2011 Agile Conference

The home ground of Automated Acceptance Testing: Mature use of FitNesse

Børge Haugset
NTNU
Trondheim, Norway
borge.haugset@sintef.no

Geir K. Hanssen
SINTEF
Trondheim, Norway
geir.k.hanssen@sintef.no

***Abstract*— This article describes a group of developers and how they successfully use FitNesse. The study is based on interviews with 4 consultants, and shows that automated acceptance testing was used in two steps every iteration: a specification step where it helped communicate requirements, and a verification step where it helped developers in the process of producing and maintaining software. The study indicates that automated acceptance testing may be a viable strategy, but that it must be used with care. Communicating requirements throughout the lifespan of the software is just as important as automating the tests, and AAT can help with both of these issues.**

Keywords-component; Automated acceptance testing, AAT, Fit, FitNesse, agile testing

The creation of high-quality software requires firstly developers and customers to create a shared understanding of the customer needs, secondly to implement a solution and thirdly to evaluate the solution with respect to the needs. Automated acceptance tests (AAT) have been proposed as a way to provide a common understanding of the needs of a system and to repeatedly and automatically test software at a business level – an approach that has gradually gained a lot of attention. One of the most used tools for this development style is called Fit, and was in [1] suggested as a way of improving the communication between the customers and developers. This was based on a common and direct description of the business requirements through concrete tests all parties understand and agree on. Ricca et al. summarize Fit and FitNesse (which is partly based on Fit) nicely, so we borrow their description [2]: *"Fit (Framework for Integrated Test) is an open source framework used to express acceptance test cases and a methodological tool for improving the communication between analysts and developers. Fit lets analysts write acceptance tests in the form of simple tables (named Fit tables) using HTML or even spreadsheets. A well-known implementation of Fit is FitNesse, substantially a Wiki where analysts/customers can upload requirements and related Fit tables. A Fit table specifies the inputs and expected outputs for the test."*

Since Ward Cunningham introduced Fit in 2002, Fit and FitNesse have now had nearly a decade to mature and to be studied. As expected the tool (and accompanied method) has gained some followers. Fit itself has laid dormant for a number of years, but FitNesse is still being developed. Other practitioners have decided that tools for automated acceptance testing simply aren't worth the cost. Interestingly, one of those skeptics is a previous coordinator of Fit; James Shore. From being a strong proponent, he sees several problems:[1] The customers don't write the tests, he claims, thereby reducing the supposedly communicational benefits. The tests also turned out to be slow and brittle, requiring too much maintenance. Some of these concerns align quite well with the conclusions from a combined literature review and case study we published earlier [3, 4]. We described that it seemed somewhat inappropriate for customers to actively express their requirements in this tabular format; they rarely did so. Ricca et al. have however shown that using FitNesse tables as a medium for communicating with customers enhances the understanding of the requirements [2]. So why do developers rarely use them this way?

A decade is not a long time for methods to get established and find its right place. Research is still ongoing, and the practitioners are themselves discussing this. Even though parts of the previously supporting community have abandoned the belief in AAT, there are still supporters. Are these supporters in denial, or do they still have a case? We suggest, quite typically, that there is a middle ground [5]. AAT might be a beneficial tactic for some projects or parts of them, and not for others. The problem lies in knowing on which shelf AAT belongs, and this has not been dealt with so far by the research community. We want to contribute to solving this, and ask the following research question:

"How and why is FitNesse used in a successful and mature software development project?"

We think researching and answering this question is important. In the years since Fits introduction many companies have tried and failed to adopt this method. The research community should be able to understand if their lack of success is due to implementing the proper method in a wrong way, if the method isn't suitable under these circumstances, or if it indeed not a good method at all.

This article is a story about a group of developers who think they successfully use AAT. We believe that by studying such a company, and relate this to findings in research, we can begin to carve a niche for AAT in research. The case is investigated by talking to the developers to get their story. The developers are the ones who know about the varieties of methods and development practices that exist. We want to

[1] http://jamesshore.com/Blog/The-Problems-With-Acceptance-Testing.html

capture the intentions of using AAT, and their impressions of its usability. These findings will provide feedback to us and other researchers about where to seek further understanding.

I. Literature

In 2008 and 2009 we described the status of research on AAT as well as providing insights from a case study [7, 8]. We found that some of the proposed benefits were realistic, but that it was not by any means a silver bullet. We refer to these articles for richer descriptions and references, and only sum up the major findings here:

Fit seemed to be an easy tool to understand and use, at least in quite simplistic student experiments. Some findings pointed out that developers were often the ones writing the Fit tests, getting the customer to join in or even write them themselves was difficult. Further, there was an extensive use of added documents, suggesting that mere Fit tables were not often enough to convey the whole requirements. One article reported that the conversation around the Fit tables were worth more than the later use of the tables themselves. It was also shown that a passing acceptance test not necessarily meant the work was finished. All in all we found that AAT seemed to have some promising benefits. One should however not believe that it would be a solution for all, and little was told about who would benefit and how to use it.

Since the literature review in 2008 there have been relatively few empirical studies of AAT. Some work has emerged with interesting results that extend the knowledge described previously. Ricca et al. have through a set of student experiments shown that the sheer presence of Fit tables (i.e. the process of creating them) improves the understanding of requirements [9]. People with limited technological background would also benefit from this use.

One can decide not to have a full coverage of FitNesse tests. If so, Ricca et al. advice practitioners to select and cover the most complex requirements [10]. Further, the developers' experience and work organization played an important role in the result. Experienced developers (PhD students) gained more from the use of Fit tables for maintenance than less experienced ones. They also found that the benefit of Fit tables was reduced when the developers used pair programming. This can be explained by the intended interaction that goes on: The two together grasp more of the domain and requirements, and are able to build on each others knowledge. Likewise, they have a continuous code inspection while coding, so some of the regression benefits from Fit would be lost as well. Ricca et al. found that the good thing about Fit tables lies in their simplicity, and that you should exclude cases in which use of such tables would become too complex [ibid]. As a final note, they state that the use of Fit does not alter the comprehension effort and/or maintenance effort. This means you could get the other perceived benefits from Fit – understanding requirements and keeping the code proper – without adding any extra cost.

A few studies look at companies using AAT, and completes the experiments by Ricca et al. Park and Maurer describe a case that contains very complex system with a plethora of acceptance tests in the form of huge Excel sheets [11]. They confirm the communicational importance of AAT for mediating domain knowledge to the developers, also acting as the feedback to see if their understanding was correct. Further they find that the medium selected for specifying the acceptance tests is important. By using Excel, a tool already used by the domain experts, they gave them more familiarity in how to express their requirements correctly. Lastly Parker and Maurer describe how the acceptance tests need to be extended with documentation showing the larger picture and business workflow. While this is nothing new, this underlines the need for communicating the larger-than-tables picture that constitutes the developer project. They do not focus on how the requirements are established, the communication surrounding these processes and the validation of them. Gallardo-Valencia and Sim describe a case where validation of the product was performed continuously; during pre-iteration, iteration planning and intra-iteration, mostly through conversations [12]. This validation involved all members such as Product Owner, programmers and testers. Like Ricca describes in the use of Fit helping to focus gathering requirements, this team used validation as a way of making sure they were on the right track. They suggest that practitioners should think about test cases and acceptance criteria during requirements elicitation, encourage knowledge sharing and question asking, and write test cases for even the most obvious success scenarios.

II. Research method

A. *Case study method*

This article is based on four in-depth interviews with developers made in a consultancy company in Norway, referred to as Konsult. All interviews were done within the same group of developers, working together on a set of incremental and parallel sub projects for an external company. Two of the interviews were made in cooperation with a Swedish research team, studying the alignment between requirements and testing the requirements. This particular case was chosen because the use of AAT is an interesting practice that links requirements directly to high-level tests. The interview guide is attached. Some parts of the interview guide were omitted during the interview, as they were not applicable to this specific software development process. These were mainly issues like key performance indexes and product line engineering. In the attached interview guide questions not relevant to the case in question are omitted.

All interviews were semi-structured, to allow the respondents to reflect and actively engage in the questions. Interesting tangents were followed up. The interviews lasted between 55 and 90 minutes. All interviews were transcribed and tagged in NVivo™ (a tool for textual data analysis) by both authors creating a shared understanding of the case, identifying insights of particular interest and relevance to our research question. Several tags can describe the same content, and gradually we build a set of main themes. In the next chapter we provide a description of these. Both here and in the discussion we provide relevant quotations from the interviews. We wish to provide a rich context in order to give the reader the possibility of seeing yet more things than we have so far described, or approach the topic from other angles.

According to Yin, case studies are the preferred research strategy "*...when a «how» or «why» question is being asked about a contemporary set of events over which the investigator has little or no control.*" ([13] p. 9). This is a very adequate description of the context of our study.

B. Limitations of the study

The study design has some limitations. The obvious question regarding case studies is its generalizability. We have interviewed a low number of developers from a single case. The interviews have been long, and this has however given us the opportunity to gain deeper insights than more and shorter interviews would have provided. We believe there are important insights to be gained from providing a rich description of a single case, and that a broad generalization in any case cannot be gained from extending this to two or a few cases. No software development company or project is equal to the other. Detailed industrial studies of automated acceptance testing and how frameworks such as Fit and FitNesse are used in practice – and the enabling context - are few. We believe that our findings and conclusions are of practical importance to other similar software organizations that consider using automated acceptance testing and other researchers who address similar topics. Also, focusing on a single organization or case enables a more detailed focus of the study, a feature we believe is important to spot and exemplify important effects and prerequisites. The richness of this particular case will later be extended upon by interviewing other roles, observing different parts of their work and reviewing documentation.

III. Results

A. The case context

The case is about a consultancy company (Konsult) that was hired by a branch of a large logistics company (Kunde). Konsult is one of the most highly regarded IT consultancy companies in Norway. They have long experience in using agile development methods, and are developing open-source tools for AAT themselves. Konsult was hired more as a developer team than as a developer project, a resource to make use of in different projects. They work on-site at Kunde.

The interviews were performed from roughly a year after Konsult started working for Kunde and over a period of one and a half years, thus providing insights into a mature environment. This section of Kunde had little experience with agile software development and on-site consultants prior to this. They had also felt the downside of outsourcing (spending a long time not getting what you need). One developer felt that this gave way to an initial lack of customer trust. While at Kunde, the developers started working on several projects running test-driven Scrum.

Konsult during this period doubled their size at Kunde. They now consist of two teams, with a total of ten developers and two interaction designers. Kunde supports the development with one project leader, product owners and domain specialists. Right now there are four ongoing projects, two per team. All of the projects are purely test-driven at a unit level, and most of the code is pair programmed. They describe unit level tests in a BDD-format[2], tests that are only to be used by developers. All projects have some degree of automated acceptance testing written in FitNesse or Cucumber[3]. We have focused on the projects with an extensive use of FitNesse, so our following breakdown is from that category. The project with the most extensive use of FitNesse, a freight guide, currently has 1100 lines of tests, the average test being 15-20 lines long. The developers were very busy at the time of project start of one project, so Kunde agreed on scoping the project. There was no set deadline or cost for this specific project. One developer had this as his first ever development project. In this project they started out by spending the first month having workshops, trying to find out what to build and decide on technologies. After 4 weeks they produced the first service with some customer value. After 4-5 months they had created a system with enough features to make it usable for the 'real' end users of this application, other web shops that use Kundes services.

B. When to use FitNesse

The developers use FitNesse as a specialist tool, and not necessarily all the time. They are continuously evaluating the emerging requirements in all of their projects to see if FitNesse is valuable, and this depends on many factors: In most situations a requirement is basically shifting data: *"...if you only move data from A to B, and there's very little business logic, you only show data from another system... then it typically isn't that good."* If so, they never use AAT. Neither is AAT used if the requirements are difficult to define in a tabular format. If the requirements contain a lot of complex logic, perhaps requiring formulas, FitNesse may be a good approach. The key is reducing uncertainty: *"If there is no room for misunderstanding, we don't bother [defining FitNesse tests]. [However,] If we are not one hundred percent certain we understand the problem, we do it in FitNesse."* He also pinpoints what he thinks is an important factor for using FitNesse for the complex and difficult requirements: *"...But FitNesse is gold because you have really documented what business probably would have done non-documented. They would have sat there, punched a little*

[2] Behavior-Driven Development, a term coined by Dan North. Mostly used as an automated acceptance-test-driven approach. For a description see http://behaviour-driven.org/

[3] Cucumber is the latest development within BDD, focusing on writing tests in plain text. This follows a 'given – when – then' scenario. See http://cukes.info/

back and forth…to verify something. Or they would probably not even have tested that much."

C. *Iteration work*

Business requirements are discussed in bi-weekly customer meetings. The most important requirements are selected for the next iteration. They are specified in a table format at a white-board, and discussed until agreed upon. The developer then turns these tables into FitNesse tests. The concept of a customer driving the requirements process by writing FitNesse tables him/herself was by one developer called *"wishful thinking",* stating it would take a very technical customer. The developers at times ends up seeing some errors when they start programming, and have to go back to discuss this with the customer. The importance of sitting on-site was evident in this process: "*It is a lot of back and forth, and if I had to write an email and wait a couple of days for a reply or take a taxi to the customer, it would never work. It's clearly a success factor."*

The developers all express how they think this domain (logistics) is very well suited for expressing business requirements in the form of Fit tables, especially in the freight guide. The tables are sent as Excel sheets to the customer who fills them with important data like edge cases, while starting coding to make these tests pass. As the customer returns the test data these are plugged in. In the end of each iteration there is a demonstration.

For the longest ongoing project the demo was strictly not necessary. One developer stated: *"…the reason they trust us is that that they have all the tests, and in the demo we show all the tests, […] so it's like there's no question about it, it's like… they hardly even need to know about it.*

In general, FitNesse was seen as giving many benefits during development. One of the problems in this group was the steady noise of differing requirements from the various 'customer' groups (managers, IT, documentation…): *"But it makes it hard for us to stick with like: This is the requirement, we have done this. We always have to think about switching requirements and that's why automated parts of test are very good. Because what the test says is actually what we've done. And if the next meeting refines the requirements from the previous meeting, then the tests must [change]."* He thought that making changes visible was the main benefit of using automated acceptance testing.

D. *Manual testing*

Being an agile project and implementing Scrum [14], testing was done partly as a manual process where the developers once per iteration would run a demo. The customer would then provide immediate feedback with respect to their requirements for the software. In addition, these meetings opened for even more detailed feedback. One of the developers explained to us how the testing was done: *"…what we regard as tests, as good programmers, are things that can be automated, and it is difficult to test a goal or a vision automated, so then it's more about agile, continuous feedback from the customer, continuously showing what you make etc… they [the customer] at all times can correct the goals. So in that regard I'd say that the demo and everything we have of customer meetings, follow-up we do, helps to direct us towards the goal…"* Another manual test was performing activities that would be too difficult to automate, such as checking cross-browser compatibility. In the freight guide project this particular task could take as little as one hour per iteration, and the customer had never performed any manual testing.

E. *Project handover*

The project with full coverage of FitNesse tests, freight guide, had no specific handover. It started out by developers and Kunde agreeing on scoping the project, leaving several parts of the wished for features out for later. The work gradually progressed, steadily implementing more and more features. One developer doubts they will ever have a handover. As he says, it is a very technical freight guide where no individual customer has a relation to the complete project. He instead agrees on it being sort of an iterative handover through the demos they hold. These demos can be quite technical, which in turn could have lead to a knowledge spread: *"We are proud of what we make in every iteration. We want to show, it is always a dilemma if we should show them XML web services we have made when we make a cool JSON mapping, but […] shows it to them because they [the developers] themselves think it is so cool, so in that sense they [Kunde] have much better knowledge about the system than they would ever be able to with one big handover when we were finished."*

He later says that he believes Kunde hasn't had a formal handover largely because there's been established a great deal of trust: *"And I also believe that us not having a formal handover, that they haven't checked our work to a large degree is based on them trusting us for the work we have done. They feel that they know what we have created, what the guide is. This is what you get when you show them JSON services and web service calls and such. They even know parts of the system that business people don't know about."*

Another developer says they have approached the handover issue by constantly aiming for keeping things simple. The complex areas are where you will find the most tests explaining border cases and the like. When all tests pass, it is ok for the customer and it can be deployed.

F. *Collaboration and co-ordination*

FitNesse tests are developed collaboratively, but from different viewpoints. Customers and developers agree that the tables represent the correct requirements. One of our subjects explained: *"In FitNesse you have data separated from code which means that you actually can show this to a business person, unlike Java code which you just can't show"*. This means that the developers do what they do best, which is develop code and maintain it, and that customers do what they do best, which is to describe their business requirements. The Fit tests become communication artifacts that supports the agile development process with the release of frequent increments. We also see that the Fit tests become valuable (automatable) documentation of what the customer side would do anyway. Or perhaps even worse, as one developer said, the customer might not actually have performed as thorough testing as the tests that they filled into the FitNesse tables.

Discussing the domain and requirements is done not only during specified meetings, but the physical proximity to the customer resulted in a continuous discussion. When asked if the developer team in reality included the whole floor, a developer responded *"Yeah, clearly. I think some of them have grown pretty bored with us in periods…"* The communication has been focused around the FitNesse tests, instead of talking loosely about the domain: *"This forces both us and them to think in the same way, we establish a good common understanding by sitting down talking about the tests."* The customer was not actively writing Fit tests, this was mostly the developers' game. That did not mean customers had not heard of it: "*They are really very conscious about it, as we have worked hard to make them so. We bring printouts, showing it do them, and such, but for them FitNesse is what we use, their link to the FitNesse tests are the Excel sheets. They are very well aware of what the Excel sheets are used for."*

No employee at Kunde had full domain knowledge, but the developers knew whom to contact. One developer said he found it easier to talk to the sales people rather than IT people, as they tended to get too technical. Sales were more value-oriented and focused about the business layer. All in all the developers seemed to feel comfortable with their placement on-site: *"Yes, we feel that we are a very well integrated part of Kunde, I know most of the people on this floor by name. Or [at least] the ones I have had contact with project wise. And that starts to be a lot of people, I have noticed, which is very pleasant, to know who in the organization I need to relate to/ask about something before I start on it, so I don't have to go through three or four persons and send ten emails every time I want to ask a simple question."*

Sitting on-site of course also lead to the point of never letting your guard down: "*This might sound horrendous, but I think… it requires developers that like their job and are self-driven. But it is extremely visible if I sit down and read news on the Net for six hours; I still sit over there. It's not cool to do that when eight people an hour walk by, and they pay my bill."*

IV. Discussion

The overall impression from our case study is that using AAT, with tools such as FitNesse, seems to be useful in certain situations and serves certain needs well. In most situations the developers found no need to use AAT, and we believe that this case is not unique in this regard. We find that it is very important to be aware of the home ground of automated acceptance testing to make the most of it. Almost as a general rule; tools such as FitNesse, which was used in our case, rarely fully replaces creative human work. We however believe we can bring nuance to this realization by looking into and understanding situations where the use of a tool may be useful and not – and some of the apparently underlying reasons.

First of all, choosing a level of specification depends on ability, complexity and uncertainty. Not all requirements are possible to describe in FitNesse. If the requirements are simple, our interviewees stated it would cost more than you gained. If solving the requirement would include complex logic, FitNesse was an obvious choice. Some business rules are easier to represent in FitNesse, and thus automate, than others. Once developer explained that *"…the further up [in complexity] you get, the higher level of the artifact, the harder it is to test"*, and automate, we may add. He also made a note that *"it is difficult to test a goal or a vision automated"*, telling us that humans still need to consider the software's ability to fulfill the intention and high level goals of the system. This is a task that has a certain level of subjectivity to it. The use of AAT was impacted by the complexity of a requirement, and the chance of developers misunderstanding it.

This resulted in the developers only scarcely deciding on using FitNesse to describe requirements in most of their projects. One project that they found particularly well suited had a full coverage of FitNesse tests, and this resulted in being able to deploy often with almost no customer verification. Gallardo-Valencia and Sim suggest that developers should write test cases for even the most obvious success scenarios [12]. While the developers we interviewed do not think addressing all requirements as FitNesse tables is a good idea, *all* requirements are described and developed using test-driven at the unit level. As such we feel that our case refines these suggestions.

One developer explained how they worked: *"First we find all the requirements, then we define where to place the automated tests for this, here we typically use FitNesse because we work with an application that is extremely easy to test in FitNesse, calculations and all sorts."* The developers describe the domain as *"difficult"* and *"complex"*, but calculations about logistics can still be an area that is easily described using the table structure in Fit. It seems that the value of the conversation around the Fit table is very high when the domain easily fits into this table format. By selecting the more difficult cases for AAT they are able to quickly cooperate with the customer to gain a solid understanding of the requirement. This is in accordance with, and extends on, how Ricca et al. state that using FitNesse as a mediator can help in understanding requirements [9, 10, 15]. They also get the benefit of automating high-level testing of the most important parts of the system.

The use of AAT – as any other type of testing – is based on the need for feedback to validate the solution up against requirements, and for agile development that feedback should be frequent. The idea of automation is that it is costly, boring and error-prone – it will also free time for developers to do other things. One respondent described a key benefit of using FitNesse: *"I can tell you briefly why we use a lot of automated acceptance tests. It is because we can release every second week without involving the customer. [...] We have a set of automated acceptance tests. When they pass we can release. There are no questions about it. So we always do."* This is of course a special case, and most of their projects were not as automated as this one. It still shows that FitNesse brings with it the possibility of close to full automation, and we have not found any other projects like this described in literature.

For tools like Fit and FitNesse, where requirements are expressed as tables (input + expected output) there is the additional effect that tests may be used or seen as well structured documentation of requirements. One developer says he didn't create documentation, like *"...Word documentation, because I have yet to see a single system documentation that actually is true. So we let our code do our documentation."* In this case the Fit table *is* the documentation for complex issues. More interestingly, we see from our case study that such tables are used partly as a "medium" for communication, back and forth until a point where the customer has managed to transfer his possibly quite tacit view of requirements. One respondent explained how he would go about defining the requirements for a calculation feature:

"We sit down with him [the customer], and say that 'Let's create a price for this product'. We show them the suggested table, and see if this is enough to get a correct model: if he is able to get the correct price by using the table we have created. Usually this isn't the case, so we talk a bit about that. We end up agreeing on a table. He leaves, and we create this in FitNesse. We then create an Excel sheet, and send this back to him. He fills this in, and sends it back to us. This might go back and forth a little, as we might find some deviation in the code... That's how we make them."

Fit tables (or Excel tables) were used specifically as the mediator, and for a reason. It was seen as a way of focusing the attention away from loose talk to gain a result in communicating about requirements, concurring with findings from Ricca et al. that Fit proved successful in this phase [2]. One developer claimed the customer failed to grasp the tool for two reasons: While reading the tables is easy, editing them is impossible unless you are a developer. The customers were not involved in writing the actual tests, and had little knowledge of how to use it. Using copy and paste to Excel meant having one less tool for the customer to learn. The developers also sat on a guest network, making customers unable to access the FitNesse servers. Rather than solving these problems, they let Excel be the mediator and rather sat down with the customer when they needed them to look at FitNesse itself. The use of Excel as a mediator resembles Park and Maurer's case [11]. Our case also resembles these studies in how developers and customers needed additional information about the context in order to grasp the full picture. In our case this is done by performing their work on-site, with heavy continuous interaction with key personnel.

The analysis of our data has inspired us to take "a knowledge view" on the development process. Software, which is codified knowledge, is created in two repeated steps:

Step 1: (Specification) Knowledge of the business and problem domain is transferred from the customer to the developers and,

Step 2: (Verification) Knowledge about how the software meets the requirements (AAT test outcome) is communicated back to the developers

These two steps constitute a cyclic learning process where iterations create the space and the arena to exchange knowledge and the increments define the scope and context of the knowledge.

Step 1 is basically performed through a constant dialogue between customer and developer [16]. The developers did not only communicate with the customers through meetings. A comment from one developer about how he thought the customer has grown bored with them. Whether true or not, this shows there has been a lot of communication also inside the iterations. We think this close cooperation is key to their success. The developers not only had to cooperate with the product owner, but also a plethora of other actors with their own agendas of what were the most important requirements. One developer replied that this was difficult: *"I find it very interesting in many ways. But it makes it hard for us to stick with like: This is the requirement, we have done this. We always have to think about switching requirements and that's why the automated parts of testing are very good. Because what the test says is actually what we've done. And if the next meeting refines the requirements from the previous meeting, then the tests must change."* AAT was used as a way of easing the cost of constantly changing their code.

From our case we have learned that this knowledge transfer is enabled and driven by a very high proximity. This lowers the communication threshold that technical remedies such as email or telephone imposes. This means that step 1 can be done as a dialogue, combining socialization (creating a shared knowledge of the problem and business domain) and externalization (creating a suite of acceptance tests as tables) [17].

Step 1 is not at all about automation. If the developers judge the requirements to be very complex or easy to misunderstand they see the Fit tables as a way to interact with the customer, and guide the attention like Ricca et al. reported was productive [10]. The Fit tables start out as a means of communication about especially complex and unclear requirements. Through learning, trial and error, through disagreements or arguments, they become a rich and explicit description of the same requirements, a description that can actually be used for development. For some kinds of requirements: "*It's very good for many aspects of the project, which are like the integration with the other vendors. How the business side wants this data to be displayed for the end-user. It's a useful tool. But for us in planning and structuring* *how we have to plan tasks, we don't want to just pick one task and just do that and be completely blind about the other tasks. We need some sort of oversight and this tool isn't very..., well the tool itself is just one tool, but it's sort of limiting."* This comment pinpoints being aware of the larger structures while looking at the details. These developers seemed to grasp this, helped by their continuous interaction with their customers.

Step 2 is about transforming requirements in the form of shared knowledge and the tables into code - in practice a new increment of the growing solution - and to verify whether the solution answers the requirements or not. We think this can also fruitfully be looked upon as communicating requirements. This verification is done *both* as a manual and automated process. Manual testing is done by running demos of the solution and collecting response from the customer. It also involves high-level tests not possible to automate so easily, such as the cross-browser compatibility. Automated testing is the execution of FitNesse-tests using the tables. These results are fed back to step 1 and drive the dialogue further. We believe that building shared knowledge and explicit knowledge in the form of tables has enabled Konsult to drive a fruitful process with a rich dialogue with Kunde. We also think this domain was very well suited for this approach.

Step 2 can be seen as the 'extra benefits' Ricca et al. [ibid.] mention you get from investing not only in Fit for communicational purposes but also for code purposes. It gives way to a number of regression benefits. It automates the validity of customer requirements, maintains quality of code and ensures easy deployment.

Being able to automate is not only a technical question, but also an issue of trust. This case is a good example of how the customers' trust in the developers has been built by demonstrating early value and being able to maintain this over iterations. A customer cares about the result; it takes trust to allow deployment without extensive manual testing. They allow this because they trust the ever-growing system to be up to speed because of the automated tests and the tight communication.

Our insights lead us to suggest a simple model that defines the "home ground" of AAT and explains how it can be used to support a development process based on a constant dialogue between the customer and the developers.

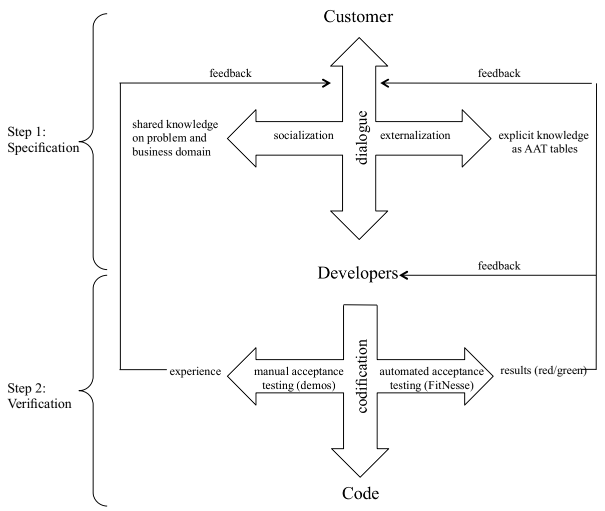

Figure 1 - A two-step knowledge creation process

We believe that step 1 relies on good communication and trust, which through iterative development creates a growing shared knowledge between the customer and the developers as well as a growing suite of automated acceptance tests in the form of FitNesse tables. We see that good communication comes partly from proximity and "access" to customers [6].We see that trust can be built through investing time (which is costly) to "get to know each other". An investment that may at first seem to be a waste (one developer claimed spending the first month without coding didn't help with trust) can give a productive environment in return. Trust in this project was also gained by having dedicated developers on board. An example of that is how one developer coded open source solutions for Kundes customers in his spare time, in order to gain speed in adoption.

We believe that step 2 may partly be automated but that it still needs a manual component to work well. Yet, we think AAT tools play two important roles in agile development in this phase: Firstly to support the dialogue between the customer and developers and secondly, to enable frequent releases of the solution under development and thus establish a constant velocity and stability in the development project.

V. Conclusion

In this article we have sought to answer the research question *"How and why is FitNesse used in a successful and mature software development project?"* through in-depth interviews with developers from a group of projects who feel they have succeeded in identifying the home ground of AAT using FitNesse.

Our main impression is that use of AAT is just as much about communicating requirements as it is about testing and automatic verification. Using AAT for communicating requirements takes place both between the customers and developers, and among developers. We believe the developers' use of FitNesse can be summed up by describing the development as two phases that take place each iteration of the project:

In the *specification phase* there is an explicit knowledge transfer from customers to developers, where requirements are described as FitNesse tests (*externalization*). Choosing to describe a requirement in FitNesse is done based on analyzing the requirement: Is it possible to describe the requirement in FitNesse? If so, is the requirement complex enough to warrant being described? The third factor relates to uncertainty, if the developers are not sure they have understood the requirement it is a candidate for FitNesse. The studies have shown that the developers think it is possible to get benefits even without having a full coverage. Neither is it necessary for the customer to actively use FitNesse, filling in data via Excel works fine. The project with full coverage dealt with logistics, so this seems a home ground for this method. Other domains may be just as good, provided the requirements can be described within the tool. The developers use FitNesse to talk about the requirements that are difficult, both to make sure they understand it and to get a proper documentation of the complex issues. In this regard, AAT helps to guide communication to evolve a shared knowledge of the domain and its quirks (*socialization*). This also includes knowledge about requirements that end up being described as unit tests rather than acceptance tests.

In the *verification phase* the developers confirm that the produced code adheres to the requirements. This is done in two parallel ways: The demonstrations (*manual verification*) align the developers and customers. Some requirements need or should be handled manually. The demos also provide experience and response to the next specification phase, in a regular agile fashion. The FitNesse tests make sure as much as possible is tested automatically (*automated verification*). As we have seen in our studies this can extend to cover all requirements, thereby reducing manual labor to a minimum. This effect is strengthened because the automated tests are able to check also older code than the newest, and provides a safety harness for new development. It also acts as documentation of requirements: agile software development using AAT brings with it a set of up-to-date acceptance-level descriptions of requirements. Agile development has generally been less inclined to describe requirements as thoroughly as this, at such a high level, and also keep this documentation up-to-date.

We do not believe that FitNesse is a perfect tool, for use in all situations. The community is also discussing if AAT is worth the cost. One developer describes this well: *"If this tool had not existed we would probably just have done it in another way, and not had any problems with that. But still, this tool did not introduce just the tool, but also the way of thinking."* We think this underlines our findings in a subtle way. Fit and its siblings have their place as specialist tools, when used with caution and understanding. Its use must be followed by not neglecting the manual feedback for requirements not testable, or the aspects of communication. There are many ways to reach a goal, and FitNesse may be one of them.

In one of the projects could the developers deployed with almost no customer interaction. This suggests that AAT can play an important role in building *trust*. Trust was also, we believe, influenced by on-site *proximity*. We cannot say that it is impossible to successfully use AAT when not on-site, but we think the communication aspect of AAT should not be underestimated. Our studies suggest there is a lot of back-and-forth when describing tests in FitNesse, and having good access to the customer makes this easier.

There are many ways to draw on this case study and point at further studies, for us and other researchers. For us this was step one, understanding how and why the developers opt to use FitNesse. The next step for us will be to look more closely at the case, and gather data from more angles. This will include a more in-depth case study at the company.

As far as we can see there has never been any research that tries to describe the use of AAT from a theoretical perspective. It seems promising to look at the FitNesse tests as canonical artifacts [18], with a set of border resources that are socially shared. Each of the actors can be seen as *communities of practice* [16], working together to solve a common task (making functional software). These theories can be combined with established models of learning, such as Kolbs model of experiential learning [19] and the SECI-model [20]. We believe drawing on work from other domains will shed light on how AAT is actually used.

Another interesting path to follow up on this study is to look at AAT as a form of continuous Requirement Engineering. Ramesh et al. have compared RE and agile RE, and describe how agile mitigates certain risks in RE and exacerbates others [21]. They did not include the use of AAT in this comparison. We believe that AAT constitutes a merge between the documentation-centric RE and the software-focused agile RE by emphasizing updated high-level documentation of requirements, and wish to extend the framework of Ramesh et al. to include AAT.

VI. References

1. Mugridge, R. and W. Cunningham, *Fit for Developing Software: Framework for Integrated Tests (Robert C. Martin)*. 2005: Prentice Hall PTR Upper Saddle River, NJ, USA.
2. Ricca, F., M. Di Penta, and M. Torchiano. *Guidelines on the use of Fit tables in software maintenance tasks: Lessons learned from 8 experiments*. in *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*. 2008.
3. Hanssen, G.K. and B. Haugset, *Automated Acceptance Testing Using Fit*, in *42d Hawaiian International Conference on System Sciences (HICSS'09)*. 2009, IEEE Computer Society: Hawaii, USA. p. 1-8.
4. Haugset, B. and G.K. Hanssen, *Automated Acceptance Testing: a Literature Review and an Industrial Case Study*, in *Agile Conference*, P. Kruchten, Y. Dubinsky, and P. Abrahamsson, Editors. 2008: Toronto, Canada. p. 27-38.
5. Boehm, B. and R. Turner, *Balancing Agility and Discipline - A Guide for the Perplexed*. 2004, Boston: Addison-Wesley. 266.
6. Keil, M. and E. Carmel, *Customer-Developer Links in Software Development.* Communications of the ACM, 1995. **38**(5): p. 33-44.
7. Hanssen, G.K. and B. Haugset. *Automated Acceptance Testing Using Fit*. in *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on*. 2009.
8. Haugset, B. and G.K. Hanssen. *Automated Acceptance Testing: A Literature Review and an Industrial Case Study*. in *Agile, 2008. AGILE '08. Conference*. 2008.
9. Ricca, F., et al., *Using acceptance tests as a support for clarifying requirements: A series of experiments.* Information and Software Technology, 2009. **51**(2): p. 270-283.
10. Ricca, F., et al., *Guidelines on the use of Fit tables in Software Maintenance Tasks: Lessons Learned from 8 Experiments*, in *2008 IEEE International Conference on Software Maintenance*. 2008. p. 317-326.
11. Park, S. and F. Maurer, *Communicating Domain Knowledge in Executable Acceptance Test Driven Development*, in *Agile Processes in Software Engineering and Extreme Programming*. 2009. p. 23-32 %U http://dx.doi.org/10.1007/978-3-642-01853-4_5.
12. Gallardo-Valencia, R.E. and S.E. Sim, *Continuous and Collaborative Validation: A Field Study of Requirements Knowledge in Agile*, in *Proceedings of the 2009 Second International Workshop on Managing Requirements Knowledge*. 2009, IEEE Computer Society.
13. Yin, R., *Case Study Research*. 2002, Thousand Oaks: Sage Publications Inc.
14. Schwaber, K., Beedle, M., *Agile Software Development with Scrum*. 2001, New Jersey: Prentice Hall.
15. Ricca, F., et al., *Are Fit Tables Really Talking? A Series of Experiments to Understand whether Fit Tables are Useful during Evolution Tasks*, in *Icse'08 Proceedings of the Thirtieth International Conference on Software Engineering*. 2008. p. 361-370.
16. Melnik, G. and F. Maurer. *Introducing agile methods: three years of experience*. in *Euromicro Conference, 2004. Proceedings. 30th*. 2004.
17. Nonaka, I. and H. Takeuchi, *The knowledge-creating company: how Japanese companies create the dynamics of innovation*. 1995: Oxford University Press.
18. Brown, J.S. and P. Duguid, *Borderline issues: social and material aspects of design.* Human-Computer Interaction, 1994. **9**: p. 3,Äì36 %U http://dx.doi.org/10.1207/s15327051hci0901_2.
19. Baker, A.C., P.J. Jensen, and D.A. Kolb, *Conversation as experiential learning.* Management Learning, 2005. **36**(4): p. 411-427.
20. Nonaka, I. and N. Konno, *The concept of "ba": Building a foundation for knowledge creation.* California Management Review, 2008. **40**(3): p. 40-54.
21. Ramesh, B., L. Cao, and R. Baskerville, *Agile requirements engineering practices and challenges: an empirical study.* Information Systems Journal, 2007. **20**(5): p. 449-480 %U http://onlinelibrary.wiley.com/doi/10.1111/j.1365-2575.2007.00259.x/full.

Interview Guide

(These is the subset of the interview guide that was relevant to the case in question)

Step 2 - Introduction: discussion of the interviewee work situation.

Tell us about your work

• Role, Daily tasks, Responsibility

Which roles/other persons do you cooperate the most with in your daily work? How?

Step 3 - The interviewer presents the goals of the project.

- Can you tell us your views of RE – V&V alignment?
- What do you see are the challenges in the alignment; can you give perhaps some examples?
- How does the conceptual model map into the context of your company?

Step 5 – General questions about requirements and testing

Which organizational processes, units and roles are involved in requirements handling? How are functional and quality requirements artefacts managed?

• Identified, Changed, Measured, Prioritized, Validated

Which organizational processes, units and roles are involved in testing? How are testing artefacts managed?

Identified, Changed, Measured, Prioritized, Validated

Is there a difference between testing functional and quality requirements?

How is the customer involved in requirements and testing phases?

Step 6 - Questions on quality and functional requirements evaluation.

What quality requirements are the most important in your work?

- Efficiency/performance/capacity
- Reliability
- Safety, Security
- Maintainability, Usability
- Others

In which development phases are quality requirements evaluated? How (through measurement, reviews, testing, etc.)?

What are the pros and cons of working with quality requirements?

Step 7 - Questions on requirements – testing alignment.

What is used now to align requirements and testing?

- Documents
- Processes, methods
- Tools
- Principles and practices

Who is responsible for alignment (organizational processes, units and roles)?

Are testers participating in requirements validation?

Is the test strategy based on requirement documents?

Is there a difference between functional and quality requirements with respect to alignment?

What are the current problems/challenges in terms of alignment? (What does not work now?)

How is the alignment maintained in case of testing outsourcing?

Step 8 - Change related questions.

What happens when a requirement is changed (or deleted)? How are the testing artefacts modified when requirements are changed (and vice versa)?

The interviewer may show the scenarios.

Scenarios:

Changed requirement -> regression testing

Chanqed requirement -> changed test cases

New Requirement -> Test effort estimation

New Requirement -> Test suite development

Step 10 - The interviewer asks about possible solutions and ideas.

Do you have other ideas to propose to improve the alignment? It can be process, methods, tools.

What are the expected benefits of an improved alignment?

Step 11 - Conclusions.

Can you think of any challenges/good things etc. that we have not covered that you think we should have asked?

If we are going to continue investigating this area and learning more, are there other roles we should talk to? Some specific names?

Step 12 - The interviewer concludes saying that the interview record will be transcribed, summarised and communicated to the interviewee for confirmation and interpretation. The interviewer thanks the interviewee and say good-bye.