

UTRECHT UNIVERSITY

**Applying the Requirements Engineering for
Software Architecture Model in Software
Products: a case study using crowdsourcing**



Abel Menkveld

Master Thesis

Supervised by Sjaak Brinkkemper and Fabiano Dalpiaz

Business Informatics, Faculty of Science

Department of Information and Computing Sciences

May 2019

Preface

It is conventional wisdom that great innovations come from disruptive inventions by entrepreneurs who are not following the market trends, but who create something no-one has thought of before. After reading Steve Jobs' biography for example, I learnt that you can't just ask customers what they want and give that to them, because "by the time you get it built, they'll want something new". The famous quote attributed to the great innovator Henry Ford illustrates it best: "If I had asked people what they wanted; they would have said faster horses."

Since I founded my own software start-up, I am wondering how to deal with customer input. The process of answering emails, taking phone calls and meeting clients to discuss their (feature) requests is time consuming and might stop the real innovation. However, did customers actually tell Henry Ford that they want faster horses, and was the posed question the right one?

While customers might be bad at telling you exactly what they want, you can still extract their needs. Before Henry Ford introduced the affordable automobile that made his fortune, people knew they needed a *faster mode of transportation*. They just didn't know the *form factors*. For software startups, this process of requirement engineering is ongoing and is one of the drivers of innovation.

This idea of crowd-centric software development formed the basis for my thesis. The connection with the technique has always been my topic of interest: how to translate these requirements into a product or service that people love to use. The RE4SA model - the main artefact for this study - combines business and IT and allows the information scientist to serve as the bridge between them.

Table of contents

1.	Introduction.....	5
1.1.	Problem Statement	6
1.2.	Objective, Scope, and Structure.....	7
2.	Research Approach	9
2.1.	Research questions	9
2.2.	Research design	11
2.2.1.	Literature study.....	12
2.2.2.	Case study.....	12
2.3.	Relevance	13
3.	Literature review	15
3.1.	The RE4SA model.....	15
3.2.	Requirements Engineering.....	18
3.2.1.	Jobs.....	19
3.2.2.	Epic Stories.....	25
3.2.3.	User Stories	27
3.2.4.	Crowdsourcing in Requirements Engineering.....	31
3.3.	Software Architecture	37
3.3.1.	Software applications.....	41
3.3.2.	Modules.....	42
3.3.3.	Features	44
3.3.4.	Software architecture recovery	45
3.4.	Summary.....	48
4.	Case study	50
4.1.	About Tournify	50
4.1.1.	Applying the principles of Canonical Action Research.....	52
4.2.	The case.....	56
4.3.	Architecture recovery	59
4.3.1.	Extract features from the graphical user interface (GUI).....	59
4.3.2.	Identify (sub)modules by abstracting.....	61
4.3.3.	Present the model	62

4.4.	Crowdsourced requirements engineering platform	62
4.4.1.	Elicitation	63
4.4.2.	Prioritization.....	66
4.4.3.	Negotiation	67
4.5.	Evaluation protocol	67
4.5.1.	Architecture recovery.....	68
4.5.2.	Crowdsourced requirements engineering platform.....	68
5.	Results & Analysis	72
5.1.	Architecture recovery	72
5.1.1.	System context.....	72
5.1.2.	Functional Architecture	73
5.1.3.	Quality of the reconstructed architecture	76
5.1.4.	Perceived usefulness of the reconstructed architecture	78
5.2.	Crowdsourced requirements engineering platform	80
5.2.1.	Perceived usefulness of the crowdsourcing platform	82
5.2.2.	Quality of the crowdsourced User Stories	83
5.2.3.	Complexity of the crowdsourced User Stories.....	85
5.3.	Summary of the main results.....	87
6.	Discussion.....	88
6.1.	Answering the sub research questions.....	89
6.1.1.	Linking requirements engineering and software architecture.....	89
6.1.2.	Software architecture	91
6.1.3.	Requirements engineering	93
6.2.	Answering the main research question.....	99
6.3.	Conclusion.....	100
6.4.	Validity threats.....	101
6.5.	Future research	102
	Bibliography	103
	Appendices.....	109
	Paper.....	125

1. Introduction

Already in 2001, Nuseibeh [1] highlighted the importance of the connection between Requirements Engineering (RE) and Software Architecture (SA) in software development. The popularization of agile development just started – it is the same year in which the manifesto for agile software development was drawn up – and shorter times-to-market are key. He speaks of incremental development and speedy delivery, facilitated by a streamlined development process. This development process is iterative and requirements and design specifications are produced in more detail progressively [1].

Recent literature in RE [2] proposes to define this process as a refinement activity that converts Jobs into high-level Epic Stories (also known as Epics in practice), and then split these Epic Stories into detailed User Stories. All these artifacts are natural-language statements. A Job (shorthand for Job-to-be-Done) captures a customer need and begins with *help me*. An Epic Story, or Job Story [2], includes a problematic situation, motivation and expected outcome. It describes a high-level product feature or roadmap theme that can be used as an input for User Story formulation. User Stories contain a role, goal and benefit and are used by over half of the software practitioners to capture requirements [3]. Each well-written User Story is atomic; it expresses a requirement for *one* feature.

The requirements engineer (or product manager) works closely together with the software architect in the development process in which they exchange artifacts (e.g. requirements or knowledge). The software architect creates an architecture in which the product is represented by different modules with specific features. To facilitate (automated) mapping from requirements to the architecture and vice versa in software development it is vital to understand the relationship and interplay between the artifacts of the requirements and architecture. Therefore, the Requirements Engineering for Software Architecture model (RE4SA) has been created by researchers from Utrecht University (Figure 1.1). To test the validity of this model, its usefulness for software development should be subject of studies.

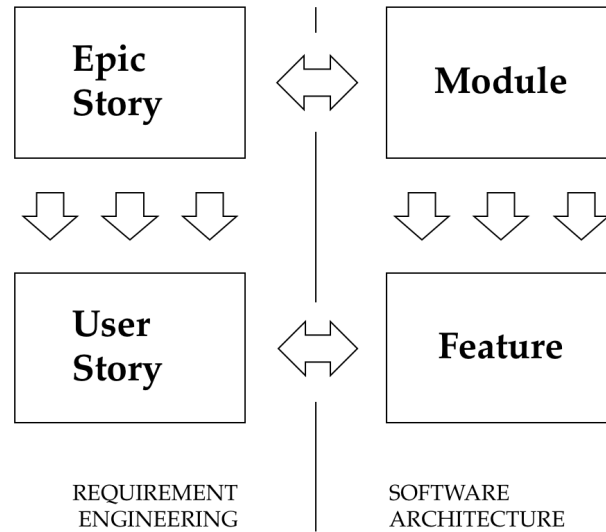


Figure 1.1: The RE4SA model [4]

In an early validation, Blessinga [5] demonstrated how the principles of the RE4SA model could be applied to specify a functional architecture for a new software product based on a set of Epic Stories and User Stories. With our study, we focus in particular on the applicability of the RE4SA model in **existing software products**. Those products often lack the up to date artifacts the RE4SA model is built upon but do have an existing implementation and user base that brings new opportunities. The implementation can be used to extract features and reconstruct a software architecture [6], while the user base offers possibilities for crowdsourcing in RE [7].

1.1. Problem Statement

The RE4SA model is created by combining existing literature from two research areas in computer science. It has recently been used for software traceability using ontologies [8] and to develop an architecture based on the requirements. Although the early results of these studies seem promising, the situation in practice is often less structured, documented and streamlined as hoped for. For example, 56% of the User Stories contain easily preventable syntactic defects [9], or User Stories are not used at all. Collaborating with users during the innovation process also comes with risks of losses of know-how, serving a niche market only or

misunderstandings [10]. The validity of the RE4SA model when applied to existing software has not been tested. In addition, the possibilities of adding crowdsourcing to the model are unknown.

Wieringa [11] distinguishes between design problems and knowledge questions. The goal of knowledge questions is to acquire theoretical knowledge. In this case however, we also want to (re)design an artifact that actually helps product managers and software architects. By doing so, this problem can be categorized as a design problem. The following schema for expressing design problems is therefore applicable [11]:

Improve *a problem context*

By *(re)designing an artifact*

That *satisfies some requirements*

In order to *help stakeholders to achieve some goal*

If we apply the template to this study, this translates to:

Improve *the interplay between RE and SA in software development*

By *validating the RE4SA model for existing software products*

That *uses crowdsourcing*

In order to *allow product managers to gather, negotiate and prioritize high-quality requirements that can be easily communicated to software architects*

1.2. Objective, Scope, and Structure

This study aims to validate a new theoretical model, so it can be applied in real business cases. The RE4SA should be applicable for software companies of any size, from start-ups to multinationals, integrated into already used agile development processes and techniques. The main objective is threefold: (1) report on the RE and SA principles underlying the RE4SA model, (2) include crowdsourcing in the RE4SA model, and (3) validate the model by applying it to an existing software product.

The scope of this research is limited to the artifacts mentioned in the model. This means that in the RE process only Epic Stories [5] and User Stories [12] are covered. We do cover Jobs [13], which serve as the basis for Epic Story generation, but other techniques, like tools such as UML [14], are out of the scope of this research. Regarding the SA, only the context viewpoint and functional viewpoint [15] will be covered which includes the Context Diagram [16], Functional Architecture Diagram [17] and Feature Diagram [18].

This thesis has the following structure: in chapter 2 the Research Approach will be covered in which the research questions, research design and relevance are covered. chapter 3 is an extensive literature review focusing on RE and SA in agile software development, and the connection between them. chapter 4 covers the Case Study at a software development start-up. The results of this Case Study will be listed in chapter 5 and discussed in chapter 6. In this chapter we answer both the sub research questions and main research question, draw the main conclusions and provide directions for future research.

2. Research Approach

2.1. Research questions

In this Master Thesis, we investigate the applicability of the RE4SA model for existing software products and the possibilities of adding crowdsourcing to the model. The research question is defined as follows:

RQ HOW CAN THE REQUIREMENTS ENGINEERING FOR SOFTWARE ARCHITECTURE MODEL (RE4SA) BE APPLIED IN EXISTING SOFTWARE PRODUCTS, WHILE MAKING USE OF CROWDSOURCING IN REQUIREMENTS ENGINEERING?

In order to give an answer to the research question, seven sub research questions are drawn up. At first, a proper understanding of the RE4SA model and its underlying theory is needed. Answering this question requires a combination of RE and SA literature and provides valuable information about the origin and foundation of the model, which serves as the main artifact in this research.

SRQ1 HOW ARE REQUIREMENTS ENGINEERING AND SOFTWARE ARCHITECTURE RELATED AND HOW IS THIS REFLECTED IN THE RE4SA MODEL?

Next, the principles in both RE and SA that are part of the model will be covered in detail. Sub question two and three relate to this and define the scope of the research. The second sub research question also covers the principles regarding crowdsourcing in RE.

SRQ2 WHAT ARE THE PRINCIPLES IN REQUIREMENTS ENGINEERING THEORY REGARDING JOBS, EPIC STORIES, USER STORIES AND CROWDSOURCING?

SRQ3 WHAT ARE THE PRINCIPLES IN SOFTWARE ARCHITECTURE THEORY REGARDING CONTEXT DIAGRAMS, FUNCTIONAL ARCHITECTURE DIAGRAMS AND FEATURE DIAGRAMS?

The second part of this research consists of a business case: a software product for which the RE4SA model is integrated in the development process. This is in contrast to the general approach in software development, where one starts with RE and creates a SA based on the requirements. Since the software product is already coded and released, the fourth research question relates to the extraction of features and the reconstruction of the architecture. In the fifth research question, the extension of the RE4SA model with the use of crowdsourcing is designed.

SRQ4 HOW CAN FEATURE EXTRACTION SUPPORT THE RECONSTRUCTION OF THE SOFTWARE ARCHITECTURE OF AN EXISTING SOFTWARE PRODUCT?

SRQ5 HOW CAN A CROWDSOURCED REQUIREMENT ENGINEERING PLATFORM BE DESIGNED TO SUPPORT THE ELICITATION, NEGOTIATION, AND PRIORITIZATION OF USER STORIES FOR AN EXISTING SOFTWARE PRODUCT?

The last two sub research questions will validate the treatment designs of the previous two questions in a business case. We will first evaluate the quality (understandability of the model and similarity to other development artifacts) and usefulness (applicability in daily work) of the reconstructed software architecture qualitatively. Then, the effect of using crowdsourcing will be assessed.

SRQ6 WHAT IS THE PERCEIVED QUALITY AND USEFULNESS OF THE RECONSTRUCTED SOFTWARE ARCHITECTURE, WHEN CREATED BASED ON FEATURE EXTRACTION FROM AN EXISTING SOFTWARE PRODUCT?

SRQ7 WHAT IS THE EFFECT OF USING A CROWDSOURCED REQUIREMENTS ENGINEERING PLATFORM TO SUPPORT THE ELICITATION, NEGOTIATION, AND PRIORITIZATION OF USER STORIES FOR AN EXISTING SOFTWARE PRODUCT?

The effect is measured by for variables: the engagement of users, the perceived usefulness of the platform by its users and the quality & complexity of the crowdsourced User Stories. The procedure to measure these variables will be described in the evaluation protocol of this thesis (section 4.5).

By combining the results of the sub research questions, we hope to find an answer to the main research question, which will be covered in the discussion of this thesis.

2.2. Research design

This study is a solution-oriented technical research: an artifact is validated by simulation. The research design comes from the book *Design science methodology for information systems and software engineering* by Wieringa [11], whose prominent work is frequently used as teaching material for students in this field of study. In section 1.1 of this thesis, the introduced problem is categorized as a design problem. Design problems are solved following the design cycle, consisting of three iterative phases: problem investigation, treatment design and treatment validation. These three phases are reflected in the sub research questions of this thesis: SRQ 1, 2, and 3 are part of the problem investigation, we design the treatment by covering SRQ 4 and 5 and validate this treatment by answering SRQ 6 and 7. The research design is summarized in Figure 2.1.

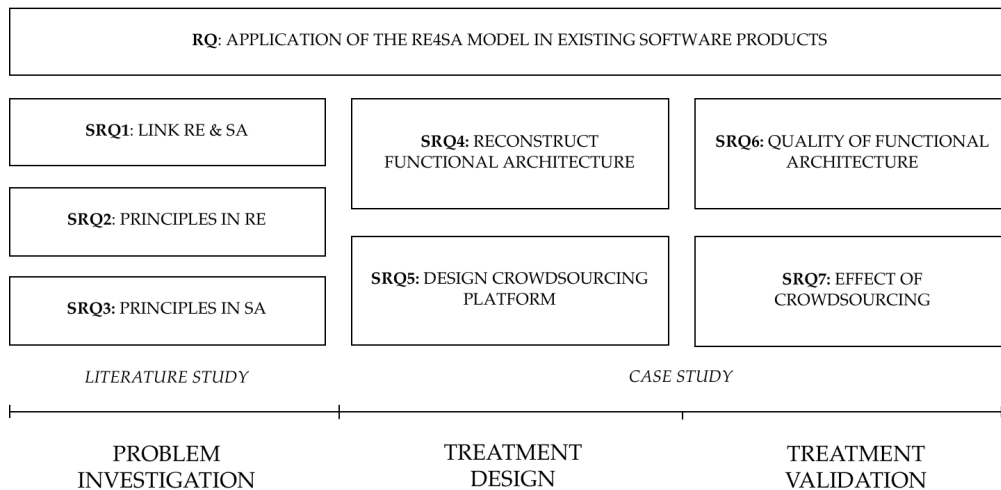


Figure 2.1: The research design of this thesis

2.2.1. Literature study

The first three sub research questions introduced in section 2.1 form the basis for the literature study. The principles in RE and SA and the connection between them will be covered. Although this thesis is obviously no review article, the guidelines of Webster and Watson [19] provide a proven technique for the identification of the relevant literature for this study. They recommend starting with major contributions from leading journals and work both *backwards* by reviewing the citations for the major contributions and *forwards* by identifying articles that cite those key articles. This technique is also called (*reverse*) *snowballing*. The works that will be used as a starting point for the literature study are two RE literature reviews [11, 12], the article from Nuseibeh [1], and the SA book from Rozanski & Woods [15]. Next to this, scientific search engines will be queried so a comprehensive view of all the relevant literature is formed. The search strings that will be used contain requirements engineering, software architecture, crowdsourcing, User Stories or Jobs To Be Done.

2.2.2. Case study

The case study is a big part of this research and entails the validation of the solution in a business context. It will be done to test the properties of the artifact under real-world conditions [11]. The single-case study is a Technical Action Research (TAR), which is important (and usually on of the last stages) in scaling up the solution from idealized conditions in a laboratory to conditions of practice in an organization. The TAR is different from observational case studies and single- case mechanism experiments, the other two types of case studies noticed by Wieringa [11]. This is because in this study we *will* interfere in the case to see the effects of the artifact in context, but also use the artifact to *help a client*. In a TAR one does not transfer the technology to the stakeholders: it is not adopted without involvement of the researcher because the artifact is still under development. It is also different from other forms of action research, since TAR is solution-oriented (artifact driven) instead of problem-oriented (solving a clients' problem without testing a particular artifact) [11]. However, the five principles and most of the associated criteria of Canonical Action Research (CAR) [22], which actually

assumes a problem-driven approach, also apply to our research [23]. Those principles and criteria are developed by Davison, Martinsons & Kock in 2004, to allow for a study in which organization problems are addressed while at the same time contributing to scholarly knowledge [22]. We will cover the principles and how we address them in this study in section 4.1.1, after we have introduced the case.

2.3. Relevance

The RE4SA model aims to support both scientists and practitioners in information sciences, which can still be seen as immature. If we look at the traditional architecture and construction disciplines for example, we can easily say that the processes and routines are much better documented than in software architecture. The documentation should be the main instrument in the communication and the design choices the product managers and software architects make must be recorded or documented accordingly. With studies like this one, we try to point the RE and SA discipline towards that level of maturity.

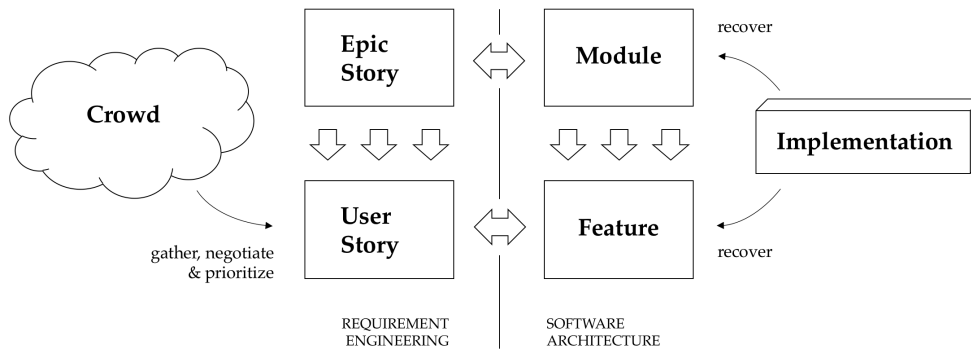


Figure 2.2: The use of crowdsourcing and architecture recovery in this study in relation to the RE4SA model

This thesis is interdisciplinary. It covers both requirement engineering and software architecture and links the artifacts that are used in agile software development to concepts like crowdsourcing and Jobs to be Done theory. This approach can stimulate innovation because it helps in developing products that customers will buy. How the involvement of the crowd via an online cloud

platform could support the elicitation, negotiation, and prioritization of requirements and how architecture recovery could support the creation of an architecture based on a products' code base or user interface, is displayed in Figure 2.2. This figure is an extension of the RE4SA model and shows the research design of this study.

3. Literature review

In this literature review, the RE4SA model will be explained in further detail. We cover its origin and highlight the RE and SA elements it is built upon. Then we aim to give an overview of the RE domain and dive deeper into the challenges that agile RE poses. We cover Jobs, Epic Stories and User Stories as practices to document requirements and discuss crowdsourcing as a method to enhance customer engagement. Lastly, we present the principles in software architecture theory and discuss software architecture modelling.

3.1. The RE4SA model

Where software development in the seventies mainly followed the waterfall model with sequential design steps, Swartout and Balzer [24] contradicted their earlier work and that of many others in the beginning of the eighties when they called for the intertwining of specification and implementation. The researchers argue that intertwining the two will result in a “more coherent and realistic structure for making modifications” [24, p. 440]. Earlier, it was claimed that specification should be completed before implementation begins. Nuseibeh [1] described this concurrent crafting of a system’s requirements and its architecture as the cornerstone of the spiral life cycle model. This model “acknowledges the need to develop software architectures that are stable, yet adaptable, in the presence of changing requirements” [1, p. 115]. In 2001, he created an adaption to the spiral life cycle model, based on his and his colleagues’ experiences in the software industry, called Twin Peaks. In the Twin Peaks model, problem structure and specification are still separated from solution structure and specifications. More detailed specifications are produced progressively and dependency on the implementation increases in the mapping from requirements to architectural design.

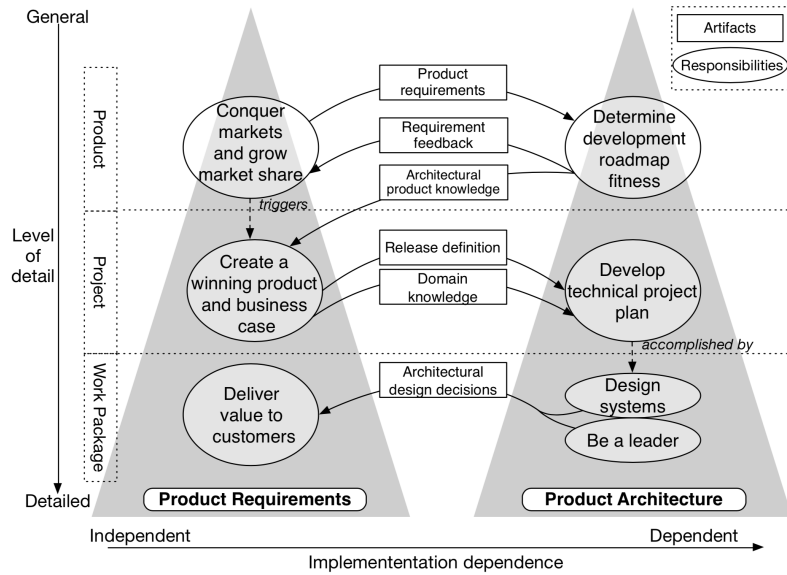


Figure 3.1: The Reciprocal Twin Peaks model of product requirements and architecture [25]

That the requirement specification and architectural design are intertwined is still the commonly accepted perception [26] but Twin Peaks itself provides no approach or guideline for requirements engineers and software architects to cooperate [27]. Therefore, it is necessary to examine the artifacts that are used by practitioners in the specification refinement process. Lucassen et al. [25] tried to ‘bridge’ the Twin Peaks for software products by introducing the Reciprocal Twin Peaks model (Figure 3.1). Software product managers and software architects contribute reciprocally to achieve their goals and exchange artifacts in the process [28]. The conceptual framework “defines *how* product managers and software architects can effectively collaborate in product software development through the exchange of concrete information artifacts” [25, p. 24]. In order to facilitate this communication, the RE4SA model helps to set conventions for the structuring of these artifacts.

The most general representation of requirements are **Jobs**, positioned on top of the requirements peak in the Twin Peaks model. This ‘Job To Be Done’ captures what the customer hopes to accomplish [13]. Christensen et al. [13, p. 56] state that when we buy a product (or service) we essentially “*hire* it to help us do a job”. In a lower representation level, this framework has been used to capture every design problem in a Job Story [29]. We renamed this into **Epic Stories** due to the existing

notion of epics in Scrum development. An Epic Story focuses on a triggering event or situation, a motivation and goal, and the intended outcome. It is different from **User Stories**, the most detailed representations of requirements. A User Story contains a persona (role), action and benefit. Although it is suggested that Epic Stories can serve as an alternative to User Stories [29], we make a distinction in the granularity of the two concepts. User Stories should have the finest granularity possible and multiple User Stories cover one Epic Story. Multiple Epic Stories relate to one Job.

In order to discuss the architecture peak of the Twin Peaks model, it is necessary to discuss view and viewpoints. An architectural view is a description of one aspect of a system's architecture and makes it possible to understand, define, and communicate a complex architecture [15]. Each view is governed by a viewpoint. We only create and discuss the context viewpoint and functional viewpoint, as reflected in the research questions and scope of this thesis. The context view is about the relationships, dependencies, and interactions between the system and its environment and is easy to be read by different stakeholders. The functional view is also easy to understand and the cornerstone of most architectural descriptions, since it defines the elements that deliver the system's functionality and it forms the basis for creation of other views afterwards [15]. Salfischberger, Van de Weerd and Brinkkemper [30] presented the Functional Architecture Framework (FAF) in 2011 to support requirements management for product software businesses. In this model, the product context is the highest level in the functional architecture model. This level defines the product scope and situates the **Applications** as intended to be used by the customers. Zooming in on the system brings up the **Modules**, described as the building blocks of the functional architecture. On the lowest level, which describes the relationships between the functional and technical architecture, each module is supported by **Features** [30].

The four medium to low-level artifacts mentioned in the previous two paragraphs are combined into one model: The Requirements Engineering for Software Architecture model (RE4SA) [4]. In his thesis, Remmelt Blessinga [5] proposed to add an extra abstraction level on top of the RE4SA model containing the Job

(requirements engineering) and Application (software architecture) we also use in this study (Figure 3.2).

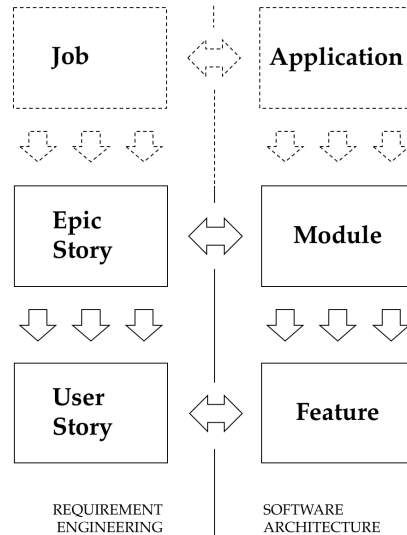


Figure 3.2: The extra abstraction level on top of the RE4SA Model [5], shown by the dotted elements

3.2. Requirements Engineering

The process of extracting informal stakeholders’ needs and translating them into formal specifications is the core principle of RE. These requirements are used as an input for software development. More specifically: they serve as the basis for project planning, risk management, trade off, acceptance testing and change control [31]. Clear statements of requirements are one of the project’s success factors, but at the same time incomplete requirements are the number one reason why projects are impaired [31].

There are two types of requirements: functional requirements and non-functional requirements. The latter are also called quality requirements and are considered to be the most expensive and complex ones to deal with [27]. They are often neglected in agile RE [21] and tend to “interfere, conflict, or contradict with each other” [27, p. 20]. Performance and security requirements are often two conflicting non-functional requirements [27]. Functional requirements can conflict as well because the needs of the stakeholders vary [31].

Together with the shift from traditional (waterfall) development to agile software development, the RE processes changed accordingly. This was necessary because traditional requirement activities – elicitation, analysis and negotiation, documentation, validation, and management – do not take the iterative processes of agile software development into account. However, agile RE does not only alleviate challenges of traditional RE, but also poses new ones. Minimal documentation, customer inability, customer agreement and inappropriate architecture are reported as some of the challenges of agile RE [21]. A proper use of artifacts can overcome the documentation problems and organizations apply different techniques to do so. We will discuss Jobs, Epic Stories and User stories in this chapter, the concepts reflected in the RE4SA model. We also present crowdsourcing as a solution to customer inability and customer agreement. In section 3.3 we write about code refactoring, which may serve as a solution for the inappropriate architecture.

3.2.1. Jobs

The Jobs To Be Done theory (JTBD) originates in the mid-eighties when Rick Pedi and his colleagues turned the marketing research technique Voice of the Customer (VOC) into a theory [29]. Around the same time, Antony Ulwick describes, the IBM PCjr – where he had worked on for 18 months – flopped because the team never identified the metrics customers use to judge the value of products. This inspired Ulwick to create and patent a process called Outcome-Driven Innovation (ODI) in 1999 [32]. The idea of ODI is that companies should “stop focusing on the product and customer and instead aim to understand the underlying process the customer is trying to execute when they are using a product or service” [32]. Ulwick discusses his ideas with Harvard Business School professor Clayton Christensen, who is well-known for his theory of disruptive innovation. Since 2003, when Christensen first mentioned JBTD and states that customers hire products to do specific jobs, the theory got popular [33]. Ten years later, researcher Alan Klement started writing about JTBD and recently published a book on Customer Jobs [29].

Although the work of Ulwick, Christensen and Klement can all be seen as JTBD research, their interpretations of the theory differ considerably. Discussing the theory won't make sense without providing a clear distinction between those interpretations. In his thesis and paper on JTBD, Utrecht University alumni Maxim van de Keuken differentiates between a qualitative approach (the Christensen/Klement camp) and quantitative approach (the Ulwick camp) [2]. Klement in his turn calls the two models Jobs-As-Progress (Christensen/Klement) and Jobs-As-Activities (Ulwick) [29], but Ulwick posed questions by this dichotomy. Christensen and Ulwick share a lot of the core tenets of JTBD theory and it is unclear to what extent Christensen agrees with Klement. Meanwhile, the researchers are involved in an online mudslinging campaign that does not benefit the theory and its understandability. On Twitter, Klement tries to paint Ulwick as a fraud, and called Christensen an "intellectual yet idiot (IYI) who doesn't understand his own theory". In his turn, Ulwick, claims that "Klement has been the source of drama in the JTBD community. He conflates terms and creates fallacies to confuse readers" [34]. For this reason, we refrain from splitting the interpretation of the JTBD theory in two camps, but we will cover the interpretation of each of the three authors separately.

Antony Ulwick

The first book Ulwick wrote was *Business Strategy Formulation* in 1999, with the goal to bring on an 'Intellectual Revolution' caused by the evolution of strategy formulation from an art to a science [35]. He introduced the Customer-Driven Mission Achievement Process (CD-MAP), a sixteen-step quantitative strategy formulation process that can be implemented in three to six months [35]. It is based upon his Universal Strategy Formulation Model (USFM) that illustrates how "the optimal solution is the one (...) that will best satisfy the largest number of important desired outcomes given the constraints imposed on the solution and the competitive position that is desired" [35, p. 40]. This so-called Outcome-Based Logic, focusing on outcomes rather than solutions, is also the basis for the JTBD theory.

Ulwick is the founder and CEO of Strategyn, a California based consultancy that uses the mentioned theory and processes to serve large organizations. In 2005, his next book *What Customers Want: Using Outcome-Driven Innovation to Create Breakthrough Products and Services* is published in which he repackages and expands upon previous ideas into the Outcome-Driven Innovation process [36]. A quote from his frequently cited article in Harvard Business Review explains his theory best:

“What usually happens is this: Companies ask their customers what they want. Customers offer solutions in the form of products or services. Companies then deliver these tangibles, and customers just don’t buy. The reason is simple - customers should not be trusted to come up with solutions; they aren’t expert or informed enough. That’s what your R&D team is for. Rather, customers should be asked only for outcomes--what they want a new product or service to do for them. What form the solutions take should be up to you, and you alone” [37, p. 2].

Ulwick’s latest book *Jobs To Be Done: Theory To Practice* [32] is particularly interesting to compare his theory and JBTD framework with the other researchers and practitioners. He states that a **core functional job** should be defined first. A proper job is stable (doesn’t change over time), has no geographical boundaries and is solution agnostic. It consists of a verb, object of the verb (noun) and contextual clarifier, like in the example:

“Listen to music while on the go”

Then, the metrics the customer uses to measure success are uncovered: the **desired outcomes**. ‘Minimize the likelihood that the music sounds distorted when played at high volume’, is an example of good measurable and controllable desired outcome. Then, you need to know what the **related functional jobs** are and define **emotional** (how customers want to feel or avoid feeling) and **social jobs** (how the customer wants to be perceived by others). It is common to list 50-150 desired outcomes, 5-20 related jobs and 5-25 emotional and social jobs for any given core

functional job. The last kind of jobs are **consumption chain jobs**, covering the product lifecycle: from the purchase itself, to learning, upgrading and disposing the product. Each consumption chain job has its own desired outcomes. Lastly, buyers decide whether or not to buy the product based on their **financial desired outcomes**.

According to Ulwick's JTBD theory, a product or service wins the marketplace if it helps customers get a job done better and/or cheaper [32].

Clayton Christensen

As a Harvard Business School Professor of Business Administration, Clayton Christensen is the foremost authority on disruptive innovation. According to Christensen, leading companies stay close to their customers and pursue sustaining innovations: they give customers something more or better in the attributes they already value. By doing so, they unintentionally open the door to disruptive innovations at the bottom of the market. These disrupters start by appealing to low-end or unserved consumers and then migrate to the mainstream market: they introduce a very different package of attributes [38].

The idea is further developed in Christensen's 1997 best-selling book *The Innovator's Dilemma* [39] and linked to JTBD in his later work [33]. He states that while the theory of disruption doesn't tell how a company should innovate to consistently grow, the theory of JTBD provides clear guidelines [30]. Christensen's milk shake dilemma can be read in most of his introductions to JTBD: A fast-food chain failed to sell more milk shakes, even after thorough marketing research and customer feedback. Only after observing the conduct of people buying a milk shake and asking them where exactly they were hiring the milk shake for, they discovered a lot of them shared one common job:

"Help me keep my morning commute to work interesting and keep me from being hungry by the time I get to work"

It turned out that the milk shake was competing with doughnuts, bagels and bananas, but did the job best because it isn't crumbling, you need only one hand and it takes some time to consume. One product can be hired for completely different jobs: in the afternoon the same milk shake can compete with a visit to the toy store or game of soccer, when the job is to "help me be a good dad and placate my children" [40]. Despite the popularity of the milk shake dilemma, critics point out that this marketing method can only be leveraged to sell more of a given product, instead of supporting innovation.

Christensen believes JTBD theory can help understanding the causal mechanism of customer behavior. It is about the progress a customer is trying to make. He emphasizes that apart from the **functional** dimensions of a job, also **emotional** and **social** dimensions that define the desired progress are critical. In his view, one job has multiple dimensions and a 'job spec' can be used to capture all relevant details. Apart from the three dimensions, the job spec can contain the tradeoffs the customer is willing to make, the competing solutions and the obstacles that must be overcome. After all, customers need to fire another product in order to hire your solution. The job spec becomes the actionable guide for innovation.

Alan Klement

Former software engineer Alan Klement is the author of the in 2016 published JTBD book *When Coffee and Kale Compete*. Besides writing books and blogs, he is a business owner / coach and investor. He defines a JTBD as "the process a consumer goes through whenever she aims to transform her existing life-situation into a preferred one, but cannot because there are constraints that stop her" [29, p. 32]. Captured in this definition is the high-level goal the customer has, referred to as 'be-goals'. This is opposed to 'do-goals', which are the activities you choose in order to fulfill this high-level goal. Klement argues that there are already a lot of design methods to design those activities or tasks and that they deal with a problem in root cause analysis. For example, if you state a customer wants to get a hole in the wall, the solution might be to buy a drill. But the need can also be to install bookshelves, and a method that doesn't require holes can serve as the

solution. But again, if you think one step further e-books might be the solution to the real need: storing books [41].

According to Klement, there are no different types of jobs. It is also not a job when you can visualize the customer acting it out, it describes something the customer doesn't like or it doesn't describe a better version of the customer [29]. A good JTBD can be used throughout the organization, gives room for interpretation but also offers boundaries, like [19]:

“Free me from the stress I deal with when figuring out what products won't harm my children, so I can have more time to enjoy being a parent”

The struggle for progress is why Jobs often begin with expressions like: give me, help me, free me, equip me, make the or take away. Klement's new insights into the theory faced heavy criticism of Ulwick, but the earlier mentioned Rick Pedi - one of the initiators of JBTD - supports his interpretations. He describes Klement's thinking as 'a breath of fresh air', in the foreword of Klement's book [29].

Conclusion

By analyzing the three different views on JTBD theory, we point out the dimensions and definition (do-goals vs be-goals) of Jobs as the two main disagreements between the authors (Table 1).

	Ulwick	Christensen	Klement
Definition	Do-goals: activities and tasks.	Do- and be-goals: progress, activities and tasks.	Be-goals: progress.
Dimension(s)	Functional. But emotional and social jobs are defined relative to the core job.	Multidimensional: functional, emotional and social.	Each Job is unique, and the type of Job is irrelevant.

Table 1: Comparing JTBD theories

After an analysis of 19 Jobs (8 from Ulwick, 6 from Christensen, 5 from Klement) that the authors gave in books or articles, we can find a common linguistic model of Jobs (Figure 3.3). Although little examples of Jobs exist in literature or in practice and a proper validation is necessary, we can argue that each Job consists of a struggle, goal and (optionally) contextual clarifier. A Job starts with an action verb and optionally an indirect verb. The examples include struggles containing phrases like help me, free me, prevent, find or listen. Then, the direct object is mostly a phrase or noun and is the object of the verb. Lastly, it is possible to bring context to the statement by providing a contextual clarifier. The contextual clarifier often starts with while, when, or so (that) but has a free form.

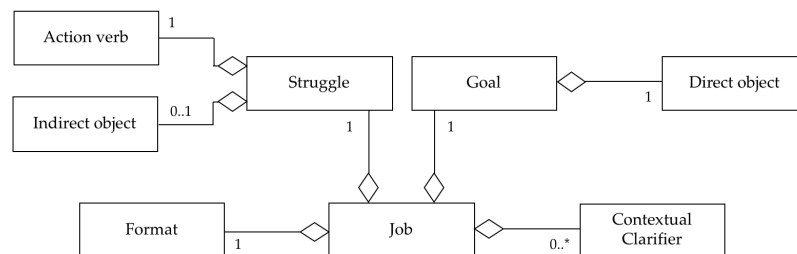


Figure 3.3: Linguistic Model of Jobs

3.2.2. Epic Stories

While a high-level Job shines a new light on a business, its customers and competition, it does not provide a tool for a design or product team to work with during software development. That is why in late 2013 the concept of Epic Stories was introduced by Klement, based on the approach of the product design team at Intercom [29]. Intercom develops a customer messaging platform. On their blog, Paul Adams (VP of product) describes how they use Clayton Christensen’s Jobs framework in their daily work: “We frame every design problem in an Job, focusing on (1) the triggering event or situation, (2) the motivation and goal, and (3) the intended outcome” [42]. An example of a good Epic Story adhering to this template is the following [43]:

“When I’m presenting my visual design and I’m worried that people will reject its merits, I want to back it up with something objective, so that people will see and discuss the design with less subjective bias.”

Klement originally named it a ‘Job Story’, but the term Epic Story is used in favor of Job Story in the RE4SA model and in this thesis. That is because an Epic is already known in agile development to describe a large User Story. User Stories are the smaller and implementable breakdowns of an Epic [44], but we will discuss the interplay between Epic Stories and User Stories in more detail at the end of this section.

Lucassen et al. [2] created a linguistic model of Epic Stories (Figure 3.4) based on an analysis of 131 Epic Stories. Both the problematic situation and expected outcome describe a situation. In this situation the problem lies or in the **action** an actor is executing, or the attributes of an actor or an object are in a problematic **state**, or the problematic situation is experienced because of an **external event** [2]. In the given example of an Epic Story, the problematic situation is a state and the expected outcome is an action. The majority of motivations comprise a subject, action verb and direct object. Most of them start with ‘I want’, like in our example, but motivations are free form text.

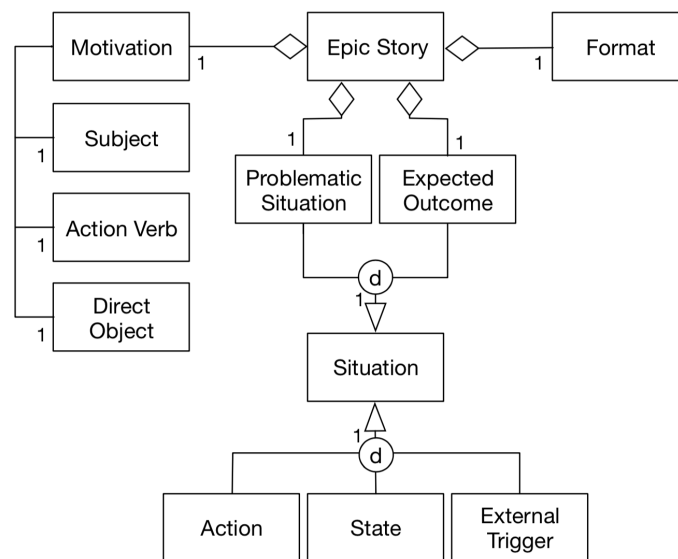


Figure 3.4: Linguistic Model of Epic Stories [2]

In a blog post, Klement covers five tips for writing Epic Stories [43]. He argues that (1) adding rich contextual information helps designing better solutions and (2) Epic Stories can only come from real customer interviews and not from personas. Furthermore, (3) you should not mix up Epic Stories with solutions. A situation can have multiple solutions, but you need to find one that fits with the other Jobs the product is solving. Therefore, Jobs should only cover situations, so they can be seen separately from solutions. It can be useful to (4) include forces in the motivation of the Epic Story and (5) you don't have to write Epic Stories from a specific point of view [43].

We have seen that Epic Stories focus on the *why*, instead of the *who* and *how*. According to the creators, this should stimulate creativity while designing the implementation. It also brings up an interesting discussion on how these Epic Stories relate to User Stories. Initially, Epic Stories are introduced as a replacement to User Stories. According to Klement, the persona in User Stories is irrelevant and there are too many assumptions about the desired solution in a User Story [45]. Klement suggests to use three 'layers' in JTBD: A higher level Job, smaller Jobs (which help resolve the higher level job) and Epic Stories [46]. In this thesis, we choose to adhere to the template of the RE4SA model. This means that a Job is always the highest level and we do not distinguish primary Jobs and secondary Jobs. A Job is broken down into Epic Stories. However, the most detailed specification of a requirement in the RE4SA model is a User Story. Thus, an Epic Story spans multiple User Stories.

3.2.3. User Stories

While the earlier discussed Jobs and Epic Stories are problem-oriented, User Stories are solution-oriented. Because of this difference in focus, it is suggested that there could be synergistic relationship between JTBD and other techniques like User Stories [2].

A User Story is "a description of a feature written from the perspective of the person who needs this. It consists of a written text, conversation about it and

acceptance criteria” [20, p. 87]. The written text is a semi-structured natural language statement. The most widespread format of a User Story is: “As a <role>, I want <goal>, so that <benefit>”, as used in the following example [3, 8]:

“As an administrator, I want to receive an email when a contact form is submitted, so that I can respond to it.”

The linguistic model of a User Story is displayed in Figure 3.5. A User Story adheres to a **template**, like the one we just described, and consists of three other parts: one role, one means, and zero or more ends [9].

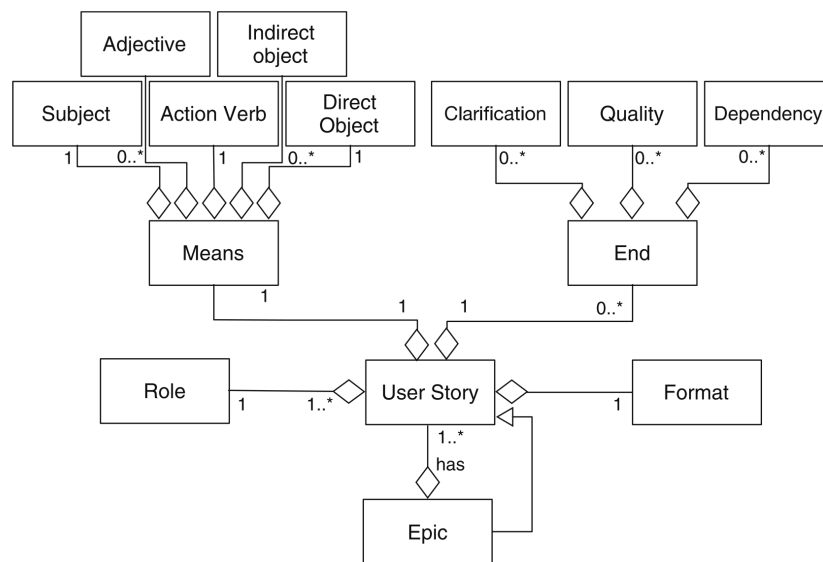


Figure 3.5: Linguistic model of User Stories [9]

The **role** defines what stakeholder or persona expresses the need. **Means** are free form text, but have in common that they have a subject, action verb and direct object (sometimes an adjective and indirect object). Although “I want’ is mostly used as subject, other phrases like ‘I am able to’ are also possible. The **end** part explains why the means are requested. They can either clarify the means, reference to a functionality which is required for the means to be realized or communicate the intended qualitative effect of the means [9].

Despite its popularity and fixed form, more than half of the User Stories contain easily preventable syntactic defects when it was tested against the Quality User

Story framework [9]. This framework defines 13 criteria for User Story quality based on its syntactic, semantic, and pragmatic correctness. For example, each User Story should be **well-formed** (has at least a role and a means) and **atomic** (expresses a requirement for just one feature). It is also required to be **minimal** (contains nothing more than role, means, and ends), **uniform** (employs the same template as the others) and **unique** (has no duplicates) [9]. Although the intrinsic User Story quality can be improved using automated natural language processing tools, as one study showed, practitioners do not perceive such a change [3]. Next to these newer quantitative quality measurements of User Stories, the INVEST framework of Bill Wake provides a widely accepted guideline to ensure the quality of User Stories in a more qualitative manner [47]. According to these criteria, a User Story should be independent, negotiable, valuable, estimable, small, and testable.

User Stories, or agile RE practices in general, often do not provide enough documentation for development teams alone. This happens when User Stories and backlogs are the only documents, causing traceability issues [21]. In order to transfer User Stories into architecture design and working code, this process can be supported by 'Delivery Stories' in large-scale projects [48]. A Delivery Story is an extension to a User Story and contains a functional specification, high level design and test scenarios. Multiple roles are involved in the translation process from a User Story to a Delivery Story (Figure 3.6).

Although the use of Delivery Stories was initiated by a project team with nearly 300 team members working for a single external client, also in smaller projects User Stories often come with a description that helps to give context to the story. In all cases, the goal is still to discourage long and complex specification documents [21].

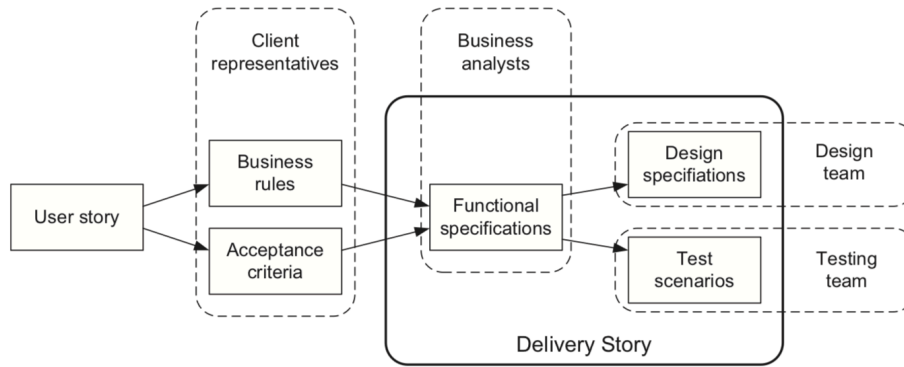


Figure 3.6: The translation from User Story to Delivery Story [48]

The RE4SA model contains an important assumption about User Stories, as indicated by the double headed arrow between User Story and feature in the model: User Stories map one-to-one on features. In terms of scope granularity, how much of the system’s functionality is implied in the User Story, it is recommended to split User Stories that take one week or more to implement [49]. This process of planning releases and iterations is covered in the well-known book by Mike Cohn [12]. A release plan consists of multiple iterations, and in each iteration a set of User Stories is delivered. Of course, next to discussing the desirability of the feature and cohesiveness to other stories, the stories cannot be prioritized without considering their costs. Developers give this estimate of the costs by assigning story points to each User Story based on its size and complexity relative to other stories, for example in a game of planning poker. User Stories are fully delivered within a single iteration, in terms of functionality and quality [12].

Next to the usefulness of User Stories in planning, it is also easy to deduct backlog items from a User Story so different team members, like a front-end UI designer and back-end developer can work on the same User Story in a sprint. Because User Stories are comprehensible, not only members of the organization can understand them, but also customers. In agile software development, User Stories are the most frequently used artifact [20] and they have proven to be of great value for both start-ups [50] and large organizations [48].

3.2.4. Crowdsourcing in Requirements Engineering

Although the principles of crowdsourcing can be traced back to the 18th century, it was first widely accepted in 2006. The definition then introduced by Jeff Howe has been adjusted by Mao et al. [51] to define Crowdsourced Software Engineering as follows:

“Crowdsourced Software Engineering is the act of undertaking any external software engineering tasks by an undefined, potentially large group of online workers in an open call format” [51, p. 61].

Three types of actors are generally involved in Crowdsourced Software Engineering: requesters, platforms, and workers (Figure 3.7).

Requesters offer software development work and post these tasks on a platform. It is a two-sided market: workers pull the tasks from this platform and provide solutions, like software code. An example of such a platform is Topcoder - an online marketplace using crowdsourcing for software development with a community of over one million software engineers [51].

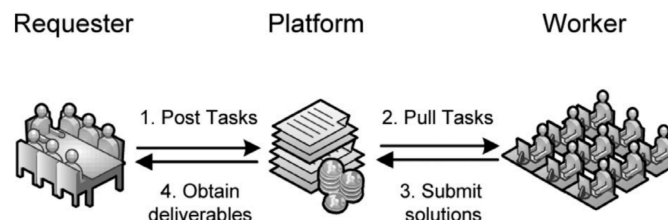


Figure 3.7: Actors in Crowdsourced Software Engineering [51]

The example of Topcoder demonstrates the broad usefulness of Crowdsourced Software Engineering: the *tasks* can cover anything from all phases of the software development life cycle, from planning to analysis, design, implementation, and maintenance. However, in this thesis we only discuss the opportunities for crowdsourcing in the RE (planning and analysis) phase. Crowdsourced RE has been investigated by a series of studies [51]. For example, Hosseini et al. [52] state that we need to rethink requirements elicitation to “accommodate the complexity

and the scale of the crowd and ensure that we get their requirements efficiently and precisely” [53, p. 2]. User involvement in RE can improve system acceptance, diminish project failure, deliver greater system understanding by the user, and improve customer loyalty and broaden the market [54].

In Crowdsourced RE, the requesters are the engineers, designers or software teams and the workers can be all stakeholders. In a case study of Snijders et al. [55], the workers were product managers, developers, experts, clients, end-users, and prospective clients. The platforms provide online (market)places for requesters and workers to meet. The most basic platform that provides this service is e-mail, but there are also several bespoke tools we will discuss in more detail.

Adepetu et al. [56] studied a conceptualized crowdsourcing platform, which they named **CrowdREquire**. In their paper, they provided a basic system, business model and market strategy for the platform. The main idea of CrowdREquire is to develop requirements for projects submitted by individuals or corporations. It uses a contest model: the final solution to a task is selected based on a competition among workers. The concept can be used as a reference for other platforms, the authors argue, but to the best of our knowledge no article about the actual implementation or validation of the CrowdREquire platform was published in the five years following the publication of the initial paper.

Soo Ling Lim, who received her doctoral degree for her dissertation on the use of social networks in large-scale requirements elicitation [57], published four well-cited articles about Crowdsourced RE [49 – 52]. She states that large-scale requirements elicitation (dozens of stakeholder groups and tens of thousands of users) deals with three problems: information overload, inadequate stakeholder input, and biased prioritization of requirements. The *Stakeholder- and Recommender-assisted method for requirements elicitation*, **StakeRare**, can address these problems by using social networks and collaborative filtering for requirements elicitation [58]. It is a four-step method in which the stakeholders are identified and prioritized first. The requirement engineer provides the initial set of stakeholders, but then these stakeholders can recommend others and give them a weight, based

on the level of influence this stakeholder has on the project. By doing so, a social network is built with stakeholders as nodes and their recommendations as links [59]. Second, the requirement engineer presents some initial requirements for the stakeholders to rate and asks them to provide additional requirements. The responses are used to create a unique profile of each stakeholder. In step three, a recommendation system is used to present the stakeholder relevant new requirements to rate. In the final step, the requirements are prioritized into a ranked list [58]. Lim et al. also developed a web-based tool that automates initially the first step [60] but now the entire process [61]. StakeRare has been validated in a case study at a software project in University College London, which provided “clear evidence that StakeRare effectively supports requirements elicitation” [58, p. 26]. It took less time to come up with an accurate list of stakeholder needs and the prioritized list of requirements was completer and more accurate compared to the existing method used in the project.

Different from commercial feature request solutions like Feature Upvote [62], Get Satisfaction [63], UserVoice [64], Cadet [65], Receptive [66], and Instabug [67] - **Requirements Bazaar** [68] is a free and open source web-based platform initiated by a research group from RWTH Aachen University. It has four pillars. First, a requirement page contains all information about a requirement. Not only basic metadata, but optionally also artifacts like User Stories and social interactions like comments, votes and commitments to help. Second, there is a co-creation workflow: from the idea generation and selection to the realization and release. During the process users can refine, negotiate, provide, test and acknowledge requirements and solutions. Third, a web plug-in allows the integration of the Requirements Bazaar into end-user and developer workspaces, like Jira. And fourth, the requirement prioritization is personalized. Just like the StakeRare method, a ranking score of a requirement is based upon different factors like the rating, importance and behavior of the stakeholders who voted for it [68].

By combining social media technologies with software engineering concepts, Greenwood et al. [69] presented the **UDesignIt** platform in order to empower communities to discuss and extract high-level design features. The researchers

explain how UDesignIt “combines Natural Language Processing with feature modelling to: identify key themes being discussed; group these themes according to their similarity to form a feature model-like structure; and automatically name the themes to ease the process of identifying the concepts being discussed” [69, p. 1321]. When they published the paper in 2012, the system was in early stages of deployment in community settings. However, no follow-up paper has been published to validate its practice, although the authors claim that the interaction with stakeholders, particularly non-technical stakeholders, could be enhanced by the platform.

Snijders et al. [7] advocate Crowd-Centric Requirements Engineering, by combining crowdsourcing and gamification to involve users in the elicitation, negotiation and prioritization of requirements. According to the researchers, this helps fostering user involvement, is valuable not only in early stages of RE and gives equal priority to both customers and end users when they are not the same. The Crowd-Centric Requirements Engineering method consists of six phases: (1) feasibility analysis, (2) context analysis, (3) crowdsourcing preparation, (4) crowd involvement, (5) requirements identification, and (6) focus group execution. The next phase is a development sprint, which is not part of the CCRE method itself [54]. The embodiment of their vision is **REfine**, a gamified platform for eliciting and refining requirements. Dalpiaz et al. [54] showed in a case study how the users perceived this crowdsourcing platform as more useful and more engaging compared with previous feedback experiences. There were also some challenges. For example, the experiment showed difficulties in engaging a lot of clients and end-users of the software product. They were also worried that the quality of requirements would not be significantly better than the quality of the experts' methods, and the requirements may not be detailed enough for a focus group or product backlog. The authors think that simple formalisms such as User Stories will improve the quality of the requirements and thereby can mitigate this risk. Näkki, Koskela and Pikkarainen [57] used such a concept of needs-based User Stories. They collected users' everyday needs and challenges regarding a specific domain. Users were involved during requirements elaboration, by commenting and rating features in order to allow the prioritization of features. It turned out

that facilitators were still needed to translate the semi-formal user data into software requirements.

The quality of crowdsourced requirements is an interesting topic that is frequently mentioned in the literature. One of the main incentives to involve the crowd in software development in general [70] and RE in specific [51] is to achieve higher quality, but it can be a challenge or limitation at the same time [45, 49, 55, 56]. Next to the proposed idea of formalizing the structure of the crowdsourced requirements, Sherief et al. [72] developed an architecture for structured feedback modelling that uses a controlled natural language engine, feedback ontology reasoner and knowledge base in order to improve the quality of users' feedback. They argue that feedback acquisition should be designed with the goal to maximize the expressiveness of users' feedback and still be able to efficiently analyze it. From online feedback forums, the researchers derived eight types of feedback: confirmation or negation, investigation, elaboration, justification, verification, problem feedback, mitigation and correction. They also noted that there are different detail types of feedback (concise, explanation, exemplification, trials, scenario, feature definition, and question) and users use four different methods to provide feedback (text, code snippets, snapshots, and links).

Based on focus groups with 14 users and developers and an online survey with 34 RE experts, Hosseini et al. [52] listed ten challenges and issues related to crowdsourcing for requirement elicitation. In Table 2, we cover the challenges by briefly highlighting the main positive and negative attributes of the crowdsourcing features.

Feature	Positive and negative attributes
Largeness	<ul style="list-style-type: none"> + Maximize accuracy, relevance and saturation + Minimize problem of missing requirements - Coordination - Strictness of platforms should not harm voluntary nature of participation
Diversity	<ul style="list-style-type: none"> + More relevant and creative requirements - Difficult to reach agreement, especially with geographical diversity

Anonymity	+ Enhances honesty of users, improves quality and quantity of comments
	+ Assuring participants' privacy and security
	- Allows malicious participants to join in
	- Discourages participants who care more about social recognition
Competence	+ High competence in the crowd
	+ Micro tasks can easily be amenable to novice crowd workers
	- Recruiting competent crowd brings additional financial costs
Collaboration	+ Realize rationale for requirements and having holistic solutions
	- Clustering and dominance of certain opinions, trends, and groups
Intrinsic motivations	+ Participants give better quality information because they are genuinely interested in the software
	- Motivation may lead to bias and strong views on what requirements the system should fulfil
Volunteering	+ Core element of crowdsourcing, no work for pay
	- Participants getting demotivated over time
Extrinsic incentives	+ Participants take responsibility for non-intrinsically interesting tasks
	- Not necessarily more reliable requirements, but higher costs and more effort
	- Extrinsic incentives can harm intrinsic motivations
Opt-out opportunity	+ Loose contractual model based on voluntary participation
	- Inadequate incentives or higher complexity of tasks may lead participants to opt-out
Feedback	+ Giving feedback can improve the performance of participants and motivate them to persevere and accept more tasks
	- Unclear how to decide what feedback to give and when
	- Can lead to elimination of diversity of opinions

Table 2: Challenges of crowdsourcing for requirements elicitation [52]

Next to the bespoke Crowdsourced RE tools we discussed, Maalej et al. [73] envision a paradigm shift towards data-driven RE. Since users can easily submit feedback in app stores, social media, or user groups and software suppliers collect analytical usage data, this information can be used for the elicitation and prioritization of requirements [73]. Researches have focused on automatically classifying user feedback, classifying stakeholders and summarizing reviews. By

doing so, feedback does not only lead to a change in the visibility and sales numbers of applications in the app stores, but also in better products.

3.3. Software Architecture

There is certainly no lack of SA definitions, with over 150 definitions collected from literature and practitioners [74]. The Software Engineering Institute at the Carnegie Mellon University published an article with the most used definitions, split into three categories: modern, classic and bibliographic [75]. Two of the researchers who worked at this institute are Len Bass and Paul Clements, who can be seen as authorities on the domain of SA. In 2003, they co-authored the book *Software Architecture in Practice* in which they defined SA as follows:

“The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them” [76, p. 3].

There are five conditions captured in this definition: (1) architecture defines elements, (2) systems can comprise more than one structure, (3) every software system has an architecture, (4) the behavior of each element is part of the architecture and (5) the definition is indifferent as to whether the architecture for a system is a good one or a bad one [76].

A second modern definition comes from the ISO/IEC/IEEE 42010:2011 International Standard: *Systems and software engineering – Architecture description*. The IEEE Standard 1471:2000 from a decade earlier is also frequently cited, but now superseded by the revised one, which includes the following definition:

“The software architecture contains the fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution” [77, p. 2].

The key ideas in this definition are that (1) an architecture names that which is fundamental or unifying about a system as a whole, (2) it is a conception of a system, and (3) it is understood in context [77]. This definition uses ‘elements’ instead of the word ‘components’, that was used in earlier definitions, because the latter was frequently misunderstood as referring to software components.

SA plays a pivotal role in supporting organizations to meet their business goals. From a technical perspective, it is extremely valuable as well. SA facilitates the communication among stakeholders, supports early design decisions and provides a transferable abstraction of a system [76]. To allow an architecture to be portable and usable by many different stakeholders, Philippe Kruchten [78] introduced what would become the widely accepted notion of views. A view is “a representation of one or more structural aspects of an architecture that illustrates how the architecture addresses one or more concerns held by one or more of its stakeholders” [15, p. 34]. Kruchten’s “4+1” View Model from 1995 (Figure 3.8) breaks the architecture down in five views [78].

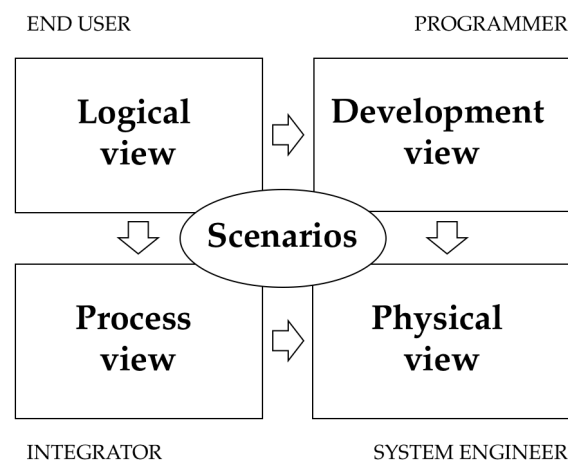


Figure 3.8: The “4 + 1” View Model of Software Architecture [78]

The **logical view** shows the components of the system and their interaction or relation. It considers the functional requirements: the services the system provides to its end users. The **process view** is about the non-functional requirements like performance and availability and shows the processes of the system and the communication between those processes. In the **development view** the

organization of the software modules is shown: the building blocks of the system from a programmer's perspective. The **physical view** shows the system execution environment and is also known as the deployment view. The architect uses the last view, the '+1', to illustrate and validate the other views by making use of **scenarios**. These scenarios are instances of use cases and describe sequences of interactions between objects and processes [78].

Each view is governed by a 'viewpoint': a concept proposed by the IEEE Standard. It is "a collection of patterns, templates and conventions for one type of view" [15, p. 36]. Rozanski & Woods [15] present seven core viewpoints for information systems architecture: context, functional, information, concurrency, development, deployment, and operational. These viewpoints help to decide what views to produce and the level of detail that goes into these views. Using different views to represent an architecture is the best way to address different stakeholder concerns, without making a model too complex.

When modeling a software architecture in a formal way, it is also valuable to adhere to a modeling language. There are roughly two choices. The first option is to use the Unified Modeling Language (UML) and adapt it to suit your needs. UML is a general-purpose semiformal modeling language, and the most used modeling language in the industry [79]. However, it does not support architectural concepts (like layers) and can be a source of ambiguity and inconsistency [27]. The alternative is to use a special-purpose notation: an architecture description language (ADL), although some people consider UML as an ADL as well [79]. An ADL is a formal graphical language for representing a software architecture [74]. Research groups around the world proposed their own ADLs, like the Carnegie-Mellon University [15]. Based on the techniques used and taught at Utrecht University, two fellow students of the Master Business Informatics proposed the uADL [80]. The modeling techniques used in this thesis are all part of this new uADL.

Interestingly, with the rise of agile software development, some developers ignored or despised SA. Kruchten touched upon these perceived tensions between

SA and agile software development in his workshop *Software Architecture and Agile Software Development—A Clash of Two Cultures?* [81]. He argues that although SA is sometimes pictured as a typical non-agile process, projects that lack architectural focus will fall behind. Organizations should balance between adaption (agility) and anticipation (architecture) in which it is important to understand the context to define how much architecture is needed for a given project [81]. Based on a survey of software developers, Falessi et al. [82, p. 25] concluded that “agile developers perceived software architectures as important and supportive to agile values”. The complexity of the project – mostly based upon the lines of codes, number of requirements, number of stakeholders, or geographic distribution – was perceived as an important decision criterion for when to focus agile development on software architecture [82]. A frequently proposed solution to avoid having an inappropriate architecture in agile software development, is code refactoring. Refactoring is an ongoing activity among agile teams [21] and used by Extreme Programming (XP) to replace upfront design [12]. Refactoring entails the action of restructuring or rewriting code so the code gets improved without changing its observable behavior [12]. It relies heavily on self-checking tests to avoid bugs. You need to refactor first before adding a feature, if the program’s code is not structured in a convenient way to easily add the feature [83].

In this chapter we first discuss the applications, modules and features that are the core elements of each software architecture. These elements can be diagrammed using respectively a Context Diagram, a Functional Architecture Diagram and a Feature Diagram. The Context Diagram gives a representation of the system and the corresponding external roles, from a context viewpoint. The Functional Architecture Diagram highlights the primary functionality of the software product, consisting of its main functions and supportive operations, from a functional viewpoint. From this same viewpoint, the lower-level Feature Diagram shows which features are present within the different modules. Lastly, we discuss software architecture recovery, a technique to extract architectural information from the source code or graphical user interface of a software system.

3.3.1. Software applications

In the second edition of the book *Software systems architecture: working with stakeholders using viewpoints and perspectives*, Rozanski & Woods [15] introduced a new viewpoint: the context viewpoint. It can be seen as the ‘overarching’ viewpoint that sets the scope for the other viewpoints. The context viewpoint describes “the relationships, dependencies, and interactions between the system and its environment (the people, systems, and external entities with which it interacts)” [15, p. 41]. It is important since few systems exist in isolation. Furthermore, many stakeholders can benefit from the context viewpoint, as it helps them to understand their responsibilities in relation to the system and their organization.

An uADL technique to create a view from a context viewpoint is a **Context Diagram**, or System Context Diagram. This view pictures the software system in the center, with associations to external entities that interact with it, either directly or indirectly. These entities (or actors) can include systems, roles, databases or other services. They are sources for input into the system and destinations for outputs from the system. The elements that are transferred in these associations can be categorized into four forms: data, signals, materials, and energy. The associations, that are modelled using arrows, can also show activities [16].

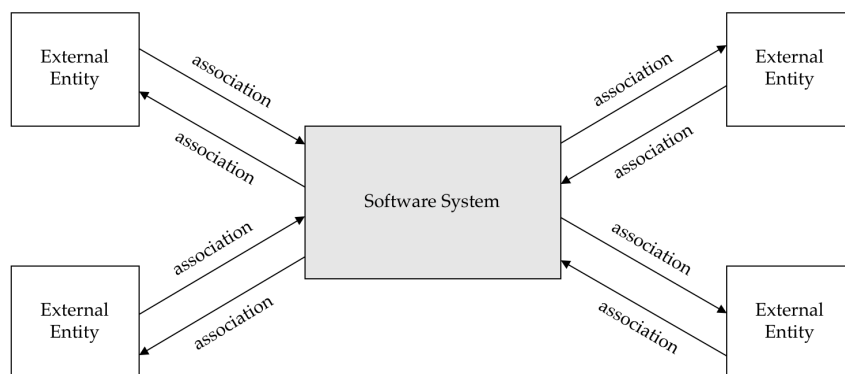


Figure 3.9: Generic Context Diagram

Since this is a high-level view, no details of the interior structure of the system are pictured. Therefore, it is known as a ‘black box diagram’. If the external entities

include other software systems as well, then they should also be pictured as a black box. This is the case when you want to model multiple applications in a software product line, for example. Kossiakoff et al. [16, p. 266] explain how a System Context Diagram can serve as “a useful starting point for describing and defining the system’s mission and operational environment, showing the interaction of a system with all external entities that may be relevant to its operation”. Figure 3.9 shows a generic context diagram containing the software system, external entities and the associations between them. Based on the example of [17], a Context Diagram for a collaborative authoring tool is shown in Figure 3.10.

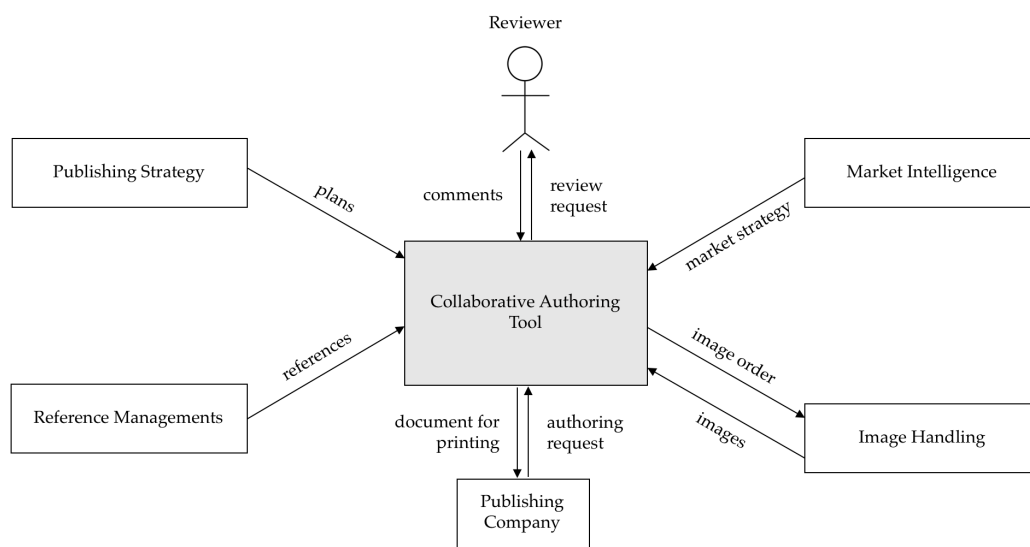


Figure 3.10: Context Diagram for a collaborative authoring tool

3.3.2. Modules

‘Zooming in’ on the software system (the ‘black box’) from the Context Diagram, will show the functional architecture. We get to a point where all architecturally significant modules are shown: it does demonstrate the functionality of the system but does not show all specific features that are part of the modules. Defining and refining a functional view takes most time, but then drives the definition of the other views. The functional view “documents the system’s functional structure – including the key functional elements, their responsibilities, the interfaces they expose, and the interactions between them” [15, p. 41]. It addresses mainly concerns of stakeholders with no technical expertise, by reflecting the software product's architecture from a usage perspective [17].

A **Functional Architecture Diagram** (FAD) contains modules that show the functions of a software product. Since functions are implemented by modules, this diagram indicates which modules need to be developed for each functionality. The goal is to create modules that can be developed independently of other modules. Next to this concept of modularity, each system should run in different operations systems, platforms, and customer organizations (variability) and a product can interoperate with external factors by the use of an interface (interoperability) [17].

Although a FAD can be designed for different levels, we take the product scope level. The breakdown of modules into features is done in the Feature Diagram, which we discuss in the next section. A FAD consists of modules, information flows and a product scope. These are modelled by respectively boxes, arrows and a rectangle, as shown in the generic FAD in Figure 3.11.

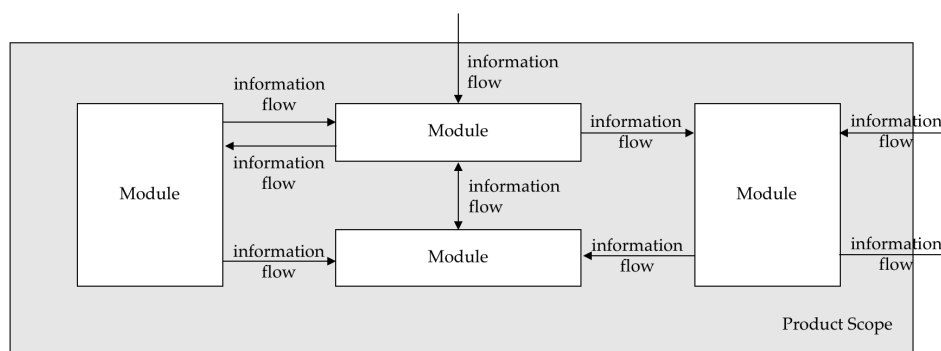


Figure 3.11: Generic Functional Architecture Diagram

It is possible to incorporate **scenarios** with the FAD in order to “visualize the flow between the product’s modules and third party applications for the implementation of the system’s functionalities” [17, p. 212]. We discussed these scenarios earlier when we covered Kruchten’s 4+1 View Model. They are an abstraction of the most important requirements [78] and represent behavior coming from the output of User Stories [80]. These customer journeys validate and illustrates if the architectural design meets the requirements. The German professor Martin Glinz presented a clear definition of a scenario in his discourse for the use of scenarios to improve the quality of requirements:

“A scenario is an ordered set of interactions between partners, usually between a system and a set of actors external to the system. It may comprise a concrete sequence of interaction steps (instance scenario) or a set of possible interaction steps (type scenario)” [84, p. 56].

In a FAD, a scenario is an overlay to the diagram. It contains arrows and sequence numbers to indicate a customer journey. It includes a written description as well. According to Glinz [84], scenarios are easy to understand, give the users a feel for what they get, provide a decomposition of a system into functions, allow short feedback cycles, and allow test cases to be directly derived from them. An example of a FAD with a scenario overlay is shown in Figure 3.12.

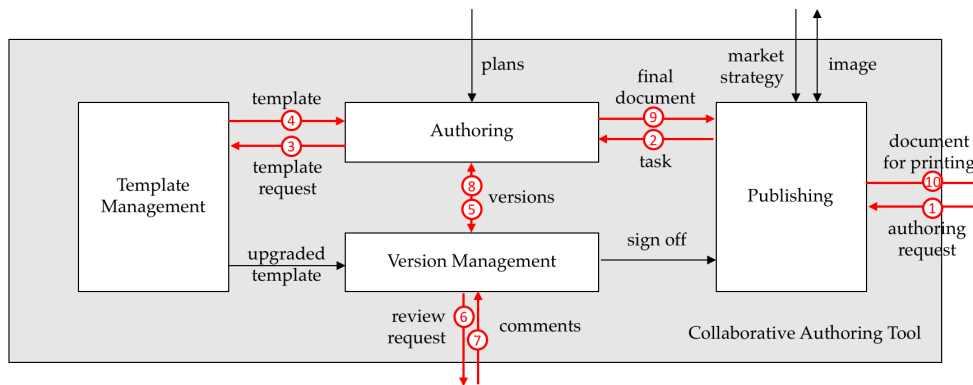


Figure 3.12: Functional Architecture Diagram for a collaborative authoring tool, with a scenario overlay

3.3.3. Features

From the same functional viewpoint, but on a lower-level, a **Feature Diagram** shows which features are present in the modules of the FAD. Although the term Feature Diagram is also used to represent different products in a software product line, the definition of the uADL limits the scope of a Feature Diagram to one specific module of one software product. Optionally, one can prioritize the features in a Feature Diagram. A Feature Diagram has a tree data structure, which is shown in the generic diagram in Figure 3.13.

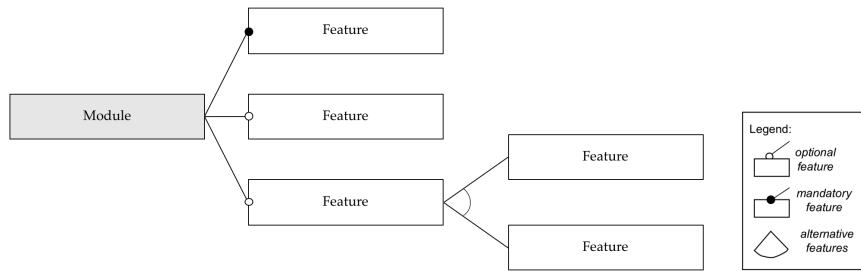


Figure 3.13: Generic Feature Diagram

While many definitions of a feature exist, the definition from the Feature-Oriented Domain Analysis [85] is frequently cited. They adapted the general definition of a feature from the American Heritage Dictionary to make it applicable for software systems:

“A feature is a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems” [85, p. 3].

It is also possible to show optional or alternative features in a Feature Diagram [18] as shown in the example in Figure 3.14.

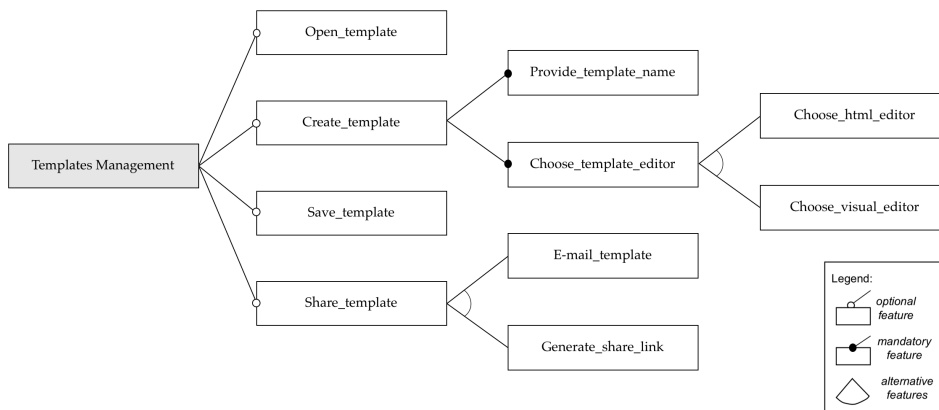


Figure 3.14: A Feature Diagram for the Templates Management module of a collaborative authoring tool

3.3.4. Software architecture recovery

Since a software product is constantly evolving, a small project with no documented software architecture can grow into a complex and big system. Or

systems are developed over a period of decades by a lot of developers. Practice and theory showed that in these cases “maintaining good quality software architectures is non-trivial” [86, p. 1458]. For web applications the problem may be even worse: a proper documentation is rare, well-known software-engineering practices are not well adopted by web developers and there is a high employee turnover rate [87]. To improve the understanding of these applications or systems, reverse engineering and system visualization techniques have been proposed [87]. Reverse engineering has its origin in the analysis of hardware, where it is common to extract the design from a finished product to improve a company’s product or analyze a competitor’s product [88]. Chikofsky & Cross [88] define reverse engineering as follows:

“Reverse engineering is the process of analyzing a subject system to identify the system’s components and their interrelationships and create representations of the system in another form or at a higher level of abstraction” [88, p. 15].

Obviously, it is the opposite of forward engineering which entails the traditional process of moving from high-level designs to physical implementation. All terms, the relationship between them and transformation processes between or within abstraction levels, are shown in Figure 3.15.

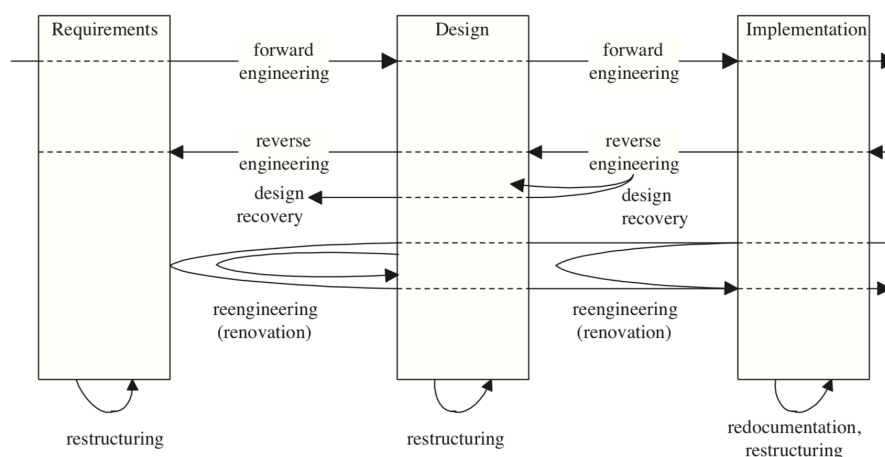


Figure 3.15: SA terms and their relationships [88]

The objectives of reverse engineering include: cope with complexity, generate alternate views, recover lost information, detect side effects, synthesize higher abstractions and facilitate reuse. But the primary purpose is to “increase the overall comprehensibility of the system for both maintenance and new development” [88, p. 16]. In contrast to the restructuring and reengineering processes, reverse engineering does not involve changing the system. Three activities are mostly part of the reverse engineering process. First, **extracting** information from system artefacts (e.g. source code, design documentation), system experts and system history. Next, **abstracting** the information to a higher (design) level. And finally, **presenting** the information to stakeholders in a friendly way [89]. We apply reverse engineering in this thesis only to recover an architecture (a design) from its implementation. This process is also called reverse architecting and helps to incrementally improve an existing system, instead of rebuilding the entire system from scratch with a new design. In his PhD thesis, former Philips software engineer, software architect and project manager, Rene Krikhaar [89] sees reverse architecting as one of the three typical activities in architecture improvement. The architecture models that are retrieved by **reverse architecting** can be balanced with the ideal architecture retrieved from **forwarding architecting** to create an improved architecture during the **re-architecting** activity (Figure 3.16).

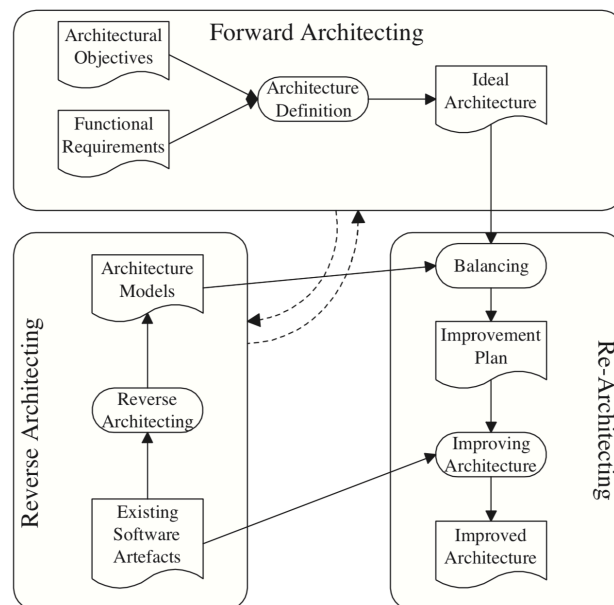


Figure 3.16: Architecture Improvement Process [89]

Automated architecture recovery techniques include the use of dependencies between code files [90], dedicated extractors for web applications [87], specialized tools that can analyze the source code [6] or graphical user interface testing tools [91].

3.4. Summary

In this literature review we covered both forward engineering and reverse engineering as techniques to support the maintenance and development of existing software products.

In section 3.2 we discussed RE as a forward engineering process in which requirements serve as a valuable input for project planning, risk management, trade off, acceptance testing and change control. Requirements are a critical success factor for projects, but JTBD theory shows that asking customers what features a product should contain is bad practice. Customers should not be trusted to come up with these solutions. Instead, a company should discover why a customer *hires* its product: what job the customer is trying to get done. In the process of capturing the JTBD, customer interaction and user involvement are vital. A crowdsourcing platform facilitates this cooperation and can reveal what customers want a product or service to do for them.

In section 3.3 we discussed how reverse engineering can help to recover a SA from an existing project based on its lower level abstractions like the user interface or code base. A SA facilitates the communication among stakeholders and plays a pivotal role in organizations to meet their business goals. In order to capture a complex system in a model and to allow the system's architecture to be portable and usable by many different stakeholders, we use different views from a context, functional, information, concurrency, development, deployment, or operational viewpoint. Although UML is frequently used as a modeling language, it is a source of ambiguity and inconsistency. Therefore, the diagrams in this thesis originate from the formal uADL. Earlier, in section 3.1 we have shown how RE and SA are linked based on the

RE4SA model. The product can be improved using re-architecting by balancing the outputs of the forward engineering and reverse engineering processes. The low-level requirements in the form of **User Stories** can be mapped to **Features**, captured in a recovered Feature Diagram. In a higher level of abstraction, features are grouped into **Modules** in a recovered Functional Architecture Diagram – in the same way in which User Stories are grouped into big **Epic Stories**. If we take the context of the **Application** into account as well, a Context Diagram shows all associations to external entities that interact with the software system. This application satisfies a high-level process the customer is trying to execute: A **Job**.

4. Case study

We discussed the use of crowdsourcing and architecture recovery in relation to the RE4SA model in the previous chapters. The single-case study we will cover in this chapter entails the design and validation of our treatment using a Technical Action Research. We start by introducing the company where we perform the case study: Tournify. This Dutch startup provides an online tournament manager for sports organizers, delivered as a service. Then, we cover the case by discussing how we recover the technical architecture and develop the requirements crowdsourcing platform. Lastly, we present the evaluation protocol which will serve as the basis for the validation.

4.1. About Tournify

Tournify is a small software development company based in Amsterdam, The Netherlands. Their services, which are provided online, are targeted at sports and e-sports tournament and competition organizers. Their main product is the Tournify tournament manager. This web application allows tournament organizers to manage the participants, create a match schedule based on a chosen tournament format and process the results as the tournament progresses. The organizer can also use the tournament manager to create a tournament website, which is used to present the event to the audience. The athletes and supporters are able to view the schedule, results and standings by visiting this tournament website on their mobile phone, as new information comes in real-time (Figure 4.1). The tournament website can also be displayed as a slideshow, when a big screen or beamer is available at the tournament venue.

We consider the tournament website as a separate application and will not include this application in our study. We will simply use 'Tournify' to refer to the Tournify tournament manager in the following of this thesis.

Tournify is written in Javascript. It uses React, an open-source JavaScript library developed by Facebook, for creating the interactive user interfaces. For the dynamic content Firebase is used: a mobile and web development platform maintained by Google that allows for storing and syncing data across multiple clients. The total lines of code (LOC) is near 25.000 and around 110 components are used.

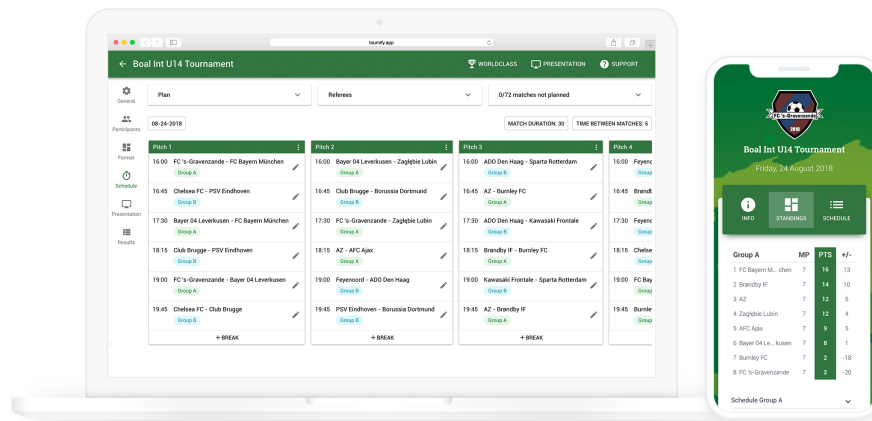


Figure 4.1: The Tournify web app consists of a tournament manager for the organizers (left) and a tournament website for the athletes and supporters (right)

The company is founded by two Information Science students, who started developing the service mid 2015 for a client in the e-sports branch. Since late 2017, the service is publicly available online and the focus of the founders pivoted towards regular sports tournament and competitions. They currently report over 10.000 registered users (tournament organizers) from The Netherlands and Belgium. Customers include professional football clubs like Ajax Amsterdam and AZ Alkmaar and many amateur sports clubs. Next to (indoor) football tournaments, Tournify has been used for different sports - including (indoor) hockey, (beach) volleyball, (table) tennis, basketball, rugby, darts, korfbal, water polo, pétanque, curling, and bowling. Tournify uses a freemium business model, as well as a subscription business model. Users can decide to upgrade from the free version to one of the premium packages, if the number of participating teams in that tournament exceeds the limit of eight teams. The subscription models are targeted at organizations that host eight or more tournaments each year.

The company is not backed by any external funding and is still run by its two founders: one software developer and one product manager / sales representative.

The English version of Tournify can be found at www.tournifyapp.com and you can get in touch by sending an email to info@tournifyapp.com. For the Dutch site, please visit www.tournify.nl.

4.1.1. Applying the principles of Canonical Action Research

It is important to mention that the researcher of this thesis is no independent researcher, as he works for and investigates on the case study concurrently. This personal involvement allows for an unbounded access to the development artifacts and stakeholders, and a comprehensive knowledge of the organization and business processes. At the same time, it raises relevant questions about possible biases and prejudices. Action Research in general has also been criticized for its “lack of methodological rigor, its lack of distinction from consulting and its tendency to produce either research with little action or action with little research” [22, p. 65]. To ensure the rigor and relevance of this study, we make sure the five principles of Canonical Action Research (CAR) are being taken into account seriously. This set of principles and associated criteria are developed by Davison, Martinsons & Kock in 2004, to allow for a study in which organization problems are addressed while at the same time contributing to scholarly knowledge [22]. We cover each principle in detail and explain how we (plan to) meet the associated criteria or try to present justified reasons for not doing so. The full table with all criteria can be found in Appendix 1.

1. The Principle of the Researcher–Client Agreement (RCA)

This agreement is described as being the guiding foundation for a project. The first four criteria of this agreement were met before the project was formally initiated. There are four main roles and responsibilities specified (criterion 1d), as described in the table in Appendix 1.

Since the product manager of the client company is also the lead researcher of this study, the wish to perform Action Research was expressed during the first meeting with the supervisor. Before the project formally started, there was already an agreement that CAR was the appropriate approach for the organizational situation according to both client employees and the supervisors (criterion 1a). They also committed themselves explicitly to the project, by the means of a thesis application form that is approved by the Board of Examiners of Utrecht University (criterion 1c). This application form also included the scope (section 1.2), research focus (section 2) and adheres to the deadlines set by the University (criterion 1b). The total duration of the project is eight months: three months for the literature study and five months for the case study.

The final two criteria for the RCA include the explicit specification of project objectives and evaluation measures (criterion 1e) and data collection and analysis methods (criterion 1f) which will be covered in the Evaluation protocol (section 4.5).

2. The Principle of the Cyclical Process Model (CPM)

The activities we undertake during this project, follow the CAR process model (criterion 2a) as shown in Figure 4.2. We expect to complete the project satisfactorily in a single cycle, but additional cycling through the stages may be possible.

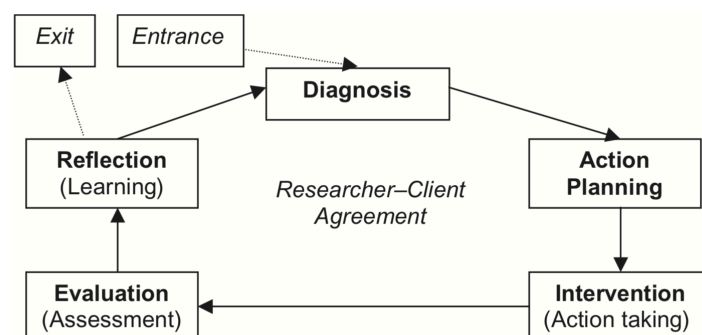


Figure 4.2: The CAR Process Model

In section 4.2, we will introduce the case and conduct a diagnosis of the organizational situation (criterion 2b). Although the case will be introduced in this part of the thesis, the researchers were all aware of this case when the project started. While CAR is problem-oriented, the TAR we perform in this study is solution-oriented. This means that although the diagnosis of the organizational situation can be considered as the entrance for this study, it is only used to discover how the RE4SA model can be best applied in the organization. The diagnosis is also not independent, since this would require a lead researcher from outside of the client organization. Because of this studies solution-oriented approach, we do not expect that this deviation of criterion 2b will harm the validity of this study. The goal of the diagnosis is to uncover development processes in which the RE4SA model may be utilized to improve the current situation.

The action that follows is twofold: the architecture will be recovered and a requirement crowdsourcing platform will be designed and implemented (criterion 2c and criterion 2d). An evaluation on the outcomes follows in the Results section of this thesis on which we will reflect in the Discussion of this thesis (criterion 2e). In this section, we will also decide if an additional process cycle is needed (criterion 2f), if the project objectives are met or that there is any other justified reason to conclude the project (criterion 2g).

3. The Principle of Theory

The principle activities in this thesis are guided by RE and SA theories (criterion 3a). In section 2.3, the relevance of this study for both scientists and practitioners in information sciences is highlighted (criterion 3b). We investigate commonly used artifacts in this domain and try to find how these artifacts are linked using the RE4SA model (criterion 3c). We apply existing techniques like (requirement) crowdsourcing and architecture recovery to the specific case study and thereby build upon the RE4SA model (criterion 3d). This guiding theory is also used to evaluate the outcome of the intervention (criterion 3e).

4. The Principle of Change through Action

In their article, Davison et al. [22, p. 75] emphasize that “taking actions in order to change the current situation and its unsatisfactory conditions” is the essence of CAR. Both client and researcher need to be motivated in order to improve the situation (criterion 4a) and any planned actions must be approved by the client (criterion 4d). Since the lead researcher of this study is also involved in the client organization, this commitment is a known fact. The other co-founder of the client organization is also committed to the project and sees the benefit of the planned actions from both a business and a scientific perspective. The timing and nature of these actions is taken clearly and will be documented in the following sections (criterion 4f). We do obviously assess the organizational situation both before and after the intervention (criterion 4e).

As mentioned earlier, the case will be introduced in section 4.2. We will specify the problem and hypothesized cause(s) based on the diagnosis of the organizational situation (criterion 4b). However, we then try to investigate how the use of the RE4SA model and its related artifacts, as well as the use of architecture recovery and crowdsourcing, can address the hypothesized cause(s) (criterion 4c). Although other solutions may exist to address the same situation, that may even produce better results, the goal of this research is to validate a model in an organizational context (solution-oriented) as opposed to solving the client’s problem in the best manner, as is usually the case in problem-oriented CAR.

5. The Principle of Learning through Reflection

The action researcher has a responsibility to the client and to the research community. The lead researcher needs to provide progress reports to the client members (criterion 5a) and other researchers. Throughout this research, this is done during bi-weekly meetings with the research group and weekly phone calls with the other client co-founder. Most of the learning through reflection happens in the final stage of the cyclical process model and can be found in chapter 5 and chapter 6 of this thesis. In these chapters, we report clearly and completely on the outcomes and keep a clear distinction between facts and judgements (criterion 5a). The researcher and client also reflect on the outcomes of the project (criterion 5b)

and the results are considered in terms of implications for further action in this situation (criterion 5c), for action to be taken in related research domains (criterion 5d), and for the research community (criterion 5e). We will also comment on the benefits and limitations of the TAR methodology with the CAR principles for this project (criterion 5g).

4.2. The case

Based on the internal observations of the two founders, two main challenges are identified regarding the software development at Tournify.

The first challenge concerns the missing of development artifacts. There is little code documentation and the software architecture has never been modelled. There is a minimal use of comment lines in the code. As the code base is growing, it becomes more difficult to keep an overview and to assess the impact a new feature has on the existing components.

A support section on the website serves as the user manual. This section contains a written description (1500 words) on the use of the application, a section with 30 frequently asked questions and related answers (1700 words) and there are five explanatory videos with a total length of 10 minutes. The support information is incomplete and a challenge to keep up to date, since new features are deployed quickly.

An online Kanban board is used to communicate requirements between the product manager and developer, but no format on the written descriptions of requirements is used. There are over 250 cards on the board which are completed or archived and approximately 100 cards still open.

The second challenge concerns the handling of customers' feature requests. In five-months' time 44 unique customers requested 77 new features (Appendix 2). Most of them are requested via email or via the support chat. The requests that came in by phone are not properly registered and therefore not included in this overview.

In the end of February 2019, a total of 12 requests has been implemented in following releases. The majority of the requests has been added to the backlog in the online Kanban board. More than half of the users who requested features are paying customers. In only 11% of the cases, we can certainly state that the requesters stopped using the application because the service was unusable for them without meeting their request(s). The requesters organize tournaments in 16 different sports and 2 different e-sports (Table 3). The requests are written in Dutch and contain 192 characters on average.

Source	#	Status	#
E-mail	44	Implemented	12
Support chat	29	Pending	65
Other	4		
Total	77	Total	77

Requesters	#	Tournament types	#
Free users	17	Unique sports	16
Paid users	27	Unique e-sports	2
Total	44	Total	18

Table 3: Feature requests proposed by the Tournify end users between September and January 2019

Further analysis of the requests and conversion of the text to User Stories caused no difficulties in 71% of the cases: the **role** was clear, a **goal** was expressed, and the potential **benefit** was highlighted. Consider the following (translation of a) request that came in:

We organize tournaments in different venues. Some have excellent WIFI, others work badly or there simply is none. So, is it possible to create a tournament online, but then continue offline to enter the match results?

We can convert this request into the following User Story:

As an organizer, I want to enter the match results in an offline mode so that the application can be used in venues with bad or no WIFI network.

With 29% of the requests, the benefit was not explicitly mentioned. Although this benefit is optional in a User Story, it may provide valuable information, especially when a request comes in without any context via email or chat. Consider the following request that came in:

Why is the number of teams I can add to a group limited? I want to place 46 teams into one group.

The corresponding User Story would be:

As an organizer, I want to place 46 teams into one group.

This is a valid User Story but raises questions since it is unknown why one wants to place this many teams into one group: even the biggest leagues in the world have place for a maximum of 20 teams. Only after further communication between the product manager and requester, it becomes clear why this user wants this feature: “We work with different minigames. I’ve added the minigames to a group and want to add all teams to this group as well, so they play ‘against’ each minigame and I can include these matches in the match schedule.” Although the solution is creative, it clearly is a workaround. The user doesn’t want to place 46 teams into one group, he wants to host a tournament with different games (currently Tournify is built to host tournaments for a single sport). Rather than focusing on making the workaround possible, this user (and most likely, many others) will benefit a lot more if a dedicated feature is developed to host multi-sports tournaments.

This missing of information is one of the reasons that makes the current workflow time-consuming for the product manager. Responding to the feature requests, even if they are clearly stated, also takes time. We roughly estimate the time it takes to process a feature request at ten minutes. This includes responding to the

requester and adding the request to the backlog if not listed already. As the business grows, more requests will come in. Another downside of the current workflow is that it does not allow for proper requirements prioritization and does only involve a small subset of the users.

4.3. Architecture recovery

As we learned from the literature review, reverse engineering consists of (1) extracting information from system artefacts, (2) abstracting the information to a higher (design) level, and (3) presenting the information to stakeholders in a friendly way [89]. The goal is to create a Feature Diagram, Functional Architecture Diagram and Context Diagram for Tournify. We will explain our approach in further detail in this section.

4.3.1. Extract features from the graphical user interface (GUI)

We used the open source workspace of Eclipse¹ for drawing the Feature Diagram, using the FeatureIDE² framework. Eclipse is mostly known for its Java integrated development environment and FeatureIDE has extensive possibilities for feature-oriented development. In our case, we only used the Feature Modeling composer which allows us to draw a hierarchical Feature Diagram in a tree-structure. FeatureIDE also has options to mark features as optional or mandatory, next to the possible 'alternative' and 'or' relationships between features and its sub features.

Or: one or more of the subfeatures must be selected

Alternative: exactly one of the subfeatures must be selected

Features can be collapsed or expanded by clicking on them so you can easily change the level of detail and view the features in a in a simple visual manner. Cross-tree constraints are also allowed. These constraints are placed underneath

¹ www.eclipse.org

² <https://featureide.github.io>

the diagram. The most common constraint is ‘implies’, indicating that ‘feature A implies the selection of feature B’, for example.

There are multiple ways to extract features from the system artifacts. We used the GUI as the artifact and extracted the features manually, after we concluded that automatic GUI testing tools would take more time to set up and yield less reliable results, especially when working with a web app like Tournify. We opened every page on the application and clicked on every button, link or entry field. The site page hierarchy was used to group features, in the same way you construct a sitemap. An example of this process for a site section is shown in Figure 4.3.

On this participants page, the three tabs on top (teams, referees, administrators) provide the main navigation. They become the first *compound* features we model in the Feature Diagram. Compound features represent a group of composed features which become available for the user when the feature is selected.

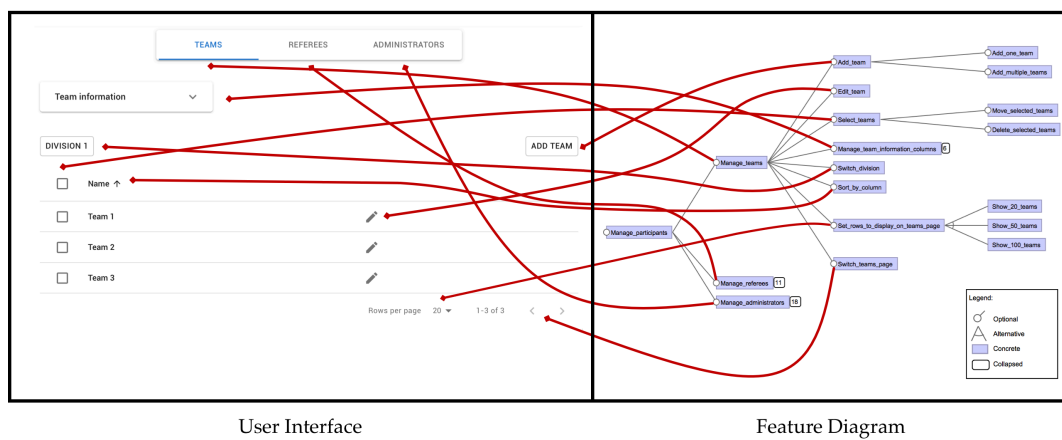


Figure 4.3: An example of mapping UI elements from the software to features in a Feature Diagram

The naming of a feature is based on the label of the UI element, if available. We rephrased some of the feature names to make them start with a verb, as is customary in a Feature Diagram (e.g. *Manage_teams* instead of *Teams*). We used an upper-case letter for starting the feature name, and an underscore to separate words, but other variants like spaces (e.g. *Manage teams*) or medial capitals (e.g. *ManageTeams*) are also acceptable. It can happen that the same feature can be

accessed from different pages in the UI. Since a merge of different branches is not possible in this type of Feature Diagram and each feature name should be unique, we added a number to a feature in the case of a consecutive occurrence of the same feature (*Set_bracket_size2*).

In the example of Figure 4.3, we continue by focusing on the teams page. The ‘rows per page’ functionality shows a dropdown with three different options. These are modelled with an alternative relationship: either one of the child features must be selected. In this case, the different options are *atomic* features: they do not have any child elements.

In some cases, it may be useful to deviate from the page hierarchy and group features according to their function instead of their presence on specific pages. For example, we modelled a subfeature *Choose_language* as a child of *Manage_account* because the user language is linked to an account, although choosing a language is done on the tournament overview page instead of on the account page.

4.3.2. Identify (sub)modules by abstracting

A Feature Diagram can easily consist of hundreds of features, making interpretation of the diagram a difficult task. Therefore, we need to abstract the information to a higher level. We do that by identifying (sub)modules in the Feature Diagram, which “correspond to the software product parts that implement the respective functions” [17, p. 204]. We take the advice from Brinkkemper & Pachidi [17], who suggest that a functional architecture is usually modelled in two or three layers. The features are supportive to the sub modules on the lowest level. The highest level is constructed first, and consists of the decomposition of all features into components, in such a way that each module embodies a manageable and well-defined functionality which can be developed relatively independent of other modules [92]. Although there are no strict rules for this mapping process, it is our guidance to use a module-size that is convenient in development and for visualization purposes (Figure 4.4), while taking into account the interaction with other (sub)modules. Arrows in a FAM are used to model this interaction in the

form of information flows. Substantivized nouns are used for the naming of the modules (e.g. Presenting instead of Present) and the scope is described in the lower-right corner of the rectangle, being either the application name (first layer) or (sub)module name (consecutive layers) that is shown.

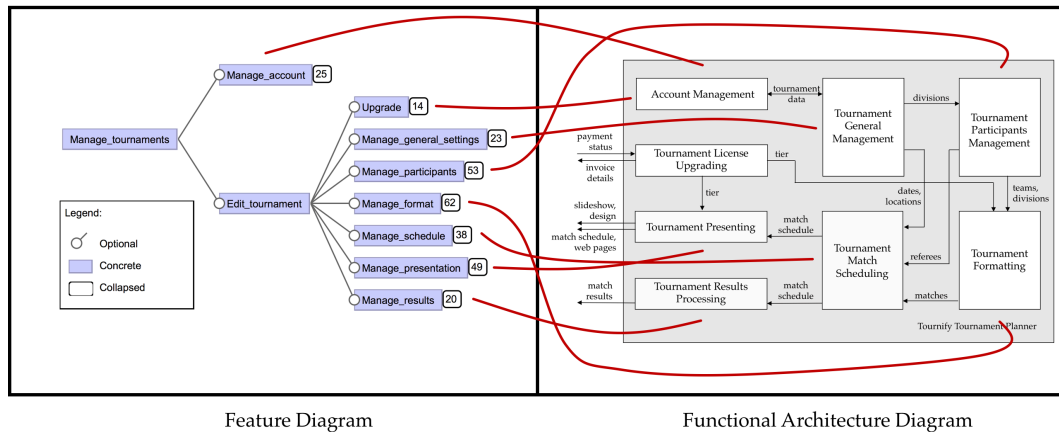


Figure 4.4: An example of mapping feature groups from the Feature Diagram to modules in the Functional Architecture Diagram

4.3.3. Present the model

The FAM provides a proven way to communicate the architecture in a developer-friendly way. Any diagramming tool can be used to draw the diagrams. Popular examples are Microsoft Visio or the online tool Draw.io. We decided to use Microsoft PowerPoint to create the FAM in an interactive presentation where one can navigate between the application overview, modules and submodules.

4.4. Crowdsourced requirements engineering platform

Based on our literature review and the case we presented at Tournify, we developed an online platform that allows users of a software application to create new requirements for it, comment on, and vote on requirements from others. For this platform, we created a set of 13 User Stories and prioritized them using the MoSCoW method (See Appendix 3). The design based upon the requirements can

be found in this thesis' public data set and the final implementation is live at www.tournify.nl/manage/features (Tournify account required). In this section we cover how the platform allows for the elicitation, negotiation and prioritization of requirements by crowd workers.

4.4.1. Elicitation

The literature review demonstrated the broad adoption of User Stories in agile software development. However, User Stories only served as an optional development artifact that could be added to a free form textual description of a requirement, when considered in Crowdsourced RE. It was never used as the basis for feature description, although these informal natural language descriptions are easy to read and adhere to a simple format. As a result, it may yield more useful, higher-quality, and more detailed requirements.

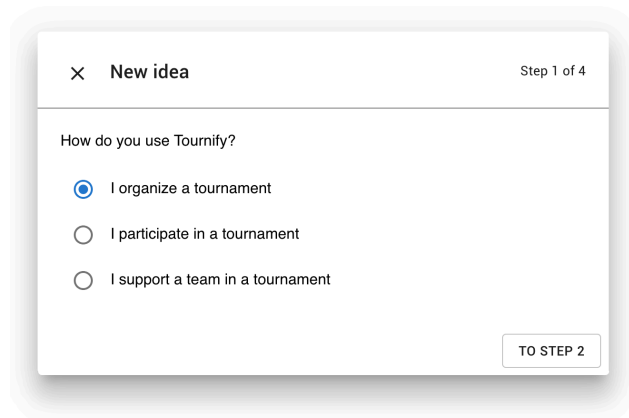
To test this hypothesis, the crowdsourced requirements engineering platform we designed needs to enable users of a software application to submit feature requests in the strict format of User Stories - containing a role, goal and benefit. In order to help users to formulate these stories, even if they have never seen or heard of a User Story before, we used a form with the following four simple self-explanatory and small steps:

1. Role selection
2. Goal expression
3. Potential benefit expression
4. Verification and categorization

We will look at each step in more detail by means of an example.

³ <http://dx.doi.org/10.17632/7r9j67wxzb.1>

Step 1: Role



× New idea Step 1 of 4

How do you use Tournify?

I organize a tournament

I participate in a tournament

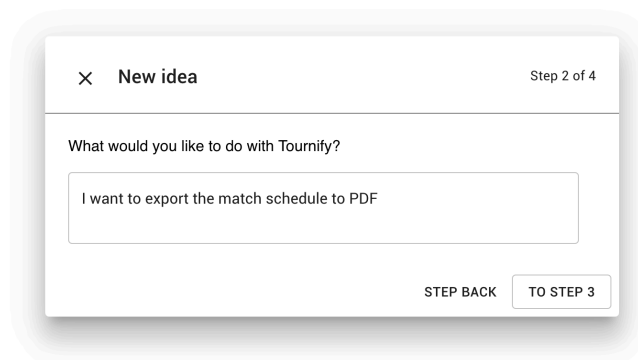
I support a team in a tournament

[TO STEP 2](#)

Figure 4.5: Formulating a requirement as a User Story: step 1

The first step is to find out in which role the requester uses the application. The user can select one of the roles from the predefined options using radio buttons. In the case of Tournify we defined three roles: organizer, participant and supporter.

Step 2: Goal



× New idea Step 2 of 4

What would you like to do with Tournify?

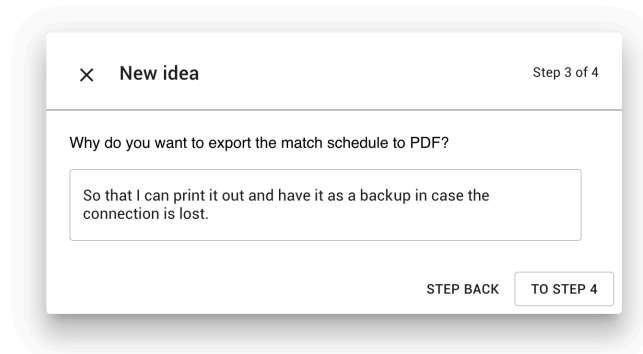
I want to export the match schedule to PDF

[STEP BACK](#) [TO STEP 3](#)

Figure 4.6: Formulating a requirement as a User Story: step 2

In the second step we ask the user what he or she wants to do with Tournify: a feature that is missing. The textbox contains static text before the user input, containing the phrase “I want to”. We wrote “export the match schedule to PDF” in our example.

Step 3: Benefit



× New idea Step 3 of 4

Why do you want to export the match schedule to PDF?

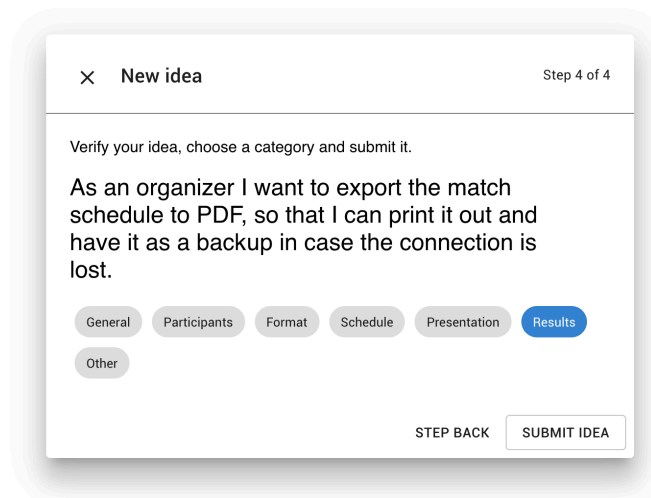
So that I can print it out and have it as a backup in case the connection is lost.

STEP BACK TO STEP 4

Figure 4.7: Formulating a requirement as a User Story: step 3

The users' text entry recurs in the formulation of the question in the third step. We ask explicitly why the user wants to have the requested feature, to know what the user sees as the potential benefit when the feature would be implemented in the software application. The answer always contains the predefined "So that" at the start.

Step 4: Verification and category selection



× New idea Step 4 of 4

Verify your idea, choose a category and submit it.

As an organizer I want to export the match schedule to PDF, so that I can print it out and have it as a backup in case the connection is lost.

General Participants Format Schedule Presentation Results Other

STEP BACK SUBMIT IDEA

Figure 4.8: Formulating a requirement as a User Story: step 4

Before submitting the idea, the user is able to verify the User Story that has been formulated based on the answers he or she provided in the first three steps. We also ask the user to select one of the predefined categories. These categories are part of the main menu of the application, so the users are already familiar with the terms. Labeling the requests with the corresponding category allows for easy categorization later on.

4.4.2. Prioritization

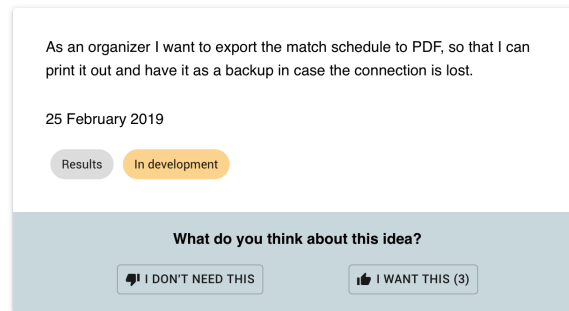


Figure 4.9: Feature request detail with the option to vote

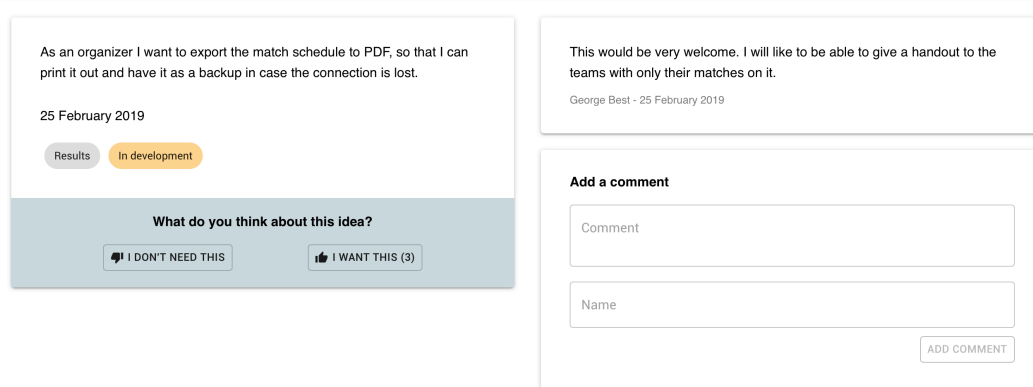
All requests are published on a feature request overview page in the Tournify web app, which can be accessed via the support menu. This is no static page, as requirements elicitation is not the only goal of the platform. The second goal is to prioritize requirements utilizing the crowd, by means of voting (Figure 4.9).

For requirements prioritization, several techniques exist. We follow the general advice of Maiden & Ncube [93], also advocated by Berander & Andrews [94], to use the simplest appropriate prioritization technique. This is especially true in crowd-centric requirement engineering [54], since end-user crowd workers are likely to be less experienced with requirement prioritization than product managers. The prioritization technique should also allow for easy reprioritization, as requirements will be added, changed or deleted continuously.

Numerical Assignment is the most common (and a very easy) prioritization technique, which is based on grouping requirements into different priority groups. Each group represents something that the stakeholders can relate to (e.g. critical, standard, optional) [94]. A downside of this is that stakeholders tend to “think that everything is critical and they will most likely consider 85 percent of the requirements as such” [94, p. 77]. Another simple feedback type is **Confirmation or Negation**, in which users agree or disagree on problems or opinions of other users [72]. This feedback type is used in the Requirements Bazaar [68] and REfine [55] platforms. After discussing these two feedback types in our research group,

we decided to implement the Confirmation or Negation method by allowing users to upvote or downvote requests (I want this, I don't need this).

4.4.3. Negotiation



The image shows a user interface for a feature request. On the left, a request is displayed: "As an organizer I want to export the match schedule to PDF, so that I can print it out and have it as a backup in case the connection is lost." It is dated "25 February 2019" and has two status buttons: "Results" (grey) and "In development" (orange). Below the request is a question "What do you think about this idea?" and two buttons: "I DON'T NEED THIS" (with a thumbs-down icon) and "I WANT THIS (3)" (with a thumbs-up icon). On the right, a comment from "George Best - 25 February 2019" reads: "This would be very welcome. I will like to be able to give a handout to the teams with only their matches on it." Below the comment is a section titled "Add a comment" with a text input field for the comment, a text input field for the name, and an "ADD COMMENT" button.

Figure 4.10: Feature request detail with the option to comment

The last goal is to facilitate negotiation of requirements. A commenting section (Figure 4.10) enables users or product managers to respond or add suggestions to the requests in order to come to a mutually satisfactory requirement.

4.5. Evaluation protocol

The quality and usefulness of the reconstructed architecture and the effect of using the crowdsourced requirements engineering platform will be evaluated. The results of this evaluation will be presented in the next chapter. In this section, we explain how we got to these results.

We used a mixed methods design, combining quantitative and qualitative research methods. The advantages of this research design are highlighted in the paper of Kaplan & Duchon [95, p. 582], *Combining Qualitative and Quantitative Methods in Information Systems Research: A Case Study*, who state that "mixing methods can lead to new insights and modes of analysis that are unlikely to occur if one method is used alone" in information systems research. This is in line with findings in other disciplines: "Using multiple methods increases the robustness of results because

findings can be strengthened through triangulation - the cross-validation achieved when different kinds and sources of data converge and are found congruent" [95, p. 575].

4.5.1. Architecture recovery

The recovered architecture will be made available in the online public data set of this thesis⁴. In this thesis we will present the context diagram and top-layer Functional Architecture Model, alongside some examples of feature diagrams for (sub)modules. We will report on:

- The number of modules and submodules
- The number of features
- The depth of the feature diagrams
- The feature degree of modules

In order to assess the quality of the recovered architecture, we will present the architecture to the developer of Tournify and interview him afterwards. In a semi-structured interview, we cover the developers' previous experiences with functional architectures and opinion on the quality and usefulness of the recovered architecture for his daily work. The interview protocol can be found in Appendix 4.

4.5.2. Crowdsourced requirements engineering platform

The crowdsourcing platform has been deployed and announced on February 25th, 2019. The announcement was published via an email to a selected group of 337 users (63% opened). These users had either requested a feature in the past, subscribed to the newsletter, or made a purchase recently. A reminder was sent one month later (55% opened). The total data collection period was five weeks, so all requests submitted after March 31st, 2019 were not included in this research. Among all requesters, voters and commenters, one free tournament upgrade has

⁴ <http://dx.doi.org/10.17632/7r9j67wxzb.1>

been raffled. Users were also informed of the feature request platform via a *snack bar* message which was shown when opening the Tournify tournament planner.

The researcher initiated the first request and commented on some of the requests during the study. He was also able to label features as *in development* or *done*. This first request by the researcher will be included in the report on the number of requests, because users were able to comment and vote on this request. However, it is not further evaluated regarding the quality and complexity criteria we discuss below. Comments from the researcher will be excluded from the results.

Engagement

We will report on the engagement of users on the platform by providing the following results:

- the number of initiated feature requests and unique requesters
- the number of votes and comments
- the number of visitors on the feature request page
- the use of the different roles in the User Stories
- the categorization of User Stories

Perceived usefulness

After the data collection process, the users who submitted an idea received an extra email with a link to a short questionnaire. This questionnaire tests the perceived usefulness of the platform from an end user perspective (Appendix 5). It contains four questions with a five-point Likert scale, asking the users to evaluate the usefulness of different aspects of the platform. One closed question is included to verify if the requester had experience with formulating User Stories before, and one open text field can be used to comment on the experience with the platform.

Quality

In section 3.2.3, we discussed the Quality User Story framework [9]. This framework will be used to assess the User Stories individually based on their

syntactic, semantic and pragmatic quality. The eight criteria and their descriptions are shown in Table 4.

Criteria	Description
<i>Syntactic</i>	
Well-formed	A user story includes at least a role and a means
Atomic	A user story expresses a requirement for exactly one feature
Minimal	A user story contains nothing more than role, means, and ends
<i>Semantic</i>	
Conceptually sound	The means expresses a feature and the ends expresses a rationale
Problem-oriented	A user story only specifies the problem, not the solution to it
Unambiguous	A user story avoids terms or abstractions that lead to multiple interpretations
<i>Pragmatic</i>	
Full sentence	A user story is a well-formed full sentence
Estimatable	A story does not denote a coarse-grained requirement that is difficult to plan and prioritize

Table 4: The eight criteria to be used to assess User Stories individually according to the Quality User Story framework [9]

Each User Story will be evaluated on its quality manually by three experts individually. The experts use the description of the criteria as shown in Table 4, as well as the additional information from the accompanying article, to analyze the User Stories. The lead researcher will analyze all User Stories. The User Stories will also be distributed among six members of the Requirements Engineering Lab at Utrecht University. They will analyze one third of the User Stories each. If there is no consensus in the judgement of the experts, majority voting is leading.

Complexity

We will make an estimation of the amount of work it would take to implement each User Story individually, based on the assessment of the lead developer of Tournify. For the scaling the Fibonacci sequence (1, 2, 3, 5, 8, 13, 21) will be used. We assign a value of '0' when it concerns a feature that has already been implemented but overlooked by the requester. The other numbers represent development hours. Since it is difficult to estimate large work items with a high degree of confidence, the upper limit for our estimation is 21 hours. In practice, User Stories who take more than 21 hours to implement can be broken down into more granular pieces. Since there is only one developer involved in the assessment of the complexity, we have decided to do the estimation in development hours instead of in Story Points. The developer is used to make these hour estimations,

as this is common practice in the organization while making price quotations for organizations requesting customization. Story Points are better suited for planning poker with multiple developers but are a lot more abstract than the concrete man hours.

We also give each User Story a complexity score based on the impact it has on the architecture. In a Feature Diagram, the leaves represent atomic features and intermediate nodes represent compound features [96]. If a new feature would be added as a leaf, we rate its architectural impact as '1'. If it would be added as an intermediate node (additional sub-features have to be developed to deploy the feature), we rate it as '2'. If the User Story requires a new submodule or has an impact on multiple modules, we rate the architectural impact as '3'. Value '4' will be assigned when it concerns an entire new module.

5. Results & Analysis

The recovered architecture and quality evaluation will be presented in this chapter. We also present the results of the feature request platform we integrated in the software application and analyze the crowdsourced User Stories in terms of quality and complexity.

5.1. Architecture recovery

In this section, we give a high-level overview of the system context and Functional Architecture Model which we have reconstructed for Tournify. We also present an example of a Feature Diagram of a particular submodule and present statistics regarding the number of modules, submodules, and features. The entire reconstructed architecture in an interactive Powerpoint presentation can be found in this thesis' public data set. In the second part of this section we present the evaluation results, covering the quality and perceived usefulness of the model based on an interview with the developer.

The interviewed developer is Jesse, responsible for all functionality of the Tournify application. He has five years of experience in programming, starting as a Python developer for Mobile Professionals – an Amsterdam-based media agency. Currently, most of his work is done in JavaScript and he has experience with the ReactJS and NodeJS libraries. He completed the Information Science bachelor at the University of Amsterdam in 2015, with a minor in programming.

5.1.1. System context

Three external entities interact with the Tournify application, as shown in the Context Diagram (Figure 5.1). The Mollie Payment Service Provider handles the online payments of upgrades via an API. The Jortt Bookeeping API is used to send

⁵ <http://dx.doi.org/10.17632/7r9j67wxzb.1>

an invoice to the customer. The Tournify Live website is the third external entity the Tournify application interacts with. This is a separate repository that does share the database with the tournament planner but is out of scope of the application we study in this research.

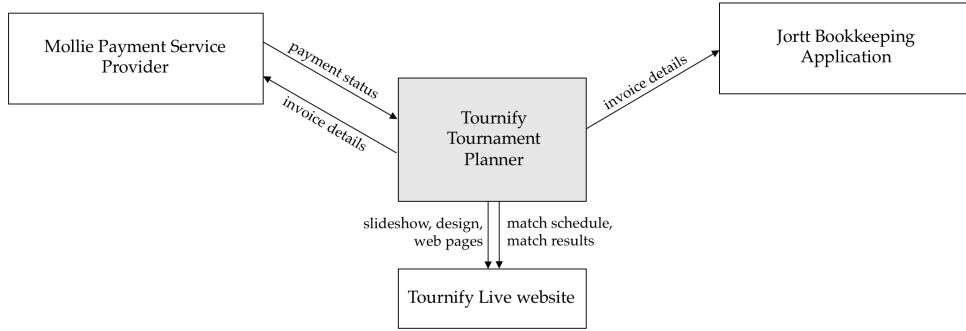


Figure 5.1: Context Diagram of Tournify

5.1.2. Functional Architecture

For the tournament planner, eight modules are identified based on the GUI. These modules and the information that flows between them, is shown in the FAD of Figure 5.2.

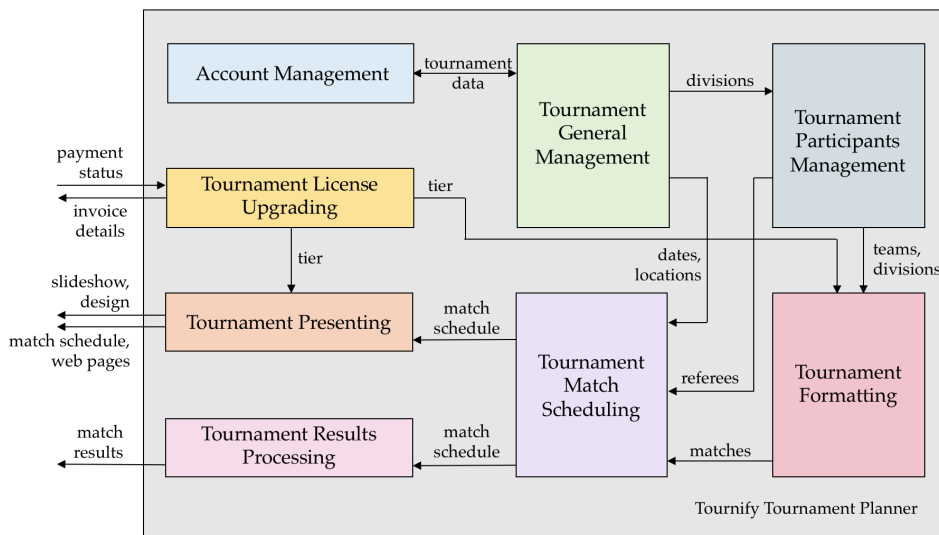


Figure 5.2: Functional Architecture Diagram of Tournify, application level

For six out of the eight modules, the module is supported by submodules, like in the example of the Tournament Participants Management module (Figure 5.3). In total, 21 submodules are used.

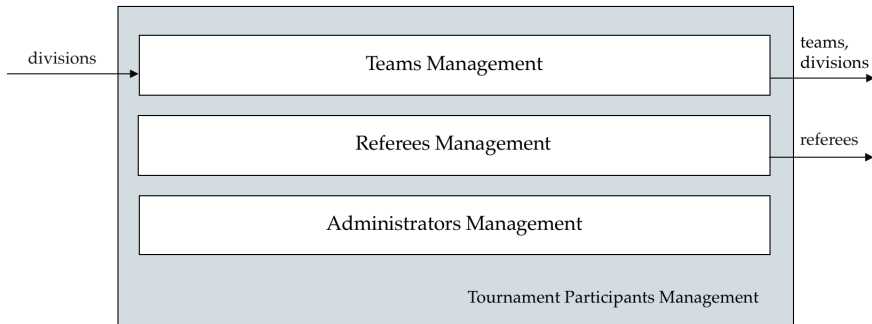


Figure 5.3: Functional Architecture Diagram of the Tournament Participants Management module

On the lowest level, each (sub)module is supported by features. Those features are represented in a Feature Diagram, like the one in Figure 5.4 of the Administrators Management subfeature. Some interesting implementation rules can be extracted from this diagram, like:

- In order to add an administrator, you have to enter an email address and assign at least one right.
- In order to delete an administrator, you first have to select one.
- You have to choose between showing either 20, 50 or 100 administrators on one page.

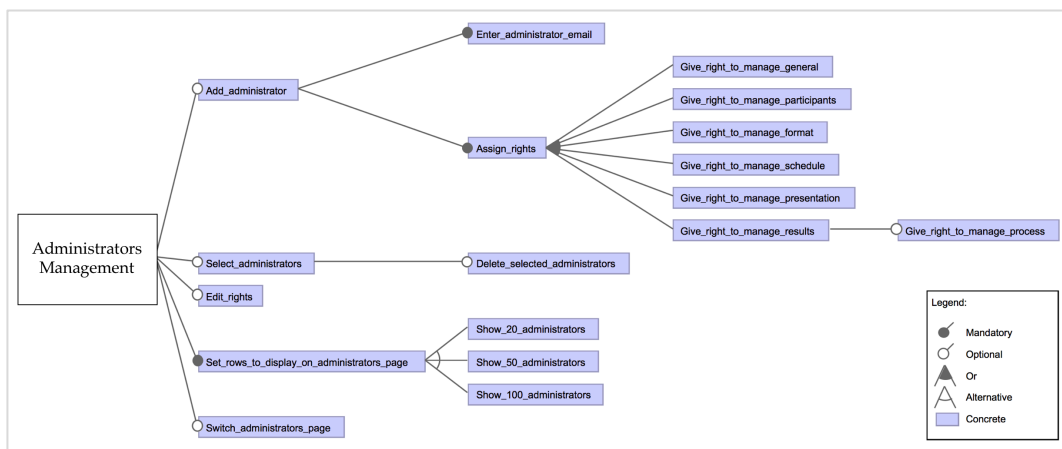


Figure 5.4: Feature Diagram of the Administrators Management submodule

In total, 198 atomic features are captured in the recovered functional architecture. That means that each module contains 25 features on average. A complete list of all modules, submodules and the number of supporting features is shown in Table 5. The table also includes the degree and depth of each (sub)module. The **degree** is the number of features that is present on the first layer of the Feature Diagram. For example, in the Feature Diagram of Figure 5.4, there are initially five features to choose from: *Add_administrator*, *Select_administrators*, *Edit_rights*, *Set_rows_to_display_on_administration_page*, and *Switch_administration_page*. The **depth** is the maximum number of layers in the diagram. In the given example, the feature *Give_right_to_manage_process* sets the depth of this diagram at four. On average, the Feature Diagrams have a degree of 4.2 and a depth of 2.2.

Modules	Submodules	Features	Degree	Depth
Account Management	Language Selecting	2	2	1
	Support Requesting	8	8	1
	Account Creating	3	3	1
	Tournament Creating	4	4	1
Tournament License Upgrading	-	11	3	2
Tournament General Management	-	17	7	3
Tournament Participants Management	Teams Management	16	8	3
	Referees Management	8	5	2
	Administrators Management	13	5	4
Tournament Formatting	Phases Management	3	3	1
	Divisioning	1	1	1
	Automatic Formatting	16	3	3
	Groups Management	8	3	3

	Brackets Management	10	3	3
	Single Matches Management	5	3	3
Tournament Match Scheduling	Planning	7	5	2
	Playing Field Management	6	4	2
	Breaks Management	6	3	2
	Matches Management	5	2	2
	Viewing	2	2	1
Tournament Presenting	Public Website Management	14	6	3
	Slideshow Management	12	4	4
	Presentation Design	7	7	1
Tournament Results Processing	-	14	7	4

Table 5: Statistics of the Tournify Functional Architecture

5.1.3. Quality of the reconstructed architecture

Jesse, the main developer of Tournify, did not receive education in software architecture during his study. Also, in his working experience, formalized functional architectures never came across: “When we started with Tournify we thought about how the application should look like and what the arrangement of the pages should be, but we did not call it an architecture or used any formalized style or technique.” He explains how the Tournify application has a sidebar that is used for the navigation. The pages on the sidebar are also the main components in the code base. Other components mainly belong to one of those pages. According to the React website⁶, this is a common approach to structure React projects. In this approach, the file structure is based on the grouping of features. When showed the reconstructed architecture, this might be one of the reasons he commented: “The architecture and modeling style are very clear. The architecture matches the code base very closely. In my feeling, it works well with React, because the application

⁶ <https://reactjs.org/docs/faq-structure.html>

is divided into components, in a similar way the architecture consists of different modules.”

Jesse explains where there are differences between the reconstructed architecture and code base: “The naming that has been given to the features and modules in the model is different than the names I used in the code.” When asked if he recognizes the features and (sub)modules in the model, it is clear that the names that are used are comprehensible and the developer knows which features and components are meant. He suggests working together with the product owner to improve both the architecture model and code base at the same time: “We can improve the code by means of this model and we can improve the model on the basis of the code. The first approach may even work best. The more the two correspond, the better.”

Not only in the naming of the features and (sub)modules, there are slight differences between the code and the functional architecture. Sometimes a component is reused on another page to avoid duplicate code, Jesse explains. This is true for the Tournament Participants Management module we have shown in Figure 5.3. Currently, the features on the teams, referees and administrators tab overlap, making Jesse decide to fit them into one component. This is definitely not the case for all submodules we included in the functional architecture. The Tournament Presenting module also contains three sub-modules, but since every tab in the UI shows very different functionality, there is a component for every submodule in the code base. Jesse argues that it may be beneficial to use those small-sized components also for the Tournament Participants Management module: “It is good to model it the way you did. At this time, the pages have very similar functionality. However, we already know that in the feature the referees page will have many different features that will not be available on the teams page. The administrators page already has different functionality, like assigning rights to administrators, so I would suggest keeping the distinction between the different submodules in the software architecture. The more the better, especially for visualization purposes.”

We also discussed the Feature Diagrams in more detail. Jesse finds the distinction between mandatory and optional features clear and convenient. The same applies to the use of ALTERNATIVE and OR relationships between subfeatures and compound features, when we explained them to him. Regarding the depth and degree, Jesse prefers an interface with a higher depth and lower degree: “In my opinion: the more layers, the simpler the interface is to use. Let’s take the export feature as an example. Currently, the user clicks ‘Export’ on the results page, then selects what information to export (match schedule or scoring sheets), the size, and sorting. If you use the right naming, users will know where to click on when they want to perform a certain action. If we show all those export features in the first step, the user gets confused.”

5.1.4. Perceived usefulness of the reconstructed architecture

Jesse’s familiarity with the code base is helpful when developing new features: “I instantly get an idea about how new features should be built, and which components are affected or should be created.” Depending on what features needs to be developed, he would create additional modules, submodules or features in the architecture, deciding where it would fit taking the depth and degree into account. “I think this a very handy way to develop features. For both the developer and the product manager. The model will also help the product manager to understand the code.” The architecture may help to show what needs to be developed for a new feature, Jesse explains. If a new requirement comes in, it is instantly clear what needs to be developed. The clearer this is, the better he can make an estimation of the workload of a new requirement. “Visualization the components that are affected is clearer to work with than a list of bullet points in Trello” he says, referring to the visual collaboration platform that is currently used internally at Tournify to manage requirements. “While building a feature, questions always come up I want to discuss with the product owner. If we would discuss a feature beforehand and note it in the architecture model, I think it will help to give me a better understanding of what we want to implement.” To illustrate this, consider the following User Story:

As an organizer, I want to limit the availability of referees to specific divisions, match days or time slots so that I can satisfy the referees' wishes.

The addition is shown in Figure 5.5, in which the features with a light blue color are added to meet the User Story we just presented. As shown, the original "Edit_name_of_referee" feature will become part of the compound feature "Edit_referee".

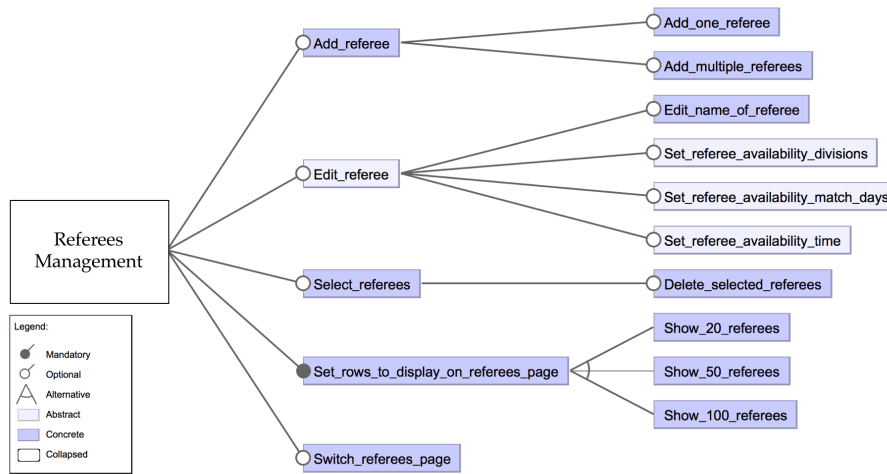


Figure 5.5: Adding new functionality to the Referees Management sub-module

The size of a Feature Diagram may be a hint regarding the workload of a feature. Jesse thinks that bigger feature diagrams usually concern more work. But he is hesitant in setting generic rules on deciding the workload based on the architecture alone: "A feature that touches upon multiple (sub)modules can be as easy or difficult to develop as an addition to one specific (sub)module. If the adjustments to the different modules are small, it can still be an easy feature to implement."

During the interview, Jesse also mentions the potential usefulness of having this architecture for new developers who join the company: "The architecture is clear and very nice for people who are not familiar with the code base. The visualization helps to understand and learn the code easier. Although experience with React is also very beneficial in that sense."

5.2. Crowdsourced requirements engineering platform

During the five weeks period, we had 157 unique visitors on the feature request platform. From those visitors, 39 users interacted with the platform by submitting an idea (23), voting on an idea (28), and/or commenting on an idea (9). Together, they submitted 57 ideas, voted 89 times and commented 14 times (Table 6). The users that interacted with the platform organize tournaments in 14 different sports and 1 e-sports. The functionality to downvote an idea ('I don't need this') was not used and in five times a requestor voted on its own idea, which was not prevented by the platform. The complete list of crowdsourced User Stories can be found in Appendix 6.

Value	#	Value	#
Page views	247	Unique page views	157
Interactions	160	Unique interactors	39
Requests	57	Unique requesters	23
Votes	89	Unique voters	28
Comments	14	Unique commenters	9

Table 6: Use of the crowdsourced requirements engineering platform

More than half of the requesters (15, 65%) submitted only one idea, two users submitted respectively two and three ideas and four users submitted five or more ideas (respectively 5, 6, 7, and 14 ideas). All requesters indicated they used Tournify as an tournament organizer; the other predefined roles (participator, supporter) are not selected.

All ideas are written in Dutch and constructed based on the template of a User Story. A screenshot of part of the Feature Requests overview page is shown in Figure 5.6. Next to the feature description, each element also contains the submission date and selected category. If applicable, the element also contains the number of votes, number of comments, and development status.

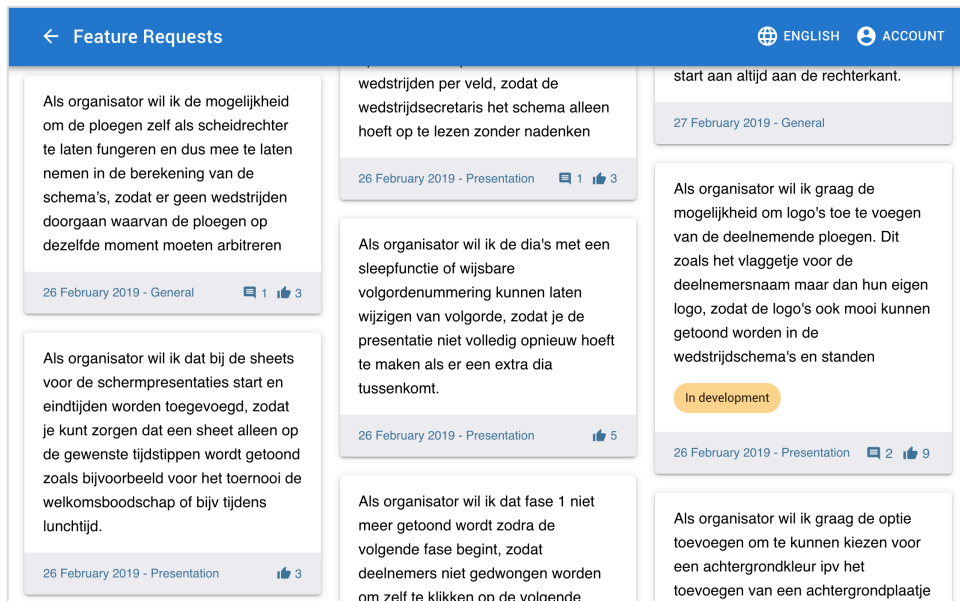


Figure 5.6: Screenshot of the feature request overview page

Two feature requests got nine upvotes, which is the most times a feature has been upvoted. Translated to English, these ideas are:

US₁ *“As an organizer I want to have the possibility to add logos of the participating teams. Like the flags in front of the participants name but instead with their own logo, so the logos could be displayed nicely in the match schedules and standings.”*

US₂ *“As an organizer I want to set the match duration per division instead of per day, so you can make the match duration longer for divisions with fewer teams, then the divisions with many teams.”*

The categorization of User Stories turned out to be a difficult task for the crowd workers, judging by the numbers. In more than half of the cases (52 percent), the category selection of the requester does not match the category assignment done by the main researcher of this study. For example, US₁ was categorized as Presentation. Although this categorization makes sense, adding logos would be implemented on the Participants page. Many User Stories were falsely classified as General by the requesters. This category was not intended to be an umbrella term for general User Stories (the category Other was meant for that purpose) but

was referring to the General Settings page in the application. US₂ was correctly assigned to the Schema category: the page that requires the most attention, together with the Presentation page, if we listen to the users (Figure 5.7).

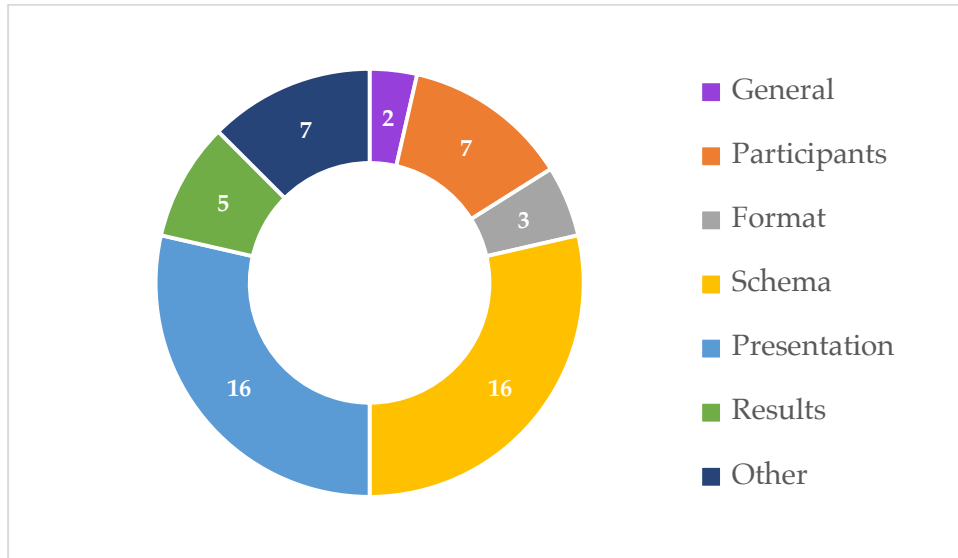


Figure 5.7: Categorization of the crowdsourced User Stories based on the evaluation of the main researcher

5.2.1. Perceived usefulness of the crowdsourcing platform

After the study period, thirteen user who interacted with the crowdsourcing platform responded to the questionnaire that was sent to them via email. Most of them (10) requested a feature themselves, the other three respondents only voted for a feature. They perceived the platform as very useful, regarding all four possible interactions when rated on a five-point Likert scale: requesting (M = 4.9; SD = 0.28), viewing (M = 4.8; SD = 0.38), voting (M = 4.5; SD = 0.88), and commenting (M = 4.5; SD = 0.66). All results are shown in Figure 5.8.

One user who requested a feature, voted for and commented on an idea and had previous experience in writing User Stories commented:

“You implemented the agile methodology in a very fun way! In such a manner the users get better involved and at least have the feeling their opinion matters”

Others found it “a fantastic way to improve the application”, “very useful to allow users to submit ideas” and see it as a way to “improve the software for your own tournament”. Another user noted how “every user gets new ideas while using Tournify on their tournament” and how this is “the best feedback to improve the application”.

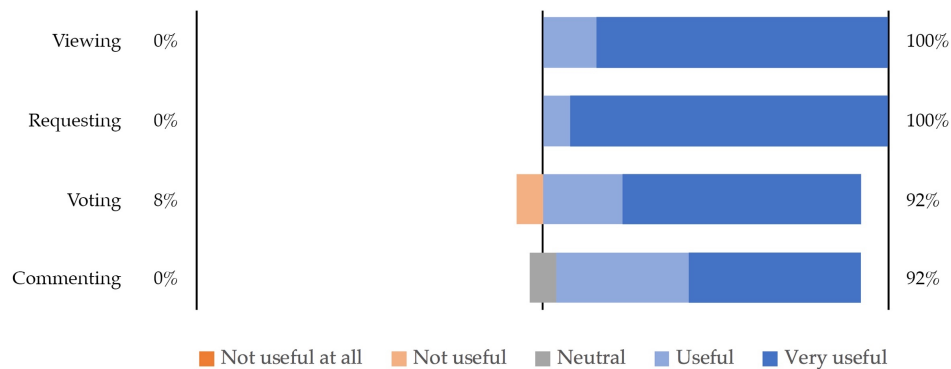


Figure 5.8: Perceived usefulness of the requirements engineering crowdsourcing platform

Out of the people who requested a feature, 70 percent has never written a User Story before. When asked if they find it helpful to formulate the ideas as User Stories, compared to free texts, the average score was 3.5 (SD = 0.85). There is hardly any preference to write the feature requests in free text (M = 3.2; SD = 1.14).

5.2.2. Quality of the crowdsourced User Stories

Each User Story ($n = 56$) has been tested against eight criteria from the Quality User Story framework [9] by three different experts independently (1344 decisions in total), as described in the evaluation protocol (section 4.5.2). Tests for inter-rater reliability show that the average pairwise percent agreement between the three judgements varies from 65.5% to 91.7% for each criterion. The results of the analysis are shown in Table 7. The entire evaluation can be found at this thesis public data set.

⁷ <http://dx.doi.org/10.17632/7r9j67wxzb.1>

Criterion	Number of User Stories with defect	Percentage of User Stories with defect	Average pairwise percent agreement
Well-formed	3	5.4	90.5
Atomic	5	8.9	84.5
Minimal	24	42.9	79.8
Conceptual	5	8.9	79.8
Problem-oriented	8	14.3	73.8
Unambiguous	9	16.0	65.5
Full sentence	19	33.9	81.0
Estimatable	3	5.4	91.7

Table 7: Quality of the crowdsourced User Stories

In total, 52 percent of the User Stories meet all requirements, meaning that 48 percent of the User Stories contains one or more easily preventable error(s) (Figure 5.9).

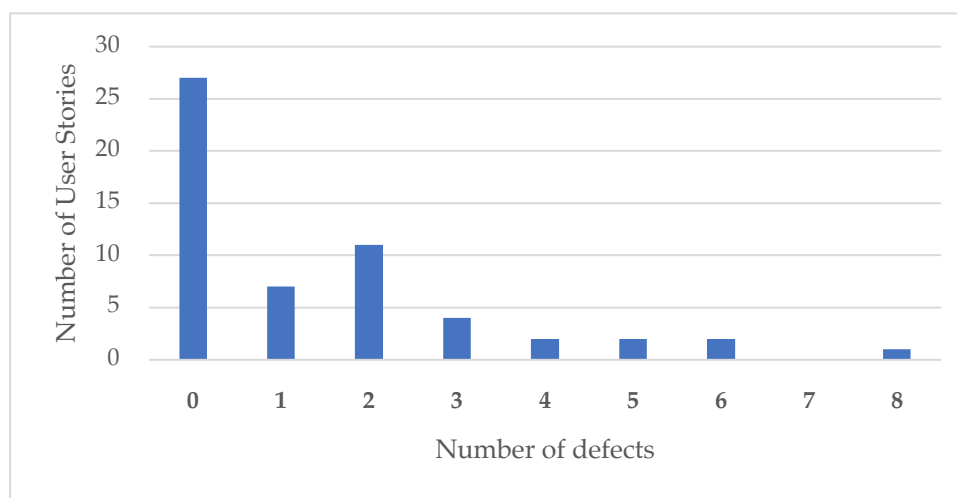


Figure 5.9: Number of defects per User Story

There is a strong association between the minimal and full sentence criteria, which is statistically significant ($X^2 = 31.6, p < .001$). Since both variables are measured at a nominal level and consist of two categorical independent groups, we performed a Pearson Chi-Square test to determine whether this association exists. The finding might provide an explanation for the higher number of User Stories

with two defects, compared to those with only one defect. Table 8 shows a sample of crowdsourced User Stories that violate one or more of the criteria.

ID	User Story	Violated criteria
US3	As an organizer I want to have the insurance that an email address that comes with a registration is valid, so the confirmation is delivered at all times. You could use a third-party check, like mailgun.com / email-verification-service, so we have more insurance about email. Viewing whether a mail has been opened is also nice. You probably use an email distribution API and you can show these metrics (sent / opened) in the GUI to the user.	<p><i>Atomic</i> address check and opening rate</p> <p><i>Minimal</i> additional information</p> <p><i>Conceptual</i> means expresses rationale, not a feature</p> <p><i>Problem-oriented</i> hints at the solution</p> <p><i>Full sentence</i> multiple sentences</p>
US4	As an organizer I want to Since recently you are able to register via the website! Very good improvement! I would also like to have the possibility to show pictures of the last tournaments' edition, for example, when people enter the website! It currently is so blank, so that more beautiful presentation!	<p><i>Well-formed</i> templated is disregarded</p> <p><i>Minimal</i> additional information</p> <p><i>Full sentence</i> multiple sentences</p> <p><i>Estimatable</i> to vague</p>
US5	As an organizer I want to change the order of slides with a drag and drop functionality or changeable ranks, so you don't have to recreate the presentation when an extra slide comes between.	<p><i>Problem-oriented</i> hints at the solution</p>
US6	As an organizer I want bigger scoring sheets in such a way they are properly distributed among the paper, so it looks better (regarding the size) and have the same dimensions after cutting	<p><i>Minimal</i> information between brackets</p> <p><i>Unambiguous</i> contains abstract terms: bigger, better</p>

Table 8: Sample of crowdsourced User Stories that violate the criteria from the Quality User Story framework

5.2.3. Complexity of the crowdsourced User Stories

The crowdsourced User Stories are evaluated based on their complexity by the developer of Tournify. Most of the crowdsourced User Stories can be developed within one workday, according to his estimation. One User Story could not be

estimated, because it was formulated to vaguely. Seven User Stories were already implemented but overlooked by the user. They are therefore not included in the estimation shown in Figure 5.10, which includes 48 User Stories.

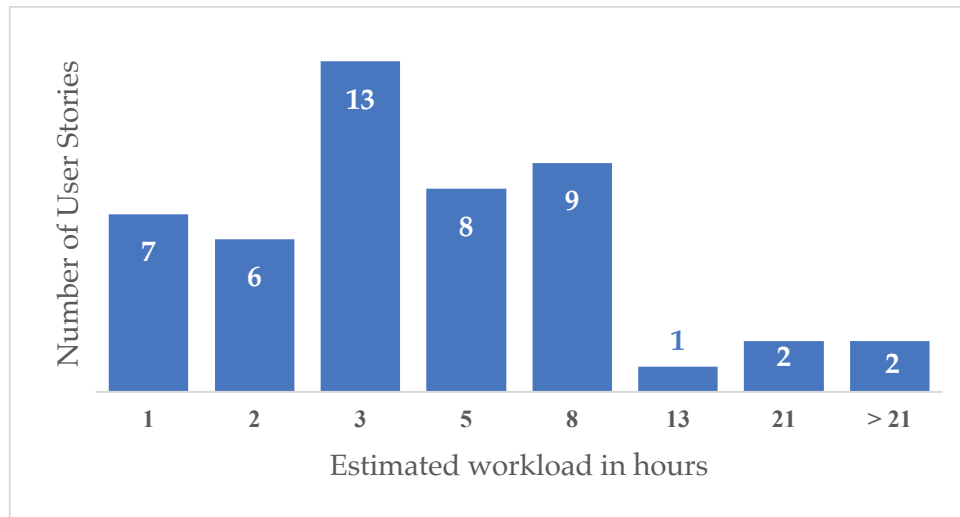


Figure 5.10: Complexity of the crowdsourced User Stories

The hour estimation has been mapped against the estimation we did based on the impact a feature would have on the functional architecture. We were able to do this for 43 User Stories. Five User Stories concerned features regarding the Tournify Live page, which is out of scope of the recovered architecture. User Stories with an architectural impact of 1 (would be added as an atomic feature) and 2 (would be added as a compound feature) show a similar level of workload. However, User Stories with an impact score of 3 (which require a new submodule to be added in the architecture diagram, or have an impact on multiple modules), are generally considered to be more complex to develop. A Spearman's rank order correlation was run to determine the relationship between the two ordinal variables. There is a moderate positive correlation between architectural impact and developers' hour estimation, which is statistically significant ($r_s = 0.56$, $p = < .001$).

5.3. Summary of the main results

- **Recovery:** the functional architecture of Tournify has been recovered based on the GUI. It contains 8 modules, 21 submodules and 198 atomic features. Each module contains 25 features on average which are shown in Feature Diagrams with an average degree of 4.2 and a depth of 2.2.
- **Documentation clarity:** the functional architecture model is clear to the developer of Tournify. The architecture matches the code base very closely, although the naming of features and modules does vary. In the process of refactoring, the code can be improved by means of the created model and the model can be improved on the basis of the code.
- **Learnability:** the visualization helps to understand and learn the code easier for people unfamiliar with the code base. The model is also helpful in the development of new features for both the developer and product manager.
- **User Story formulation:** a feature request platform has been integrated to involve users in requirements engineering. The platform allows users to submit ideas in the form of User Stories using a four-step data collection form.
- **Engagement:** the 39 users that interacted with the platform during a five-weeks period submitted 57 ideas, voted 89 times and commented 14 times.
- **Perceived usefulness:** users interacting with the platform perceived it as very useful, regarding all four interaction possibilities: requesting features, viewing ideas from other users, voting on ideas, and commenting on them. Interestingly, 70 percent of the requesters has never written a User Story before. There is hardly any preference to write the feature requests in free text.
- **Quality:** when tested on quality based on the Quality User Story framework, 48% of the crowdsourced User Stories contains one or more easily preventable error(s). Most frequent occurring defects are User Stories violating the minimal criterion (42.9%) or not being written as one full sentence (33.9%).
- **Complexity:** most of the crowdsourced User Stories can be developed within one workday. There is a moderate positive correlation ($r_s = 0.56$, $p = < .001$) between this estimated complexity as done by the Tournify developer, and the assessment from the researcher based on the impact a feature has on the reconstructed functional architecture diagram.

6. Discussion

The importance of connecting Requirements Engineering and Software Architecture in software development is long known. The rise of agile development drove the wide adoption of User Stories by requirements engineers and the introduction of microservice architecture created independently deployable modules running in different processes. While User Stories are small, implementable units of work or simply descriptions of features, the higher-level Epic Stories describe roadmap themes that can be used as an input for User Story formulation, focusing explicitly on the motivation and expected outcome of such a group of User Stories. The Requirements Engineering for Software Architecture (RE4SA) model has been proposed as a guidance to improve communication between requirements engineers and software architects, grounded in the idea that they contribute reciprocally to achieve their goals and exchange artifacts in the process [25].

In this thesis, we aimed to find out how the RE4SA model could be applied in an existing software product, using architecture recovery to reconstruct a functional architecture and crowdsourcing to gather, negotiate and prioritize new requirements. A set of sub questions, introduced in section 2.1, has been drawn up to cover the different aspects of this interdisciplinary study. We combined an extensive literature review with a case study at Tournify. This Amsterdam-based software development company provides an online tournament manager, used by sports and e-sports tournament and competition organizers. The web application provides functionality to manage participants, create match schedules based on any tournament format and process results as the tournament processes.

The results from the literature study as well as the case study are used to formulate an answer to the sub research questions, which we will discuss in the subsequent section of this chapter. Then, we will answer the main research question and draw the main conclusions from this study. In the last sections, we cover both internal and external validity threats and provide directions for further research.

6.1. Answering the sub research questions

The seven sub research questions formulated for this thesis will be covered in three different subsections. We start by providing a theoretical foundation for the link between requirements engineering and software architecture. We do not cover the sub research question in numerical order but focus on each side of the RE4SA model separately. We first discuss the principles in software architecture, show how we extracted features and modules from the graphical user interface and analyze the quality and usefulness of the recovered architecture. Lastly, we discuss the principles in requirements engineering theory, cover how we designed a crowdsourcing platform and evaluate its effect.

6.1.1. Linking requirements engineering and software architecture

The core principle of requirements engineering (RE) is to extract informal stakeholders' needs and translating them into formal specifications, ready to be used as an input for development of a software system [31]. A software architecture (SA) of such a software system contains "its fundamental concepts or properties in its environment, embodied in its elements, relationships and the principles of its design and evolution" [77, p. 2]. SA facilitates the communication among stakeholders, supports early design decisions and provides a transferable abstraction of a system [76]. A SA needs to be "stable, yet adaptable, in the presence of changing requirements" [1, p. 115]. In other terms: intertwining specification and implementation is required [24].

The first sub question concerns this relationship between RE and SA and can be answered based on the literature review. The Twin Peaks Model [1] demonstrates how more detailed specifications are produced progressively and dependency on the implementation increases in the mapping from requirements to architectural design. Lucassen et al. [25] showed how software product managers and software architects contribute reciprocally to achieve their goals and exchange artifacts (like "product requirements" and "architectural design decisions") in the process.

SRQ1 HOW ARE REQUIREMENTS ENGINEERING AND SOFTWARE ARCHITECTURE RELATED AND HOW IS THIS REFLECTED IN THE RE4SA MODEL?

The RE4SA model (Figure 6.1), developed at Utrecht University, aims to provide approaches or guidelines for requirements engineers and software architects to cooperate. The relationship between RE and SA is reflected in the RE4SA model by four artifacts they exchange. Artifacts the requirements engineers and software architects may be already familiar with, because they are frequently used in Agile development.

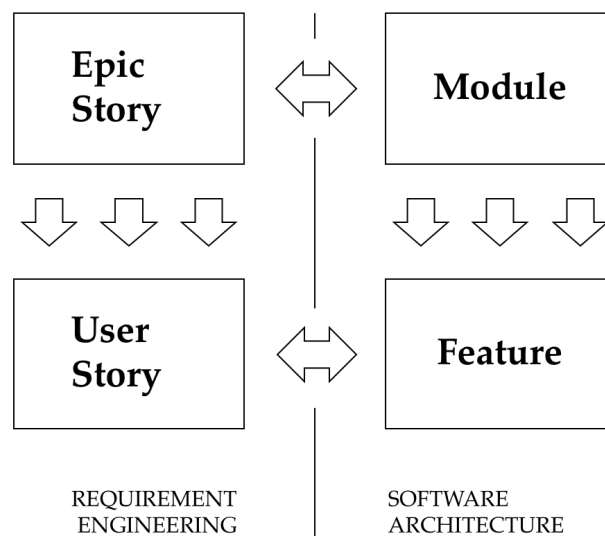


Figure 6.1: The RE4SA model [4]

User Stories are the most detailed representation of requirements and contain a persona (role), action and benefit, formulated in one full sentence. The requirements engineers communicate these User Stories to the software architects, who use them to position **Features** in a Feature Diagram. A group of features is called a **Module** and a Functional Architecture Diagram is used to visualize the modules of an application and the information that flows between them. Modules are on the same representation level as **Epic Stories**, which are used by the requirements engineers to group User Stories and frame every design problem, focusing on (1) the triggering event or situation, (2) the motivation and goal, and (3) the intended outcome.

6.1.2. Software architecture

Although SA is sometimes pictured as a typical non-agile process, projects that lack architectural focus will fall behind. Organizations should balance between adaption (agility) and anticipation (architecture) in which it is important to understand the context to define how much architecture is needed for a given project [81].

In order to answer SRQ3, we cover each type of architectural diagram mentioned in the question in more detail based on the literature review.

SRQ3 WHAT ARE THE PRINCIPLES IN SOFTWARE ARCHITECTURE THEORY REGARDING CONTEXT DIAGRAMS, FUNCTIONAL ARCHITECTURE DIAGRAMS AND FEATURE DIAGRAMS?

We start from a context viewpoint, which is used to describe the relationships, dependencies, and interactions between the system and its environment [15]. A technique to create a view from a context viewpoint is a **Context Diagram**. This diagram can serve as “a useful starting point for describing and defining the system’s mission and operational environment, showing the interaction of a system with all external entities that may be relevant to its operation” [16, p. 266]. The context viewpoint sets the scope for the functional viewpoint. A functional view “documents the system’s functional structure – including the key functional elements, their responsibilities, the interfaces they expose, and the interactions between them” [15, p. 41]. A **Functional Architecture Diagram (FAD)** is a view from this viewpoint, representing the primary functionality of a software product [17]. The functional architecture can be modeled in multiple layers, usually in two or three layers. On the lowest layer, each (sub)module is supported by features. These features, “prominent or distinctive user-visible aspects, qualities or characteristics” [85, p. 3], are represented in a **Feature Diagram**. In the three-structure of such a diagram we can mark features as optional or mandatory and define relationships between groups of features.

Creating a Feature Diagram was the first step in reconstructing the software architecture of an existing application (SRQ4). We used the graphical user interface (GUI) to extract features manually. We opened every page on the application and clicked on every button, link or entry field. The site page hierarchy was used to group features and the naming of a feature is based on the label of the UI element, if available.

SRQ4 HOW CAN FEATURE EXTRACTION SUPPORT THE RECONSTRUCTION OF THE SOFTWARE ARCHITECTURE OF AN EXISTING SOFTWARE PRODUCT?

Since the reconstructed Feature Diagram consisted of nearly 200 features, we needed to abstract the information to a higher level in order to facilitate interpretation. We did that by identifying (sub)modules in the Feature Diagram, which correspond to the software product parts that implement the respective functions. We then created an interactive presentation where one can navigate between the application overview, modules and submodules.

The recovered architecture contains eight modules. For six out of the eight modules, the module is supported by 3.5 submodules on average. In total, 21 submodules are used, and 198 atomic features are captured in the recovered functional architecture. That means that each module contains 25 features on average. The Feature Diagrams have an average degree of 4,2 and average depth of 2,2, indicating the number of features that is present on the first layer of the Feature Diagram and maximum number of layers in the diagram respectively.

SRQ6 WHAT IS THE PERCEIVED QUALITY AND USEFULNESS OF THE RECONSTRUCTED SOFTWARE ARCHITECTURE, WHEN CREATED BASED ON FEATURE EXTRACTION FROM AN EXISTING SOFTWARE PRODUCT?

The reconstructed architecture has been evaluated based on quality and usefulness in an interview with the lead developer of the reconstructed application (SRQ6). He states that the architecture and modeling style are very clear: “The architecture matches the code base very closely. In my feeling, it works well with React, because the application is divided into components, in a similar way the architecture

consists of different modules.” React is an open-source JavaScript library used for the creation of the interactive user interfaces of the application. Mainly the naming of features and (sub)modules varies between the model and code base. He suggests working together with the product owner to improve both the architecture model and code base at the same time: “We can improve the code by means of this model and we can improve the model on the basis of the code. The first approach may even work best. The more the two correspond, the better.”

A feature diagram with a high depth indicates that multiple clicks needs to be performed in order to perform an action by the user. Although this may seem to complexify the application, the developer argues that grouping features and giving them the right naming actually may simplify the user interface.

The reconstructed architecture is perceived as useful in three aspects: it will (1) improve communication between the product owner and software architect, may help to (2) make an estimation of the workload of new developing additional features, and to (3) prepare new developers unfamiliar with the code base – compared to the previous situation in which no documentation existed.

6.1.3. Requirements engineering

Traditional requirement activities – elicitation, analysis and negotiation, documentation, validation, and management – do not take the iterative processes of agile software development into account and changed accordingly over the years. However, agile RE does not only alleviate challenges of traditional RE, but also poses new ones. Minimal documentation, customer inability, customer agreement and inappropriate architecture are reported as some of the challenges of agile RE [21]. A proper use of artifacts can overcome the documentation problems and organizations apply different techniques to do so.

In order to answer SRQ2, we cover each artifact mentioned in the question in more detail based on the literature review.

SRQ2 WHAT ARE THE PRINCIPLES IN REQUIREMENTS ENGINEERING THEORY REGARDING JOBS, EPIC STORIES, USER STORIES AND CROWDSOURCING?

The notion that customers hire products to do specific jobs for them is the basic principle of the theory of Jobs To Be Done (JTBD). Customers should not be asked what they want, but what they want as an outcome instead. We've analyzed the three different views on the theory by the authorities on this domain: Antony Ulwick, Clayton Christensen and Alan Klement. Their main disagreements concern the definition of JTBD, in which we distinguish between do-goals (activities and tasks) and be-goals (progress), and the dimensions of JTBD. We can still argue that each **Job**, written in natural language, consists of a struggle, goal and (optionally) contextual clarifier. While a high-level Job shines a new light on a business, its customers and competition, it does not provide a tool for a design or product team to work with during software development. Therefore, each design problem can be framed as an **Epic Story**, focusing on the triggering event or situation, the motivation and goal, and the intended outcome. To stimulate creativity while designing the implementation, an Epic Story covers the *why* instead of the *who* and *how*. A **User Story** does include a persona and can be defined as a description of a feature written from the perspective of the person who needs this. A User Story contains a role, goal and benefit.

There is a tight coupling of User Stories with Agile methods, as shown by Lucassen et al. [97] based on a large survey among practitioners in the software industry. Despite the rise of **crowdsourcing** in RE, those User Stories are still mostly written by professionals from inside the organization. However, Crowd-Centric Requirements Engineering has been a proven way to help fostering user involvement and has been perceived by users as more useful and more engaging compared with previous feedback experiences [7], [54]. Therefore, we designed a crowdsourced requirements engineering platform to allow end-users to gather, negotiate and prioritize requirements in the form of User Stories (SRQ5).

SRQ5 HOW CAN A CROWDSOURCED REQUIREMENT ENGINEERING PLATFORM BE DESIGNED TO SUPPORT THE ELICITATION, NEGOTIATION, AND PRIORITIZATION OF USER STORIES FOR AN EXISTING SOFTWARE PRODUCT?

The crowdsourced requirements engineering platform needs to enable users of a software application to submit feature requests in the strict format of User Stories - containing a role, goal and benefit. In order to help users to formulate these stories, even if they have never seen or heard of a User Story before, we used a form with four simple self-explanatory and small steps. In the first step, the user needs to select one of the roles from the predefined options using radio buttons, to find out in which role the requester uses the application. In the second step, we ask the user to type in what he or she wants to do with the application and provide an input field starting with the static text "I want to". In step 3, we ask explicitly why the user wants to have the requested feature, to know what the user sees as the potential benefit when the feature would be implemented in the software application. The answer always contains the predefined "So that" at the start. In the last step, the user is able to verify the User Story that has been formulated based on the answers he or she provided in the first three steps. We also ask the user to select one of the predefined categories, so we can group the feature requests.

All requests are published on a feature request overview page in the application, which also allows other users to view ideas, comment on ideas and vote on ideas using the confirmation or negation technique.

SRQ7 WHAT IS THE EFFECT OF USING A CROWDSOURCED REQUIREMENTS ENGINEERING PLATFORM TO SUPPORT THE ELICITATION, NEGOTIATION, AND PRIORITIZATION OF USER STORIES FOR AN EXISTING SOFTWARE PRODUCT?

In order to answer SRQ7, the feature request platform has been deployed and we measured the results for a period of five week. In those weeks, had 157 unique visitors on the feature request platform. From those visitors, 39 users interacted with the platform by submitting an idea (23), voting on an idea (28), and/or commenting on an idea (9). Together, they submitted 57 ideas, voted 89 times and commented 14 times. More than half of the requesters (15, 65%) submitted only

one idea, two users submitted respectively two and three ideas and four users submitted five or more ideas (respectively 5, 6, 7, and 14 ideas).

We evaluated the perceived usefulness of the platform based on a questionnaire (13 respondents) among the users who interacted with the platform. They perceived the platform as very useful, regarding all four possible interactions when rated on a five-point Likert scale: requesting ($M = 4.9$; $SD = 0.28$), viewing ($M = 4.8$; $SD = 0.38$), voting ($M = 4.5$; $SD = 0.88$), and commenting ($M = 4.5$; $SD = 0.66$). These findings are in line with the work of Snijders et al. [55] who demonstrated how voting and commenting on a gamified crowdsourcing platform was perceived as very useful. The users also felt more engaged compared to previous feedback experiences, whereas we did not explicitly compare our platform to other feedback forms or alternative notations to express requirements. Almost 77% of the respondents has never written a User Story before. When asked if they find it helpful to formulate the ideas as User Stories, compared to free texts, the average score was 3.6 ($SD = 0.87$). There is hardly any preference to write the feature requests in free text ($M = 3.2$; $SD = 0.99$).

Each User Story has been tested against eight criteria from the Quality User Story framework. Most frequent occurring defects are User Stories violating the minimal criterion (42.9%) or not being written as one full sentence (33.9%). Lucassen et al. [9] tested 1000+ User Stories written by professionals from different companies and found that the minimal criterion is violated in 13.3% of the cases. Based on this observation we can conclude that crowdsourced User Stories are currently over three times more likely to contain comments, descriptions of the expected behavior, or testing hints, when compared to those written by professionals. This additional information should be left to the comment section of the platform. The violation of the minimal criterion is also reflected in the length of the crowdsourced User Stories. The goal is expressed in 108 characters on average and crowd workers needed 97 characters on average to formulate the potential benefit. When compared to 551 real-world English User Stories from eight different projects, retrieved from a publicly available data set [98], we found the means plus end of the Dutch crowdsourced User Stories (204 characters) to be over two times

longer than the User Stories written by professionals (97 characters), which had an average goal description of 51 characters and benefit expression in 55 characters, if present. The length of the crowdsourced User Stories is similar to the length of the feature requests that were sent in by email or the support chat (192 characters) prior to the deployment of the platform. Note that we did not count the terms from the User Story format ('I want to' and 'so that') and did not control for the information density of the different languages the User Stories are written in. Moreover, 17% of the real-world User Stories lack a description of its benefit. There is also a major difference in the use of roles. We defined three roles for the Tournify application (organizer, participant, supporter). All requesters indicated they are organizers, whereas professionals use 12 roles on average in their User Story set.

The crowdsourced User Stories and User Stories written by professionals show a similar number of defects regarding the well-formed criterion (5.4% crowd, 4.5 professionals) and atomic criterion (8.9% crowd, 10.3% professionals). In total, 52% of the crowdsourced User Stories meet all requirements, meaning that 48% of the User Stories contains one or more easily preventable error(s). Lucassen et al. [9] conclude that 56% of User Stories written by professionals have at least one defect as detected by their automatic testing tool. However, these results are difficult to compare as Lucassen et al. [9] tested against less, but different, criteria from the framework than we did.

Based on our results, we see opportunities for improving the crowdsourced requirements engineering platform to enhance the quality of the User Stories. Defects on the minimal and full sentence criteria can be prevented with simple means like a spelling checker and warnings when there is additional text after a dot, hyphen, semicolon, or other separating punctuation marks. Text between brackets should also trigger a warning message on the screen.

Lastly, the crowdsourced User Stories have been evaluated based on their complexity by the developer of Tournify. Most of the crowdsourced User Stories (90 percent) can be developed within one workday. The hour estimation has been mapped against the estimation of the complexity we made based on the impact a

feature would have on the functional architecture, showing a moderate positive correlation ($r_s = 0.56$, $p = < .001$). This result shows that end-users mainly require small additions or changes to the software application and also sets the way for further analysis on (automatically) linking new requirements to existing functional architecture modules for the purpose of workload estimation.

Since we also have information on the requirements management practices of Tournify prior to the deployment of the platform, it is valuable to dive deeper into the impact the platform had on the organization. One of the aspects it may influence is the workload of the requirements engineer, who spends an average of ten minutes to process each request that comes in by phone or chat. During the testing period, 17 features were requested through one of those media, bypassing the feature request platform (Table 9). In most cases, those users were unaware the platform existed. When corrected for the duration of the measurement, the number of ideas that were sent in every day remained unchanged. However, since the number of organizers using the service increased with over 175%, this saves the requirements engineer an estimated two hours of work per month. Although this time saving seems currently insignificant, it will have an impact when the business grows. Still it is fair to say that the main benefit of having a crowdsourced requirements platform is to engage users and gather, prioritize and negotiate high-quality requirements, rather than replacing the work of the requirements engineer.

	Pre-introduction	Post-introduction
Unique page views	5678	2363
Duration of measurement (days)	153	36
Unique page views per day (average)	37	66
Requested ideas via email or chat	77	17
Ideas per 1000 unique page views	13,6	7,2
Ideas per day	0.50	0.47

Table 9: Feature requests via email or chat pre- and post-introduction of the crowdsourced requirements engineering platform

6.2. Answering the main research question

In this study we aimed to find an answer to the following main research question:

RQ HOW CAN THE REQUIREMENTS ENGINEERING FOR SOFTWARE ARCHITECTURE MODEL (RE4SA) BE APPLIED IN EXISTING SOFTWARE PRODUCTS, WHILE MAKING USE OF CROWDSOURCING IN REQUIREMENTS ENGINEERING?

The RE4SA model intends to improve communication between requirements engineers and software architects through “simple communication means, clear structural guidelines, and consistent domain terminology” [4]. This is done by linking existing artifacts that are already used by practitioners in the software industry. In the RE4SA model, a relationship is established between Epic Stories, User Stories, Modules, and Features. By applying the principles of the RE4SA model to the case of an existing software application, we aimed to find out how organizations can benefit from it, even if artifacts are missing. We combined the RE4SA model with crowdsourcing in requirements engineering. The answer to the research question is twofold:

1. For reconstructing a functional architecture based on the principles of the RE4SA model, the graphical user interface (GUI) of an application is a valuable asset. This process consists of (1) feature extraction to create a Feature Diagram, (2) abstracting the information to identify (sub)modules for a Functional Architecture Diagram and (3) presenting the information in a friendly way so both requirements engineers and software architects can understand each functionality of a software product. The reconstructed architecture is useful for code refactoring, improves internal communication and serves as a resource in continuous development.
2. A newly developed crowdsourced requirements engineering platform allows users of an application to express feature requests in the form of User Stories. This platform uses a form with four small and interconnected steps. Not only requesting features, but also viewing features from other users, rating features, and commenting on features are perceived as very

useful by the crowd workers. Although enhancements to the platform may be necessary to produce User Stories with less violations of the minimal and full sentence quality criteria, more than half of the User Stories are written flawlessly by people mostly unfamiliar with the concept. Establishing links between the crowdsourced User Stories and the functional architecture based on the RE4SA model gives an indication of the development workload and solution design.

6.3. Conclusion

Since we answered all sub research questions and formulated an answer to the main research question of this study, we can now draw the main conclusions from this research project. We combined a literature review with a single-case Technical Action Research to assess the effects of the RE4SA principles in a business context, while helping a client at the same time.

During the research, we recovered the functional architecture of Tournify, a web application used by sports associations to organize tournaments. The RE4SA model and its related artifacts (features and modules) and modeling techniques (Feature Diagrams and Functional Architecture Diagrams) have been used in the architecture recovery, leading to an architecture that is useful for code refactoring, improves internal communication and serves as a resource in continuous development.

At the same time, an addition to the application has been made. A new feature request platform allows end users to submit feature requests in the form of User Stories. The platform was not only perceived as very useful by the customers who interacted with it, but also delivered requirements that did not inferior to those written by professionals, when tested on quality. However, *minimality* (a User Story contains more information than necessary) was significantly worse than User Stories written by professionals. Newer versions of the feature request platform will focus on improving the quality of requirements by providing real time feedback to requesters, who mostly have never written User Stories before.

Finally, we have performed a first attempt to establish links between the crowdsourced User Stories and the functional architecture based on the RE4SA model. Our findings show how the recovered architecture can be used as an indication of the development workload of new User Stories, written by crowd workers.

6.4. Validity threats

Reflecting on the validity of this study, we cover both external and internal validity threats. Regarding the generalizability of this study, the main concern is that we performed a single-case study. The nature of this study is explorative, and this is one of the first attempts to bring the RE4SA principles from the literature study to conditions of practice in an organization. To mitigate this risk, multiple students from the Requirement Engineering lab at Utrecht University are working on other cases at different organizations. The results of this thesis have to be considered in this broader perspective. In our study, for example, there was only one developer we could interview regarding the software architecture. Not only the (size and type of the) organization can have an impact on the results but also the size and programming language of the application, and presence of development artifacts need to be taken into account. Furthermore, we focused solely on the applicability of the artifacts from the RE4SA model in our case study, and therefore refrained from comparing our techniques to alternative formalisms or notations to express requirements and architectures.

Most of the work in this thesis has been done manually by only one researcher, like the theory construction, extraction of features and identification of modules. Although we did our best to describe the taken procedures carefully and made all materials available in the appendices or online, this causes validity threats regarding the reproducibility of this study. The personal involvement of the main researcher at Tournify, and how we dealt with this concern, has been extensively covered in section 4.1.1. Even despite we followed the steps of Canonical Action Research carefully to ensure the rigor and relevance of this study, it is still possible

that the results are influenced because respondents modified their responses because they knew they were part of a study (known as the *Hawthorne effect*).

The biggest limitation regarding the crowdsourced requirements engineering platform is the small sample size. Over half of the crowdsourced User Stories is written by four users. This means that their expertise highly influenced the overall results regarding the quality evaluation.

6.5. Future research

There have been several studies focusing on User Stories in recent years. Also crowdsourced requirements engineering gained attention. However, this is the first study to combine the two interest fields, by focusing explicitly on User Story writing by crowd workers. We can continue this work by implementing direct feedback techniques during the User Story formulation to improve the syntactic quality. Further research can also focus on the usefulness of having a *role* in User Stories written by crowd workers, can focus on the interplay between Epic Stories and User Stories, and possibility to combine crowdsourcing with crowdfunding in the development of new features. The expertise of crowd workers, in relation to their involvement with the software product is also worth investigating in further research.

Regarding the functional architecture (recovery) more research is needed on (automated) architecture recovery techniques and how to keep an architecture up to date. Dedicated (web-based) modelling tools based on the RE4SA model can be developed and tested to support traceability between requirements and architecture. Also interesting is the use of Natural Language Processing (NLP) to support traceability and estimate the workload of new requirements by linking text in the User Stories to names of features and (sub)modules in the architecture. Lastly, we are interested in the relation between the depth and degree of a Feature Diagram and the perceived complexity of a user interface.

Bibliography

- [1] B. Nuseibeh, "Weaving together requirements and architectures," *Computer (Long Beach, Calif.)*, vol. 34, no. 3, pp. 115–119, 2001.
- [2] G. Lucassen, M. van de Keuken, F. Dalpiaz, S. Brinkkemper, G. W. Sloof, and J. Schlingmann, "Jobs-to-be-Done Oriented Requirements Engineering: A Method for Defining Job Stories," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*, 2018, pp. 227–243.
- [3] G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper, "Improving user story practice with the grimm method: A multiple case study in the software industry," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*, 2017, pp. 235–252.
- [4] S. Brinkkemper, "The Requirements Engineering for Software Architecture Model," unpublished.
- [5] R. Blessinga, "Designing for the Automated Greenhouse - Matching Requirements and Architecture for Startup Product Specification Using Epic Stories," Utrecht University, 2018.
- [6] G. Rasool and N. Asif, "Software architecture recovery," *Int. J. Comput. Information, Syst. Sci. Eng.*, vol. 1, no. 3, 2007.
- [7] R. Snijders, F. Dalpiaz, M. Hosseini, A. Shahri, and R. Ali, "Crowd-centric requirements engineering," in *Proceedings - 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC 2014*, 2014.
- [8] S. Martens, S. Brinkkemper, and F. Dalpiaz, "Ontological Traceability for Software: A new avenue for software traceability," 2018.
- [9] G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper, "Improving agile requirements: the Quality User Story framework and tool," *Requir. Eng.*, no. 21(3), pp. 383–403, 2016.
- [10] E. Enkel, C. Kausch, and O. Gassmann, "Managing the risk of customer integration," *Eur. Manag. J.*, 2005.
- [11] R. Wieringa, *Design Science Methodology for Information Systems and Software Engineering*. 2014.
- [12] M. Cohn, *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.
- [13] C. M. Christensen, T. Hall, K. Dillon, and D. S. Duncan, "Know Your Customers' Jobs to Be Done," *Harv. Bus. Rev.*, 2016.
- [14] N. Medvidovic, D. S. Rosenblum, D. F. Redmiles, and J. E. Robbins, "Modeling software architectures in the Unified Modeling Language," *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 1, pp. 2–57, 2002.
- [15] N. Rozanski and E. Woods, *Software systems architecture: working with stakeholders using viewpoints and perspectives*. 2005.
- [16] A. Kossiakoff, W. N. Sweet, S. J. Seymour, and S. M. Biemer, *Systems engineering principles and practice*, vol. 83. John Wiley & Sons, 2011.

- [17] S. Brinkkemper and S. Pachidi, "Functional architecture modeling for the software product industry," in *European Conference on Software Architecture*, 2010, pp. 198–213.
- [18] M. Riebisch, "Towards a more precise definition of feature models," *Model. Var. Object-Oriented Prod. Lines*, pp. 64–76, 2003.
- [19] J. Webster and R. T. Watson, "Analyzing the Past to Prepare for the Future: Writing a Literature Review.," *MIS Q.*, 2002.
- [20] E.-M. Schön, J. Thomaschewski, and M. J. Escalona, "Agile Requirements Engineering: A systematic literature review," *Comput. Stand. Interfaces*, 2017.
- [21] I. Inayat, S. S. Salim, S. Marczak, M. Daneva, and S. Shamshirband, "A systematic literature review on agile requirements engineering practices and challenges," *Computers in Human Behavior*. 2015.
- [22] R. Davison, M. G. Martinsons, and N. Kock, "Principles of canonical action research," *Inf. Syst. J.*, vol. 14, no. 1, pp. 65–86, 2004.
- [23] R. Wieringa and A. Morali, "Technical action research as a validation method in information systems design science," in *International Conference on Design Science Research in Information Systems*, 2012, pp. 220–238.
- [24] W. Swartout and R. Balzer, "On the inevitable intertwining of specification and implementation," *Commun. ACM*, 1982.
- [25] G. Lucassen, F. Dalpiaz, J. M. Van Der Werf, and S. Brinkkemper, "Bridging the twin peaks: the case of the software industry," in *Proceedings of the Fifth International Workshop on Twin Peaks of Requirements and Architecture*, 2015, pp. 24–28.
- [26] E. Stachtari, A. Mavridou, P. Katsaros, S. Bliudze, and J. Sifakis, "Early validation of system requirements and design through correctness-by-construction," *J. Syst. Softw.*, vol. 145, pp. 52–78, 2018.
- [27] A. Alebrahim and M. Heisel, *Bridging the Gap Between Requirements Engineering and Software Architecture*. Springer, 2017.
- [28] G. Lucassen, J. M. E. M. van der Werf, and S. Brinkkemper, "Alignment of software product management and software architecture with discussion models," in *Software Product Management (IWSPM), 2014 IEEE IWSPM 8th International Workshop on Software Product Management (IWSPM)*, 2014, pp. 21–30.
- [29] A. Klement, *When coffee and kale compete: Become great at making products people will buy*. CreateSpace Independent Publishing Platform, 2018.
- [30] T. Salfischberger, I. van de Weerd, and S. Brinkkemper, "The Functional Architecture Framework for organizing high volume requirements management," in *Software Product Management (IWSPM), 2011 Fifth International Workshop on*, 2011.
- [31] J. Dick, E. Hull, and K. Jackson, *Requirements engineering*. Springer, 2017.
- [32] A. W. Ulwick, *Jobs to be done: theory to practice*. Idea Bite Press, 2016.
- [33] C. M. Christensen and M. E. Raynor, *The Innovator's Solution: Creating and Sustaining Successful Growth*. 2003.
- [34] A. W. Ulwick, "Alan Klement's War On Jobs-To-Be-Done," 2016. [Online]. Available: <https://jobs-to-be-done.com/alan-klements-war-on-jobs-to-be-done-dad8eae567c>. [Accessed: 18-Oct-2018].

- [35] A. W. Ulwick, *Business strategy formulation: theory, process, and the intellectual revolution*. Quorum Books Westport, 1999.
- [36] A. W. Ulwick, *What customers want: Using Outcome-Driven Innovation to Create Breakthrough Products and Services*. McGraw-Hill Professional Publishing, 2005.
- [37] A. W. Ulwick, "Turn customer input into innovation.," *Harv. Bus. Rev.*, vol. 80, no. 1, pp. 91–97, 2002.
- [38] J. L. Bower and C. M. Christensen, "Disruptive technologies: catching the wave," *Harv. Bus. Rev.*, 1995.
- [39] C. M. Christensen, *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail*. Boston, MA: Harvard Business School Press, 1997.
- [40] C. M. Christensen, T. Hall, K. Dillon, and D. S. Duncan, *Competing Against Luck: The Story of Innovation and Customer Choice*. New York: HarperBusiness, 2016.
- [41] D. Norman, *The design of everyday things: Revised and expanded edition*. Constellation, 2013.
- [42] P. Adams, "The dribbblisation of design," *Inside Intercom*, 2013. [Online]. Available: <https://intercom.com/blog/the-dribbblisation-of-design/>. [Accessed: 16-Oct-2018].
- [43] A. Klement, "5 Tips For Writing A Job Story," *JTBD*, 2013. [Online]. Available: <https://jtbtd.info/5-tips-for-writing-a-job-story-7c9092911fc9>. [Accessed: 16-Oct-2018].
- [44] G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper, "Forging high-quality user stories: towards a discipline for agile requirements," in *Requirements Engineering Conference (RE), 2015 IEEE 23rd International*, 2015, pp. 126–135.
- [45] A. Klement, "Replacing The User Story With The Job Story," *JTBD*, 2013. [Online]. Available: <https://jtbtd.info/replacing-the-user-story-with-the-job-story-af7cdee10c27>. [Accessed: 16-Oct-2018].
- [46] A. Klement, "Designing Features Using Job Stories," *JTBD*, 2013. [Online]. Available: <https://jtbtd.info/designing-features-using-job-stories-41d20fc7ade6>. [Accessed: 16-Oct-2018].
- [47] B. Wake, "INVEST in Good Stories, and SMART Tasks," 2003. [Online]. Available: <https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/>. [Accessed: 18-Oct-2018].
- [48] M. Daneva *et al.*, "Agile requirements prioritization in large-scale outsourced system projects: An empirical study," *J. Syst. Softw.*, vol. 86, no. 5, pp. 1333–1353, 2013.
- [49] O. Liskin, K. Schneider, F. Fagerholm, and J. Münch, "Understanding the role of requirements artifacts in kanban," in *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering - CHASE 2014*, 2014, pp. 56–63.
- [50] N. Tripathi *et al.*, "An anatomy of requirements engineering in software startups using multi-vocal literature and case survey," *J. Syst. Softw.*, vol. 146, pp. 130–151, 2018.
- [51] K. Mao, L. Capra, M. Harman, and Y. Jia, "A survey of the use of crowdsourcing in software engineering," *J. Syst. Softw.*, vol. 126, pp. 57–84, 2017.

- [52] M. Hosseini, A. Shahri, K. T. Phalp, J. Taylor, R. Ali, and F. Dalpiaz, "Configuring crowdsourcing for requirements elicitation," in *The IEEE Ninth International Conference on Research Challenges in Information Science (RCIS'15), 13--15 May 2015, Athens, Greece*, 2015.
- [53] M. Hosseini, K. Phalp, J. Taylor, and R. Ali, "Towards crowdsourcing for requirements engineering," in *CEUR Workshop Proceedings*, 2014.
- [54] F. Dalpiaz, R. Snijders, S. Brinkkemper, M. Hosseini, A. Shahri, and R. Ali, "Engaging the crowd of stakeholders in requirements engineering via gamification," in *Gamification*, S. Stieglitz, C. Lattemann, S. Robra-Bissantz, R. Zarnekow, and T. Brockmann, Eds. Springer, 2017, pp. 123–135.
- [55] R. Snijders, F. Dalpiaz, S. Brinkkemper, M. Hosseini, R. Ali, and A. Özüm, "REFine: A gamified platform for participatory requirements engineering," in *1st International Workshop on Crowd-Based Requirements Engineering (CrowdRE'15), Co-Located with RE'15, 24 August 2015, Ottawa, Canada.*, 2015.
- [56] A. Adepetu, K. A. Ahmed, Y. Al Abd, A. Al Zaabi, and D. Svetinovic, "CrowdREquire: A Requirements Engineering Crowdsourcing Platform.," in *AAAI Spring Symposium: Wisdom of the Crowd*, 2012, pp. 2–7.
- [57] S. L. Lim, "Social networks and collaborative filtering for large-scale requirements elicitation," University of New South Wales, 2011.
- [58] S. L. Lim and A. Finkelstein, "StakeRare: using social networks and collaborative filtering for large-scale requirements elicitation," *IEEE Trans. Softw. Eng.*, vol. 38, no. 3, pp. 707–735, 2012.
- [59] S. L. Lim, D. Quercia, and A. Finkelstein, "StakeNet: using social networks to analyse the stakeholders of large-scale software projects," in *Proceedings of the 32nd International Conference on Software Engineering*, 2010, pp. 295–304.
- [60] S. L. Lim, D. Quercia, and A. Finkelstein, "StakeSource: harnessing the power of crowdsourcing and social networks in stakeholder analysis," in *Proceedings of the 32nd International Conference on Software Engineering*, 2010, vol. 2, pp. 239–242.
- [61] S. L. Lim, D. Damian, and A. Finkelstein, "StakeSource2. 0: using social networks of stakeholders to identify and prioritise requirements," in *Proceedings of the 33rd International Conference on Software Engineering*, 2011, pp. 1022–1024.
- [62] Barbary Software SL, "Feature Upvote," 2019. [Online]. Available: www.featureupvote.com.
- [63] Sprinkl, "Get Satisfaction," 2019. [Online]. Available: www.getsatisfaction.com.
- [64] UserVoice Inc, "UserVoice," 2019. [Online]. Available: www.uservoice.com.
- [65] Pranav Singh, "Cadet," 2019. [Online]. Available: www.getcadet.com.
- [66] Pendo, "Receptive," 2019. [Online]. Available: www.receptive.io.
- [67] Instabug Inc, "Instabug," 2019. [Online]. Available: www.instabug.com.
- [68] D. Renzel, M. Behrendt, R. Klamma, and M. Jarke, "Requirements bazaar: Social requirements engineering for community-driven innovation," in *21st IEEE International Requirements Engineering Conference (RE)*, 2013, pp. 326–327.
- [69] P. Greenwood, A. Rashid, and J. Walkerdine, "UDesignIt: Towards social media for community-driven design," in *The 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 1321–1324.

- [70] K.-J. Stol and B. Fitzgerald, "Two's company, three's a crowd: a case study of crowdsourcing software development," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 187–198.
- [71] T. D. LaToza and A. van der Hoek, "Crowdsourcing in software engineering: Models, motivations, and challenges," *IEEE Softw.*, vol. 33, no. 1, pp. 74–80, 2016.
- [72] N. Sherief, W. Abdelmoez, K. Phalp, and R. Ali, "Modelling users feedback in crowd-based requirements engineering: An empirical study," in *IFIP Working Conference on The Practice of Enterprise Modeling*, 2015, pp. 174–190.
- [73] W. Maalej, M. Nayebi, T. Johann, and G. Ruhe, "Toward data-driven requirements engineering," *IEEE Softw.*, vol. 33, no. 1, pp. 48–54, 2016.
- [74] P. Clements *et al.*, *Documenting software architectures: views and beyond*. Pearson Education, 2002.
- [75] "What is your definition of software architecture?," *Software Engineering Institute*, 2010. [Online]. Available: <https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=513807>. [Accessed: 01-Nov-2018].
- [76] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Addison-Wesley Professional, 2003.
- [77] ISO, "Systems and software engineering—architecture description," ISO/IEC/IEEE 42010, 2011.
- [78] P. B. Kruchten, "The 4+ 1 view model of architecture," *IEEE Softw.*, vol. 12, no. 6, pp. 42–50, 1995.
- [79] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, and A. Tang, "What industry needs from architectural languages: A survey," *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 869–891, 2013.
- [80] N. Jansen and J. van Rhijn, "Utrecht Architecture Description Language," 2018.
- [81] P. Kruchten, "Software architecture and agile software development: a clash of two cultures?," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering—Volume 2*, 2010, pp. 497–498.
- [82] D. Falessi, G. Cantone, S. A. Sarcia, G. Calavaro, P. Subiaco, and C. D'Amore, "Peaceful coexistence: Agile developer perspectives on software architecture," *IEEE Softw.*, vol. 27, no. 2, 2010.
- [83] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.
- [84] M. Glinz, "Improving the quality of requirements with scenarios," in *Proceedings of the second world congress on software quality*, 2000, vol. 9, pp. 55–60.
- [85] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1990.
- [86] D. A. Tamburri and R. Kazman, "General methods for software architecture recovery: a potential approach and its evaluation," *Empir. Softw. Eng.*, vol. 23, no. 3, pp. 1457–1489, 2018.
- [87] A. E. Hassan and R. C. Holt, "Architecture recovery of web applications," in *Proceedings of the 24th International Conference on Software Engineering*, 2002, pp. 349–359.

- [88] E. J. Chikofsky and J. H. Cross, "Reverse engineering and design recovery: A taxonomy," *IEEE Softw.*, vol. 7, no. 1, pp. 13–17, 1990.
- [89] R. L. Krikhaar, *Software architecture reconstruction*. Philips Electronics, 1999.
- [90] T. Lutellier *et al.*, "Measuring the impact of code dependencies on software architecture recovery techniques," *IEEE Trans. Softw. Eng.*, vol. 44, no. 2, pp. 159–181, 2018.
- [91] T. E. J. Vos, P. M. Kruse, N. Condori-Fernández, S. Bauersfeld, and J. Wegener, "Testar: Tool support for test automation at the user interface level," *Int. J. Inf. Syst. Model. Des.*, vol. 6, no. 3, pp. 46–83, 2015.
- [92] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Commun. ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.
- [93] N. A. Maiden and C. Ncube, "Acquiring COTS software selection requirements," *IEEE Softw.*, vol. 15, no. 2, pp. 46–56, 1998.
- [94] P. Berander and A. Andrews, "Requirements prioritization," in *Engineering and managing software requirements*, Springer, 2005, pp. 69–94.
- [95] B. Kaplan and D. Duchon, "Combining qualitative and quantitative methods in information systems research: a case study," *MIS Q.*, pp. 571–586, 1988.
- [96] P. Paskevicius, R. Damasevicius, and V. Štuikys, "Quality-Oriented Product Line Modeling Using Feature Diagrams and Preference Logic," in *International Conference on Information and Software Technologies*, 2012, pp. 241–254.
- [97] G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper, "The use and effectiveness of user stories in practice," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*, 2016, pp. 205–222.
- [98] F. Dalpiaz, "Requirements data sets (user stories)," 2018. [Online]. Available: <http://dx.doi.org/10.17632/7zbk8zsd8y.1>.

Appendices

1. Criteria to ensure and to assess the rigor and relevance of Canonical Action Research [22] and the roles and responsibilities defined for this project

Criteria for the RCA

- 1a Did both the researcher and the client agree that CAR was the appropriate approach for the organizational situation?
- 1b Was the focus of the research project specified clearly and explicitly?
- 1c Did the client make an explicit commitment to the project?
- 1d Were the roles and responsibilities of the researcher and client organization members specified explicitly?
- 1e Were project objectives and evaluation measures specified explicitly?
- 1f Were the data collection and analysis methods specified explicitly?

Criteria for the CPM

- 2a Did the project follow the CPM or justify any deviation from it?
- 2b Did the researcher conduct an independent diagnosis of the organizational situation?
- 2c Were the planned actions based explicitly on the results of the diagnosis?
- 2d Were the planned actions implemented and evaluated?
- 2e Did the researcher reflect on the outcomes of the intervention?
- 2f Was this reflection followed by an explicit decision on whether or not to proceed through an additional process cycle?
- 2g Were both the exit of the researcher and the conclusion of the project due to either the project objectives being met or some other clearly articulated justification?

Criteria for the Principle of Theory

- 3a Were the project activities guided by a theory or set of theories?
- 3b Was the domain of investigation, and the specific problem setting, relevant and significant to the interests of the researcher's community of peers as well as the client?
- 3c Was a theoretically based model used to derive the causes of the observed problem?
- 3d Did the planned intervention follow from this theoretically based model?
- 3e Was the guiding theory, or any other theory, used to evaluate the outcomes of the intervention?

Criteria for the Principle of Change through Action

- 4a Were both the researcher and client motivated to improve the situation?
- 4b Were the problem and its hypothesized cause(s) specified as a result of the diagnosis?
- 4c Were the planned actions designed to address the hypothesized cause(s)?
- 4d Did the client approve the planned actions before they were implemented?
- 4e Was the organization situation assessed comprehensively both before and after the intervention?
- 4f Were the timing and nature of the actions taken clearly and completely documented?

Criteria for the Principle of Learning through Reflection

- 5a Did the researcher provide progress reports to the client and organizational members?
- 5b Did both the researcher and the client reflect upon the outcomes of the project?
- 5c Were the research activities and outcomes reported clearly and completely?
- 5d Were the results considered in terms of implications for further action in this situation?
- 5e Were the results considered in terms of implications for action to be taken in related research domains?
- 5f Were the results considered in terms of implications for the research community (general knowledge, informing/re-informing theory)?
- 5g Were the results considered in terms of the general applicability of CAR?

Abel Menkveld	Sjaak Brinkkemper	Fabiano Dalpiaz	Jesse
Lead researcher (Utrecht University)	First supervisor (Utrecht University)	Second Supervisor (Utrecht University)	Software developer (Tournify)
Product manager (Tournify)			
Graduate Student Business Informatics (Utrecht University)	Professor (Utrecht University)	Assistant Professor (Utrecht University)	Co-founder (Tournify)
Co-founder (Tournify)			
Responsible for the thesis including the literature review, treatment design and validation, and conclusions.	Responsible for providing feedback during bi-weekly group meetings and individual meetings. Also responsible for the review of the thesis.	Responsible for the review of the project proposal and final deliverable.	Responsible for the implementa- tion of the crowdsourcing platform and will be interviewed in the validation of the recovered architecture.

2. Feature requests proposed by email or chat

Date	Medium	Feature Request (Dutch)
01-09-18	E-mail	Er kwamen veel vragen binnen omtrent spelregels. Ik dacht: is het geen idee om een extra optionele kopje te plaatsen met daarin de spelregels? Dat scheelt een hoop uitleg en gedoe.
05-09-18	E-mail	We organiseren een dart toernooi tussen 7 of 8 cafés. Nou zouden we graag een schema willen genereren waarbij elk café 2 teams heeft, waarbij elk team tegen alle andere teams gooit. Voorwaarde is dat er altijd één team thuis gooit. We spelen 3 wedstrijden op een avond. Zie je kans om hiervoor een schema te maken met Tournify?
06-09-18	Tijdschr.	De tussenstand wil ik ook kunnen bekijken in het invoerscherm.
06-09-18	Tijdschr.	Er mist uitleg bij opstellen van een toernooischema.
06-09-18	Tijdschr.	Het zou handig zijn om de beheer van een toernooi te kunnen delen, zonder het hele Tournify-account te delen.
06-09-18	Tijdschr.	Beschikbaarheid van scheidsrechters beperken tot een deel van het toernooi.
07-09-18	Chat	Ik zou graag een upgrade willen naar de 40 euro versie van Tournify. Alleen ons bedrijf werkt alleen met factuurbetalingen. Is het mogelijk per factuur te betalen?
09-09-18	E-mail	Ik wil ervoor zorgen dat teams niet meer dan 2 wedstrijden achter elkaar spelen. Ik zou dat in Tournify willen kunnen instellen, of oplossen door wedstrijden op een tijd te plannen. Zo kan ik bijvoorbeeld een poule een wedstrijdronde laten overslaan.
11-09-18	E-mail	Je kunt geen extra sets (2e en eventuele 3e) invullen, wat noodzakelijk is voor ons jaarlijkse recreatieve volleybaltoernooi.
11-09-18	E-mail	Aansluitend op bovenstaand punt: weergave wedstrijdpunten per wedstrijd nodig (set gewonnen 2 punten, gelijkgespeeld 1 punt, verloren 0 punten en dan van alle 2 of 3 de sets bij elkaar opgeteld).
11-09-18	E-mail	Per wedstrijd moeten dus de scores van alle sets + wedstrijdpunten van de gehele wedstrijd getoond worden (set gewonnen 2 punten, gelijkgespeeld 1 punt, verloren 0 punten en dan van alle 2 of 3 de sets bij elkaar opgeteld).
11-09-18	E-mail	Mogelijkheid tot instellen puntenopbouw van wedstrijdpunten (bij ons per set 0 voor verlies, 2 bij winst en 1 bij gelijkspel).
11-09-18	E-mail	Geen berekening van quotiënt aanwezig (saldo voor gedeeld door saldo tegen). Aantal decimalen zouden dan instelbaar moeten zijn of voldoende ruim. Wij hebben momenteel 3 decimalen.
11-09-18	E-mail	Missende overzicht beste nummers 1, 2, 3, 4, etc. Gegevens in deze weergave zouden moeten zijn: Saldo voor, saldo tegen, saldo totaal (saldo voor minus saldo tegen) en quotiënt. Het komt in ons toernooi regelmatig voor dat het aantal teams voor niet helemaal mooie verdelingen over de poules zorgt. Het is dan belangrijk om beslissingen te kunnen maken op basis van overzichten die de verschillen tussen bijvoorbeeld alle nummers 1 van elke poule kunnen laten zien.
11-09-18	E-mail	Optionele countdown op dia's zou fijn zijn zodat men kan zien hoe snel ze moeten lezen.
11-09-18	E-mail	Printweergave van alle wedstrijden zodat je een uitdraai voor in een programmaboekje kunt maken (of wij moet in ons papieren programmaboekje dat elk team vooraf ontvangt al naar onze website verwijzen).

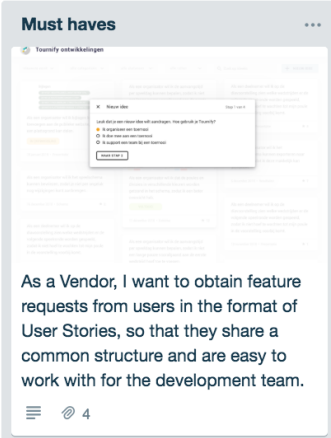
11-09-18	E-mail	Mogelijkheid tot opmerking toevoegen bij een wedstrijd met een belangrijke mededeling. Bijvoorbeeld dat een wedstrijd van een andere poule is. Het komt bij ons nog wel eens voor dat er 1 of meerdere poules zijn met meer teams dan de andere poules en we laten dan altijd meerdere wedstrijden tegelijk spelen uit die poule verspreid over de velden van andere poules die dan niet spelen. De opmerking moet dan ook opvallend op het scorebord te zien.
11-09-18	E-mail	Ook zou het fijn als ik linkjes aan teams kan toevoegen naar onze eigen website zodat als je erop klikt je meer info over het team kan zien met een eventuele teamfoto.
11-09-18	E-mail	Een API om gegevens uit te lezen en plaatsen op Tournify zou ook fijn. Zo kunnen inschrijvingen op onze eigen website dan automatisch ook doorgevoerd worden op Tournify. Dat scheelt wat handmatig werk.
11-09-18	E-mail	Het zou fijn als er een app is die automatisch pushberichten stuurt met updates van scores, poule-indelingen, etc.
12-09-18	Chat	Ik wil graag een toernooi maken voor het tafelfuotballen op het werk. Het probleem waar ik tegen aan loop is dat we 2 tegen 2 willen spelen in roulerende teams. We zijn met 7 spelers en we willen graag een schema maken waarin we in alle verschillende samenstellingen 1x tegen elkaar spelen
12-09-18	Chat	Hoe ga ik om met een schema van 32 waar ik 24 teams heb, waarvan een x aantal dus een bye krijgen in de eerste ronde en ik die wedstrijden al kan invullen met een bye en evt ook planning erop aanpassen?
12-09-18	E-mail	Kunnen we ook een lunchschema toevoegen? Zodat het in schema is opgenomen?
13-09-18	Chat	Ik wil een toernooi organiseren (elke woensdagmiddag) met 10 kinderen. Deze spelen op 2 veldjes 2 tegen 2, waardoor er dus steeds 2 aan de kant staan en pauze hebben. Elk kind speelt steeds met een ander kind in het team. Met 10 kinderen betekent dat ieder kind 9 wedstrijdjes speelt. Is het mogelijk om een dergelijk toernooi opzet te genereren?
15-09-18	E-mail	Ik zou liever de wedstrijd lengte / pauze tussen de wedstrijden ingeven in cijfer en niet via een schuifbalk (het koste mij meerdere pogingen om 10 minuten te selecteren)
15-09-18	E-mail	Ik zou graag een oplees functie hebben. Een kort een wat groter overzicht van de volgende ronde, zodat je dat makkelijk kan oplezen
15-09-18	E-mail	Ik zou graag een eenvoudige print functie hebben om de standen ook op papier bij te houden. Als de internetverbinding wegvalt heb ik graag iets achter de hand.
16-09-18	Chat	Waarom zit er een maximum aan het aantal teams dat je kunt toevoegen aan een poule? Het lijkt erop dat er maximaal 18 in kunnen en ik wil in 1 poule 46 teams plaatsen.
20-09-18	E-mail	We hebben nu een competitie opgezet voor de JO11, JO13 en JO15. Nu willen we van alle 3 de competities de dia's laten lopen, maar dat lukt denk ik niet he? Zo ja, hoe kunnen we dat doen? Is het daarnaast handig om van alle 3 de toernooien (3 verschillende linkjes), 1 toernooi en 1 link van te maken? Kan dat? We moeten dan wel eigenlijk een startscherm hebben, voorafgaand aan de competitie. 3 verschillende knoppen om zeg maar naar de competitie te gaan.
20-09-18	Chat	Zou het ooit mogelijk zijn om een oud toernooi tegen betaling te heropenen voor hergebruik? Dit in plaats van het toernooi als nieuw te moeten opbouwen?
24-09-18	Chat	Ik ben volop Tournify aan het testen voor ons maandelijks dartstornooi. Moeten in de poules de teams manueel worden toegewezen of kan Tournify

		dit zelf "loten"? Ons doel was om via software (zoals Tournify) handmatig loten te vermijden zodat er geen discussies / fouten kunnen ontstaan.
30-09-18	E-mail	Bij de FAQ's zien we dat online inschrijven momenteel niet mogelijk is. Is dat nog steeds zo en zo ja, wanneer wordt dat wel mogelijk? We hadden de link naar de inschrijvingsmodule van Tournify namelijk graag opgenomen in onze publieke communicatie die deze week de deur uitgaat.
02-10-18	Chat	Hoe kunnen mensen zich opgeven voor het toernooi?
05-10-18	E-mail	Bij de vlaggetjes werkt Tournify met Verenigd Koninkrijk, maar de landen zijn gewend aan Engeland, Wales, Schotland en Ierland.
14-10-18	Chat	Is het ook mogelijk om spelers aan clubs toe te voegen om op deze manier topscoorders bij de te houden?
15-10-18	Chat	Wij hebben poules van 5 teams en als ik het programma laat indelen door jullie software moeten teams 2x achter elkaar spelen. Ik heb een schema waarin dat niet hoeft. Nu kan ik ze natuurlijk handmatig verzetten, maar door het programma dit te laten doen is natuurlijk veel makkelijker
15-10-18	Chat	Een logo of wedstrijdshirt van de teams toevoegen zou dat ook in de toekomst mogelijk zijn?
23-10-18	Chat	Je moet een begin en einddatum invullen, zou er een optie kunnen komen dat je dit weg kan laten?
24-10-18	Chat	Ik wil Tournify gebruiken voor een biljart toernooi. Daar maakt niet iedereen hetzelfde aantal caramboles. Bijvoorbeeld Speler A moet er 15, maar Speler B moet er 10. ... Als de uitslag dan 14 - 10 is, heeft toch Speler B gewonnen in dit voorbeeld. Zou je ook aan kunnen geven wie de winnaar is (en waar dus de punten naar toe gaan) na het invullen van de uitslag
22-10-18	E-mail	Kunnen we als een team gediskwalificeerd is, 1 punt in mindering geven?
22-11-18	E-mail	Is er ook een iFrame voorzien voor bv het klassement?
05-12-18	E-mail	Kunnen we de puntentabel handmatig aanpassen voor de punten van onze andere activiteiten, deze zijn als volgt: BuurtBabbel, BuurtBijdrages, BuurtBattle(Wedstrijden), Fairplay en totaal aantal punten die de uiteindelijke klassering betalen.
06-12-18	E-mail	Kan het schema vanuit 3 domeinen worden gekoppeld?
06-12-18	Chat	Kunnen wij de achtergrond aanpassen van de buttons van de verschillende divisies en fases? Het valt nu niet op voor de bezoekers dat je daarop kan klikken.
10-12-18	Chat	Is er ook een Tournify app in de Play Store?
14-12-18	Chat	Kunnen we een print maken van het hele schema voor onszelf, zodat we niet steeds op de pc hoeven te kijken?
15-12-18	E-mail	Het viel me op dat in een schema voor 6 teams het eerste team 5x "thuis" speelt en de rest allemaal 2x. Ik heb dit tijdens het toernooi aangepast maar misschien is dit door jou ook aan te passen?
16-12-18	E-mail	Bij het plannen van een toernooi heb je natuurlijk te maken met een algoritme. Deze deed het in juli van dit jaar nog niet helemaal top waardoor je als gebruiker zelf moet gaan schuiven met wedstrijden.
16-12-18	E-mail	Het zou in mijn optiek prettig zijn als de 'wedstrijdkaarten' wat meer 'vertellen'. Zo zou je bijvoorbeeld de divisies nog kunnen weergeven in het kaartje (middels een box-shadow bijv. Zie bijlage)
16-12-18	E-mail	Om te voorkomen dat teams te vaak achter elkaar spelen zou het fijn zijn om dat snel inzichtelijk te hebben. Je zou bijvoorbeeld op het moment dat op een teamnaam 'hovers' de andere wedstrijdkaarten waar dat team in voorkomt ook kunnen oplichten.

16-12-18	E-mail	Een scheidsrechters/wedstrijdzaken (web)app zou heel erg handig zijn. Nu komen de 'wedstrijdpapiertjes' van de scheidsrechters (zeker bij een groot toernooi) pas na lange tijd binnen waardoor het speelschema in verdere fasen pas erg laat bekend wordt. Uitbreiden zou richting de deelnemers kunnen ('abonneer op alle wedstrijden van team x, y en z) of richting de scheidsrechter (toon al mijn wedstrijden + eenvoudig registreren van de uitslag)
19-12-18	E-mail	Hoewel het fantastisch is dat de tijden automatisch worden geüpdatet, is het zo dat de app het speelschema ook update naar de ingestelde tijdzone van de gebruiker. Zo hadden we een team uit Kaliningrad (+1 uur t.o.v. Amsterdam) die daardoor dachten dat er om 11 uur gespeeld zou worden, dit gaf de nodige rompslomp en organisatorische beslissingen. Ik snap dat wanneer iemand de uitslagen op afstand wil volgen de juiste tijdzone belangrijk is, maar aangezien de app voornamelijk gebruikt wordt door teams die op dezelfde locatie zijn is dit een puntje om nogmaals naar te kijken.
19-12-18	E-mail	Is het ook mogelijk om aanmeldingen te beheren met Tournify? Voorgaande jaren heb ik dit met de hand gemanaged: mensen vullen een formulier in op onze website, en ik stuur een bevestiging, stuur de factuur, houdt bij of er iets wijzigt en of iedereen op tijd betaalt, etc.. Dit bezorgt me ieder jaar heel wat werk, en ik denk zo dat er bij deze klussen heel wat te automatiseren valt. Is er dergelijke functionaliteit bij Tournify, of valt die toe te voegen?
27-12-18	Chat	Hallo! Ik vroeg me af of ik ook een printje kan maken van de wedstrijschema`s. is er een printversie?
29-01-19	Chat	De bestandsnaam van een afbeelding in een diavoorstelling tonen, zodat je ziet welk plaatje op welke sheet staat.
02-01-19	Chat	Kan je ook een toernooi kopiëren omdat ik op 1 avond 2 toernooien heb met zelfde team namen?
03-01-19	Chat	We werken met een 3e official bij de wedstrijden, die kunnen we niet kwijt. Kunnen jullie deze in de komende versie opnemen?
03-01-19	Chat	We hebben een wedstrijdduur per categorie/per dag. Dus de C speelt langer dan de D. Is het mogelijk om dat variabel te maken?
03-01-19	Chat	Wanneer er te veel wedstrijden in een poule zijn, dan getoond worden in de diavoorstelling geeft hij aan "plus meer wedstrijden". Ik wil die echter ook graag tonen.
03-01-19	Chat	We hebben behoefte aan: 1) een overzicht van alle teams van een categorie die meedoen die dag. 2) een overzicht van alle wedstrijden die op een veld gespeeld worden.
03-01-19	Chat	Hoe zou ik om kunnen gaan met gele/rode kaarten. We ontkomen er eigenlijk niet aan om alle spelers te gaan melden bij het toernooi, of zie jij een andere mogelijkheid?
03-01-19	Chat	Tussen de teams die meedoen aan ons toernooi zit veel niveauverschil. Het zal ideaal zijn als ik een team kan laten beginnen met 1, 2 of 3 of meer punten.
07-01-19	E-mail	De "beste nummer 3" moet komen uit een specifiek aantal poules, zodat een team in de tweede fase nooit bij eenzelfde team in de poule terecht komt.
09-01-19	E-mail	Vanuit het aanmelden is het een belangrijk verbeterpunt om meerdere teams aan te kunnen melden.
09-01-19	E-mail	Ik mis printfuncties (wedstrijdoverzicht voor omroeper, wedstrijdbriefjes op A5 of 2 per A4).
09-01-19	E-mail	Het vakje waarin je teams moet toevoegen om de planning te kunnen maken is klein waardoor je steeds moet scrollen. Het zou handig zijn als dat groter kan. Soms was je de cursor kwijt als hij op het tweede scherm was. Je moet

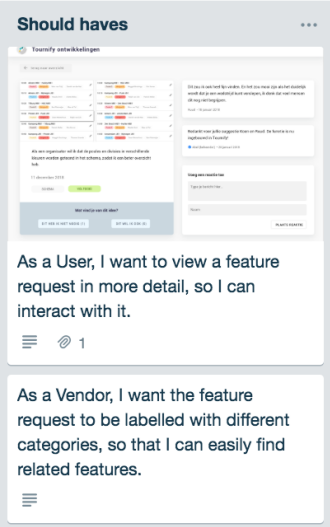
		dan onhandig het scherm draaien of uit de wagen lopen om hem weer te vinden en terug te zetten.
09-01-19	E-mail	Bij Plannen 2e ronde zou het handig zijn als bij het selecteren de 1e ronde poules niet meer zichtbaar zijn.
09-01-19	E-mail	Bij printen wedstrijdformulieren moet je in de 2e en finaleronde erop letten dat je de bladen van de voorgaande ronde(s) niet print: als dit anders kan worden ingericht
18-01-19	E-mail	Is het mogelijk om op de dia's de volgende ronde te laten zien ipv de verschillende poules?
18-01-19	E-mail	Wij houden toernooien op verschillende locaties. De één heeft fantastische WIFI, de ander slechte of zelfs geen. Is het mogelijk om het toernooi online te maken en in te vullen als offlineversie?
18-01-19	E-mail	Mocht het systeem onverhoopt haperen of offline gaan moeten wij verder op papier. Om dat op te kunnen vangen zouden wij vooraf graag het gemaakte schema uitprinten. We kunnen nu wel losse wedstrijden printen maar we zouden graag het hele schema incl. volgende rondes achter de hand hebben.
18-01-19	E-mail	Graag wilde ik voor tennis dubbel voor 8 spelers en 48 wedstrijden organiseren. Opzet is dat je steeds een andere partner hebt en steeds andere tegenstanders. Kan dit met Tournify?
23-01-19	E-mail	Is de puntdeling (dus 3 - 1 - 0) punten aan te passen? Reden hiervoor is dat de verschillende clubs strijden om een algehele prijs, maar elke club heeft wel weer een verschillend aantal teams welke meedoen, de puntdeling moet dan ook op een andere manier ingevuld worden om het zo eerlijk mogelijk te laten zijn. Het betreft een toernooi voor 3 lokale clubs, alleen Club A heeft meer teams dan Club B en Club C heeft weer meer teams dan Club A en B. Aan het eind van het toernooi worden alle punten bij elkaar opgeteld, de club met de meeste punten wint.
23-01-19	Chat	Is het ook mogelijk om dezelfde poules ook in verschillende diavoorstellingen te zetten?
27-01-19	Chat	Als ik mijn toernooi op het beginscherm van mijn iPhone wil toevoegen, bewaart hij de link naar de index.html i.p.v. mijn toernooi.
29-01-19	Chat	Goedemorgen, is het ook een mogelijkheid om de uitslagen te integreren binnen de eigen clubwebsite? Tournify lijkt ons een erg mooi systeem, maar het zou natuurlijk ideaal zijn als het binnen de eigen website geïntegreerd kan worden. Dan kunnen deelnemers en fans direct de uitslag via de eigen clubsite bekijken. Ik hoor graag wat de mogelijkheden zijn!
29-01-19	Chat	Is het mogelijk om een foto's te plaatsen bij de beschrijving die ik op de publieke website wil tonen?

3. User Stories for the Feature Request Tool



Must haves

As a Vendor, I want to obtain feature requests from users in the format of User Stories, so that they share a common structure and are easy to work with for the development team.



Should haves

As a User, I want to view a feature request in more detail, so I can interact with it.

As a Vendor, I want the feature request to be labelled with different categories, so that I can easily find related features.

Could haves

As a User, I want to filter the list of feature requests based on the category, so that I can easily find related features.

As a User, I want to filter the list of feature requests based on the status, so that I can easily see which features are in development and which features are implemented.

As a User, I want to sort the list of feature requests based on the number of upvotes, so that I can easily see which requests are popular.

As a User, I want to search for feature requests, so that I can check whether my wish has already been requested by another user.

As an Admin, I want to attach images to feature requests, so that I can demonstrate how a feature has been implemented.

As a User, I want to view feature requests from other users, so that I know which requests are already made.

een plattegrond kan delen.

IN ONTWIKKELING

18 januari 2019 • Presentatie

As an Admin, I want to assign a status to a feature request, so that the users can see if a feature is already in development or implemented.

As a User, I want to comment on feature requests, so that I have room to elaborate on the idea and bring up possible solutions.

As an Admin, I want to be able to change, delete or comment on a feature request, so that I can moderate the requests.

4. Interview protocol: functional architecture evaluation

Introduction

- Thank you for your willingness to cooperate on this interview
- This interview is being recorded
- We expect the interview to last 1 hour
- We want to cover three topics today: your previous experience with functional architectures, opinion on the quality of the reconstructed architecture and the usefulness of having an architecture in your daily workflow.

Previous experience with functional architectures

- How long have you been coding?
 - o For which clients and as which roles?
 - o What type of projects and in which languages? Which platforms?
- Are you used to work with an architectural design up front?
- Did you have any education in software architecture?
- Did you ever create an architecture yourself?
- What do you consider as a good architecture, and a good GUI?
- How are your experiences in working with or without a software architecture?
- How do you currently decide which feature belongs to which component or when to create a new one?
 - o Do you consider this process easy or difficult? Why?
- Have you seen a Functional Architecture Model before? If not, explain.
- Have you seen a Feature Diagram before? If not, explain. (SPL. Tournament License Upgrading as example)

Quality of the reconstructed architecture

- What are your thoughts on the quality of the reconstructed architecture?
- Is the modelling style comprehensible to you?

- What do you think about the decision to split-up the application into the layers: application, module, submodule, features?
 - o Are these layers detailed enough?
 - o Do you think the diagram would still be readable if we remove the submodule layer?
- Can you instantly identify the features based on their name?
 - o What does the depth of the feature diagrams tell you? (Results Processing)
 - o What does the degree of the feature diagrams tell you?
- To what extent do you see an overlap with the way your code base is organized, or does it differ a lot?
- What is missing in this architecture / what can we do to improve it?

Usefulness of architecture

- How would you use this architecture in your daily work?
- Do you think it will benefit your work?
 - o What can be the advantages of using such an architecture?
 - o What would be the disadvantages of using an architecture?
- Do you think the quality of the code based would have been better if there was an architecture upfront?
 - o Do you think having an architecture from now on will improve the quality of the code base?
 - o Do you think having an architecture makes it easier to estimate the workload of new requirements?
- What can be done to keep the architecture up to date?
 - o How can we make a link to the codebase?
- What do you think would be a good way to transfer your knowledge about the code base to another developer? Will this architecture be beneficial in transferring the knowledge?

Closing

- Do you have any remarks, comments or questions on the architecture or the interview?

5. Feature request platform evaluation questionnaire

#	Question	Type
1	What is your opinion on the possibility to submit feature requests via the Tournify Website?	5-point scale not useful at all – very useful
2	What is your opinion on the possibility to view ideas from other users?	5-point scale not useful at all – very useful
3	What is your opinion on the possibility to vote on ideas?	5-point scale not useful at all – very useful
4	What is your opinion on the possibility to comment on ideas?	5-point scale not useful at all – very useful
5	Why do you find the platform useful or not?	Open text field
6	All ideas are constructed in the form of a User Story with a clear structure (As a <role>, I want to <goal>, so that <benefit>. Did you ever write a User Story before?	Closed yes/no

6. Crowdsourced User Stories

Date	Feature Request (Dutch)	Votes	Comments
25-02-19	Als organisator wil ik het wedstrijdschema kunnen exporteren naar PDF, zodat ik het makkelijk kan printen	7	3
25-02-19	Als organisator wil ik niets. Maar wij organiseren nu een FIFA 2019 toernooi en denk dat dit heel veel gebeurt. Geen teams maar individuele spelers, mogelijk met een teamnaam, veel spelers meer dan in menig toernooi, kortere wedstrijden, geen velden maar consoles. Ik kan me voorstellen dat je Tournify daarvoor ook een template geeft of iets anders aanpast, zodat het makkelijk wordt ook e-Sports toernooien te organiseren. Denk dat het technisch nu al wel kan maar doe eens een check wat er beter kan.	1	
25-02-19	Als organisator wil ik teamfoto's toevoegen, zodat het persoonlijker wordt.	1	
25-02-19	Als organisator wil ik het account kunnen inperken om het te delen, zodat ik met meerderen een toernooi kan organiseren maar niet iedereen alle admin-rechten heeft.		
25-02-19	Als organisator wil ik beschikbaarheid van scheidsrechters beperken tot een deel van een dag of weekeinde, zodat ik tegemoet kan komen aan wensen die men heeft voor de eigen beschikbaarheid.	4	
25-02-19	Als organisator wil ik ook voordat het toernooischema gemaakt wordt, een inschrijvingssite voor een toernooi kunnen maken, zodat ik ook het aanmelden door andere teams voor het toernooi in dezelfde Tournify kan afhandelen.		
25-02-19	Als organisator wil ik als toernooi organisator als er meerdere leeftijds categorieën op hetzelfde moment spelen de scheidsrechters ook apart in kunnen plannen, dit kan nu niet alleen handmatig en niet automatisch, zodat dit sneller gebeurt is		
26-02-19	Als organisator wil ik de zekerheid dat een e-mailadres dat bij inschrijving wordt doorgegeven ook echt valide is, zodat een bevestiging ook zeker weten aankomt. Hiervoor zouden jullie gebruik kunnen maken van een third party check, zoals https://www.mailgun.com/email-verification-service , zodat we meer zekerheid hebben over de mail. Kunnen zien dat een mail geopend is, is ook wel fijn. Waarschijnlijk gebruiken jullie een mail distributie API en kun je die metrics (verzonden/geopend) gewoon in de GUI tonen aan de gebruiker.		
26-02-19	Als organisator wil ik grotere lettertypes kunnen gebruiken bij de schermen. Groter dan de huidige oplossing met H1 en H3, zodat de tekst op de schermen duidelijker is.		
26-02-19	Als organisator wil ik een betere oplossing van het huidige uitvalscherm bij planning/schema zonder dat er dubbele schuifbalken ontstaan, zodat het plannen overzichtelijker en foutlozer gaat		
26-02-19	Als organisator wil ik dat de (losse) wedstrijden op het scherm getoond worden op tijd en niet op volgorde waarin de wedstrijden zijn gemaakt, zodat er geen wedstrijden op onlogische wijze getoond worden op de schermen zodra je achteraf een wijziging gaat doorvoeren in het wedstrijd schema.		

26-02-19	Als organisator wil ik graag vaste tijdblokken met getoonde tijd en omschrijving kunnen toevoegen, zodat je ook lunch of andere activiteiten zichtbaar kunt maken voor de deelnemers in hun app.	3	
26-02-19	Als organisator wil ik de ranking kunnen tonen op de schermen, zodat alle deelnemers eenvoudig kunnen zien welke plaats ze uiteindelijk behaald hebben.		
26-02-19	Als organisator wil ik graag meer statistieken meteen zichtbaar hebben in de beheermodule in een hoofdscherm, zodat je niet continue hoeft door te klikken om betaalde aantal te achterhalen zoals aantal teams, aantal velden, aantal scheidsrechters, aantal wedstrijden etc.		
26-02-19	Als organisator wil ik graag de volgorde van de velden in het beheerscherf kunnen wijzigen of door te slepen of door het veldnummer te wijzigen waardoor de volgorde wijzigt, zodat velden die alsnog worden toegevoegd op de juiste plaats komen te staan. Of om zo tijdelijk even velden naast elkaar te zetten voor een beter overzicht.	2	
26-02-19	Als organisator wil ik graag memovelden voor intern gebruik, zodat je belangrijke info kunt opslaan of delen met anderen die toegang hebben tot de beheermodule.		
26-02-19	Als organisator wil ik graag het symbool voor verplaatsen van deelnemers naar andere teams laten vervangen door de tekst: verplaats ipv een pijltje die wijst naar de prullenbak, zodat duidelijker is dat dit een functie is van verplaatsen van deelnemers en het niet lijkt dat het is voor het verplaatsen naar de prullenbak.		
26-02-19	Als organisator wil ik graag de optie toevoegen om te kunnen kiezen voor een achtergrondkleur ipv het toevoegen van een achtergrondplaatje in die kleur, zodat op alle schermen en devices de juiste achtergrondkleur getoond wordt en je niet kunt scrollen voorbij een achtergrondaafbeelding.		
26-02-19	Als organisator wil ik dat fase 1 niet meer getoond wordt zodra de volgende fase begint, zodat deelnemers niet gedwongen worden om zelf te klikken op de volgende fase.		
26-02-19	Als organisator wil ik dat bij de sheets voor de schermpresentaties start en eindtijden worden toegevoegd, zodat je kunt zorgen dat een sheet alleen op de gewenste tijdstippen wordt getoond zoals bijvoorbeeld voor het toernooi de welkomstboodschap of bijv tijdens lunchtijd.	3	
26-02-19	Als organisator wil ik de dia's met een sleepfunctie of wijsbare volgordenummering kunnen laten wijzigen van volgorde, zodat je de presentatie niet volledig opnieuw hoeft te maken als er een extra dia tussenkomt.	5	
26-02-19	Als organisator wil ik graag de mogelijkheid om logo's toe te voegen van de deelnemende ploegen. Dit zoals het vlaggetje voor de deelnemersnaam maar dan hun eigen logo, zodat de logo's ook mooi kunnen getoond worden in de wedstrijdschema's en standen	9	2
26-02-19	Als organisator wil ik de mogelijkheid om de ploegen zelf als scheidrechter te laten fungeren en dus mee te laten nemen in de berekening van de schema's, zodat er geen wedstrijden doorgaan waarvan de ploegen op dezelfde moment moeten arbitreran	3	1

26-02-19	Als organisator wil ik een opleesschema per ronde de wedstrijden per veld, zodat de wedstrijdsecretaris het schema alleen hoeft op te lezen zonder nadenken	3	1
27-02-19	Als organisator wil ik graag de wedstrijden in een poule van 4 en 5 anders ingepland hebben. Ivm uit en thuis wedstrijden poule van 4 starten met 1-2 dan 3-4, 3-1, 4-2, 1-4, 2-3. In de poule van 5 starten met 2-1, 4-3, 5-1, 3-2, 4-5, 1-3, 5-2, 1-4, 3-5, 2-4, zodat teams dan uit en thuis wedstrijden hebben. bv eerst genoemde heeft bal uit of thuis team start aan altijd aan de rechterkant.		
27-02-19	Als organisator wil ik dat je bij een toernooi de mogelijkheid hebt dat bv. De beste 2es doorstoten automatisch, zonder dat je dit manueel moet natellen en aanpassen. Dit is handig bij toernooien met minder deelbare aantallen en zo kan je vlotter overschakelen naar de volgende fase, zodat je vlotter kan overschakelen naar de volgende fase		2
28-02-19	Als organisator wil ik als een wedstrijd niet doorgaat zou ik die zelf naar een andere datum willen schuiven, zodat dit proces soepeler verloopt.		
28-02-19	Als organisator wil ik graag kunnen werken met sets. Bij volleybal is het gebruikelijk om twee sets te spelen. Nu is de workaroud dat we twee wedstrijden inplannen maar het zou prettig zijn om 1 wedstrijd te voorzien van 2 sets, zodat wedstrijden overzichtelijker in kaart gebracht kunnen worden en de eindstanden realistischer worden. Ons volleybaltoernooi werkt niet met doelsaldo maar met punten op basis van de set-uitslagen.	1	
03-03-19	Als organisator wil ik een import functie, zodat ik bepaalde gegevens van het ene toernooi naar het volgende kan brengen. Bijv. Terreinen, leeftijdscategorie, indelingen, etc...		
04-03-19	Als organisator wil ik de wedstrijdduur aanpassen per divisie ipv algemeen per dag, zodat je divisies met minder ploegen langere wedstrijden kan laten spelen, dan de divisies met veel ploegen.	8	
07-03-19	Als organisator wil ik Sinds kort kan je ook via de website inschrijven! Uitstekende toevoeging! Graag zou ik dan zien de mogelijkheid tot het tonen van bijvoorbeeld foto's van de vorige editie als mensen op de site komen! Hij is nu zo kaal, zodat Mooiere presentatie!		
08-03-19	Als organisator wil ik kunnen zien hoeveel bezoekers mijn publieke website heeft gehad, zodat ik kan zien of deze vorm echt aanslaat en publiek ook de website gebruikt.	9	
11-03-19	Als organisator wil ik graag een mogelijkheid zien dat een deelnemer kan meedoen aan verschillende onderdelen. Bijvoorbeeld Open en 40+ categorie. Nu moet ik de deelnemers bij elke categorie toevoegen waardoor het eigenlijk andere personen zijn. Er zou dan ook een check kunnen zijn of spelers niet tegelijk moeten spelen in beide categorieën, zodat ik blijer wordt?	1	
12-03-19	Als organisator wil ik een kleedkamer overzicht kunnen maken, zodat men bij de ingang al weten welke kleedkamer ze hebben.	3	
12-03-19	Als organisator wil ik de optie/vakje om een cijfer voor sportiviteit op het geprinte wedstrijdkaartje in te vullen. Dit gebruiken we bij een schoolvoetbaltoernooi, zodat de scheidsrechters herinnerd worden dit in te vullen en we het niet handmatig op de wedstrijdkaartjes hoeven te zetten. Daarnaast lijkt het een kleine/makkelijke aanpassing ;-)	3	

14-03-19	Als organisator wil ik graag de mogelijkheid dat de poules automatisch worden ingedeeld door een knop, zodat je meteen kan starten met het toernooi		
15-03-19	Als organisator wil ik graag de mogelijkheid om de spelers en coach van ieder team te delen via Tournify, zodat alle deelnemers en vrijwilligers de indeling van de teams online (ipv als bijlage of op papier) kunnen zien	1	1
15-03-19	Als organisator wil ik graag de mogelijkheid om een overzicht van de velden als visual toe te voegen (ipv bijlage), zodat teams en coaches makkelijk 'hun' veld kunnen vinden	1	
19-03-19	Als organisator wil ik graag een link kunnen geven aan een team, zodat ze direct hun wedstrijdschema kunnen zien ipv via de website meerdere keuzes maken (dus een link van een zoekopdracht)	6	1
19-03-19	Als organisator wil ik graag grotere wedstrijdbriefjes en dan zo dat deze goed verdeeld zijn over het papier, zodat dit er beter uitziet (wat betreft formaat) en hetzelfde formaat heeft na het snijden/knippen	1	
19-03-19	Als organisator wil ik graag de mogelijkheid hebben om enkele wedstrijden te laten vervallen als deze automatisch zijn ingedeeld, zodat ik niet alle wedstrijden apart naar het veld moet slepen	1	
19-03-19	Als organisator wil ik graag de mogelijkheid om de teams nog een keer in te zetten in een losse wedstrijd, zodat deze opnieuw kunnen worden ingedeeld op bijvoorbeeld een spelelement die buiten de competitie of toernooi poule valt	1	1
19-03-19	Als organisator wil ik graag makkelijker 2 poules over 3 velden verdelen, zodat dat minder tijd kost. Als je nu de twee velden automatisch vult met de wedstrijden, kun je niet een wedstrijd naar het 3e veld slepen	2	
19-03-19	Als organisator wil ik graag de mogelijkheid dat de deelnemende teams ook automatisch scheidsrechter zijn, zodat dit niet dubbel hoeft te worden ingevuld. Kanttekening: dan krijg je met een zoekopdracht zowel de wedstrijden van het team als ook de te fluiten wedstrijden te zien in een overzicht. Dit kan worden voorkomen door bijvoorbeeld een teken voor de teamnaam te plaatsen van de scheidsrechter. Bijvoorbeeld 'Voetbalclub A' en '#Voetbalclub A'	3	1
19-03-19	Als organisator wil ik graag een aanpassing zien op de maten van de kolommen op het speelschema op de publieke site en op de diavoorstelling, zodat de namen van beide scheidsrechters zichtbaar zijn	1	
23-03-19	Als organisator wil ik dat het mogelijk is om een wedstrijd live te streamen via twitch of youtube. Dat kan al via andere programma's maar ik zou graag zo'n balkje linksboven in de hoek hebben met de tijd erop en de punten, zodat ik professioneel kan livestreamen.		
25-03-19	Als organisator wil ik graag een knop zien waarbij je alle wedstrijden naar niet gepland plaatst, zodat je vanuit daar een bestaand schema kunt gaan vullen?		
25-03-19	Als organisator wil ik graag een knop waarmee ik alle geplande scheidsrechter verwijder, zodat ik met een schone lei kan beginnen met indelen?	1	

25-03-19	Als organisator wil ik graag de mogelijkheid hebben om per dag een aanvangstijd te definiëren. Nu dus opgelost met toevoegen van pauzes, zodat het intuïtiever werkt.	2	
27-03-19	Als organisator wil ik dat ik bij het uitlopen van het tijdschema een pauze voor alle velden kan toevoegen om de nieuwe starttijden voor ronde 2 gelijk te trekken, zodat de tijden weer corresponderen met de werkelijkheid		
27-03-19	Als organisator wil ik hij het plannen van de 2e ronde optioneel bij het selecteren de 1e ronde poules zichtbaar maken (standaard niet tonen is praktischer), zodat plannen nog makkelijker wordt.		
27-03-19	Als organisator wil ik graag het formaat wedstrijdbriefje 1 per A5 (liggend) kunnen instellen, zodat we naast 6 per A4 (staand) of 1 per A4 (staand) deze extra keuze hebben.		
27-03-19	Als organisator wil ik graag teams plannen in een groter vakje. Het vakje waarin je teams moet toevoegen om de planning te kunnen maken is klein, zodat je niet steeds hoeft te scrollen ('frame in frame'). Het zou handig zijn als dat groter kan.	1	
27-03-19	Als organisator wil ik graag extra product kunnen opvoeren om te kiezen in het bestelproces, zodat bijv. consumptiebonnen / aanmeldkosten / bbq / toernooi-shirt e.d. meteen bij de inschrijving aangekocht en betaald kunnen worden	1	
28-03-19	Als organisator wil ik graag de wedstrijdbriefjes per scheidsrechter kunnen selecteren, zodat je per scheidsrechter een overzicht van zijn wedstrijden hebt		
28-03-19	Als organisator wil ik een 'herstel' functionaliteit, zodat ik bij foutjes weer naar de vorige setting kan herstellen	1	1
30-03-19	Als organisator wil ik extra activiteiten/spellen kunnen toevoegen aan het toernooi waarbij deelnemers punten kunnen scoren, zodat je deze activiteiten en scores ook weer kunt zien in de tournify app op je telefoon als deelnemer. Je ziet in je schema hoe laat je een andere activiteit hebt en je kunt de resultaten van iedereen zien in deze spellencompetitie.		

Paper

User Story Writing in Crowd Requirements Engineering: the case of a web application for sports tournament planning

Abel Menkveld, Sjaak Brinkkemper, and Fabiano Dalpiaz
Utrecht University, The Netherlands

Abstract—Although users feel more engaged when they are involved in the elicitation, negotiation and prioritization of requirements for a product or service they are using, the quality of crowdsourced requirements remains an issue. Simple formalisms like user stories have been highly adopted by practitioners in agile development to capture requirements for a software product, but utilization in crowdsourced requirements engineering seems scarce. Through a case study of a web application for sports tournament planning, we demonstrate how a dedicated platform for user story writing in crowd requirements engineering is valued by its users and delivers high-quality requirements that are not inferior to those written by professionals.

I. INTRODUCTION

The process of extracting informal stakeholders' needs and translating them into formal specifications is the core principle of Requirements Engineering (RE). These requirements are used as an input for software development. More specifically: they serve as the basis for project planning, risk management, trade off, acceptance testing and change control [1]. Clear statements of requirements are one of the project's success factors, but at the same time incomplete requirements are the number one reason why projects are impaired [1].

Together with the shift from traditional (waterfall) development to agile software development, the RE processes changed accordingly. This was necessary because traditional requirement activities – elicitation, analysis and negotiation, documentation, validation, and management – do not take the iterative processes of agile software development into account. However, agile RE does not only alleviate challenges of traditional RE, but also poses new ones. Minimal documentation, customer inability, and time estimation are reported as some of the challenges of agile RE [2]. A proper use of artifacts can overcome these.

In this study we focus on one of those type of RE artifacts: user stories (USs). USs are applied by over half of the practitioners in the software industry to capture requirements and there is tight coupling of USs with agile methods [3]. A US is “a description of a feature written from the perspective of the person who needs this” [4]. The written text is a semi-structured natural language statement. The most widespread format of a US is: “As a <role>, I want <goal>, so that <benefit>”, as used in the following example [5]:

As an administrator, I want to receive an email when a contact form is submitted, so that I can respond to it.

Next to the use of simple formalisms like USs to capture requirements, user involvement is vital in RE. Involving users in RE can not only improve system acceptance, diminish project failure, and deliver greater system understanding by the user; it also helps to improve customer loyalty and broaden the market [6]. Therefore, crowdsourced RE has been investigated by a series of studies [7]. For example, a research group from RWTH Aachen University developed *Requirements Bazaar*, an open source web-based platform for crowd-based RE [8]. Snijders et al. [9] advocate Crowd-Centric RE, by combining crowdsourcing and gamification to involve users in the elicitation, negotiation and prioritization of requirements. According to the researchers, this helps fostering user involvement, is valuable in all stages of RE and gives equal priority to both customers and end users when they are not the same. The embodiment of their vision is REfine, a gamified platform for eliciting and refining requirements. Dalpiaz et al. [6] showed in a case study how users perceived this crowdsourcing platform as more useful and more engaging compared to previous feedback experiences. However, they were worried that the quality of requirements would not match the quality resulting from experts' methods, and the requirements may not be detailed enough for a focus group or product backlog. This is an interesting observation, since one of the main incentives to involve the crowd in software development in general [10] and RE in specific [7] is to achieve higher quality. It can be a challenge or limitation at the same time [11], [12], [13], [8]. It is argued that simple formalisms such as USs may improve the quality of crowdsourced requirements and can therefore mitigate this risk [6], but to the best of our knowledge no study has yet been performed on US writing by crowd workers.

Therefore, in this paper we will demonstrate how a crowdsourced RE platform can be employed to enable crowd workers to express requirements in the form of USs. We implemented and validated the platform in the case of a web application for sports tournament planning.

The paper is written in the following structure. Section II describes the case study, covering both the company and its existing practices regarding RE. In Section III, the crowdsourced RE platform is presented and the evaluation protocol is described. The results will be listed in Section IV and discussed in Section V, in which we also present directions for further research.

II. CASE

The single-case study we performed entails the design and validation of a crowdsourced RE platform using a Technical Action Research.

A. The company

Tournify is a software development company based in Amsterdam, The Netherlands. Their services, which are provided through an online application, are targeted at sports and e-sports tournament and competition organizers. The main product is the Tournify tournament manager. This web application allows tournament organizers to manage participants, create a match schedule based on a chosen tournament format and process the results as the tournament progresses. The organizer can also use the tournament manager to create a tournament website to present the event to the audience. The athletes and supporters are able to view the schedule, results and standings by visiting this tournament website or by looking at a big screen, as new information comes in real time.

Tournify is written in Javascript. It uses React, an open-source JavaScript library developed by Facebook, for the creation of the interactive user interfaces. For the dynamic content Firebase is used: a mobile and web development platform maintained by Google that allows storing and syncing data across multiple clients. The total lines of code (LOC) is near 25.000 and around 110 components are used. They serve over 10.000 registered users (tournament organizers) and host over 25.000 created tournaments since the website¹ became publicly available in late 2017.

B. The case

The company deals with the handling of customers' feature requests. In five-months' time 44 unique customers requested 77 new features. Most of them are requested via email (57%) or via the support chat (38%). The requesters organize tournaments in sixteen different sports and two different e-sports. Tournify can be used free of charge, which 39% of the requesters did. The other 61% of the requesters upgraded at least one tournament to one of the paid packages, a requisite to host tournaments with more than eight participating teams.

Further analysis of the requests and conversion of the texts to USs caused no difficulties in 71% of the cases: the role was clear, a goal was expressed, and the potential benefit was highlighted. In the other cases, the benefit was not explicitly mentioned. Although this benefit is optional in a US, it may provide valuable information, especially when a request comes in without any context via email or chat. Consider the following request that came in:

Why is the number of teams I can add to a group limited? I want to place 46 teams into one group.

The corresponding US would be:

¹www.tournifyapp.com

As an organizer, I want to place 46 teams into one group.

This is a valid US but raises questions since it is unknown why one wants to place this many teams into one group: even the biggest leagues in the world have place for a maximum of 20 teams. Only after further communication between the product manager and requester, it becomes clear the user does not want to place 46 teams into one group, he wants to host a tournament with different games (currently Tournify is built to host tournaments for a single sport). Rather than focusing on making the workaround possible, this user (and most likely, many others) will benefit a lot more if a dedicated feature is developed to host multi-sports tournaments.

This lack of context information is one of the reasons that makes the current workflow time-consuming for the product owner. Responding to the feature requests, even if they are clearly stated, also takes time. And as the business grows, the number of requests will increase. Another downside of the current workflow is that it does not allow for proper requirements prioritization and does only involve a small subset of the users.

III. CROWDSOURCED RE PLATFORM FOR US WRITING

We designed a crowdsourced RE platform, integrated into Tournify, which enables users of the software application to submit feature requests in the strict format of USs - containing a role, goal and benefit. In order to help users formulate these stories, even if they have never seen or heard of a US before, we use a form with four simple self-explanatory and small steps (Figure 1).

The figure shows four sequential screenshots of a web form titled 'New idea' with a progress indicator 'Step 1 of 4' through 'Step 4 of 4'.
Step 1: 'How do you use Tournify?' with three radio button options: 'I organize a tournament' (selected), 'I participate in a tournament', and 'I support a team in a tournament'. A 'TO STEP 2' button is at the bottom right.
Step 2: 'What would you like to do with Tournify?' with a text input field containing 'I want to export the match schedule to PDF'. 'STEP BACK' and 'TO STEP 3' buttons are at the bottom.
Step 3: 'Why do you want to export the match schedule to PDF?' with a text input field containing 'So that I can print it out and have it as a backup in case the connection is lost.' 'STEP BACK' and 'TO STEP 4' buttons are at the bottom.
Step 4: 'Verify your idea, choose a category and submit it.' with a list of categories: 'General', 'Participants', 'Format', 'Schedule', 'Presentation', 'Results', and 'Other'. The 'Results' category is selected. 'STEP BACK' and 'SUBMIT IDEA' buttons are at the bottom.

Fig. 1. The four steps to formulate a requirement as a US

Step 1: Role The first step is to find out in which role the requester uses the application. The user can select one of the roles from the predefined options using radio buttons. In the case of Tournify, three roles are defined: organizer, participant and supporter.

Step 2: Goal In the second step the user is asked what he or she wants to do with Tournify: a feature that is missing. The

textbox contains static text before the user input, containing the phrase ‘I want to’.

Step 3: Benefit The users’ text entry recurs in the formulation of the question in the third step. It is explicitly asked why the user wants to have the requested feature, to know what the user sees as the potential benefit when the feature would be implemented in the software application. The answer always contains the predefined ‘so that’ at the start.

Step 4: Verification and category selection Before submitting the idea, the user is able to verify the US that has been formulated based on the answers he or she provided in the first three steps. The user also has to select one of the predefined categories. These categories are part of the main menu of the application, so the users are already familiar with the terms. Labeling the requests with the corresponding category allows for easy categorization later on.

All requests are published on a feature request overview page in the Tournify web app, which can be accessed via the support menu. This is no static page, as requirements elicitation is not the only goal of the platform. The second goal is to negotiate and prioritize requirements utilizing the crowd. This is done by two simple means: voting and commenting. For the requirements prioritization, several techniques exist. We follow the general advice of Maiden & Ncube [14], also advocated by Berander & Andrews [15], to use the simplest appropriate prioritization technique. This is especially true in crowd-centric requirement engineering [6], since end-user crowd workers are likely to be less experienced with requirement prioritization than product managers. The prioritization technique should also allow for easy reprioritization, as requirements will be added, changed or deleted continuously. We used the Confirmation or Negation feedback type, in which users agree or disagree on problems or opinions of other users [16]. This feedback type is also used in the Requirements Bazaar [8] and REfine [12] platforms. Lastly, a commenting section enables users and product managers to respond or add suggestions to the requests.

The crowdsourcing platform was deployed and announced on February 25th, 2019. The announcement was sent via an email to a selected group of 337 users (63% opened). These users had either requested a feature in the past, subscribed to the newsletter, or made a purchase recently. A reminder was sent one month later (55% opened). The total data collection period was five weeks, so all requests submitted after March 31st, 2019 are not included in this research. Among all requesters, voters and commenters, one free tournament upgrade has been raffled. Users were also informed of the feature request platform via a snack bar message which was shown when opening the Tournify tournament planner. The researchers initiated the first request and commented on some of the requests during the study. They were also able to label features as *in development* or *done*. This first request by the researchers will be included in the report on the number of requests, because users were able to comment and vote on this request. However, it is not further evaluated regarding the

TABLE I
THE EIGHT CRITERIA TO ASSESS USs INDIVIDUALLY FROM THE QUALITY US FRAMEWORK [5].

Criteria	Description
<i>Syntactic</i>	
Well-formed	A US includes at least a role and a means
Atomic	A US expresses a requirement for exactly one feature
Minimal	A US contains nothing more than role, means, and ends
<i>Semantic</i>	
Conceptually sound	The means expresses a feature and the ends expresses a rationale
Problem-oriented	A US only specifies the problem, not the solution to it
Unambiguous	A US avoids terms or abstractions that lead to multiple interpretations
<i>Pragmatic</i>	
Full sentence	A US is a well-formed full sentence
Estimatable	A story does not denote a coarse-grained requirement that is difficult to plan and prioritize

quality and complexity criteria we discuss below. Comments from the researchers are excluded from the results.

Perceived Usefulness After the data collection process, the users who submitted an idea received an extra email with a link to a short questionnaire. This questionnaire tests the perceived usefulness of the platform from an end user perspective. It contains four questions with a five-point Likert scale, asking the users to evaluate the usefulness of each functionality of the platform: requesting, viewing, voting and commenting. One closed question is included to verify if the requester had experience with formulating USs before, and one open text field can be used to comment on the experience with the platform.

Quality The Quality US framework [5] was used to assess the USs individually based on their syntactic, semantic and pragmatic quality. The eight criteria and their descriptions are shown in Table I. Each US was evaluated on its quality manually by three experts individually. The experts used the description of the criteria from Table I, as well as the additional information from the accompanying article, to analyze the USs. The USs were distributed among six members of the RE Lab at Utrecht University. They analyzed one third of the USs each and the lead researcher analyzed all USs. If there was no consensus in the judgement of the experts, majority voting is leading.

Complexity We made an estimation of the amount of work it would take to implement each US individually, based on the assessment of the lead developer of Tournify. For the scaling the Fibonacci sequence (1, 2, 3, 5, 8, 13, 21) was used. We assigned a value of ‘0’ when it concerned a feature that has already been implemented but overlooked by the requester. The other numbers represent development hours. Since it is difficult to estimate large work items with a high degree of confidence, the upper limit for our estimation was 21 hours. In practice, USs who take more than 21 hours to implement have to be broken down into more granular pieces.

TABLE II
USE OF THE CROWDSOURCED RE PLATFORM

Value	Total	Unique users
Page views	247	157
Interactions	160	39
Requests	57	23
Votes	89	28
Comments	14	9

IV. RESULTS

During the five-weeks period, we had 157 unique visitors on the feature request platform. From those visitors, 39 users interacted with the platform by submitting an idea (23), voting on an idea (28), and/or commenting on an idea (9). Together, they submitted 57 ideas, voted 89 times and commented 14 times (Table II). The functionality to downvote an idea ('I don't need this') was not used and in five times a requester voted on its own idea, which was not prevented by the platform.

More than half of the requesters (15, 65%) submitted only one idea, two users submitted respectively two and three ideas and four users submitted five or more ideas (respectively 5, 6, 7, and 14 ideas). All ideas are written in Dutch and constructed based on the template of a US. A screenshot of part of the Feature Requests overview page is shown in Figure 2.

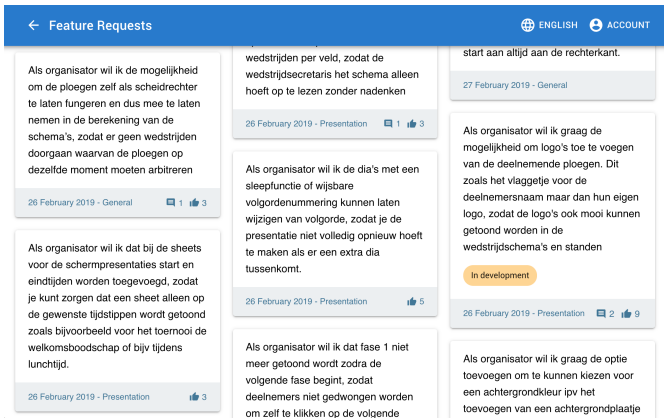


Fig. 2. Screenshot of the feature request overview page

Next to the feature description, each element also contains the submission date and selected category. If applicable, the element also contains the number of votes, number of comments, and development status. Two feature requests got nine upvotes, which is the most times a feature has been upvoted. The categorization of USs turned out to be a difficult task for the crowd workers, judging by the numbers. In more than half of the cases (52%), the category selection of the requester does not match the category assignment done by the main researcher of this study.

After the study period, thirteen users who interacted with the crowdsourcing platform responded to the questionnaire that was sent to them via email. Most of them (10) requested a

TABLE III
QUALITY OF THE CROWDSOURCED USs

criterion	# USs with defect	% USs with defect
Well-formed	3	5.4
Atomic	5	8.9
Minimal	24	42.9
Conceptual	5	8.9
Problem-oriented	8	14.3
Unambiguous	9	16.0
Full sentence	19	33.9
Estimatable	3	5.4

feature themselves, the other three respondents only voted for a feature. They perceived the platform as very useful, regarding all four possible interactions when rated on a five-point Likert scale: requesting ($M = 4.9$; $SD = 0.28$), viewing ($M = 4.8$; $SD = 0.38$), voting ($M = 4.5$; $SD = 0.88$), and commenting ($M = 4.5$; $SD = 0.66$). One user who requested a feature, voted for and commented on an idea and had previous experience in writing USs commented:

“You implemented the agile methodology in a very fun way. In such a manner the users get involved better and at least have the feeling their opinion matters”

Others found it *“a fantastic way to improve the application”*, *“very useful to allow users to submit ideas”* and see it as a way to *“improve the software for your own tournament”*. Another user noted how *“every user gets new ideas while using Tournify on their tournament”* and how this is *“the best feedback to improve the application”*.

Out of the people who requested a feature, 70% have never written a US before. When asked if they find it helpful to formulate the ideas as USs, compared to free texts, the average score was 3.5 ($SD = 0.85$). There is hardly any preference to write the feature requests in free text ($M = 3.2$; $SD = 1.14$).

The results of the quality analysis are shown in Table III. In total, 52% of the USs meet all requirements, meaning that 48% of the USs contains one or more easily preventable error(s). A Pearson Chi-Square test showed that there is a strong association between the minimal and full sentence criteria, which is statistically significant ($X^2 = 31.6$, $p = < .001$). This might provide an explanation for the higher number of USs with two defects (11%), compared to those with only one defect (7%).

The crowdsourced USs are evaluated based on their complexity by the developer of Tournify. Nine out of ten crowdsourced USs can be developed within one workday, according to his estimation. One US could not be estimated, because it was formulated to vaguely. Seven USs were already implemented but overlooked by the user. They are not included in the estimation shown in Figure 3, which includes 48 USs.

V. ANALYSIS & DISCUSSION

Our results show that the use of crowdsourcing in RE is perceived as very useful by the end-users of a software product, while at the same time empowering the product owner

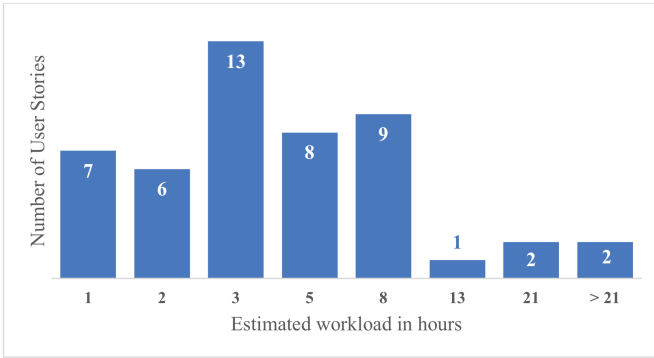


Fig. 3. Complexity of the crowdsourced USs

with a better overview of feature requests. Commenting and voting on ideas is not only valued by the crowd workers, but also helps in the prioritization and negotiation process. Interestingly, our four-step US formulation wizard was not perceived as an extra difficulty by the users while expressing feature requests: there is hardly any preference to submit ideas in free text instead. This is promising, as research has shown how “stakeholders enjoy working with USs, using a common template benefits RE and the simple structure of USs enables developing the right software” [3]. Furthermore, we have shown how 95% of the crowdsourced USs are both easy to estimate and easily implementable based on our quality analysis and hour estimation as done by the main developer. Almost 90% of the feature requests can even be implemented within one workday.

In terms of US quality, the most frequent occurring defects are USs violating the minimal criterion (42.9%) or not being written as one full sentence (33.9%). Lucassen et al. [8] tested 1000+ USs written by professionals from different companies and found that the minimal criterion is violated in 13.3% of the cases. Based on this observation we can conclude that crowdsourced USs are currently over three times more likely to contain comments, descriptions of the expected behavior, or testing hints, when compared to those written by professionals. This additional information should be left to the comment section of the platform. The violation of the minimal criterion is also reflected in the length of the crowdsourced USs. The *goal* is expressed in 108 characters on average and crowd workers needed 97 characters on average to formulate the *potential benefit*. When compared to 551 real-world English USs from eight different projects, retrieved from a publicly available data set [17], we found the *means plus end* of the Dutch crowdsourced USs (204 characters) to be over two times longer than the USs written by professionals (97 characters), which had an average *goal* description of 51 characters and *benefit* expression in 55 characters, if present. The length of the crowdsourced User Stories is similar to the length of the feature requests that were sent in by email or the support chat (192 characters) prior to the deployment of the platform. Note that we did not count the terms from the

US format (*I want to* and *so that*) and did not control for the information density of the different languages the USs are written in. Moreover, 17% of the real-world USs lack a description of its *benefit*. There is also a major difference in the use of *roles*. Three roles are defined for the Tournify application (organizer, participant, supporter). All requesters indicated they are organizers, whereas professionals use 12 *roles* on average in their US set.

The crowdsourced USs and USs written by professionals show a similar number of defects regarding the well-formed criterion (5.4% crowd, 4.5 professionals) and atomic criterion (8.9% crowd, 10.3% professionals). In total, 52% of the crowdsourced USs meet all requirements, meaning that 48% of the USs contains one or more easily preventable error(s). Lucassen et al. [8] conclude that 56% of USs written by professionals have at least one defect as detected by their automatic testing tool. However, these results are difficult to compare as Lucassen et al. [8] tested against less, but different, criteria from the framework than we did.

Based on our results, we see opportunities for improving the crowdsourced RE platform to enhance the quality of the USs. Defects on the minimal and full sentence criteria can be prevented with simple means like a spelling checker and warnings when there is additional text after a dot, hyphen, semicolon, or other separating punctuation marks. Text between brackets should also trigger a warning message on the screen. With this case study, however, we already demonstrated how a dedicated platform for US writing in crowd RE is valued by end-users and delivers requirements that do not inferior to those written by professionals.

During the five-weeks testing period, 17 features were requested via email or the support chat, bypassing the feature request platform. In most cases, those users were unaware the platform existed. When compared to the engagement prior to the deployment of the platform, while correcting for the duration of the measurement, the number of ideas that were sent in every day remained unchanged. However, since the number of organizers using the service increased with over 175%, this saves the requirements engineer currently an estimated 2 hours of work per month. This estimation is based on the 10 minutes it takes the requirements engineer to process each request, and the decrease in the number of requests via email or chat per 1000 unique page views from 13.6 to 7.2. Although this time saving seems currently insignificant, it will have an impact when the business grows. Nevertheless, the main incentive to employ a crowdsourced requirements platform should be to engage users and gather, prioritize and negotiate high-quality requirements, rather than to replace the work of the requirements engineer.

A. Validity threats

The main concern regarding the generalizability of this study is that we focused on a single case. However, the nature of this study is explorative and this is one of the first attempts to let crowd workers write USs for a software product.

The personal involvement of the first author (who is one of the co-founders of Tournify) allows for an unbounded access to the development artifacts and stakeholders, and a comprehensive knowledge of the organization and business processes. At the same time, it raises relevant questions about possible biases and prejudices. Action Research in general has been criticized for its “*lack of methodological rigor, its lack of distinction from consulting and its tendency to produce either research with little action or action with little research*” [18]. To ensure the rigor and relevance of this study, we made sure the five principles of Canonical Action Research (CAR) were taken into account at all stages of the research. This set of principles and associated criteria are developed by Davison, Martinsons & Kock in 2004, to allow for a study in which organization problems are addressed while at the same time contributing to scholarly knowledge [18]. However, it is still possible that the results are influenced because respondents modified their responses because they knew they were part of a study (known as the Hawthorne effect).

The biggest limitation is the small sample size. Over half of the crowdsourced USs is written by four users. This means that their expertise highly influenced the overall results regarding the quality evaluation, even though we can argue that more software products will have a group of highly engaged users with presumably more technical expertise.

B. Future research

There have been several studies focusing on USs recently. Also crowdsourcing in RE gained attention. However, to the best of our knowledge this is the first study to combine the two interest fields, by focusing explicitly on US writing by crowd workers. We can continue this work by implementing direct feedback techniques during the US formulation to improve the syntactic quality. Further research can also focus on the usefulness of having a *role* in USs written by crowd workers, can focus on the interplay between Epic(s) (Stories) and USs, and possibility to combine crowdsourcing with crowdfunding in the development of new features. The expertise of crowd workers, in relation to their involvement with the software product is also worth investigating in further research.

REFERENCES

[1] J. Dick, E. Hull, and K. Jackson, *Requirements engineering*. Springer, 2017.

[2] I. Inayat, S. S. Salim, S. Marczak, M. Daneva, and S. Shamshirband, “A systematic literature review on agile requirements engineering practices and challenges,” *Computers in human behavior*, vol. 51, pp. 915–929, 2015.

[3] G. Lucassen, F. Dalpiaz, J. M. E. van der Werf, and S. Brinkkemper, “The use and effectiveness of user stories in practice,” in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2016, pp. 205–222.

[4] E.-M. Schön, J. Thomaschewski, and M. J. Escalona, “Agile requirements engineering: A systematic literature review,” *Computer Standards & Interfaces*, vol. 49, pp. 79–91, 2017.

[5] G. Lucassen, F. Dalpiaz, J. M. E. van der Werf, and S. Brinkkemper, “Improving agile requirements: the quality user story framework and tool,” *Requirements Engineering*, vol. 21, no. 3, pp. 383–403, 2016.

[6] F. Dalpiaz, R. Snijders, S. Brinkkemper, M. Hosseini, A. Shahri, and R. Ali, “Engaging the crowd of stakeholders in requirements engineering via gamification,” in *Gamification*. Springer, 2017, pp. 123–135.

[7] K. Mao, L. Capra, M. Harman, and Y. Jia, “A survey of the use of crowdsourcing in software engineering,” *Journal of Systems and Software*, vol. 126, pp. 57–84, 2017.

[8] D. Renzel, M. Behrendt, R. Klamma, and M. Jarke, “Requirements bazaar: Social requirements engineering for community-driven innovation,” in *2013 21st IEEE International Requirements Engineering Conference (RE)*. IEEE, 2013, pp. 326–327.

[9] R. Snijders, F. Dalpiaz, M. Hosseini, A. Shahri, and R. Ali, “Crowd-centric requirements engineering,” in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE, 2014, pp. 614–615.

[10] K.-J. Stol and B. Fitzgerald, “Two’s company, three’s a crowd: a case study of crowdsourcing software development,” in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 187–198.

[11] M. Cohn, *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.

[12] R. Snijders, F. Dalpiaz, S. Brinkkemper, M. Hosseini, R. Ali, and A. Ozum, “Refine: A gamified platform for participatory requirements engineering,” in *2015 IEEE 1st International Workshop on Crowd-Based Requirements Engineering (CrowdRE)*. IEEE, 2015, pp. 1–6.

[13] S. L. Lim, D. Damian, and A. Finkelstein, “Stakesource2.0: using social networks of stakeholders to identify and prioritise requirements,” in *2011 33rd International Conference on Software Engineering (ICSE)*. IEEE, 2011, pp. 1022–1024.

[14] N. A. Maiden and C. Ncube, “Acquiring cots software selection requirements,” *IEEE software*, vol. 15, no. 2, pp. 46–56, 1998.

[15] P. Berander and A. Andrews, “Requirements prioritization,” in *Engineering and managing software requirements*. Springer, 2005, pp. 69–94.

[16] N. Sherief, W. Abdelmoez, K. Phalp, and R. Ali, “Modelling users feedback in crowd-based requirements engineering: An empirical study,” in *IFIP Working Conference on The Practice of Enterprise Modeling*. Springer, 2015, pp. 174–190.

[17] F. Dalpiaz, “Requirements data sets (user stories),” Mendeley Data, v1, 2018, <http://dx.doi.org/10.17632/7zdk8zsd8y.1>.

[18] R. Davison, M. G. Martinsons, and N. Kock, “Principles of canonical action research,” *Information systems journal*, vol. 14, no. 1, pp. 65–86, 2004.