The highest maturity achieved (10) is capability D in

estimating effort. The lowest maturity achieved (3) is capability A in acceptance criteria. The average maturity level of team B is 6.8.

Matrix v1										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Management & Planning										
Roadmapping			A		B		C		D	
Stakeholder involvement	A		B		C			D		E
Sprint planning	A			B			C		D	E
Ideation										
Goals & Vision	A					B		C		D
Eliciting requirements		A		B		C				D
Specification										
Documenting requirements	A			B		C		D		E
Templates			A		B		C		D	
Refining requirements		A			B		C			D
Acceptance criteria		A		B				C	D	
Implementation										
Prioritizing requirements	A			B		C		D		
Estimating effort			A	B		C				D
Validating requirements	A		B		C		D		E	
Traceability & Documentation			A		B		C		D	
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

Table 12: The filled in matrix for team B

Filling in these models in practice with a team member gave several lessons that should be addressed in the future. For some capabilities it is easy to establish whether they are achieved. An example is capability A from *templates*. All that needs to be checked is whether a template is used. For other capabilities it is very difficult, like capability E from *Eliciting requirements*. A next step for the model would be to establish clear criteria for capabilities to determine whether they are achieved.

Additionally, in its current state the capabilities can still be interpreted incorrectly by team members. When an assessor is present with good understanding of the model (e.g. the author) this is not a big issue. However, in order to assess a large number of teams it would be easier to have teams self-assess their capabilities using an assessment instrument and descriptions of capabilities should be unambiguous or supplemented with clear metrics or satisfaction criteria.

Finally, an inherent principle of the maturity matrix is that capabilities later in a focus area are dependent on capabilities occurring earlier in said focus area. However, this dependency is not always enforced in the real world, i.e. it is entirely possible (and already happened several times for only two teams) that capabilities

are achieved by a team without achieving the capability they have a dependency on. While the validation tried to ensure that capabilities within a focus area show the ‘preferred’ way of improving in a focus area according to the experts these scenarios do occur. This also occurred in the work of Gorschek et al, where not a single team was able to achieve maturity level 1 yet many teams had some capabilities achieved in higher maturity levels [54].

7. Conclusion

This chapter serves to answer the research questions established in chapter 1.

RQ1: What are common RE practices in Scrum used by software development teams?

To answer the first research question a literature review was performed. The review discussed common RE practices in Scrum, as well as the benefits and challenges of applying RE in Scrum. Traditional RE used to be performed prior to the actual software development, but in Scrum RE and development are practiced simultaneously during the entire lifetime of the product/system. Agile RE is characterized by evolving requirements and constant reprioritization. A key concept of Scrum is that teams are self-organizing and should apply best-practices where needed. Literature clearly showed that it is unclear which RE practices are 'best-practice', which is reflected by the fact that many studies looking into Scrum RE in practice had low frequencies and a high variance of different practices. In agile RE, there are several studies discussing the quality of agile requirements and techniques/frameworks to ensure quality. Quality frameworks, such as INVEST, are used in some form in a majority of the teams analyzed in studies. The user story is by far the most commonly used technique to document functional requirements, although templates differ. While agile RE has benefits, such as lower times to market, better customer involvement and better suited for dealing with volatile requirements, there are also significant challenges noted by several studies. These studies often reported similar problems such as a lack of customer involvement (or no onsite customer), negligence of non-functional requirements, lack of documentation, insufficiency of the user story format to capture all relevant information and estimation issues. As one study noted, there is a relationship between a team's maturity and patterns of requirements problems.

Because the literature was found to be lacking either due to contradicting findings between studies and the small sample sizes a multiple case study was performed on 32 software teams. We concluded that there were five main categories of activities, processes or artifacts in which RE plays a big part in Scrum. These are;

- *Management & planning*, which covers focus areas that describe capabilities that influence the management and planning activities within agile RE.
- *Ideation*, containing everything related to the extraction of an idea or wish to be able to formulate a requirement.
- *Specification*, which has a strong focus on all that is involved in the process of specifying requirements before they can be built.
- *Implementation*, covering all that is involved in correctly implementing and validating requirements.

The findings of this research question were used as input for the creation of the maturity matrix.

RQ2: How can RE maturity in Scrum be defined?

Several existing studies have been discussed that try to define RE maturity by either proposing some best practices or defining maturity by designing maturity models. There are maturity models that focus on agile/Scrum or requirements engineering, but none combine the two topics. Some models that focus on agile do incorporate some topics/techniques that relate to requirements engineering. The maturity models that were discussed in this study that focus on requirements engineering all (except for one) focused on traditional RE and do not consider how agile methods changed RE, making them inapplicable to agile RE. An important finding is that defining maturity is very dependent on the context of a software development team. This makes it difficult to determine whether a team is mature based only on the judgings of a maturity model. A complicated process (like acceptance-test driven development) may be a sign of a mature team but that does not mean that a team that deliberately chooses not to implement ATTD is less mature, as their different contexts can justify both decisions. The important take away is that a mature team is *aware* of processes and artifacts it can implement and can choose the right capabilities to achieve that will benefit them the most.

RQ3: How can RE maturity be measured and compared among different software development teams?

In order to measure and compare RE practices in Scrum maturity models are a useful tool to assess the current practices in a team, to compare teams with one another and to guide teams toward adopting good practices. Two types of maturity models were considered for this study; *fixed-level* maturity models and *focus area* maturity models. Fixed-level maturity models often consist of several levels of maturity and each level is associated with several processes that need to be implemented/conformed to. A limitation of these models is that dependencies between processes are unclear and therefore provide little guidance on the order in which processes should be implemented. Focus area maturity models consist of activities and a maturity level for each activity. The difference between the two models is that fixed-level models have global descriptions of maturity are described for each level, there are no individual descriptions for each activity at each maturity level, whereas a focus area model identifies process areas and activities with their individual maturity level. All models we encountered (both agile and RE focused) are fixed-level models. These are limited due to the fact that growth is not as linear as they describe and dependencies between capabilities are unclear. Therefore, we opted for the design of a focus area maturity model using the findings of the literature study and the case study.

RQ4: Is the constructed artifact valid?

The maturity matrix was validated on an iterative basis using expert interviews. The initial version was discussed with an expert on agile RE who proposed issues and improvements. After four iterations, the model and unresolved issues were presented to a focus group. The validation with the focus group resulted in the finalized version of the maturity matrix. As all improvements have been implemented and no more issues resulted during the focus group we can say that the matrix is valid. However, two test assessments of the model on actual software teams brought an issue forward. An inherent principle of the maturity matrix is that capabilities later in a focus area are dependent on capabilities occurring earlier in that focus area. However, this dependency is not enforced in the real world, i.e. it is possible and already happened several times that capabilities are achieved by a team without achieving the capability they have a dependency on. While the validation and experts tried to ensure that capabilities within a focus area show the 'preferred' way of improving in a focus area these aforementioned scenarios do occur.

MRQ: How to develop a maturity matrix for RE practices in Scrum?

This study developed a maturity matrix by executing several steps. First, a literature study was performed to gain an understanding of agile RE by answering RQ1 and RQ2. Due to the literature being unclear on common RE practices and processes an additional multi-case study was performed. With RQ1 and RQ2 answered, we opted to design a focus area maturity model. The actual design of the model was done by answering RQ3. Focus areas and capabilities were based on the case study and literature. With the designed model, there were several validation rounds in order to answer RQ4. Each research question served as a step to develop a maturity matrix, resulting in the following development method (figure 25).

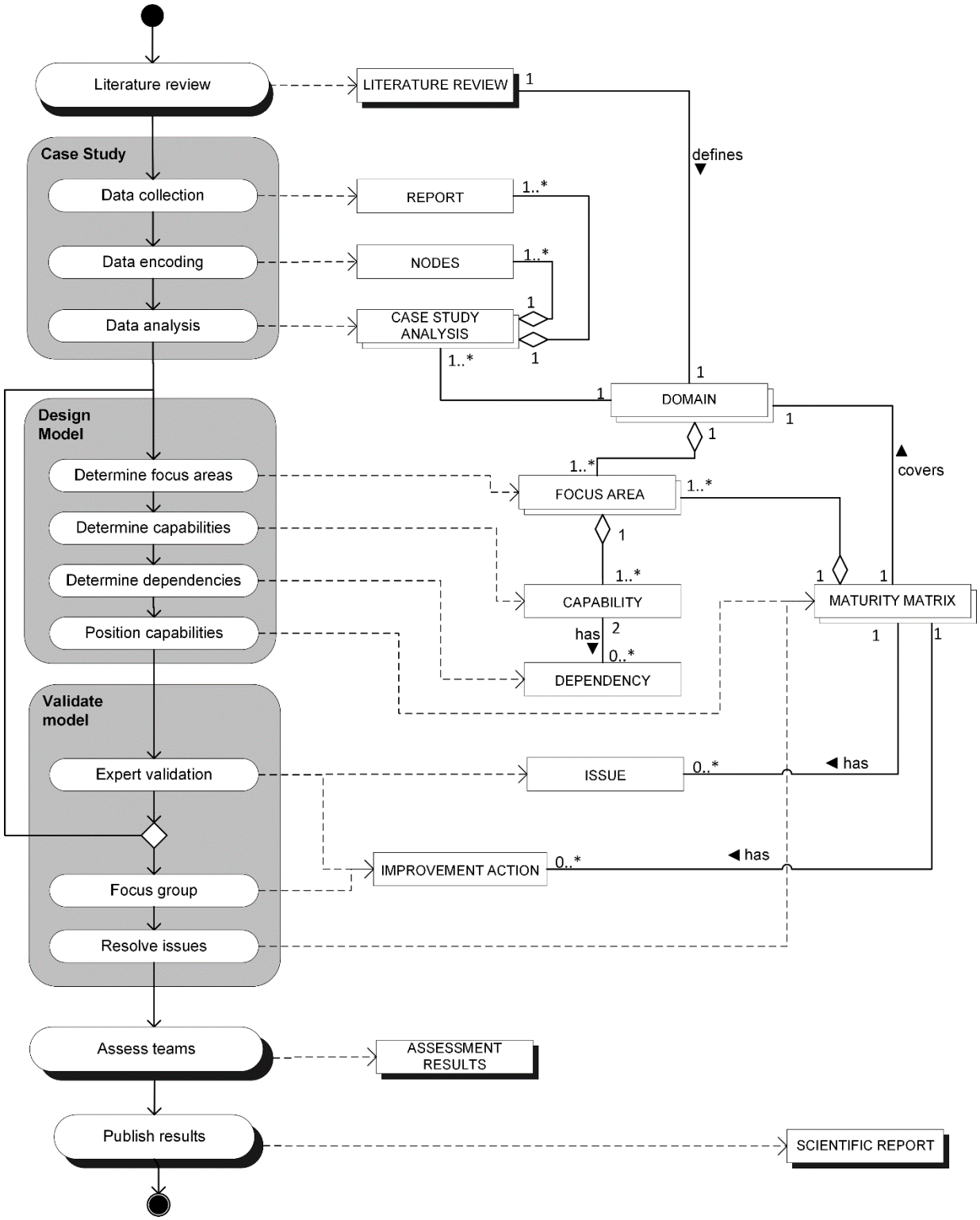


Figure 25: Development method for a maturity matrix for agile RE in Scrum

8. Discussion

8.1 Threats to trustworthiness

Validity and reliability are important concepts in quantitative research. In qualitative research, the concepts of validity and reliability cannot be addressed in the same manner [71]. With over 6400 citations, four well known and accepted criteria that should be considered for qualitative research over validity and reliability were proposed in the work by Guba [72]. These are:

- Credibility
- Transferability
- Dependability
- Confirmability

We will discuss these criteria and if threats toward these criteria are present in this study.

Credibility

Credibility is mostly aligned with internal validity. One suggestion by Lincoln & Guba that is not adhered to was random sampling of individuals. In this study, sampling occurred on the basis of convenience. This may result in a slight bias, as the daily supervisor often proposed useful individuals since the author was unfamiliar with the host company. However, this convenience sampling only occurred on 4 out of the 32 teams that participated. For the other 28 teams it is unknown how their authors sampled a participating team for their report. It is likely some authors had some sort of connection to a person or company, but this is out of the sphere of influence of the author.

Transferability

Transferability is mostly concerned with the generalizability of the study, similar to external validity. We tried to ensure the generalizability of the model by having a large and varied sample size. The software teams included in the case study come from a large number of organizations. These organizations are of varied size (from 10 to 10.000+) and they are different types of organizations (from startups to multinationals). For some of the participating organizations core business is involved with software development and for other organizations software development is merely a side activity to support their core business. However, the results of a qualitative study must always be understood within the context in which the fieldwork was performed [72]. The designed model should be applied to more teams who did not participate in the study to further examine its transferability.

Dependability

The chosen method to validate the constructed artifact of expert opinion has a threat; this validation method only works if the expert has a good understanding the artifact [8]. The reliability of the experts was ensured by choosing experts who had experience with RE related roles, such as product owners or business analysts, who had operated in an agile context. However, the true expertise and quality of an expert is hard to determine [73]. Having stricter criteria, e.g. having at least 5 years of experience, would have had a negative impact on finding experts willing to participate.

Confirmability

Confirmability is mostly concerned with the objectivity of measurement instruments and serves to ensure that findings of the study the result of the experiences and ideas of the informants, instead of the researcher's characteristics and preferences [71]. A technique to reduce the effect of investigator bias is triangulation. In this study, triangulation was applied by using both expert opinions and focus groups, as their distinct characteristics result in individual strengths. The combination of different methods compensate for their individual limitations and exploits their respective benefits [71]. The use of methods was also influenced by the available case study of other students. In their case studies, the students collected their data by means of an expert interview. In order to combine the case studies performed by the author and the students the case studies at Info Support were structured and performed in a similar way to that of the students' case study.

There are several other limitations of this study. A limitation of the dataset is that it is very likely that teams are included that claim to work with Scrum but in reality omit many principles and practices of the method. This phenomenon is often referred to as 'Scrum in name only' [74,75]. The phenomenon is described as Scrum anti-patterns in the work by Eloranta et al.; *"The practical implementation of Scrum in companies rarely follows the text book ideals. Typically, companies deviate from the proposed Scrum practices for different reasons. While some deviations may be well motivated and reasonable, companies are often tempted to adjust Scrum for the company without clearly understanding the consequences of the deviations"* [74]. It is hard to determine how many teams in our dataset apply Scrum anti-patterns and to what extent these patterns have a negative or positive impact on their RE practices. In the future, case studies should take into consideration what these anti-patterns are as to recognize teams which maybe should be omitted from the dataset.

Another risk of this study is the risk of over-generalizing the model. By using a broad dataset we try to fit all types of teams in the model with all kinds of different projects (e.g. webdev, high-risk projects, technical projects) and different kinds of companies (software products vs SaaS). It is possible that there are teams for who the model is not applicable at all, for example in safety-critical projects. This is something that could be further tested by applying the model on a much larger dataset.

8.2. Future research

First and foremost, future research could try and expand on the existing model. This research focus only on the maturity of RE in scrum and it could be integrated in a bigger model in scrum which contains other important capabilities, for example in the field of the actual development or testing. Maturity models for Scrum already exist and were discussed in chapter 3.2.2. However, none of these are focus area maturity models.

A limitation of this study is that the resulting maturity model is only applied to two teams (due to time constraints). Not only does this make it impossible to discover any trends or patterns for our dataset, it also means that more improvements for the model can be missed. The two assessments of the teams already give some striking insights (chapter 6). In this maturity model all focus areas and capabilities are given the same importance. However, it is very likely that some capabilities make a bigger impact than others, and should be prioritized. Capabilities can and perhaps should be given a weight, in order to better compare teams. For example, a team that has many capabilities of lesser importance will currently end up with a higher average maturity level than a team that has achieved some very important capabilities yet overall achieved less capabilities. It is unclear to what extent it is possible to calculate some maturity score by having weight per capability and to compare some overall maturity scores of teams. Or perhaps the difference in context and team preferences has such a big impact that maturity between teams can never be accurately be compared.

The lack of many assessments of the model on actual teams is also due to the fact that the development method by van Steenbergen et al. was only followed up to the activity of the actual development [35]. The remaining activities *Instrument development* and *Implementation & Exploitation* were not performed. In a follow-up study, these activities should be performed. The development of an assessment instrument can establish some clear metrics for when capabilities are achieved, allowing team members to more easily assess their own team and not rely on an expert of the model to do the assessment. We discussed some studies with measurement models in section 3.2.3. with models like the ones produces Niazi or Sommerville but did not have time to set up such a model [50,53]. Establishing a clear measurement framework could be a follow up study in the future and is a necessary step for an assessment instrument that can be used without having to include an expert on the model.

Originally, there were two other research questions (RQ5 and RQ6) which have been removed from this study due to a lack of time. These questions should be answered in a follow-up study. Once the maturity model is designed and validated, we originally wanted to use it to answer RQ5 "*In which ways does high maturity provide benefits?*" and RQ6 "*Should software development teams strive for high maturity?*". The idea was to apply the model on all teams in the dataset, but due to the fact we only applied it on

two teams the questions have not been answered. These questions should be answered in a follow-up study.

References

1. 13th Annual State Of Agile Report [Internet]. CollabNet VersionOne; 2019 May [cited 2019 May 14]. Available from: https://www.stateofagile.com/#ufh-i-521251909-13th-annual-state-of-agile-report/473508

2. Schwaber K, Sutherland J. The Scrum Guide™ [Internet]. 2017 [cited 2019 May 9]. Available from: https://www.scrumguides.org/scrum-guide.html

3. Diebold P, Ostberg J-P, Wagner S, Zendler U. What Do Practitioners Vary in Using Scrum? In: Lassenius C, Dingsøyr T, Paasivaara M, editors. Agile Processes in Software Engineering and Extreme Programming. Springer International Publishing; 2015. p. 40–51. (Lecture Notes in Business Information Processing).

4. Reddy A. The Scrumban [R]Evolution: Getting the Most Out of Agile, Scrum, and Lean Kanban. Addison-Wesley Professional; 2015. 519 p.

5. Hofmann HF, Lehner F. Requirements Engineering as a Success Factor in Software Projects. IEEE software 4. 2001;18:58–66.

6. Dick J, Hull E, Jackson K. Requirements Engineering. Springer; 2017. 254 p.

7. Hall T, Beecham S, Rainer A. Requirements problems in twelve software companies: an empirical analysis. IEE Proceedings - Software. 2002 Oct;149(5):153–60.

8. Wieringa RJ. Design Science Methodology for Information Systems and Software Engineering [Internet]. Berlin Heidelberg: Springer-Verlag; 2014 [cited 2019 May 8]. Available from: https://www.springer.com/gp/book/9783662438381

9. Dul J, Hak T. Case Study Methodology in Business Research. Routledge; 2007. 321 p.

10. Englander M. The Interview: Data Collection in Descriptive Phenomenological Human Scientific Research. Journal of Phenomenological Psychology. 2012 Jan 1;43(1):13–35.

11. Weerd I van de, Brinkkemper S. Meta-Modeling for Situational Analysis and Design Methods. Handbook of Research on Modern Systems Analysis and Design Technologies and Applications. 2009;35–54.

12. Heikkilä VT, Damian D, Lassenius C, Paasivaara M. A Mapping Study on Requirements Engineering in Agile Software Development. In: 2015 41st Euromicro Conference on Software Engineering and Advanced Applications. 2015. p. 199–207.

13. Schön E-M, Thomaschewski J, Escalona MJ. Agile Requirements Engineering: A systematic literature review. Computer Standards & Interfaces. 2017 Jan 1;49:79–91.

14. Hoda R, Salleh N, Grundy J. The Rise and Evolution of Agile Software Development. IEEE Software. 2018 Sep;35(5):58–63.

15. Beck K, Beedle M, Van Bennekum A, Cockburn A, Cunningham W, Fowler M, et al. Manifesto for Agile Software Development. 2001;10.

16. Augustine S, Payne B, Sencindiver F, Woodcock S. Agile project management: steering from the edges. Commun ACM. 2005 Dec 1;48(12):85–9.

17. Curcio K, Navarro T, Malucelli A, Reinehr S. Requirements engineering: A systematic mapping study in agile software development. Journal of Systems and Software. 2018 May 1;139:32–50.

18. Inayat I, Salim SS, Marczak S, Daneva M, Shamshirband S. A systematic literature review on agile requirements engineering practices and challenges. Computers in Human Behavior. 2015 Oct 1;51:915–29.

19. Medeiros JDRV, Alves D, Vasconcelos A, Silva C, Wanderley E. Requirements engineering in agile projects: a systematic mapping based in evidences of industry. ESELAW, CIBSE Ibero-American Conference on Software Engineering. 2015;460.

20. Lucassen G, Dalpiaz F, Werf JMEM van der, Brinkkemper S. The Use and Effectiveness of User Stories in Practice. In: Daneva M, Pastor O, editors. Requirements Engineering: Foundation for Software Quality. Springer International Publishing; 2016. p. 205–22. (Lecture Notes in Computer Science).

21. Wang X, Zhao L, Wang Y, Sun J. The Role of Requirements Engineering Practices in Agile Development: An Empirical Study. In: Zowghi D, Jin Z, editors. Requirements Engineering. Springer Berlin Heidelberg; 2014. p. 195–209. (Communications in Computer and Information Science).

22. Cao L, Ramesh B. Agile Requirements Engineering Practices: An Empirical Study. IEEE Software. 2008 Jan;25(1):60–7.

23. Ramesh B, Cao L, Baskerville R. Agile requirements engineering practices and challenges: an empirical study. Information Systems Journal. 2010;20(5):449–80.

24. Heck P, Zaidman A. A systematic literature review on quality criteria for agile requirements specifications. Software Qual J. 2016 Sep 15;26(1):127–60.

25. Heck P, Zaidman A. A framework for quality assessment of just-in-time requirements: the case of open source feature requests. Requirements Eng. 2017 Nov 1;22(4):453–73.

26. Paetsch F, Eberlein A, Maurer F. Requirements engineering and agile software development. In: WET ICE 2003 Proceedings Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. 2003. p. 308–13.

27. Medeiros J, Goulão M, Vasconcelos A, Silva C. Towards a Model about Quality of Software Requirements Specification in Agile Projects. In: 2016 10th International Conference on the Quality of Information and Communications Technology (QUATIC). 2016. p. 236–41.

28. Lucassen G, Dalpiaz F, Werf JMEM van der, Brinkkemper S. Forging high-quality User Stories: Towards a discipline for Agile Requirements. In: 2015 IEEE 23rd International Requirements Engineering Conference (RE). 2015. p. 126–35.

29. Lucassen G, Dalpiaz F, van der Werf JMEM, Brinkkemper S. Improving agile requirements: the Quality User Story framework and tool. Requirements Engineering. 2016 Sep 1;21(3):383–403.

30. Elghariani K, Kama N. Review on Agile requirements engineering challenges. In: 2016 3rd International Conference on Computer and Information Sciences (ICCOINS). 2016. p. 507–12.

31. Fernández DM, Wagner S, Kalinowski M, Felderer M, Mafra P, Vetrò A, et al. Naming the pain in requirements engineering. Empir Software Eng. 2017 Oct 1;22(5):2298–338.

32. Wendler R. The maturity of maturity model research: A systematic mapping study. Information and Software Technology. 2012 Dec 1;54(12):1317–39.

33. Fraser P, Moultrie J, Gregory M. The use of maturity models/grids as a tool in assessing product development capability. In: IEEE International Engineering Management Conference. 2002. p. 244–9 vol.1.

34. Paulk MC, Curtis B, Chrissis MB, Weber CV. Capability maturity model, version 1.1. IEEE Software. 1993 Jul;10(4):18–27.

35. van Steenbergen M, Bos R, Brinkkemper S, van de Weerd I, Bekkers W. The Design of Focus Area Maturity Models. In: Winter R, Zhao JL, Aier S, editors. Global Perspectives on Design Science Research. Springer Berlin Heidelberg; 2010. p. 317–32. (Lecture Notes in Computer Science).

36. Maier AM, Moultrie J, Clarkson PJ. Assessing Organizational Capabilities: Reviewing and Guiding the Development of Maturity Grids. IEEE Transactions on Engineering Management. 2012 Feb;59(1):138–59.

37. Turner R, Jain A. Agile Meets CMMI: Culture Clash or Common Cause? In: Wells D, Williams L, editors. Extreme Programming and Agile Methods — XP/Agile Universe 2002. Springer Berlin Heidelberg; 2002. p. 153–65. (Lecture Notes in Computer Science).

38. Schweigert T, Nevalainen R, Vohwinkel D, Korsaa M, Biro M. Agile Maturity Model: Oxymoron or the Next Level of Understanding. In: Mas A, Mesquida A, Rout T, O'Connor RV, Dorling A, editors. Software Process Improvement and Capability Determination. Springer Berlin Heidelberg; 2012. p. 289–94. (Communications in Computer and Information Science).

39. Sutherland J, Jakobsen CR, Johnson K. Scrum and CMMI Level 5: The Magic Potion for Code Warriors. In: Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008). 2008. p. 466–466.

40. Marcal ASC, Soares FSF, Belchior AD. Mapping CMMI Project Management Process Areas to SCRUM Practices. In: 31st IEEE Software Engineering Workshop (SEW 2007) [Internet]. Columbia, MD: IEEE; 2007 [cited 2019 May 28]. p. 13–22. Available from: http://ieeexplore.ieee.org/document/4402760/

41. A Guide to Scrum and CMMI®: Improving Agile Performance with CMMI [Internet]. CMMI Institute; 2016 Dec. Available from: https://cmmiinstitute.com/resource-files/public/marketing/document/a-guide-to-scrum-and-cmmi®-improving-agile-perfor

42. Glazer H, Dalton J, Anderson D, Konrad M, Shrum S. CMMI®or Agile: Why Not Embrace Both! [Internet]. Software Engineering Institute; 2008 Nov. Available from: https://resources.sei.cmu.edu/asset_files/TechnicalNote/2008_004_001_14924.pdf

43. Henriques V, Tanner M. A Systematic Literature Review of Agile and Maturity Model Research. Interdisciplinary Journal of Information, Knowledge and Management. 2017;12:53–73.

44. Patel C, Ramachandran M. Agile maturity model (AMM): A Software Process Improvement framework for agile software development practices. International Journal of Software Engineering IJSE. 2009;2:3–28.

45. Yin A, Figueiredo S, da Silva MM. Scrum maturity model. Proceedings of the ICSEA. 2011;20–9.

46. Ozcan-Top O, Demirörs O. Assessment of Agile Maturity Models: A Multiple Case Study. In: Woronowicz T, Rout T, O'Connor RV, Dorling A, editors. Software Process Improvement and Capability Determination. Springer Berlin Heidelberg; 2013. p. 130–41. (Communications in Computer and Information Science).

47. Sidky A, Arthur J, Bohner S. A disciplined approach to adopting agile practices: the agile adoption framework. Innovations Syst Softw Eng. 2007 Sep 1;3(3):203–16.

48. Fontana RM, Albuquerque R, Luz R, Moises AC, Malucelli A, Reinehr S. Maturity Models for Agile Software Development: What Are They? In: Larrucea X, Santamaria I, O'Connor RV, Messnarz R, editors. Systems, Software and Services Process Improvement. Springer International Publishing; 2018. p. 3–14. (Communications in Computer and Information Science).

49. Niazi M, Cox K, Verner J. A measurement framework for assessing the maturity of requirements engineering process. Software Qual J. 2008 Jun 1;16(2):213–35.

50. Niazi M. An Instrument for Measuring the Maturity of Requirements Engineering Process. In: Bomarius F, Komi-Sirviö S, editors. Product Focused Software Process Improvement. Springer Berlin Heidelberg; 2005. p. 574–85. (Lecture Notes in Computer Science).

51. Sommerville I, Sawyer P. Requirements Engineering: A Good Practice Guide. 1st ed. New York, NY, USA: John Wiley & Sons, Inc.; 1997.

52. Daskalantonakis MK. Achieving higher SEI levels. IEEE Software. 1994 Jul;11(4):17–24.

53. Sawyer P, Sommerville I, Viller S. Requirements process improvement through the phased introduction of good practice. Software Process: Improvement and Practice. 1997;3(1):19–34.

54. Gorschek T, Svahnberg M, Tejle K. Introduction and Application of a Lightweight Requirements Engineering Process. In 2003 [cited 2019 Jun 27]. Available from: http://urn.kb.se/resolve?urn=urn:nbn:se:bth-9641

55. Patel C, Ramachandran M. Story Card Maturity Model (SMM): A Process Improvement Framework for Agile Requirements Engineering Practices. Journal of Software. 2009 Jul;4(5):422–35.

56. Patel C, Ramachandran M. Best Practices Guidelines for Agile Requirements Engineering Practices. Handbook of Research on Software Engineering and Productivity Technologies: Implications of Globalization. 2010;1–14.

57. Aschauer B, Hruschka P, Lauenroth K, Meuten M, Rogers G. CPRE Advanced Level RE@Agile - Handbook [Internet]. International Requirements Engineering Board; 2018. Available from: https://www.ireb.org/en/downloads/#cpre-advanced-level-re-agile-handbook

58. Bekkers W, van de Weerd I, Spruit M, Brinkkemper S. A Framework for Process Improvement in Software Product Management. In: Riel A, O'Connor R, Tichkiewitch S, Messnarz R, editors. Systems, Software and Services Process Improvement. Springer Berlin Heidelberg; 2010. p. 1–12. (Communications in Computer and Information Science).

59. van de Weerd I, Bekkers W, Brinkkemper S. Developing a Maturity Matrix for Software Product Management. In: Tyrväinen P, Jansen S, Cusumano MA, editors. Software Business. Springer Berlin Heidelberg; 2010. p. 76–89. (Lecture Notes in Business Information Processing).

60. De Bruin T, Freeze R, Kaulkarni U, Rosemann M. Understanding the Main Phases of Developing a Maturity Assessment Model. In: Campbell B, Underwood J, Bunker D, editors. CD-ROM: Australasian Chapter of the Association for Information Systems; 2005 [cited 2019 Jun 4]. p. 8–19. Available from: https://eprints.qut.edu.au/25152/

61. de Feijter R. Towards the adoption of DevOps in software product organizations: A maturity model approach [Internet]. Utrecht University; 2017. Available from: https://dspace.library.uu.nl/handle/1874/350740

62. Wagner S, Fernández DM, Felderer M, Kalinowski M. Requirements Engineering Practice and Problems in Agile Projects: Results from an International Survey. arXiv:170308360 [cs] [Internet]. 2017 Mar 24 [cited 2019 Oct 1]; Available from: http://arxiv.org/abs/1703.08360

63. Heck P, Zaidman A. A Quality Framework for Agile Requirements: A Practitioner's Perspective. arXiv:14064692 [cs] [Internet]. 2014 Jun 18 [cited 2019 May 22]; Available from: http://arxiv.org/abs/1406.4692

64. Pichler R. Agile Product Management with Scrum: Creating Products that Customers Love (Addison-Wesley Signature Series (Cohn)) 1st Edition. Addison-Wesley Professional; 1 edition; 2010. 160 p.

65. Zowghi D, Coulin C. Requirements Elicitation: A Survey of Techniques, Approaches, and Tools. In: Aurum A, Wohlin C, editors. Engineering and Managing Software Requirements [Internet]. Berlin, Heidelberg: Springer Berlin Heidelberg; 2005 [cited 2019 May 27]. p. 19–46. Available from: https://doi.org/10.1007/3-540-28244-0_2

66. Patton J, Economy P. User Story Mapping: Discover the Whole Story, Build the Right Product. O'Reilly Media, Inc.; 2014. 324 p.

67. Trumler W, Paulisch F. How "Specification by Example" and Test-Driven Development Help to Avoid Technial Debt. In: 2016 IEEE 8th International Workshop on Managing Technical Debt (MTD). 2016. p. 1–8.

68. Berander P, Andrews A. Requirements Prioritization. In: Aurum A, Wohlin C, editors. Engineering and Managing Software Requirements [Internet]. Berlin, Heidelberg: Springer Berlin Heidelberg; 2005 [cited 2019 Jul 19]. p. 69–94. Available from: https://doi.org/10.1007/3-540-28244-0_4

69. Leffingwell D. Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise. Addison-Wesley Professional; 2010. 562 p.

70. Inayat I, Moraes L, Daneva M, Salim SS. A Reflection on Agile Requirements Engineering: Solutions Brought and Challenges Posed. In: Scientific Workshop Proceedings of the XP2015 [Internet]. New York, NY, USA: ACM; 2015 [cited 2019 Sep 4]. p. 6:1–6:7. (XP '15 workshops). Available from: http://doi.acm.org/10.1145/2764979.2764985

71. Lincoln YS, Guba EG. But is it rigorous? Trustworthiness and authenticity in naturalistic evaluation. New Directions for Program Evaluation. 1986;1986(30):73–84.

72. Guba EG. Criteria for assessing the trustworthiness of naturalistic inquiries. ECTJ. 1981 Jun 1;29(2):75.

73. Dorussen H, Lenz H, Blavoukos S. Assessing the Reliability and Validity of Expert Interviews. European Union Politics. 2005 Sep 1;6(3):315–37.

74. Eloranta V-P, Koskimies K, Mikkonen T, Vuorinen J. Scrum Anti-Patterns – An Empirical Study. In: 2013 20th Asia-Pacific Software Engineering Conference (APSEC). 2013. p. 503–10.

75. Eloranta V-P, Koskimies K, Mikkonen T. Exploring ScrumBut—An empirical study of Scrum anti-patterns. Information and Software Technology. 2016 Jun 1;74:194–203.

Appendix A: Interview template for data collection

It should be noted that the topics/questions in the protocol were guidelines. It is possible some were omitted or changed during an interview to adapt to a different context of the interviewee. The interview was a semi-structured interview.

Permission to record interview? [0]
Company;
Interviewee;

1. Context

Name of the company, location, country, name of contact	
General information on the company: history, sector, target markets, mission, etc.	
Key figures such as size, turnover, number of customers, etc.	
An overview of the company's products/services: description of main functionality, product/service platform, customer markets.	
What is the product/project the team is currently working on?	
Perceived maturity of the team? (scale 1-5)	
How is the product roadmap established and updated?	
How are the boundaries of a release defined (for a release-based product)?	
At what stage are user stories introduced?	

How is a roadmap or a release definition refined into a set of user stories?	
Are there intermediate artifacts that are utilized to obtain user stories?	
How are customers involved in the RE practices? How often is there interaction between customer and developers?	
How is RE elicitation done? What techniques?	
Length of sprints	

2. Requirements specification

How are requirements specified? (more than only user stories?)	
Who and what role is writing user stories?	
What template is used, if any, and why?	
Do they make use of epics, user stories, and themes?	
What automated tools are used for managing user stories?	
How much importance does the benefit part (“so that”) have?	
Does the company use a quality framework; if so, which one and why?	
Are user stories complemented by other requirements notations (e.g., for quality requirements)?	
Are user stories complemented with acceptance tests, and using which notation?	

What is the technique or process that is employed for effort estimation: expert opinion, story points, planning game, function points?	
To what extent is the team satisfied with their effort estimation practices for user stories?	
What is the prevalent team opinion on the effectiveness of working with user stories?	
Do you use any requirements modeling technique?	
How are requirements verified, if at all?	

3. Post US specification

Do developers work directly on user stories, or are those refined?	
If refined, what notations are used to do so?	
How are user stories validated after they have been implemented?	
Are user stories changed after their initial specification, when and why?	
Are user stories traced to other software development artifacts, and vice-versa?	
How are user stories prioritized?	
What other information is included in a backlog item, aside from the user story?	

5. Additional information

How does the team deal with challenges that originate from literature?
Examples, people;

- High turnover

- Motivation issues towards RE
- Customer availability
- Budget and time estimations

Examples, resources;
- Lack of specialized tools
- Lack of documentation (e.g. tracing)
- Inappropriate architecture
- Growing technical debt
- Distributed teams, communication problems?

Did you encounter any of these challenges?
How did you solve them?
How did you mitigate them?

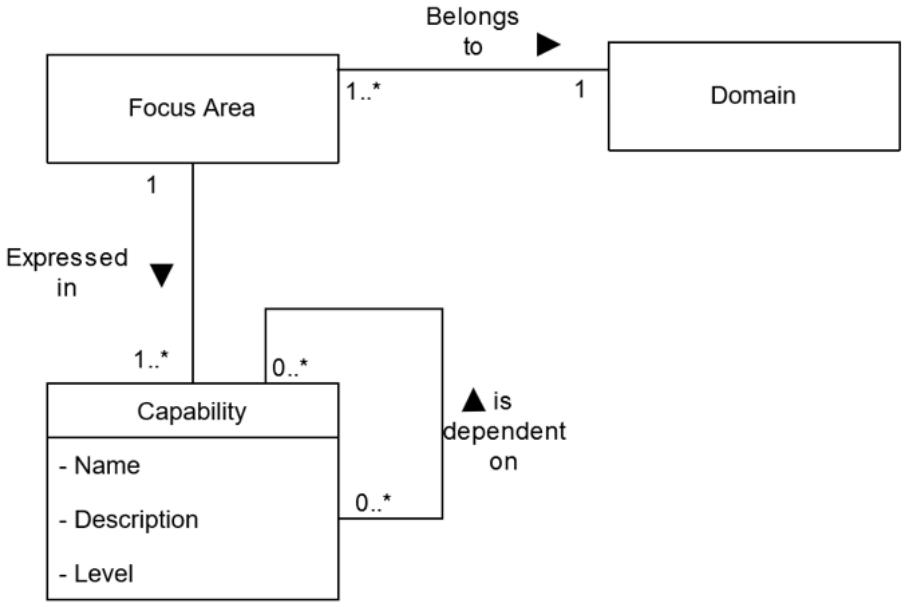
Appendix B: Case study dataset & analysis

The case studies were stored and analyzed in NVIVO. The NVIVO file (46mb) can be found here https://drive.google.com/file/d/172BxDdYHY2zJ8Wd-tFIrk2KPfLLCnzaU/view?usp=sharing.

Note: it can only be opened using NVIVO and be accessed with an account of Utrecht University. If you seek access and are not a member of Utrecht University, please submit a request for access to l.dereeder@students.uu.nl.

Appendix C: Maturity matrix changelog

Bold capabilities are capabilities that have a dependency on another capability in a different focus area.
Italic capabilities are capabilities that are situational.



The image above shows a metamodel of a maturity model. The validation sessions could result in a number of changes. The revisions that can be encountered in the changelog are;

- Addition, removal or merging of focus areas
- Addition, removal or merging of capabilities in a focus area
- Moving a capability to a different focus area
- Adding a dependency between capabilities
- Changing the content of a capability in a focus area

V0.1 First version of the matrix based on the case study & literature

Matrix v0.1									
Focus area	**Maturity level**								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Ideation									
Goals & Vision	A					**B**			C
Epics	A	**B**		C					
Elicitation		A		B		C		*D*	
Specification									
Creation	A		B		C			D	
Templates		A	B		C		D		
Estimation	A		B	C			*D*		
Acceptance criteria		A		B		C			
Implementation									
Prioritization									
Refinement			A		B		C		
Validation	A		**B**				**C**	D	E
Traceability & Documentation				A		B			C
Management & Planning									
Roadmap		A	B		*C*	*D*			
Customer involvement & stakeholder management	A		B		C		D		*E*
Responsibilities	A	B				C			D
Sprint planning	A		**B**	**C**		**D**			
Support									
Tools	A	B			C		D		
Definitions of Done/Ready	A	B							
Shared understanding									
Metrics									
Focus area	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
	Maturity level								

V0.2 Changelog based on the case study & literature
, 6 revisions

R.1. Removed shared understanding from focus areas. Shared understanding is an important concept in scrum, but does not warrant its own focus area. This is because there are certain activities which can be done to *establish* a shared understanding, such as refinement sessions and stakeholder management on a certain level. Therefore, capabilities need to reflect that they add to a shared understanding.

R.2. Removed definition of done & ready from focus areas. The two capabilities are divided to other focus areas. Capability A (there exists a definition of done that is identical for all PBI's) is added to Validation capability A. Capability B (there exists a definition of ready) is added to the focus area sprint planning as capability B. This means that these capabilities no longer have a dependency, as their original focus area has disappeared.

R.3 Added Prioritization capabilities

R.4 Changed capabilities in the elicitation focus area. Capability C is new which contains 3 amigo sessions.

R.5 Creation focus area renamed to documenting

R.6 Focus area epics is renamed to high-level requirements, as it better reflects the content of the focus area. Epics, are high-level requirements. But not all high-level requirements are epics, yet the focus area discusses high-level requirements and not necessarily epics.

Matrix v0.2										
Focus area	**Maturity level**									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Management & Planning										
Roadmapping		A		B			*C*		*D*	
Stakeholder involvement	A		B		C			D		*E*
Responsibilities	A		B			C			D	
Sprint planning	A		**B**	**C**		**D**				
Ideation										
Goals & Vision	A					**B**			C	
Epics	A	**B**		C						
Elicitation		A		B		C		*D*		
Specification										
Documenting	A			B			C			D
Templates		A			B		C		D	
Estimation	A			B			C			*D*
Acceptance criteria		A			B			C		
Implementation										
Prioritization	A			B						
Refinement			A		B		C			
Validation	A		**B**				**C**	D	E	
Traceability & Documentation				A		B			C	
Support										
Tools	A	B			C		D			
Metrics										
Focus area	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	Maturity level									
Matrix v0.2										

V0.3 Changelog after validation with expert A and discussion with lecturers. 14 Revisions

R.1. Customer involvement & Stakeholder management renamed to stakeholder involvement (by advice from the supervisor)

R.2. Stakeholder involvement capability E has been made situational, as the end-user does not always exist or is available

R.3 Sprint planning order is revised by advice from Remi; B becomes D, D becomes C, C becomes B

R.4 The expert mentioned that the topic of living documentation is something for high maturity teams and a capability discussing it could be added somewhere in acceptance criteria or documentation & traceability. A capability D was added in documentation & traceability dedicated to *living documentation*.

R.5 Capabilities have been balanced over the 10 maturity levels

R.6 Responsibilities A and B have been removed after discussion with Remi. The reasoning behind this is that it does not concern agile RE, and the given capabilities

are rules required to adhere to Scrum. As few capabilities remain, the expert was asked if the focus area is relevant enough to remain, or that the remaining capabilities could be placed in other focus areas. The expert states that the capabilities could be included in other focus areas. Responsibilities C and D are relevant for agile RE. Capability C is added to prioritization, capability D still needs to find a new focus area

R.7 Goals & vision have been merged with epics as they have many similarities. Goals & Vision often directly lead to the creation of epics, Remi even stated 'your goal & vision *are* your epics, except they are not yet formulated in this way'.

R.8 Part of capability A from focus area documenting is moved to sprint planning capability A, concerning the part 'the team ensures there are enough backlog items to enter the upcoming sprint'. The expert states this has more to do with sprint planning than documenting.

R.9 Capability C from focus area validation has been removed. This capability concerns the use of unit tests. While unit tests are a good practice to test code coverage and the quality of code, the expert argued, it does not in any way validate whether a requirements has been correctly implemented. It is therefore considered out of scope for the model. Capability D (a backlog item is verified before it will enter a sprint) has been moved to sprint planning and, as it was rarely done by teams, is added there as capability E.

R.10 In agreement with the expert, the focus area of metrics has been removed, as there are no capabilities to be found. While most teams track some metrics it is often unclear what the purpose for this tracking is. The expert also states that the use of metrics is not related to RE.

R.11 The focus area of tools is removed. The capabilities are divided over other focus areas. Capability A & B (use of a tool for documenting & managing requirements) are moved to focus area Documenting as capability A. Capability C (the use of collaboration software such as confluence) is removed, as this collaboration software is often also used to manage requirements but provides other, non RE-related, functions for team collaboration, which are good but also out of scope for the model. Capability D (the use of a tool to check the quality of a requirement) has also been removed. This was advised by Fabiano as it is important to use the more detailed quality framework (such as QUS) and whether this is done with a specific tool should not matter. There already existed such a capability in documentation (capability D)

R.12 With the deletion of the metrics and tools focus groups, the category of *Support* has been removed.

R.13 Added capability D to traceability & documentation, stating that teams should practice BDD development in order to create 'living documentation', which is documentation that automatically updates itself. BDD documentation is a way to

share the definition of your features and have a common dialogue for business and technological stakeholders.

R.14 The expert suggested that capability E (a PBI does not enter the sprint before it is verified with the customer) from sprint planning should be removed, as it is already covered by capability D. The expert argues that having a definition of ready should already cover this criteria. The problem with this is that the expert assumes that having a definition of ready will include this criteria. However, the definition of ready is defined by the team itself, thus having a definition of ready does not guarantee verification is actually done. This issue is not yet resolved, and will be discussed with the next expert.

Matrix v0.3										
Focus area	**Maturity level**									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Management & Planning										
Roadmapping		A		B			*C*		*D*	
Stakeholder involvement	A		B		C			D		*E*
Sprint planning			A	**B**			**C**		D	E
Ideation										
Goals & Vision		A			B			C		D
Eliciting requirements		A		B		C				
Specification										
Documenting requirements	A			**B**				C		
Templates		A			B		C		D	
Estimating effort	A			B		C				*D*
Acceptance criteria			A		B			C		
Implementation										
Prioritizing requirements	A					B		C		
Refining requirements			A		B		C			
Validating requirements	A		**B**				C			
Traceability & Documentation		A		B		C		D		E
Focus area	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	Maturity level									
Matrix v0.3										

V0.4 Changelog after validation with expert B, 14 revisions, 2 issues

R.1 Added capability D to traceability & documentation, stating that teams should practice BDD development in order to create 'living documentation', which is documentation that automatically updates itself. BDD documentation is a way to share the definition of your features and have a common dialogue for business and technological stakeholders.

R.2 Capability C from roadmapping has been expanded to include 'the continuous updating of the roadmap'. Expert M mentioned that although it is important to schedule a roadmap this roadmap needs to be continuously updated and reprioritized.

R.3 Part of capability A from goals & vision *'The team uses different levels of granularity to distinguish between high-level requirements (such as epics) and low-level requirements (such as user stories)'* is moved to capability A from documenting requirements as it is more related to this focus area according to the expert. Capability C has been changed, as the expert states that mature teams do not blindly follow some goal & vision from higher up the hierarchy chain and teams should establish their own vision & goal, regardless of the techniques used.

R.4 The elicitation focus area has a situational capability D added. The expert states that he sees teams that are really mature will no longer have to elicit many low-level requirements, as they understand the stakeholders needs so well they can immediately work out new requirements. The expert acknowledges implementing requirements without consulting the stakeholders can also be a huge risk and is therefore an indication of the high maturity of a team. This may be a controversial item to add to a high maturity level and therefore this issue will be presented to the **focus group** who will have the final verdict.

R.5 The documenting requirements focus area has a new capability C, the old capabilities C and D are now D and E. Expert B states that it is very important (and also stated by the scrum guide) that backlog items are transparent and can be accessed by all stakeholders. The expert states that this is also the value of a tool like azure devops or jira as it allows for easy transparency over a file like excel to keep a backlog.

R.6 The estimating of epics was included in capability C for estimation, but is moved to roadmapping

R.7 The capability B from refining requirements is removed, as the expert stated this is not and should not be a hard demand. After checking the scrum guide, this was found to be correct. The expert stated there are plenty of times where it can be entirely reasonable to invest large amounts of sprint capacity into refining. Capability B has been replaced by having a definition of ready that teams use to identify whether an item still needs refinement or not. This also introduces a dependency from Capability D from sprint planning to this capability.

R.8 Capability C from refining requirements has been expanded to include the fact that teams should use some kind of reference story or tipping point to determine when a story is too big and should be split up.

R.9 Capability A from validating requirements has been split up. The part 'There is a definition of done which is identical for all PBI's' has been added as capability B. Previous capability B is now C, and C is now D. The reason for splitting this up is because the expert stated he has seen many teams who did not establish a definition of done, and it should therefore be a capability B on its own.

R.10 Capability C from validating requirements has been split up. The part 'there are review meetings in which the team and important stakeholders discuss which items are done' has been merged with capability D as they conveyed the same meaning.

R.11 Capability E has been added to validating requirements. The expert states that while review meetings are a great way to validate, he believes there is an additional level of maturity. The expert states that showing of the progress during a review meeting is quite different from actually validating in the real-world environment in which the system operates. Therefore, (capability E) the customer should validate PBI's himself in the working environment of the system. However, the expert did not really know how this capability should be formulated. Therefore, this issue will be presented to the **focus group** who will have a final verdict on the issue.

R.12 Capability A from traceability and documentation has been removed, as the expert noted that the capability states that there is no effort towards the focus area, and therefore it should not be a capability A (but a capability 'zero' or non-existent)

R.13 Capability A from traceability and documentation (which was previously capability B) has been expanded by including the process of mapping dependencies between stories. The expert states that this is important when changes happen to see which other PBI's may be affected.

R.14 Capability D from estimating effort is introduced by the expert. The expert states that when estimating it is very important that the team uses some reference backlog item or baseline which estimations are based on. Otherwise, everyone can have their own interpretation of what a 'work hour' or 'story point' means.

Matrix v0.4										
Focus area	**Maturity level**									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Management & Planning										
Roadmapping		A		B			*C*		*D*	
Stakeholder involvement	A		B		C			D		*E*
Sprint planning			A	**B**			**C**		**D**	E
Ideation										
Goals & Vision		A			B			C		D
Eliciting requirements		A		B		C		*D*		
Specification										
Documenting requirements	A			**B**		**C**		D		E
Templates		A			B		C		D	
Estimating effort	A			B		C				D
Acceptance criteria			A		B			C		
Implementation										
Prioritizing requirements	A					B		C		
Refining requirements			A		B		C			
Validating requirements	A		B		**C**		D		E	
Traceability & Documentation		A		B		C		D		
Focus area	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	Maturity level									
Matrix v0.4										

Changelog v0.5 after validation with expert C, 5 revisions, 1 issue
R.1 Expert B argued that capability E in sprint planning is already covered by capability D. The argument given is that a definition of ready should already include some criteria that this verification activity took place, and thus capability D would already guarantee that capability E took place. This issue was discussed with expert C, who argued that while a good definition of ready should indeed cover this, not all teams who have a definition of ready have a 'good' definition of ready and therefore, the capability has remained the same. The issue will be presented to the **focus group** for a final verdict.

R.2 Expert C states that the recent addition from expert B in capability C from Roadmapping ("The roadmap is continuously updated and reprioritized.") is a good addition. The expert states that a roadmap needs to 'live', instead of being a static artifact that is only created at the beginning of the project. The expert states this capability is not situational for product software companies however, and it should therefore be moved to capability B. Expert C states that capability D is incomplete and should contain something about the SAFe framework (scaled agile framework), which acknowledges two types of roadmaps; short- and long term roadmaps.

R.3 The expert suggested changing the order of capability B and C in the templates focus area. The reason for this is that in the opinion of the expert, the A and C capability are both very pragmatic about *what* needs to be built and B is about why it needs to be built. The expert is of the opinion that this needs to be at a higher maturity level, thus they have been switched.

R.4 The expert suggests that the step in the focus area of acceptance criteria between A and B is fairly large (there are a lot of teams that use acceptance criteria, and there are way fewer teams that use a formal language such as gherkin). The expert did not know what should be in between however. Thus this issue will be presented to the **focus group**.

R.5 The expert suggested a new capability D for the refining requirements focus area. While it is important that there are enough refined items to enter a sprint a team should also be careful not to 'over-refine', i.e. there should not be enough refined items for the upcoming four sprints. This would lower the flexibility of the team and the relevance of items may be debatable 4 sprints later, making the refinement of them a wasteful activity.

Matrix v0.5										
Focus area	**Maturity level**									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Management & Planning										
Roadmapping		A		B			*C*		*D*	
Stakeholder involvement	A		B		C			D		*E*
Sprint planning			A	**B**			**C**		**D**	E
Ideation										
Goals & Vision		A			**B**			C		D
Eliciting requirements		A		B		C		D		
Specification										
Documenting requirements	A			**B**		**C**		D		E
Templates		A			B		C		D	
Estimating effort	A			B		C				D
Acceptance criteria			A		B			C		
Implementation										
Prioritizing requirements	A					B		C		
Refining requirements			A		B		C			D
Validating requirements	A		B		**C**		D		E	
Traceability & Documentation		A		B		C		D		
Focus area	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	Maturity level									
Matrix v0.5										

Changelog v0.6 after validation with Expert D, 4 revisions, 2 issues

R.1. The expert argued that capability C in stakeholder involvement should be changed. 'Systematic identification of stakeholders is done at the beginning of a project' should be supplemented with a up-to-date overview of stakeholders, as stakeholders can change during a project.

R.2. The expert argued that an additional capability for eliciting requirements would be involving the end-user in elicitation. However, this is already covered in stakeholder involvement capability E and is therefore not adjusted.

R.3. The expert argued that there are more factors to involve in prioritization, and that these could be included in the prioritizing requirements focus area. However, he had no ideas on how these should be included. Therefore, this issue is discussed with the **focus group** who will give a final verdict.

R.4. The expert argued that really mature teams no longer need to add a lot of information to some requirements, arguing that less=more for mature teams. The expert did not know how this should be added however, as it is a high maturity capability but does not really describe a process improvement. Therefore, this issue is discussed with the **focus group** who will give a final verdict.

Matrix v0.6										
Focus area	**Maturity level**									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Management & Planning										
Roadmapping		A		B			*C*		*D*	
Stakeholder involvement	A		B		C			D		*E*
Sprint planning			A	**B**			**C**		**D**	E
Ideation										
Goals & Vision		A			**B**			C		D
Eliciting requirements		A		B		C		D		
Specification										
Documenting requirements	A			**B**		**C**		D		E
Templates		A			B		C		D	
Estimating effort	A			B		C				D
Acceptance criteria			A		B				C	
Implementation										
Prioritizing requirements	A					B		C		
Refining requirements			A		B		C			D
Validating requirements	A		B		**C**		D		E	
Traceability & Documentation		A		B		C		D		
Focus area	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	Maturity level									
Matrix v0.6										

Changelog v0.6 after validation with Focus Group, 5 revisions

Name	Function	Experience in current function (years)	Experience working for current organization (years)
FocusGroupExpert A	Service delivery manager	2	14
FocusGroupExpert B	Product owner / Business analyst	4	4
FocusGroupExpert C	Agile coach / Project manager	6	18
FocusGroupExpert D	Arealead Way of Working / Manager / Senior Consultant	1.5	11
FocusGroupExpert E	Senior Consultant	6	13

The expert validations resulted in five issues which could not be resolved during these validation sessions. These five issues have been discussed and resolved by the focus group;

Issue 1: Expert C was of the opinion that there was a capability missing in the acceptance criteria focus area between the current capabilities A and B. However, the expert was unsure as to what this capability should be. The focus group was asked on their opinion of the current state of the focus area and its capabilities and whether they could identify a 'missing' capability. The group noted that the main idea behind acceptance criteria is that the team has a shared understanding when the item is finished. The experts agreed that a great way to perform acceptance criteria and achieve good shared understanding is the use of specification by example. Therefore, a new capability B has been added which describes the use of specification by example to describe acceptance criteria. The experts were rather surprised that there are teams who perform capability C without also performing D, as a team should perform C because they want to do D as well. Only performing C seems like a waste of effort. However, after some discussion the experts agreed that only performing C does provide some other benefits aside from allowing to progress to capability D. Therefore, these capabilities have not been merged.

Issue 2: Expert B was of the opinion that there was an additional capability in the eliciting requirements focus area. This capability was explained by the expert; *"Very mature teams no longer need to elicit small details from stakeholders. If they are mature enough, they can think for themselves well enough to fill in these details by themselves. I consider this to be a final step of maturity"*. This issue caused quite a bit of discussion among the focus group as many experts took this to mean there is no contact with the stakeholder at all. The final verdict of the group was that yes, this is an indicator

of a very mature team. However, there does need to be a dependency to the validation focus area. If elicitation with the stakeholder is omitted the corresponding requirements absolutely need to be validated with the stakeholder after implementation, even more so than if elicitation were not omitted. As a result of this discussion a new capability E has been added to the eliciting requirements focus area with a dependency to the validating requirements focus area capability E, as the experts stated that when omitting elicitation the validation needs to be *thorough* and not on a lower level validation capability.

Issue 3: Expert B was of the opinion that there is an additional capability in the validating requirements focus area. This capability E was defined as 'supplementing review meetings, the customer checks all PBI items in the working environment'. This capability was still rather vaguely defined and therefore the focus group was asked to give their opinion on the focus area and capability. The experts agreed that this is an important final step, but stated that the current text should be changed. According to the experts 'supplementing review meetings' should be omitted, as they could see scenarios in which review meetings would be turned down, for example; in projects with fast release cycles (e.g. every day) review meetings might be done only once every sprint while there are 14 releases in the same sprint. These should be validated by the customer/stakeholder in live production. Because of this, the capability has remained the same with some textual changes about the presence of review meetings.

Issue 4: Expert D was of the opinion that there are more aspects involved in the prioritization of requirements, but was unsure how this should be included in the capabilities. Therefore, the focus group was asked for their opinion on the current focus area prioritizing requirements and whether they had any remarks or addition of elements. There was quite some discussion between the experts what constitutes as 'business value' and 'cost of delay'. An expert stated that he missed the fact that prioritization is often linked to a certain goal. The main argument against capability B was that prioritizing on business value can mean anything from 'business value is based on one factor (e.g. costs)' all the way up to 'business value is based on all factors that have a positive contribution to the work item (e.g. costs, customer demand, impact)'. The conclusion was that capability A should state that the argumentation by the product owner is not explained to the team. 'The product owner is responsible for the prioritization of backlog items' should be removed as this is already a prerequisite by Scrum.

FocusGroupExpert C suggests that capability B and C should be merged and there should be a new capability C stating that the team is contributing to the prioritization. This is a sign of a mature team because the team is also involved in the development process.

The discussion resolved around two questions. What do we think is a sign of a mature team regarding the prioritization process?

A. The team can reach an objective prioritization by prioritizing with a lot of argumentation.
B. The team can reach a consensus when prioritizing.

In the end, the following order of new capabilities was proposed and agreed upon by experts.

Prioritizing requirements			
A	B	C	D
PBI's are prioritized by the product owner without proper argument to the team.	The prioritization of PBI's is based on a proper argument, which can include factors such as business value, the cost of delay & the results from previous sprint	Prioritization is a co-production with the development team.	The prioritization is made measurable with facts and numbers. For example, costs have been calculated, risks are predicted.

Issue 5: Expert D was of the opinion that in the focus area Templates there should be an additional capability E. According to the expert, the most mature teams have established such a high level of shared understanding that it is no longer necessary to include all kinds of information in backlog items, arguing that sometimes less=more. The focus group was asked for their opinion on the issue. Expert D argued that a template should never be a goal on its own and that it is a way to achieve a shared understanding among team members. FocusGroupExpert E and FocusGroupExpertC argue that the statement less = more is dangerous, and they both encountered situations where not adding some information resulted in misunderstanding within the team. However, they do agree with the argument provided by expert D that very mature teams know each other so well that this (less=more) argument can be true in some situations. The experts argue that its 'good' if a team uses a template but very hard to argue what a 'good' template should contain, as it is very context specific. The experts argue that teams should use a template and adjust the template to their own needs. The final capability has been adjusted to reflect this. Teams should use a template but adjust it to suit their own needs.

Issue 6: Before the focus group, all focus areas were ordered based on improvement steps that the experts argued were logical/how they saw them in practice/how important they were. However, between the focus areas there are also areas that are more important for an immature team to establish capabilities in *first* over other focus areas. The model does not yet convey this correctly. For example, focus areas stakeholder involvement and estimating effort both have a capability A at maturity level 1, suggesting that an absolute immature team should prioritize implementing these capabilities first over, for example, 'A' capabilities from other focus areas that start at either maturity level 2 or 3, like Sprint planning capability A.

The focus group was asked to try and identify which focus areas should have priority over others to start implementing. This discussion results in quite a big shift of the first capabilities in focus areas. While the experts argued that for later focus areas it is much more arbitrary and almost impossible to say which capability needs to be done over another due to the specific context being an enormous factor, they did identify some priorities for early capabilities, focus areas that should definitely be prioritized at the start of a completely immature team. The table below shows how the experts identified the importance of the focus areas, and it has been used to redistribute (mostly) capabilities that occur early in a focus area, like the A and B capabilities.

1	2	3
		Roadmapping
Stakeholder involvement		
Sprint planning		
Goals & vision		
	Eliciting requirements	
Documenting requirements		
		Templates
		Estimating effort
	Acceptance criteria	
Prioritizing requirements		
	Refining requirements	
Validating requirements		
		Traceability & Documentation

Matrix v1	
Focus area	**Maturity level**

	1	2	3	4	5	6	7	8	9	10
Management & Planning										
Roadmapping			A		B		*C*		*D*	
Stakeholder involvement	A		B		C			D		*E*
Sprint planning	A			**B**			**C**		**D**	E
Ideation										
Goals & Vision	A					**B**		C		D
Eliciting requirements		A		B		C		**D**		
Specification										
Documenting requirements	A			**B**		**C**		D		E
Templates			A		B		C		D	
Refining requirements		A			B		**C**			D
Acceptance criteria		A		B				C	D	
Implementation										
Prioritizing requirements	A			B		C		D		
Estimating effort			A	B		C				D
Validating requirements	A		B		**C**		D		E	
Traceability & Documentation			A		B		C		D	
Focus area	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	Maturity level									
Matrix v1										

Matrix v1										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Management & Planning										
Roadmapping			A		B		C		D	
Stakeholder involvement	A		B		C			D		E
Sprint planning	A			B			C		D	E
Ideation										
Goals & Vision	A					B		C		D
Eliciting requirements		A		B		C		D		
Specification										
Documenting requirements	A			B		C		D		E
Templates			A		B		C		D	
Refining requirements		A			B		C			D
Acceptance criteria		A		B				C	D	
Implementation										
Prioritizing requirements	A			B		C		D		
Estimating effort			A	B		C				D
Validating requirements	A		B		C		D		E	

Traceability & Documentation			A		B		C		D	
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Matrix v1										

Matrix with stats v1											
Focus area	**Maturity level**										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Capabilities per focus area**
Management & Planning											
Roadmapping			A		B		*C*		*D*		4
Stakeholder involvement	A		B		C			D		*E*	5
Sprint planning	A			**B**			**C**		**D**	E	5
Ideation											
Goals & Vision	A					**B**		C		D	4
Eliciting requirements		A		B		C		**D**			4
Specification											
Documenting requirements	A			**B**		**C**		D		E	5
Templates			A		B		C		D		4
Refining requirements		A			B		**C**			D	4
Acceptance criteria		A		B				C	D		4
Implementation											
Prioritizing requirements	A			B		C		D			4
Estimating effort			A	B		C				D	4
Validating requirements	A		B		**C**		D		E		5
Traceability & Documentation			A		B		C		D		4
Number of capabilities per maturity level	6	3	6	6	6	5	6	6	6	6	**Total = 56 capabilities**
											Average capabilities per focus area = 4.30 **Average capabilities per maturity level = 5.6**

Levels 1-3: Mostly A capabilities, 13 out of 15 are A, 2 out of 15 are B
Levels 4-5: Mostly B capabilities, 10 out of 12 are B, 2 out of 12 are C
Levels 6-7: Mostly C capabilities, 9 out of 11 are C, 1 out of 11 is B, 1 out of 11 is D
Levels 8-10: Mostly D and E capabilities, 12 out of 18 are D, 4 out of 18 are E, 2 out of 18 are C