Agile Requirements Engineering with Prototyping: A Case Study

Marja Käpyaho
Aalto University School of Science, Futurice Oy Finland
marja.kapyaho@iki.fi

Marjo Kauppinen
Aalto University School of Science Finland
marjo.kauppinen@aalto.fi

Abstract—The rise of agile software development methods has led to the abandonment of many traditional practices especially in requirements engineering (RE). Agile RE is still a relatively new research area and the growing use of agile methods in large projects is forcing companies to look for more formal practices for RE. This paper describes experiences gained from a case study of a large agile project. The goal of this case study was to explore how prototyping can solve the challenges of agile RE. Our findings indicate that while prototyping can help with some challenges of agile RE such as lack of documentation, motivation for RE work and poor quality communication, it also needs complementary practices to reach its full potential. These practices include reviewing the big picture regularly, keeping track of quality requirements, and using ATDD (Acceptance Test Driven Development).

Index Terms: agility, requirements engineering, prototyping, RE process, good practices

I. INTRODUCTION

Agile software development methods have become increasingly popular in the past decade by claiming lower costs, better productivity, better quality and better customer satisfaction [1]. According to Savolainen et al. [2], the promises of improved efficacy have led to a wide adoption of agile development practices. The advocates of agile philosophy see traditional software development methods as too rigid in today’s highly dynamic environment since they rely more on pre-defining customer requirements and are less flexible in adapting to change. It is attitudes towards RE that most differentiate agile and more traditional philosophies [3].

Although the industry has seen some improvements in software project success rates after the adoption of agile practices [4], the number of the projects successfully utilizing these approaches is still quite small. According to the CHAOS report in 2011 [4], the size of a software project is still a better predictor of success than the type of method used. Today, as more and more complex software is developed with agile methods, it is becoming challenging to rely so heavily on face-to-face communication. According to Savolainen et al. [2], companies adopting agile practices tend to first minimize processes to near zero before realizing that large-scale use of agile methods requires practices not so different from traditional ones.

Combining more traditional RE practices with agile methods can be an interesting endeavour. Traditional RE puts emphasis on determining a set of requirements in a separate phase before starting actual software design and development [5]. The iterative and less formal approach of agile methods is quite different and combining traditional RE with agility has gained much interest in recent years. Research on agile RE reveals that companies have already adapted many traditional practices to better suit the agile workflow. One of these is prototyping, which is seen as a good way to gain better communication between various stakeholders [6, 7].

However, as far as we know, there are no comprehensive studies on prototyping with agile software development. Furthermore, Cao and Ramesh [6] point out that little is known about how real agile projects conduct RE. There have been studies on the special characteristics of agile RE that also address possible challenges [6, 7, 5]. The goal of our case study was to investigate how prototyping can solve these challenges of agile RE. The case study focused on user interface (UI) prototyping.

In the next section, we describe challenges of agile RE based on existing literature. We also provide more background by presenting related work on the topics of agile RE and prototyping. Section III describes the case study and Section IV presents the most important results. In Section V, we compare these results to the findings from literature. Finally, Section VI presents conclusions on this study and proposes some future research topics.

II. RELATED WORK

Findings from literature show agile projects to have many RE problems that are inherent in project work. Handling changes and ensuring there is enough communication are some of the challenges faced in most projects. However, agile principles, in an effort to flexibly guarantee meeting customer needs, do bring out some challenges of their own. On the other hand, research on prototyping has quite old roots but there is surprisingly little research on combining it with agile approaches. This section presents an overview of the current state of research on both agile RE and prototyping.

A. Agile RE

Software engineering methodologies can be crudely divided into two categories: classical and agile [3]. These two schools of thought have been seen as conflicting partly because agile software engineering was originally developed to solve the
problems of its classical counterpart and therefore its rise has demanded abandoning belief in the old way of doing things. However, in their comprehensive study of the history of software engineering Jiang and Eberlein [3] found that, in the end, classical and agile philosophies actually share a lot of the main principles like good customer communication and the importance of competent developers. Similarly, Savolainen et al. [2] studied large-scale adaptation of agile methods and stated that although at first development teams enjoyed the freedom from traditional processes, later they found that they had to re-adopt many techniques similar to traditional approaches.

In the end, it is actually attitudes towards RE that separate the schools of thought the most [3]. According to Paetsch et al. [7] RE is a traditional software engineering process with the goal to identify, analyze, document and validate requirements for the system to be developed. Traditionally, RE has relied heavily on documentation and has aimed to find out a reasonable set of requirements in a separate phase before starting development to decrease the risk of costly changes later. As Cao and Ramesh [6] emphasize, the business environment where most companies these days operate is very dynamic and this can lead to big challenges with traditional approaches. Requirements are changing rapidly and can be hard to define in advance. Defining requirements too early can lead to obsolete specifications and ultimately to software that doesn’t meet customer needs. Also, as Waldmann [8] concludes, in real projects we usually have limited time and resources for RE activities. This means that even if we could identify everything in advance, it is usually not feasible. In practice many researchers have come to the conclusion that continuous requirements change is simply inevitable [9].

Solutions to many problems with the traditional planning driven models have been sought in agile software development methods [8, 2]. After all, agile methods were developed to adapt and thrive on change [7]. However, agile development has had a complex relationship with traditional RE and some agile communities have seen traditional RE as artificial and not addressing true software development problems [2]. This leads to question how RE should be carried out in agile projects. The answer to some extent is included in the methods themselves. Fowler and Highsmith [10] note that in agile methods, design is mostly a continuous activity and that the processes assume and even encourage the alteration of requirements during the entire project. Different agile methodologies differ somewhat in their specific guidelines but the basic idea is to carry out design sessions in each iteration and to do as much continuous refactoring as needed. For example, the most popular agile method, Scrum, has requirements built in as a simple list of backlog items [2].

However, the guidelines for RE provided by agile methods themselves are often vague [7], and defining agile RE is difficult as the concepts are still being developed. Research on agile RE relies heavily on either studying how companies are actually carrying out RE in agile projects (Cao and Ramesh [6], Bjarnason et al. [5]) or on existing literature on agile methods (Paetsch et al. [7]). Although it may be difficult to derive a unanimous definition for agile RE, research does offer some common characteristics of agile RE such as heavy reliance on face-to-face communication, requirements activities and design that are carried out iteratively throughout the project and continuous requirements prioritization. Other characteristics that come up are the use of prototyping or other modeling to make sense of requirements, test driven development (TDD) to ensure quality and review meetings and acceptance testing to ensure the right direction. All of these characteristics also contain some inherent challenges which are discussed further in the next section.

B. Challenges in Agile RE

According to Fancott et al. [11], agile RE has traditionally relied largely on conversations and implicit knowledge of the stakeholders. However, these non-rigorous means do not necessarily scale as well to the current environment of industrial software development [11]. To reach a better understanding of the problems of agile RE, we selected three studies (Cao and Ramesh [6], Bjarnason et al. [5], Paetsch et al. [7]) and tried to identify and categorize common agile RE challenges. In Table I you can find summaries of the biggest challenges identified by these studies. The grouping of the challenges shows that the most discussed challenges have to do with communication, lacking documentation, neglecting quality requirements and not understanding the big picture. Other issues have to do with motivating the team for RE and quality work, and problems with prototyping in particular. Below you can find more detailed descriptions of each problem area.

Managing with very little documentation and specifications A challenge closely related to communication is the lack of comprehensive documentation in agile projects. According to Cao and Ramesh [6], the customer may have difficulties to trust the agile process if they have no tangible requirements to look through. Also Paetsch et al. [7] note that lack of documentation may also become a problem with staff changes during the project or when the software transfers to maintenance.

Motivation issues related to RE work Keeping any over-head documentation up to date can be a daunting task for many teams. In the case studied by Bjarnason et al. [5], user stories and automated acceptance tests were used because writing automated test cases was believed to be more motivating than writing pure documentation. According to Cao and Ramesh [6], TDD treats writing tests as a requirements/design activity where you specify wanted functionality before developing it. However, the adoption of TDD is often undermined by lack of skills and motivation to learn [6].

Achieving enough quality communication with customer Despite the heavy focus on direct communication, agile projects still encounter a lot of communication issues. This is partly because, in the absence of comprehensive documentation, there is much more need for quality communication throughout the project. Agile methods assume a very accessible customer representative that participates in daily project activities [7]. This is often not realistic, and according to Cao and Ramesh [6], not achieving high-quality interaction with the customer can
lead to inadequate or even wrong requirements. The customer representative can often be either too busy to take part in the project or as Paetsch et al. [7] note, may lack the right data or authority to make decisions. Lastly, having several customer groups with conflicting needs and views can make achieving consensus in the short development cycles difficult [6].

**Not understanding the big picture** As agile methods encourage just-in-time information processing and iterative RE, there is a risk of partial optimization. In traditional RE this was prevented by making a more holistic plan of requirements in a separate phase. In agile RE there is minimal pre-planning and as Bjarnason et al. [5] note, missing a clear requirements picture in the beginning of development can lead to considerable rework and frustration later on. Missing the complete scope of requirements in the beginning can also make cost and schedule estimation extremely difficult [6]. Processing information just-in-time also makes reaching a suitable architecture challenging and can lead to uncovering system-level issues later in the project [6]. Constant refactoring is seen as an important part of agile philosophy. However, according to Cao and Ramesh [6] refactoring as an ongoing activity to improve the design doesn’t often completely address the problem of inadequate or inappropriate architecture. Finally, according to Cao and Ramesh [6], constant re-prioritization may also lead to instability. Changing focus too often can be a symptom of missing the big picture.

**Neglecting quality requirements** According to Paetsch et al. [7], handling of non-functional requirements is badly defined in agile approaches. Things related to maintainability, portability, security or performance can often be neglected as prioritizing happens with an emphasis on new visible features [6, 7]. Also, using business value as the main prioritization criterion may lead to serious quality compromises in the long run [6].

**Problems related to prototyping** Using user interface (UI) prototypes in requirements work can lead to the customer having unrealistic expectations of development speed [6]. For a non-technical customer, the UI often equals the software and if the prototype is done with too much detail they can think that most of the work is already done. According to Cao and Ramesh [6] they may then be unwilling to accept longer development cycles that are required to develop more scalable and robust implementations. Another risk with too much detail in the prototype is that there may be temptation to reuse the code also in production-ready software [6], which can lead to quality problems later on.

### C. Prototyping

In their article Baumer et al. [12] define prototyping as a development approach used to improve planning and execution of software projects by developing executable software systems for experimental purposes. There are many definitions for

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Managing with very little documentation</td>
<td>Customer having difficulties trusting a process without tangible results</td>
<td>[6]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lack of documentation</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>No detailed requirements documented to do a formal verification against</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Writing tests first requires low-level refinement of specifications beforehand</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Motivation issues related to RE work</td>
<td>Motivating team for constant requirements work</td>
<td></td>
<td>[5]</td>
</tr>
<tr>
<td></td>
<td>Developers are not used to writing tests first</td>
<td></td>
<td>[6]</td>
</tr>
<tr>
<td>Achieving enough quality communication</td>
<td>Achieving sufficient customer presence in project</td>
<td></td>
<td></td>
</tr>
<tr>
<td>with customer</td>
<td>Reaching consensus among more than one customer group</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Customer is not available for writing acceptance tests</td>
<td>[6]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ensuring sufficient customer competence</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not understanding the big picture</td>
<td>Not understanding the big picture of requirements at the start of the project</td>
<td>[5]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Uncovering system level issues late in the project</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Refactoring is inevitable</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cost and schedule estimation</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Constant re-prioritization can lead to instability</td>
<td>[5, 6, 7]</td>
<td></td>
</tr>
<tr>
<td>Neglecting quality requirements</td>
<td>Neglect of non-functional requirements</td>
<td></td>
<td>[5, 6, 7]</td>
</tr>
<tr>
<td></td>
<td>Using only time-to-market in prioritization can lead to neglecting quality issues</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problems related to prototyping</td>
<td>Use of prototype code in production can be risky</td>
<td></td>
<td>[6]</td>
</tr>
<tr>
<td></td>
<td>Early UI prototypes can cause customer to have unrealistic expectations</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
the practice but the purpose of prototyping is nevertheless always to gain more insight into designing the final product. Studies on the topic of prototyping have rather unanimously found prototyping to be a highly effective way to do user-centered design where you involve the user early on in development [13, 14, 9, 15, 16, 17]. As Acosta et al. [18] state, rapid prototyping techniques have also been recognized as an important technology for RE. Already in the early 90s Windsor and Storrs [15] found that using a prototype provides a user with an understandable specification and the prototype, which shows the designer’s interpretation of the requirements, can reduce the risk of misunderstandings and different interpretations of system specifications. Also other early researchers have noted that throw-away prototyping is cost-effective and improves specifications [16] and that it can be even seen as the cheapest, most practical method of producing a truly iterative approach to interface design [14].

Today, we have multiple fast and cheap ways to produce prototypes even in the earliest phases of the development cycle. The use of prototypes to capture requirements fits the agile thinking well - constructing the UI can be seen as an efficient way of reaching consensus on what we are doing. As Windsor and Storrs [15] state it, we need a model of the evolving design that is comprehensible to users and solicits feedback as much as possible. It is extremely difficult to visualise the behaviour of an interactive system with written, often lengthy and complex specifications [15]. Because of the natural tendency of humans to understand complex systems better as a visualization, prototyping can be seen as a good way of achieving a mutual understanding.

Although most of the research discussed here is focused on involving end users in prototyping, it should be noted that the same laws of understanding apply when the communication partner is the customer and not the end user. We will be discussing the upsides of prototyping related to agile development and particularly agile RE later when discussing our results in the light of other research. For now it is enough to state that, as far as we know, there has been very little research on the relationship between agile RE and prototyping.

III. THE CASE STUDY

This paper describes a case study that was conducted as part of a master’s thesis of one of the authors [19]. The case study was based on a one-year period of a large agile project. The original research was conducted with in-depth interviews and qualitative analysis. In the process of writing this paper the results were reviewed. The goal of this paper was to gain a deep understanding of how RE was practiced with prototyping and whether this helped in solving some problems inherent in agility. The case study also contained elements of insider action research [20]. The first author of the paper was an active member of the case study project. She was the lead designer and lead UI developer and the one producing the UI prototypes. Therefore, she was able to make observations and gain a deep understanding of prototyping and other software-engineering activities performed during the project. The validity threats related to the active role of the first author are discussed in Section V.C.

A. Overview of the Project

In May 2011, two developers were sent to customer premises to build a very easy-to-use UI for collecting working hours of store managers in a big retail company. The customer company was encountering big organizational changes and had already tried to automate its processes but the previous costly project had failed due to usability problems of the complex system. The new approach was to build the needed components iteratively and incrementally using a very agile approach. During the first year, the project size grew from two to ten developers in the core team and thirty people in the entire project. During the year the team developed a portal containing multiple applications related to hour tracking and human resources. The core team also took over the development of the backend systems. Although the project was delayed multiple times and encountered overruns, three years later the project had finally reached most of its goals with a few thousand users using the built software daily.

We classify the case study project to be large for the following three reasons: 1) number of team members and stakeholders, 2) duration of the project, and 3) complexity and size of the system. First, the maximum size of the core team was fifteen with at least another fifteen stakeholders supporting the project. Secondly, the project lasted three years with a lot of member turnover during this period. Thirdly, the system developed included most of the customer’s support processes related to salaries and human resources and ended up having a few thousand end users. There were also other suppliers involved which further complicated the management of the project.

B. Data Collection and Analysis

To collect information on the case we carried out semi-structured and open-ended interviews with four team members and one customer representative. The team members included two backend architects, a person in charge of maintaining the data and the person that, in addition to the author, was most involved in RE during the project. The customer interviewee was the project owner for six months and was the person who knew most about the systems and processes of the company. All participants that were interviewed had been involved in the project one year at minimum.

We chose semi-structured and open-ended interviews as the method because we wanted to gain rich information about agile RE and prototyping. In addition, the interviewees were able to provide information more freely than a questionnaire would have made possible. The interviews were semi-structured meaning that there were certain themes that we discussed in each interview and the questions were used as a frame for the discussion. The main themes of the interviews were: 1) challenges encountered in the project, 2) benefits from prototyping, and 3) improvement suggestions to the processes used. Additional questions were also asked depending on the interviewee’s role in the project. For example, effects
Elicitating initial requirements In the beginning of each iteration we had a brainstorming session with the customer. The customer usually had some preliminary ideas of the features they wanted implemented in the following iteration. These ideas were discussed very informally in these sessions. The meetings had a big participation rate including most of the development team and from two to eight customer representatives. There were always technical team members present, so we were also able to discuss implementation issues. During the meetings some notes were written and some drawings were made on the whiteboard. The meetings ended when we had a good enough picture of what should be done and we could start building a prototype.

Building the prototype The prototyping was also a fairly unstructured process and was usually started right after the initial requirements meeting. The meeting had produced some vision of the UI and one of the team members would start by doing concrete HTML/CSS to see if the preliminary ideas would work. We also made the prototypes interactive with JavaScript to enable a better understanding of the intended UI flow. After the initial version, some team members reviewed the design together. In some cases, the design process was quite complicated and we first had to discuss different alternatives and how they would affect the overall architecture. We usually showed what we were working on to the customer already before the next meeting and getting this informal firsthand feedback was crucial in setting the design in the right direction.

Reviewing the prototype The prototypes were developed in one to two days and then we held a review meeting with the customer. The participants of these meetings usually included most team members but only key customer representatives. We went through the prototype features together and there were questions and discussion along the way. The changes to prototypes that came up in reviews could include minor tweaks or a total refactoring of the whole prototype. Some designs only stabilized after many iterations as the customer kept realizing new viewpoints when seeing the prototype.

Using the prototype in implementation The prototypes we created were always developed directly in the same technologies as the final product. We felt that it was faster to develop real UI prototypes than to create paper prototypes because the prototype code could be used as a reference for the actual implementation. It was important that the final prototype was public to the whole development team so that also backend developers could use it as a reference for software architecture design.

B. Most Important Findings

The case project was a valuable example of agility and its problems in a big company-wide project. The circumstances were challenging as the customer organization was in a transition to new internal processes. On one hand, the project was successful because it was agile and able to adapt to change. On the other hand, the case study revealed many problems of agility and especially agile RE. We found that while prototyping helped in some areas of agile RE, it would have required...
support in other areas. Table II gives an overview of the effects of prototyping on challenges identified from previous literature. The effects have been derived from a comparison of challenges in literature and in the case study project [19]. The following paragraphs describe both benefits of prototyping, and problems where complementing practices would have been beneficial.

**Managing with very little documentation and specifications** This was the area where we gained most benefits from UI prototypes. It is characteristic of agile methods that documentation is minimal. However, this can cause problems as the customer may have difficulties trusting the process and understanding what the developers are doing. In the case study project the prototypes acted as a comprehensive but low maintenance documentation for the plans of one iteration. The prototypes also made writing acceptance tests easier as they provided relatively detailed specifications for features.

**Motivation issues related to RE work** We had few motivation issues related to RE work because the prototypes were concrete and actually fun to work with. It was perceived more motivating to update the visual prototype than textual documentation. Our project still faced challenges in practicing TDD (Test Driven Development) and the development team agreed that the method should have been utilized more than it was. Quality assurance was sometimes lacking and many bugs were only found after completing a feature.

Using ATDD (Acceptance Test Driven Development) could have stabilized some features earlier as the goals of the iteration would have been clearer. The project team did practice some ATDD by using a test framework that enabled integration tests to act more as acceptance tests. These automated tests were written in natural language because we were hoping the customer would then participate more in writing acceptance tests. The framework worked well, but we simply did not utilize it as well as we could have and also did not get the customer to participate enough.

**Building acceptance tests early on based on the prototypes** would have been a good improvement in the case study project. The UI prototypes could have served as a good base for building acceptance tests more efficiently. Not realizing this potential was mainly due to lack of skills in TDD and the fact that requirements seemed to change too much even during iterations. However, several interviewees noted that clearer acceptance tests would have given the process a better backbone. As it was, we were often unsure when the features were actually production ready.

**Achieving enough quality communication with customer** Despite the fact that the project team worked in customer premises and used UI prototypes for better communication we still experienced problems in this area. Nevertheless, we did gain some benefits in this area from prototyping. It was sometimes difficult to achieve sufficient customer presence, but when we did, the prototypes enabled better communication and a quick way to achieve shared vision among different stakeholders.

The improvement idea that came up in the interviews was to **educate the customer on agile methods and on being a product owner**. The role of product owner comes from Scrum where it means the person responsible for representing the customer and being the voice of the stakeholders. Common to most agile methods is the importance of the customer role in the project. Sufficient participation is often hard to achieve and we also failed to educate our customer enough on his role. Building a genuinely agile project requires agility from both the development team and the customer who should be educated on their responsibilities right from the start.

**Not understanding the big picture** One of the biggest problems we faced in the project was that, as we concentrated on a few features or one small UI at a time, we sometimes missed the mark on the business needs of the customer. Keeping the business needs in mind at all times might have led to some less perfect partial implementations but would have been better for the project as a whole. A good example of this was the human resources UI that was originally designed for thousands of store managers but at some point was transformed into an administrative UI for ten users causing some quality issues. Had we understood the internal politics while doing the original design, we could have included a way for the administrative personnel to remain in control of the data. The bigger picture in this project would also have included better understanding the components that other suppliers were developing. Despite these difficulties we also had some positive effects from prototyping in this area. The system architects reported that the prototypes made architecture design easier at times and helped in basing important decisions on something concrete.

As an improvement, it would have been beneficial to start the project by **taking time to understand the big picture** and making a larger-scale plan. This plan could then have been reviewed every few iterations. Although it would have changed a lot during the project we might have been able to catch some major problems earlier. The review sessions should have included participants from as many customer groups as possible so that all views could have been considered. It would also have been beneficial to keep a record of the envisioned roadmap for the project in a flow chart or some other simple illustration. Overly focusing on end user satisfaction was also further strengthened by prototyping. It enabled too much focus on partially optimized solutions. When using prototyping and very iterative approaches it is important to remember that user needs and customer needs can differ and are both important to consider when building comprehensive solutions. The customer is not always right, and a team of software development professionals should be able to make the customer see that on those occasions.

**Neglecting quality requirements** Our prototyping approach only served to enforce the natural tendency of agile methods to neglect quality requirements. In the case study project we mainly focused on visible features based on user interface prototypes. For example, there were some architectural requirements concerning our database structure that were mentioned every now and then by the customer, but that ended up not having any effect until a visible feature demanded it. By this time it was already extremely costly and demanded a
lot of rewriting and a total re-factoring of the existing database structure.

Keeping better track of quality requirements came up in the interviews as a needed improvement. We could have kept track of quality and architectural requirements separately of other requirements. At minimum this documentation could have consisted of a checklist that would have been read and updated in each iteration to guide both design and development throughout the project. This was an idea that our product owner presented in the interviews. The idea would be to have the list and then go through it whenever a new feature is taken into the backlog.

Problems related to prototyping We did not experience too much of the problems related to prototyping mentioned in literature. It did take some expectation management to make especially the non-technical customer groups understand that prototypes are not ready for production. We also mostly did not use prototype code directly in production but used it as a reference for the actual implementation. This worked well and improved the quality of the code. Sometimes, if time pressures forced the project team to use bits of prototype code in the ready product, it resulted in lower quality.

One improvement related to prototyping would have been to build less finalized prototypes to enable the team to take more responsibility for design. When prototypes look too finalized, it can be hard for the customer to stay out of design issues. Using a bit less design effort in the prototyping phase could prevent the reviews from going into too much detail too early. This would also prevent the customer from having too high expectations, which was mentioned as a problem in both literature and in one project interview.

C. Summary of Findings

The case study revealed that prototyping solved some challenges in agile RE while having very little or even a negative effect on other agile RE problems. The biggest benefit from prototyping was having unambiguous documentation in the form of prototypes. The prototypes helped create trust and acted as clear documentation. Another big benefit was that prototyping provided better motivation for RE work. Working with prototypes was perceived to be more motivating, and they also made it easier to practice ATDD when appropriate. The third area where we gained benefits was in ensuring improved communication among stakeholders. Prototyping did not make the customer participate more in daily activities but it did help in reaching consensus more easily by providing a common language that everyone could understand.

On the other hand, in the areas of not understanding the big picture and neglecting quality requirements, prototyping did not help significantly and may have even highlighted these problems. Other challenges that were not helped by prototyping were problems with communicating the customer role more clearly, motivating the team for ATDD and issues with too detailed prototyping. Figure 2 presents five improvement ideas we found to solve these problems, added into the previously shown case process.
EVERY FEW ITERATIONS

- Reviewing big picture
- Educating customer on being product owner

EACH ITERATION

- Eliciting initial requirements
- Building simplified prototype

New features from backlog

- Using prototype in implementation
- Reviewing prototype with quality requirements

Building acceptance tests based on prototype

Fig. 2. Improved process for RE with prototyping. Improvements highlighted with red color.

V. DISCUSSION

In this section, we compare the case study results to current literature. The case study revealed much of the same issues as previous literature on agile RE challenges, including lack of customer participation, neglect of quality requirements and instability due to constant change. However, looking at the results it seems that some problems that had plagued other projects were counteracted by prototyping in the case study project. Next we go through benefits and shortcomings of prototyping as found in the case study, and compare them to literature.

A. Benefits from Prototyping

According to literature, prototyping can complement the agile philosophy well and Memmel et al. [21] even suggest that the role of prototypes was less important in software engineering before agile approaches refocused attention on them as vehicles for inspections and testing. Here we compare our positive findings regarding prototyping with previously published work.

Having unambiguous documentation Prototyping helped in many of the challenges related to managing with very little documentation that often plague agile RE. Acosta et al. [18] also report that developing executable prototypes as part of the requirements specification process diminishes ambiguity, incompleteness and inconsistency that easily become prominent when developing complex software systems. Lack of tangible plans and specifications didn’t come up as a challenge in the case project because the prototypes provided sufficient documentation. Also, Sillitti et al. [22] point out that companies using agile development seek to create trust by integrating the customer into the development process. However, when the customer involvement is insufficient there may not be much base on which to build trust. In our case the prototypes provided this base and there were very few trust issues.

In their study Haugset and Stålhane [23] identify similarly the absence of clear documentation in agile RE as a problem worth solving and they propose that extensive use of automated ATDD would enable combining the traditional RE focus on documentation and the agile focus on iterative communication. However, Haugset and Stålhane [23] also report that textual descriptions are often not enough to convey understanding between the customer and the developers. On the other hand, Savolainen et al. [2] suggest that if requirements change too often there might not be any sense in recording them, but to instead get feedback from the customers by having frequent small releases. However, always finding out requirements were misunderstood only at the end of the iteration could lead to inefficiency. Prototyping can help to mitigate this risk by validating requirements before implementation.

Better motivation for RE work Our findings indicate that updating a prototype was considered to be more motivating than writing requirements documents. The reason was that it was easy to go through the visual results with both the customer and the development team. Also the use of target technologies made it possible to test implementation strategies while building the prototypes. We did not find a lot of research on RE motivation issues but Bjarnason et al. [5] report that automated acceptance tests were believed to be more motivating to write because they consisted of code and not pure documentation. Prototypes could have motivated us to utilize more (A)TDD by providing comprehensive specifications. However, we found that they did not help us overcome lack of skills nor motivation to learn the practice, which according to Cao and Ramesh [6] can be challenges in agile projects.

Improved communication among stakeholders We could not make the customer participate in all activities like writing acceptance tests but prototypes did help us make sure that the communication we had was of better quality and that mutual understanding was reached faster. Schrage [9] suggests that instead of a ‘risk-adjusted’ set of requirements, the main medium of communication between the development team and the customer should be a prototype or model that would better capture the real needs of the customer. It can often be hard to increase the amount of communication with the customer and therefore reaching better quality is one key to improved communication.

B. Complementary Practices to Use with Prototyping

Our findings indicate that prototyping is no panacea and it needs other good practices to complement it. Several studies on agile RE conclude that achieving sufficient customer involvement is difficult in practice [5, 6, 7] and we found this to be true even while working at the customer premises. Our suggestion would be to state customer responsibilities more clearly from day one and so to educate the customer on agile methods and on being a project owner.

Keeping better track of quality requirements is another important RE practice that needs to be integrated with prototyping and this was also supported by other studies. Paetsch et al. [7] and Cao and Ramesh [6] report that quality requirements
tend to get neglected in agile projects. The results of our case study concur with this finding as visible features were the main target of the prototypes.

The need to take time to understand the big picture is one of our key findings and also Bjarnason et al. [5] report that the lack of understanding of the big picture is an agile RE problem. Related topics such as difficulties in cost and schedule estimation and uncovering system level issues too late are documented by Cao and Ramesh [6]. Our results also indicate that business requirements of the customer are critical when forming the big picture. According to our knowledge, both business requirements and big picture understanding are underrepresented in literature on agile RE.

Our results suggest that ATDD is one of the practices that can support the usage of prototypes. Haugset and Stålhane [23] report that the number of studies on ATDD is low and that they did not find any research linking ATDD to a RE framework. Our case study indicates that building acceptance tests early on based on the prototypes could bring benefits to agile projects. These benefits could include having automatic definitions to indicate when a feature is production ready and having documentation that could be updated also after we have abandoned the prototype during implementation. Automatic test definitions could be written utilizing a natural language based test framework. Their key idea is to specify use case descriptions so systematically that they can be used to test the implemented features automatically with the test framework.

Cao and Ramesh [6] report that prototypes can create too high expectations and deploying prototype code into production can become a problem. Our findings support these results although neither was perceived to be a serious problem in the case project. However, both problems can possibly be counteracted by Building less finalized prototypes that indicate clearly that the features are not ready.

C. Limitations of the Study

The fact that one of the researchers was an active member of the case study project may have affected the internal validity of the results. According to Coghlan [20], one disadvantage of being close to the data can be that the researcher may assume too much and not be able to probe as deeply as an outsider would. We tried to reduce this validity threat by two actions. Firstly, the researchers consciously tried to adopt and maintain a stance of neutrality during the data collection and analysis activities. The interview questions were designed to be as neutral as possible. The experiences of the first author were also explicitly recorded and then compared with the results from the five interviews. Secondly, the data analysis was conducted in an iterative manner. The results of the first analysis were published as a master’s thesis and they were reviewed by the second author of this paper prior to publication. This paper contains the results of the second analysis. According to Coghlan [20], insider action researchers need to show how they challenged their interpretations of what was happening in the project and the results of the analysis need to be exposed to critique by means of public reflection.

The results of this study are based on one project, which is a threat to external validity. Our case study indicates that the agile RE approach combined with prototyping can be useful in a large project. However, every project is unique and our findings can be questioned because we cannot know whether other approaches might have worked as well. Existing research suggests that prototyping can be a good practice for overcoming challenges encountered in agile development. In the study of Cao and Ramesh [6], 11 of the 16 companies were using prototyping to assist in communicating about the requirements with their customer. This seems to support our findings of the practical value of prototyping.

VI. Conclusions

The findings of the case study indicate that while prototyping helps in solving some of the agile RE challenges, it can also strengthen other problems often encountered in agile projects. Therefore, it seems that prototyping is most beneficial for agile projects when it is used with other complementing practices. One of the biggest benefits from prototyping is that the prototypes act as tangible plans that can be relied on when discussing changes. Prototypes also seem to improve motivation to do requirements work as they force participants to discuss changes to requirements more concretely. Finally, although the case study once again demonstrated the difficulty of achieving enough communication, prototypes do seem to improve the quality of communication both between team members and between stakeholders.

On the other hand, prototyping does not solve the problem of partially optimized and volatile requirements in agile projects. If prototyping is used only to demonstrate the plans of one iteration it can strengthen the emphasis on short term goals. By directing the focus on visible features, prototyping can also worsen the problem of neglecting quality and architectural requirements. The most important practices that could mitigate these remaining challenges would be reviewing the big picture at steady intervals, reviewing prototypes with quality requirements at hand and building acceptance tests based on prototypes. The first two tackle the problems of lacking big picture and quality requirements head-on. The last one, building acceptance tests, could help to keep the vision alive even after abandoning the prototype during implementation.

The results of this study point to several topics for future research. First, it would be interesting to investigate how ATDD can be combined with prototyping in agile projects. The prototype developed at the beginning of each iteration could help in writing acceptance tests. Furthermore, it could be relevant to explore how architecture design can benefit from prototyping. Prototyping is often related to usability, but in our case study also system architects reported gaining benefits from prototypes. Finally, there is need for more case studies on prototyping as an agile RE practice. Further studies on the use of prototyping in agile projects would strengthen the findings of this paper.
REFERENCES


