Continuous and Collaborative Validation: A Field Study of Requirements Knowledge in Agile

Rosalva E. Gallardo-Valencia
University of California, Irvine
rgallard@ics.uci.edu

Susan Elliott Sim
University of California, Irvine
ses@ics.uci.edu

Abstract

We present the results of our field study that describe how requirements validation was performed at an industrial software company using agile software practices. As is common in agile processes, the team did not capture requirements knowledge in a comprehensive specification document. Instead, requirements knowledge was captured in user stories, automated acceptance tests, personal notes, and conversations. Validation was performed continuously, during pre-iteration, iteration planning, and intra-iteration using mainly conversations. Validation was also collaborative and involved all team members, including the Product Owner, programmers, and testers. The results of our field study have implications for both agile and validation methods. This successful arrangement of agile practices is instructive for agile practitioners and for researchers studying validation.

1. Introduction

Requirements engineering is a knowledge-intensive activity. The correct and clear understanding of requirements among stakeholders and software team members is key to the success of a software product.

Ideally, requirements knowledge is captured in a written format called a requirements specification document that includes all details that specify customer needs. However, recent studies on requirements engineering experts have shown that requirements engineers prefer short documents that include only the important details at an appropriate level of abstraction [1]. Written requirements knowledge is complemented by requirements knowledge that is shared through conversations among stakeholders and software team members.

We are interested in understanding how requirements validation takes place in environments where little requirements knowledge is written down. For this reason, a software team using agile practices offers a good setting for studying requirements knowledge.

Requirements validation is an activity that requires different techniques in agile software development. Existing approaches to validation rely heavily on a requirements specification document which is not available in agile. This leads to a false assumption that rigorous validation is only necessary on projects that produce a complete set of written requirements. However, producing software that does not do what the customer wants is a major risk [2], which means validation is necessary on every project. The question then arises: how can we perform validation on an agile project without comprehensive documentation?

We conducted a field study of a company using agile requirements and analyzed the data qualitatively. We use the phrase agile requirements to refer to the requirements managed using agile techniques such as user stories [3], iterations, among others. In this field study, we saw agile validation being carried out at a software company using existing agile practices and artifacts, in particular Scrum [4], user stories, and automated acceptance testing [5]. Not surprisingly, the software team did not capture requirements knowledge in a comprehensive specification document. Instead, they captured requirements knowledge in user stories, improvised checklists, and automated acceptance test cases. The written component of these artifacts and practices was minimal; they served only as reminders. There was a dependence on ongoing conversations and feedback. It was in these interactions that we found requirements validation activities.

We believe that this approach worked at this company for two reasons. First, requirements validation was conducted continuously throughout an iteration. Validation was not an activity performed only at the end of the iteration. Instead, each user story was validated on an ongoing basis, using mainly conversations to share knowledge about requirements.

Second, *every* team member participated in requirements validation. Although the Product Owner

(PO) played a key role, the Scrum Master, manager, programmers, and testers were also actively involved. They worked together to define the code that needed to be implemented and the test cases that should pass to complete a user story. Validation was a collaborative activity shared by all team members.

Just as agile requirements differ from the requirements documentation found in other processes, we found that validation in agile differs from classical forms of validation. Agile validation is a form of validation that is continuous and collaborative. Although, our study was conducted at only one organization, we do not believe that agile validation is unique to this team. Scrum, user stories, and automated acceptance testing are used throughout the agile community. This paper provides an explication of agile validation methods based on qualitative analysis and reflections on our findings. These results will be instructive for practitioners and researchers of agile and of validation.

2. Field Study – Easy Retirement

We conducted a field study at Easy Retirement[1], an Internet-based 401k service provider. The company's main product is a web application that allows individuals to manage their own retirement investment plans, and is sold as a service to customers. The company has a total of 26 employees, and the software team consists of ten members including the Scrum Master, the Product Owner, a technical manager, programmers, and testers. We observed the software team at work, attended planning meetings, interviewed team members, and collected software artifacts.

2.1. Agile Practices

The software team follows a number of agile methods closely, including Scrum, daily stand-up meetings, user stories, continuous integration, on-site customer, and automated acceptance testing.

Each Sprint or iteration lasts two weeks, and starts and ends on a Friday. On the first day of an iteration, the team holds a Sprint Planning meeting when user stories are broken down into tasks and estimated. During the Sprint programmers and testers will work closely to complete the user stories and to ensure that all the user stories are accepted by the end of the Sprint. It is a challenge to get everything done in an orderly fashion, because testing and development are mutually dependent.

[1] This name is a pseudonym to protect the company's privacy.

2.2. User Stories

The primary unit of work for requirements is the user story, which includes written reminders, test cases, and conversations. Their most public written representation is the index cards on the Scrum board using the format *"As a ... I want .... so that ....."* But individuals also have their own representations that they use in their work. The Product Owner and Scrum Master each maintain a personal checklist in a spreadsheet. The testers store additional details in a wiki [6].

2.3. Test Cases

During the Sprint, testers start creating the test cases while programmers are adding new functionality. Testers use Fitnesse [7], an automated acceptance testing tool, to create the test cases. On the first Monday of the Sprint, testers meet with the Product Owner to ask questions about the high-level requirements and to create acceptance tests. The programmers also use these tests to guide their design work. Programmers are expected to write their own unit tests and to write fixtures for Fitnesse.

All team members were involved in creating test cases, especially for acceptance testing, though the programmers were primarily responsible for unit tests.

2.4. Conversations

All team members have conversations about the requirements on a daily basis. These conversations are mainly about the user stories, high-level test cases, and acceptance test cases. This face-to-face knowledge sharing facilitates the validation of requirements.

We analyzed the flow of requirements at Easy Retirement to identify the potential interaction points that are used to validate requirements. We identified three stages around each iteration when requirements are engineered. These stages are: 1) before the iteration; 2) during the iteration planning; and 3) during the iteration.

3. Method

We used qualitative research methods [8] to collect and analyze data at our field site. We observed the software team for two days in December 2007. This included observations of stand up and iteration planning meetings. We also conducted six semi-structured interviews that lasted between 30 and 68 minutes. Our interview participants had various roles

in the company, including the Scrum Master, Product Owner, two programmers, a tester, and the owner of the company. We also collected software artifacts such as user stories, user stories checklist, and test cases.

After transcribing the interviews, we analyzed them using focus coding [8]. This technique helped us to identify points of interest in the transcripts such as those related to validation of requirements. We grouped the identified points of interest across transcripts and used this information to determine the flow of requirements and how and when they are validated.

4. Requirements Knowledge in Agile

We begin our discussion of agile validation by looking first at the requirements process. This perspective is necessary, because the two activities are inextricably linked. In agile, requirements knowledge is gathered iteratively and incrementally, with a customer focus. Agile validation builds on and is enabled by these processes. We use the phrase requirements knowledge to refer to the requirements themselves including both requirements that are written down and requirements that are shared through conversations.

The requirements process and artifacts are also closely tied to each other. In particular, user stories are both artifacts and activities, because they include a written reminder, test cases, and conversations. It is difficult to draw a stark line between an activity, such as a conversation or testing, and the artifacts, such as user story cards and wiki pages. We use the term artifact-activity to refer to this tight coupling of an artifact and an activity.

Some requirements knowledge is kept in written formats such as user stories, user story checklists, task cards, and test cards. All these formats are short and only include relevant details. *Written requirements knowledge* is also kept in high-level test cases in a wiki, acceptance test cases, unit test cases, and code.

A considerable part of requirements knowledge is shared in conversations between business people, Scrum Master, Product Owner, programmers, and testers. We will refer to this unwritten requirements knowledge as *live requirements knowledge*. The validation of these requirements is done through conversations and there is no written record of it.

It is also difficult to situate requirements knowledge precisely in one form of representation or another. While this coupling makes our data analysis more complicated, it is one of the strengths of agile. In other words, there is a web of collaborative and interlocking practices that provide a safety net for adaptive software development.

Consequently, we describe the requirements process at Easy Retirement using activity diagrams, as shown in Figure 1 - Figure 3. By modeling the process in this manner, we are able to see both the activities and artifacts. This representation also facilitates our examination of validation activities. The stick figures along the top of the diagrams represent the different roles involved in requirements. Horizontal arrows show questions and answers being exchanged. Large boxes that are bordered with a dashed line depict meetings. Smaller rectangular boxes represent artifacts that are created or used. Down the left hand side of the diagrams is a shorthand notation for the kinds of requirements activities taking place. The first letter shows whether the interaction is performed through a conversation ('C') or through other means ('-'). The next character in the shorthand is a vertical bar to separate the next group of characters. The last four characters correspond to the four canonical activities in requirements engineering, elicitation ('E'), modeling ('M'), communication ('C'), and validation ('V').

In our data analysis, we found that there were three time periods that correspond to different opportunities for requirements engineering. These stages were: 1) pre-iteration showed in Figure 1; 2) iteration planning showed in Figure 2; and 3) intra-iteration showed in Figure 3.

4.1. Pre-Iteration

Pre-iteration is the time leading up to the start of an iteration. During this stage, the Product Owner received requirements from business people and clarified their expectations (arrows 1.1 and 1.2 in Figure 1). Then, the software team held a Brainstorming Meeting (1.3-1.5 in Figure 1) to make preliminary estimates of user stories, which are prioritized in a later meeting (1.6 in Figure 1).

The Product Owner records what he has learned from these interactions with the business people to create a personal user story checklist in a spreadsheet. The Product Owner uses this checklist to help him think of questions to ask the businessperson regarding details and test cases, manage details of the user story, understand scope, manage risks, and document conversations and decisions. The checklist also helps him to participate more effectively in Iteration Planning.

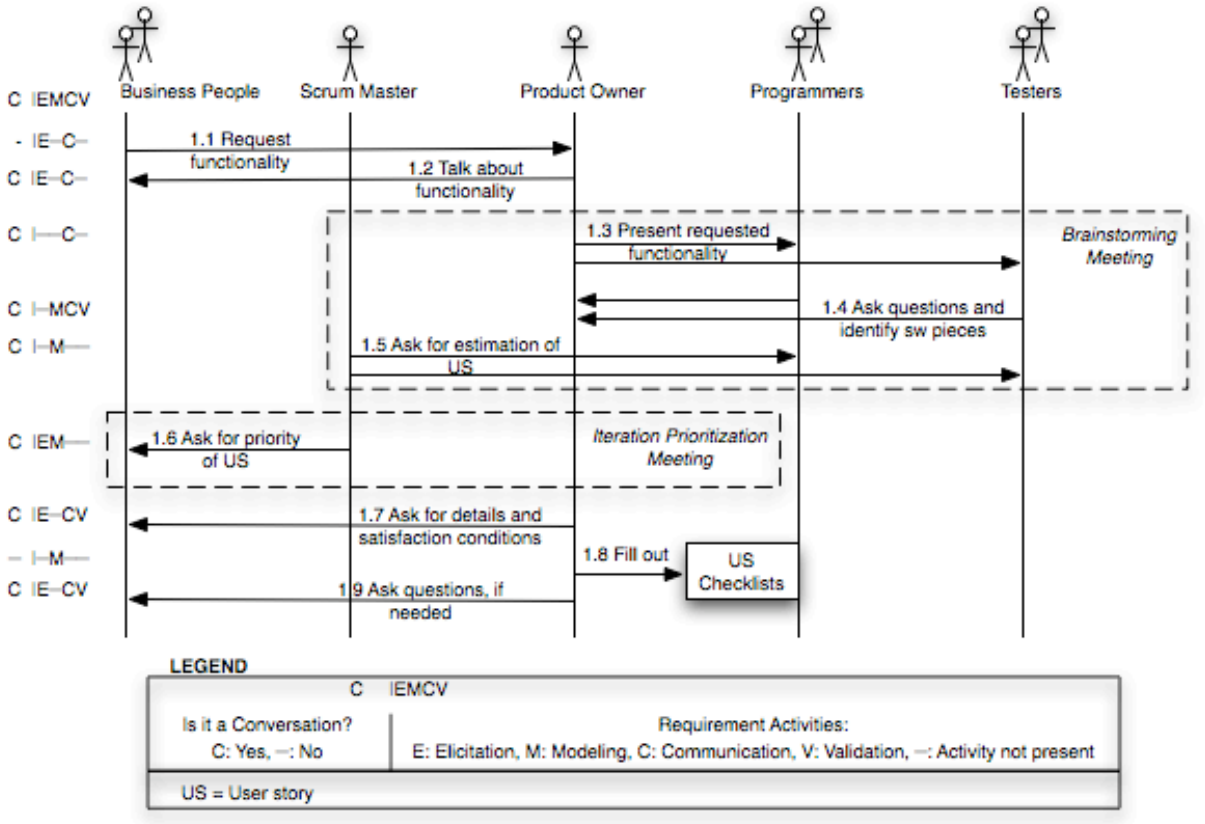

Figure 1. Pre-Iteration flow of requirements at Easy Retirement

The user story checklist includes the satisfaction conditions that are needed to consider the user story completed, as well as a story description, its assigned story points, expected delivery date, the iteration it belongs to, output of the story, and related user stories.

The user story checklist is an instrument to validate requirements because it helps the Product Owner think about questions related to requirements and user story completeness. When the Product Owner asks these questions to the business people, the answers received serve to validate the information the Product Owner already has.

4.2. Iteration Planning

Every Friday, an Iteration Planning Meeting (2.1-2.5 in Figure 2) is held. At this time, user stories, tasks cards, and test cards are written on an index card and placed on the Scrum Board. All the team members, including the Product Owner, Scrum Master, technical manager, programmers, and testers work collaboratively to create these cards. The index cards do not have much information on them, just a couple of sentences. However, the requirements that they represent are validated when programmers and testers ask questions about details.

4.3. Intra-Iteration

During the iteration, as showed in Figure 3, testers write tests, programmers implement new functionality, and testers run the acceptance test cases. Additionally, the Product Owner checks that the software works as expected.

Testing plays a particularly important role during the Intra-Iteration stage. For the first time, requirements are expressed as test cases. This information is stored on a wiki, based on the detailed information obtained by the testers in conversations with the Product Owner. Programmers are free to look at the information, though they rarely do. Often, there is friction between the testers and programmers, because the testers must enforce the acceptance criteria from the business people passed on by the Product Owner.

These criteria are written into automated acceptance tests within Fitnesse. Thus, we can consider that when these test cases are run the requirements are being validated.

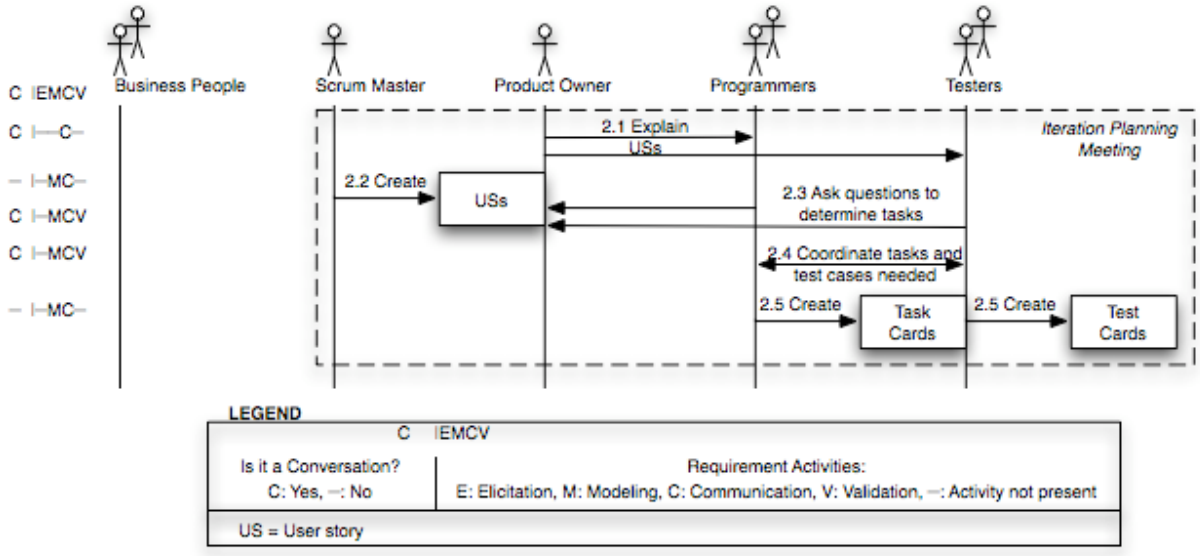

Figure 2. Iteration planning flow of requirements at Easy Retirement

High-level test cases are also instrumental on requirements validation. When they are written, testers think about questions related to the acceptance criteria of user stories. Testers ask the Product Owner questions, and the answers help them to validate the requirements they have in the wiki.

5. Agile Validation

In the previous section, we described the agile requirements process at Easy Retirement. Consistent with the values in the Manifesto for Agile Development [9], the process does not use comprehensive documentation, relies on collaboration, and emphasizes individuals and interactions. It should come as no surprise that validation has the same characteristics.

Agile validation is continuous and collaborative. This activity is carried out continuously during the development of the software product and mainly through conversations. We saw that validation activities are performed by different roles in the development team during the whole iteration. Consequently, the whole team contributes to software quality.

5.1. Continuous Validation

The concept of continuous validation applies to both artifacts and activities. Because the two are tightly coupled in agile, both perspectives need to be considered because they suggest different focal points for validation activities.

User stories are the primary unit of work in agile. Consequently, each user story is validated one at a time. This focused validation allows programmers to gain a detailed understanding of small pieces of functionality. Validation occurs when a user story is discussed, when it is broken down into tasks and tests cards, and when it is written on index cards. High-level test cases and automated acceptance test cases are created specifically to validate a user story. In addition, programmers and testers are able to validate requirements (arrows 1.4, 2.3, 2.4, 3.2, 3.6, 3.10) even before a single line of test or code has been written for a user story. The entire team has a shared understanding of the requirements related to each user story.

The iteration also serves as a focal point for validation. Work is done in anticipation of an iteration, to plan for an iteration, and during an iteration. Working out the requirements for the whole project is avoided. The team focuses on doing only what needs to be done for the current iteration. As a result, the Product Owner will be able to concentrate on getting detailed requirements, communicate those to the development team, and also validate these requirements through conversations in the time frame of the Iteration. This focused validation of requirements occurs during pre-iteration, iteration planning, and intra-iteration. These time-boxes allow programmers to have feedback on their code intra-iteration, when they still have the current user stories fresh in their minds. Also, programmers and testers are able to see their code working and being tested by the Product Owner and business people.

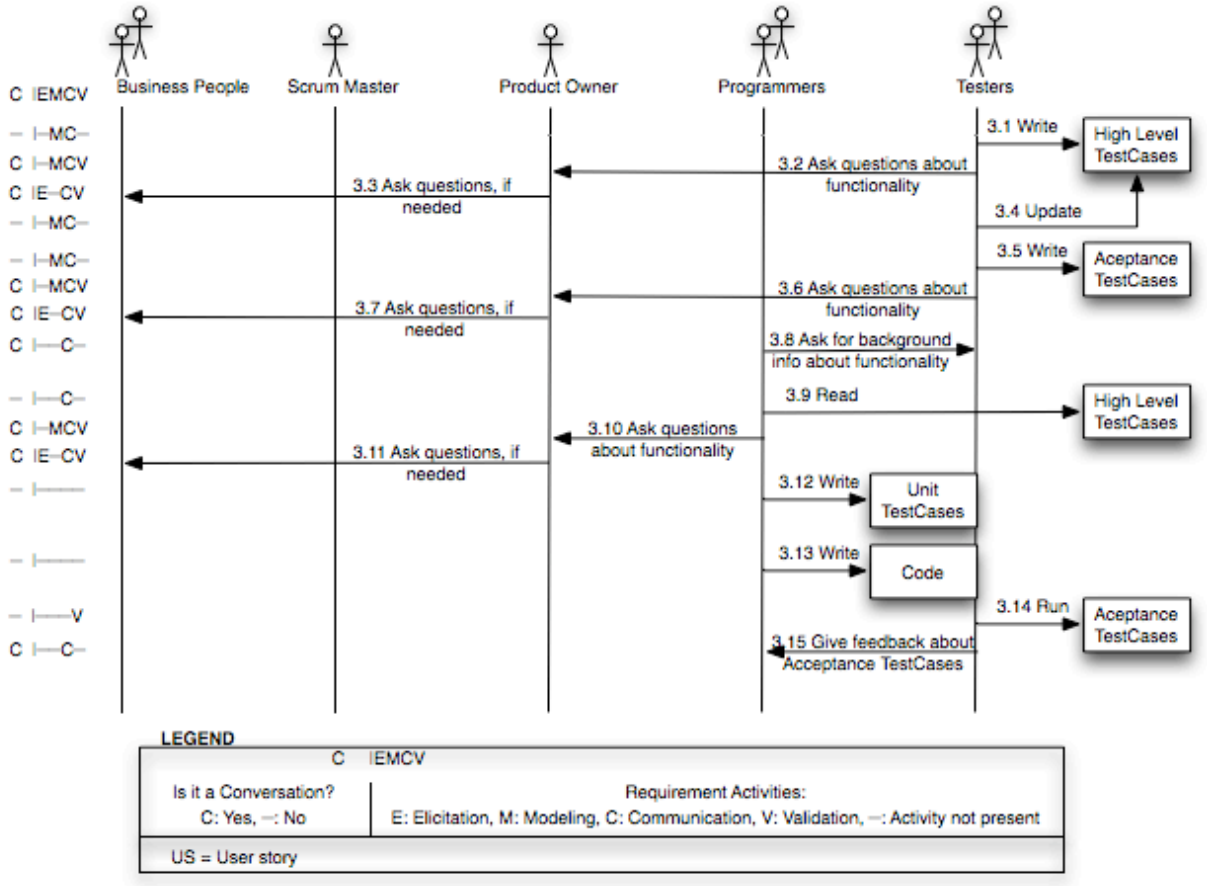

Figure 3. Intra-Iteration flow of requirements at Easy Retirement

If adjustments are needed, they can be part of the next iterations. Validation is carried out in each iteration, until the whole project is done.

Continuous validation is possible due to the practices of agile, such as Iterations, user stories, and test-driven development. Similarly, collective and collaborative validation is possible due to people who perform different roles such as business people, Product Owner, Scrum Master, programmers, and testers, who trust on each other, are able to share live requirements knowledge, and are available to ask and answer questions of each other. Live requirements knowledge is used to validate requirements mainly through conversations. In Table 1, we identify the shared knowledge, both written and live, that is involved in requirements validation.

Iterations promote conversations among stakeholders and the development team during the Iteration Planning and during the whole iteration. User stories are triggers for conversations about requirements. Test-driven development (TDD) [10] promotes early communication among testers, programmer, and Product Owner. Thus, practices enable team members to rely on conversations to communicate and validate requirements knowledge.

Table 1. Written and live knowledge used to validate requirements

Stage	Written Knowledge	Live Knowledge
Pre-Iteration	-	- Conversations between Programmers and PO, and Testers and PO. - Conversations between PO and Business People.
Iteration Planning	-	- Conversations between Programmers and PO, and Testers and PO. - Conversations between Programmers and Testers.
Intra-Iteration	Acceptance Test Cases	- Conversations between Testers and PO, PO and Business People, and Programmers and PO.

5.2. Collaborative Validation

At Easy Retirement, all members of the development team worked together collaboratively to validate requirements. There was no *one* person who was responsible for validation, because everyone was responsible for this activity. We observed that validation was initiated by Product Owner (1.7, 1.9, 3.3, 3.7, 3.11), programmers (1.4, 2.3, 2.4, 3.10), and testers (1.4, 2.3, 2.4, 3.2, 3.6). All of them asked questions to make sure that they have the correct understanding of what business people want.

At first glance, one would expect validation activities to be centered on the Product Owner, because he is the conduit between business people and engineers. However, this was not the case at Easy Retirement. Programmers and testers both asked and answered validation questions. This interaction happens during the iteration planning where programmers and testers talk to each other to validate their understanding of the functionality and create task cards and test cards (2.4). It also happens in intra-iterations when test cases and fixtures are exchanged.

High quality software is the shared goal of the development team. All members cooperate to achieve this goal. Although the Product Owner is the first point of contact, he does not have sole responsibility for requirements engineering, including validation. It is always possible that Product Owner missed some details or questions. However, whenever user stories change hands between team members an opportunity arises for programmers and testers to raise questions. The Product Owner might have an answer, but if he does not, it reveals a gap in knowledge that needs to be addressed. Thus, validation was a collaborative activity; if one team member misses something, others pitch in, help out, or catch the mistake.

6. Discussion

In this section, we discuss the implications of agile validation for both practitioners and researchers. Based on our field study, we have a number of insights and observations relevant to the application of agile and the study of validation.

6.1. Effectiveness

A natural question to ask is: *Does agile validation actually work?* We do not have quantitative data to show that it does. However, Easy Retirement's executives and business people are satisfied. In comparison to approaches that they used previously, it is much faster and easier to get the functionality that they want. The software programmers also feel that the process is a vast improvement over what they used before, even the ones who resist interacting with end users. The company is growing rapidly and attracting new customers. Beyond these anecdotes, we were not able to collect metrics that showed a quantitative change. However, improvements in these areas get to the heart of agile, the value of working software.

Other questions also arise regarding the application and use of the agile validation process itself.

Isn't it bad to have one person be the gatekeeper for all requirements? At first glance, it appears that the Product Owner tracks all the requirements. In practice, any member of the development team is welcome to communicate with executives, business people, and customers, but they never do. Having one person serve as a communicator and a translator was invaluable. After the requirements are communicated by the Product Owner to the development team, and requirements are validated through questions asked by programmers and testers, and answered by the Product Owner, requirements become part of the shared knowledge of the whole team. Members of the development team should be willing to share knowledge about requirements and be interested on having them clear, so that they could also serve as validations points of requirements.

Do you not need one person in charge of validation to make sure it happens? Agile validation is decentralized, meaning that no single person is responsible for checking or ensuring that it occurs. This is not a problem if every team member is pitching in. Mechanisms for ensuring accountability, without assigning blame can be helpful here.

What if people do not ask questions? Agile validation depends on people sharing knowledge and asking questions to make sure that requirements were understood correctly. But people do not always ask. It could be due to situational factors, e.g., distractions on a given day, or personality factors, e.g., a coder who is not social. In our case study, the Scrum Master frequently has to help people communicate to each other or arrange a meeting where people can talk (sometimes just by reminding others to "go talk to each other"). For this type of validation to work, teams should assure an open environment where asking and answering question is considered a fundamental practice.

This process does not produce any records showing that validation has been done. A working system is the best evidence of effective validation. Generally, people want work records to protect themselves when things go wrong. In such cases, a record of validation is useful for creating culprits and victims, but not at all helpful for fixing problems. Effort would be much

better spent on creating working software rather than creating records for contract negotiation or worst-case scenarios.

6.2. Implications for Practitioners of Agile

Lessons can be learned from the Easy Retirement experience, whether or not one intends to adopt agile validation or even agile itself. Some of the practices at the field site suggest improvements to requirements engineering and validation that are generally applicable.

Think about test cases and acceptance criteria during requirements elicitation. Good requirements can be mapped easily onto test cases and test cases can reveal gaps in understanding of requirements. Consequently, thinking about test cases in early requirements can improve requirements. It can also be beneficial to have a tester participate in requirements engineering. This idea of early validation is not new [11] and it has been already proposed by other researchers [12]. Similarly, the idea of performing Validation and Verification early and during the whole development cycle has already been proposed [13], [14].

Encourage knowledge sharing and question asking. Team members should be encouraged to share knowledge and ask questions about requirements throughout the software lifecycle. Questions can be helpful even when using the Waterfall model, especially in the early phases where detecting and fixing a misunderstood requirement could be less expensive and time consuming. Asking questions about requirements will help team members gain a better understanding of high-level requirements and share a common understanding of requirements. This common understanding is key to have all team members participating in a collaborative validation where every member can trigger validation of requirements or validate them.

Write test cases even for the obvious success scenarios. Agile processes depend on continuous feedback and frequent conversations. Additionally, comprehensive documentation is avoided, sometimes at the risk of not having enough documentation. As a result, there is a great deal of tacit knowledge on agile projects, including requirements that seem obvious at the time. However, these requirements may not be obvious to team members as the project ages or to newcomers to the project. Consequently, tests should be written for the simple, obvious scenarios too. Test cases are actually part of user stories and provide high-level details.

6.3. Implications for Validation Researchers

Validation has been traditionally defined as "The process of evaluating software at the end of the software development process to ensure compliance with software requirements." [13] This definition was extended by Boehm to include the activity of determining the fitness or worth of a software product for its operational mission. This definition and extension work well for phased software process models as Waterfall where requirements are gathered up front, validation is based on a requirement specification document, and there is one main release of the product.

This traditional validation process does not include activities that ensure that a software system is being built to the customer's satisfaction in the absence of a specification document. There is also no room for artifact-activities, such as user stories, test cases, and conversations. Yet some form of validation must be taking place, because software is being built successfully using agile. Perhaps it is time to broaden the concept of validation to make it applicable to a wider variety of software processes.

7. Limitations

We are aware that our study has some limitations. The data reported in this paper is based on observations and interviews conducted in one company. Our findings cannot be generalized to other companies due to the specific settings of Easy Retirement, the nature of the software product, the organizational culture of the company, among other factors. However, our findings could be helpful for agile practitioners that use similar agile methods and for researchers working on requirements validation.

This paper reports on data collected and analyzed using qualitative methods. This work would be improved by future development of quantitative metrics to supplement the qualitative ones.

Although our study has some limitations, the results obtained represent the findings of an initial study and provide us with some useful empirical data to understand how requirements validation is done in agile environments.

8. Related Work

Requirements validation for agile teams has not been widely explored in the literature. There have been some studies on agile requirements and many others on traditional validation of requirements. There has been

little research on validation of agile requirements and we know of no other field studies on this topic.

Eberlein and Sampaio do Prado Leite [15] in their position paper studied agile requirements through the lenses of Requirements Engineering (RE). They suggested some ways to improve agile requirements verification using traditional RE techniques such as the use of formal models and inspections. The authors recommended the use of checklists to inspect requirements. In our field study, we observed that the software team used user story checklists to validate requirements.

Martin and Melnik [11] examined the close relationship between requirements and testing. The authors argued that the early writing of acceptance tests cases should be used as a requirements engineering technique. Thus, tests cases can specify system behavior and the behavior can be verified by executing the tests. Similar to Martin and Melnik's statement, we observed that the software team at Easy Retirement writes high-level test cases to detail requirements and then the testers write acceptance test cases. Later, the acceptance test cases are run to validate the requirements. In our field site, the acceptance test cases are written in Fit [7] style. Martin and Melnik recommend this style because it is easy for the stakeholders to read and write.

Kovitz [16] surveyed the skills that support phased and agile requirements. He identified six skills for agile requirements. In our field site, we observed all six skills but four were relevant for requirements validation: breaking big things into tiny things, writing meaningful tests, conversation, and tools for fast cycle times.

Some empirical studies have also been done in agile requirements and agile in general. Cao and Ramesh [17] conducted an empirical study of 16 software companies and identified seven agile requirements practices. While this study does not focus on the validation process itself, it does mention characteristics of agile validation also identified in our study, such as early and constant validation and frequent review meetings to validate requirements. Sharp and Robinson [18] reported an ethnography on a company working with Extreme Programming (XP) [19] with the aim to understand the culture and community of agile. Similar to our study, they also found that agile teams have a shared purpose, understanding, and responsibility. They found that communication was mainly face-to-face and the only documentation they observed was user stories. However, we found that agile teams also use user stories checklists, high-level test descriptions, and automated acceptance test cases.

Similar to our study, Chau et al's [20] comparison of knowledge sharing in agile methods and traditional methods reported that the interactions among team members are key to knowledge sharing and that agile techniques rely on communication and collaboration to share tacit knowledge.

9. Conclusions

In this paper, we reported on a field study of the requirements practices of an agile software development team. Our analysis focused on requirements validation, because we were struck by the gap between conventional, academic concepts of validation and what we found the team was doing. In the software engineering literature, requirements knowledge is captured in specification documents and requirements validation is an activity that relies on this detailed document. Such an artifact is rare on agile projects, which do not value comprehensive documentation. Instead, we saw a set of artifact-activities that were centered around the basic unit of work in agile, the user story. A user story consists of a written reminder, test cases, and conversations, and which means that it is both something that is created and something that is done. At Easy Retirement, the user story was also the basic unit of validation. The conversations, test cases, and written reminders served to help team members validate their knowledge and understanding of the requirements.

Agile validation at the field site was continuous and collaborative. Validation was done even before the iteration starts, during planning meetings, and throughout the entire iteration. Since the basic unit for requirements was the user story, validation was also performed per each user story. The user stories served as a focal point for gathering details, giving context to conversations, and relating test cases. Agile validation practice depended on team members working together constantly to ensure that the software being built met the customer's expectations. No one person was in charge of validation, but everyone was responsible for getting it done. Agile validation relied on team members thinking critically and asking questions about the program and test cases. While agile validation is compatible with values in the Manifesto for Agile Development, such as valuing individuals and interactions, it is not trivial to implement. We feel that both agile practitioners and academics studying validation can learn from requirements knowledge techniques used for validation at Easy Retirement.

10. Acknowledgements

Thanks to all the participants and supporters at Easy Retirement. This research was possible due to their

generosity in sharing their time and work with us. Thanks to Marisa Cohn for her help in data collection and to Anahita Fazl for helping us with transcriptions. This research was supported in part by a grant from the Agile Alliance Academic Research Program.

11. References

[1] S. E. Sim, T. Alspaugh and B. Al-Ani, "Marginal Notes on Amethodical Requirements Engineering: What Experts Learned from Experience," *in Proceedings of Requirements Engineering Conference (RE 2008),* pp. 105-114, 2008.

[2] B. W. Boehm, "Software Risk Management: Principles and Practices," *IEEE Software,* vol. 8, pp. 32-41, 1991.

[3] M. Cohn, *User Stories Applied: For Agile Software Development.* Addison Wesley, 2004.

[4] K. Schwaber, *Agile Project Management with Scrum.* WA, USA: Microsoft Press Redmond, 2004.

[5] L. Crispin and J. Gregory, *Agile Testing: A Practical Guide for Testers and Agile Teams.* Addison-Wesley Professional, 2009.

[6] B. Leuf and W. Cunningham, *The Wiki Way: Quick Collaboration on the Web.* Addison-Wesley Professional, 2001.

[7] R. Mugridge and W. Cunningham, *Fit for Developing Software: Framework for Integrated Tests.* Prentice Hall, 2005.

[8] J. Lofland, D. Snow, L. Anderson and L. Lofland, *Analyzing Social Settings: A Guide to Qualitative Observation and Analysis.* Belmont, CA: Wadsworth/Thomson Learning, 2006.

[9] M. Fowler and J. Highsmith, "The Agile Manifesto," *Software Development,* vol. 9, pp. 28-32, 2001.

[10] D. Astels, *Test-Driven Development: A Practical Guide.* Upper Saddle River, New Jersey: Prentice Hall PTR, 2003.

[11] R. C. Martin and G. Melnik, "Tests and Requirements, Requirements and Tests: A Möbius Strip," *IEEE Software,* vol. 25, pp. 54-59, 2008.

[12] J. C. S. P. Leite and P. A. Freeman, "Requirements Validation through Viewpoint Resolution," *IEEE Transactions on Software Engineering,* vol. 17, pp. 1253-1269, 1991.

[13] B. W. Boehm, "Verifying and Validating Software Requirements and Design Specifications," *IEEE Software,* vol. 1, pp. 75-88, 1984.

[14] D. R. Wallace and R. U. Fujii, "Software Verification and Validation: An Overview," *IEEE Software,* vol. 6, pp. 10-17, 1989.

[15] A. Eberlein and J. C. S. P. Leite, "Agile Requirements Definition: A View from Requirements Engineering," *in Proceedings of the International Workshop on Time-Constrained Requirements Engineering (TCRE'02),* 2002.

[16] B. Kovitz, "Hidden Skills that Support Phased and Agile Requirements Engineering," *Requirements Engineering,* vol. 8, pp. 135-141, 2003.

[17] L. Cao and B. Ramesh, "Agile Requirements Engineering Practices: An Empirical Study," *IEEE Software,* vol. 25, pp. 60-67, 2008.

[18] Sharp, "An Ethnographic Study of XP Practice," *Empirical Software Engineering,* vol. 9, pp. 353, 2004.

[19] K. Beck, *Extreme Programming Explained: Embrace Change.* Mass: Addison Wesley, 2000.

[20] T. Chau, F. Maurer and G. Melnik, "Knowledge Sharing: Agile Methods Vs. Tayloristic Methods," *in IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03),* pp. 302-307, 2003.