Automated Acceptance Testing: a Literature Review and an Industrial Case Study

Børge Haugset[1], Geir Kjetil Hanssen[2,1]
[1]SINTEF ICT, (borge.haugset, ghanssen)@sintef.no
[2] Norwegian University of Science and Technology

Abstract

Automated acceptance testing is a quite recent addition to testing in agile software development holding great promise of improving communication and collaboration. This paper summarizes existing literature and also presents a case study from industry on the use of automated acceptance testing. The aim of this paper is to establish an up to date overview of existing knowledge to benefit practice and future research. We show that some of the proposed benefits are realistic but that further research and improvements are needed to get the full potential value.

1. Introduction

Testing in agile software development [1] is fundamentally important as it enables visibility and enhances communication and feedback to developers. Unit testing and test-driven development are the most known, practiced and researched approaches to agile testing [2, 3] so far. However - over the past few years a complementary testing approach in agile development has emerged - automated acceptance testing (AAT). In principle, the customer or his representative is given the role of expressing requirements as input to the software paired with some expected result. Whilst unit testing is about testing low level units such as methods, acceptance tests integrate at a higher level between the business logic and the user interface or directly with the user interface. Doing acceptance testing manually will in most cases be tedious, expensive and time consuming. Automation of acceptance tests may thus seem as a promising initiative to ease and improve this process. The basic idea of AAT is to document requirements and desired outcome in a format that can be automatically and repeatedly tested – very much based on the same philosophy as for unit testing.

The aim of this paper is twofold: Firstly, to reinforce existing knowledge and experience by identifying topics and concepts investigated so far – have the most relevant topics been addressed already, or do we see the need to expand this scope? Secondly, we aim to establish a good overview of existing experience on AAT in agile software development (based on the identified topics and concepts). To do so we have carefully identified and reviewed existing experience reports and publications on AAT within the context of agile development. To build on and complement this literature review we have also conducted an industrial case study to both add to and extend the knowledge base on AAT. We hope that this work will be a valuable asset for both practitioners and researchers to better apply and develop AAT as a valuable practice in agile development projects. This study is explicitly focused on AAT and does not delve into Storytest-driven development (STDD) [4].

We see from our summary of the identified experience reports and our own case study that AAT may deliver some of the intended benefits but that it still is an immature testing practice, and not yet filling its full potential. We claim that some topics and concepts so far have been overlooked or addressed too weakly by the research community.

We continue this paper by first describing our research method for the literature review and then our approach for the case study - see chapter 2. In chapter 3 we report the results from the literature review. Results from the following case study are reported in chapter 4. In chapter 5 we present a joint discussion to identify central concepts of relevance from the studies and how they relate to each other. Chapter 6 summarizes and concludes our work and points to future research.

2. Research methods for literature review and case study

The literature review was done in order to establish an understanding of the state of the research on AAT. The goal is to identify topics that have been addressed in existing research and to systematize results related to these.

The following case study is of a medium sized Norwegian software consultancy company and two of

978-0-7695-3321-6/08 $25.00 © 2008 IEEE
DOI 10.1109/Agile.2008.82

their projects applying Fit and Selenium respectively. Fit[1] is a framework for integrated testing originally defined by Ward Cunningham. It is about testing business rules, and acceptance tests are expressed as simple table statements of input and expected output. These statements can be automatically executed and validated by an underlying testing framework. FitNesse is a framework that uses Fit and a wiki to automate the testing process. As for a detailed description of Fit, FitNesse and its internal workings see the book by Mugridge and Cunningham [5]. The FitNesse[2] tool itself is available for free.

Selenium[3] is a test tool for web applications, and can be run inside a regular browser. Just like Fit, Selenium automates testing of predefined test statements and evaluates the outcome. The main difference is that Fit-tests usually integrate directly with the business layer whilst Selenium integrates with the user interface layer.

2.1. Literature review method

The type of literature review reported in this paper can be classified as an integrative literature review [6] which identifies publications of relevance, evaluates and selects publications for review and systematize them such that common central concepts are identified and structured. Thus, an integrative literature review will develop an overview of existing research which initially is not related and form a basis for further research. The literature review was executed through three steps:

1. Identification of potentially relevant literature
2. Evaluation of relevance and quality to select literature for analysis and synthesis
3. Analysis and synthesis of literature

The aim of performing this review was to identify all published studies on AAT.

Review step 1 was done by searching the most relevant indexing services for research reports on software engineering. This included searches in ACM Digital library[4], IEEE Explore[5] and Web of Science[6]. Together these indexes cover close to all relevant scientific journals, magazines and conferences related to software development. In addition, the proceedings from the first annual conference on Extreme Programming and Flexible Processes in Software Engineering [7] were manually searched (as these proceedings are not indexed electronically).

All sources were searched using the following six search phrases:

1. "automated acceptance test"
2. "executable acceptance test"
3. "user acceptance test"
4. "customer acceptance test"
5. "fit AND software"
6. "fitnesse"

Two researchers (the authors) reviewed the title and the source (journal, magazine or conference) of the identified papers to exclude findings obviously not relevant to software development. We also removed editorials, prefaces, article summaries, interviews, news, reviews, correspondence, discussions, comments, reader's letters and summaries of tutorials, workshops, panels and poster sessions. This resulted in 50 unique publications of which 13 presenting empirical results.

Review step 2 was to evaluate each paper to ensure that it either explains or discusses the concept of - or use of - AAT or/and that it reports empirical findings. Based on a review of abstracts 31 papers were removed, keeping 19 which constitute the basis for the state-of-the-practice part of this paper. During the study of the full text, 5 more papers were found to be not relevant to our review and thus removed, leaving 14 papers in total for the synthesis.

Review step 3 was the synthesis of the resulting papers. The resulting collection of papers differs with respect to thematic variation, study type and scientific quality. We found that only 8 of the 19 papers reported results from empirical studies in the sense that they describe the study method, analysis and validity of results – the rest of the papers can be classified as experience reports or elaborations of concepts or ideas. AAT is a relatively new and shaping practice; we decided to keep also strictly non-empirical papers such as lighter experience reports. This was done to make sure that potentially relevant findings and trends in this new field of research are not overlooked initially. As a result of this, one can not consider this synthesis a full systematic review [8] but an integrative literature review [6]. Thus this forms an overview of the current state of research of automated acceptance testing, showing topics and concepts that have been addressed in research so far and a summary of this experience. We believe that this is useful initially, but that this

[1] http://fit.c2.com/
[2] http://www.fitnesse.org/
[3] http://selenium.openqa.org/
[4] http://portal.acm.org/dl.cfm
[5] http://ieeexplore.ieee.org/
[6] http://apps.isiknowledge.com/

knowledge base should be verified and extended with later sound empirical studies.

In addition we identify topics and concept that have not been investigated so far but that should be included in future research. This consideration is based partly on the book describing AAT with Fit [5] and conversations with people involved in acceptance testing.

The analysis of the selected literature was inspired by the practice of constant comparison [9] where central concepts and topics emerge through repeated analysis of the reviewed literature.

2.2. Case study method and context

To partly build on and to complement the results from the literature review we conducted a case study at a Norwegian medium-sized software consultancy company. We were given access to recently finished or ongoing projects currently using AAT. Two projects were available and four developers from each project were appointed as interview respondents (convenience sample).

Case project 1 has 10-15 participants (varying over time), a cost frame of approximately 2.5 M€ (fixed price) and a duration of 18 months (still ongoing at the time of the study). The product owner role is fulfilled by an internal, representing the customer. The solution being developed is a web-system fronting a database. The project applies Scrum with two geographically separated teams, and uses the Selenium test framework.

Case project 2 is an extension of a previous delivery starting in 2003 and is still ongoing at the time of the study. The team varies from 5 to 10 persons. The customer consecutively pays for resources spent. The system being maintained is part of a larger system of subsystems delivered by other suppliers. The project follows the Scrum process, and uses the Fit test framework.

We developed a common interview guide for the eight interviews based on the following aspects:

1. Questions of special interest to the CTO at the case company

2. Some of the topics and concepts identified in the literature review:
   - Training, support and introduction
   - Communication
   - Cooperation
   - Requirements management
   - Process control
   - Customer satisfaction
   - Product quality
   - Maintenance of tests

3. Additional topics that we missed from the literature review that we find potentially relevant based on the description of automated acceptance testing [5]:
   - Tests as system documentation
   - Usage pattern (who, when, how)
   - Positive and negative tests
   - Complexity of tests
   - Discovery of new requirements
   - Frequency of finding errors/issues
   - Main negative and positive experiences

Our intention is to add experience to the existing knowledge base and to contribute to extend this for future practice and research. The interview guide is not included due to space limitations, but can be obtained from any of the authors.

Each interview was made on-site lasting for 30-40 minutes and in a semi-structured way, to allow the respondents to reflect. The interviews were recorded and transcribed by a professional language service company. The resulting eight interviews were analyzed according to the principles of constant comparison [9]. In the analysis we followed the structure of the interview guide to identify conformance or divergence in the respondents replies. This summary of the interviews are reported in chapter 4. As a final part of the analysis we collate and discuss our findings in both the literature review and the case studies. Hopefully, this constitutes an up-to-date understanding of the implications of AAT, based on all available studies on the topic so far.

2.3. Bias and limitations

The literature review was done by two researchers meaning that the selection of literature, the analysis and systematizing are potentially biased by the researchers experience and expectations. To compensate we required both researchers to agree on all decisions. Each case of dissension was resolved by discussion. We were also open to relevant topics in the analysis to identify both obvious and more distant topics. Some of the literature included in the review can not be classified as thorough empirical studies. However – given that AAT is a rather new practice we chose to also include non-empirical articles as we appraise the value of reporting the current state on the

topic. As the field evolves new topics will be investigated and described.

Our case study consists of eight interviews of employees working on two projects in one organization. This small sample size affects our possibilities to generalize. We still believe that this study could be of interest to practitioners and researchers of the emerging practice of automated acceptance testing.

3. Results from the literature study

The results from the analysis of the literature presented here is structured according to the most prominent and relevant topics identified. This does not mean that this list of 11 topics is the ultimate set of topics relevant to AAT – it merely reflects topics investigated and discussed in the set of the identified literature.

3.1. Learning Fit

Judging from the reviewed articles, Fit seems an easy tool to learn. The average introduction is a 3-4 hour crash course supplemented with relevant readings. Training usually only covers the basics of the tool, so any advanced use of Fit is bound to be gained by continuous use. Melnik et al. reports on a university course where 90% of the students managed to deliver their Fit test assignments after a brief introduction, indicating that the students found it easy to learn [10]. These students were however computer graduates. How easy is it for less technology-savvy people, like most customers are, to learn Fit? Unfortunately there isn't much clear evidence. In another study by Melnik et al. they found that while business graduates (playing the role as customers) initially had some more problems learning the tool (50% found it hard to learn), they eventually produced good specifications with ease [11]. The study reports that customers in partnership with IT professionals (both roles played by students) can effectively specify acceptance tests. An interesting finding in this article is that teams mixed of business and computer graduates were no better at describing a good quality specification than pure computer graduate teams, even if the mixed teams should have more knowledge of the domain. The authors speculate that using Fit somehow bootstraps the business grads to the same level as the computer grads. These students were already taking an elected course in software engineering, and may so have been more interested and knowledgeable in this area. Related to this, Melnik and Maurer report that a company successfully adopted and used Fit tests. However, this specific customer had an information systems background [12].

One way of judging the ease of use is to check if people will continue to use it later or recommend it. Students that were asked if they would recommend using executable acceptance tests to colleagues, were on average positive to this (5 very likely, 6 somewhat likely out of 17) [11]. 52% of students in another study reported that they would chose Fit to define requirements, while more than two thirds would consider it [13]. Findings show that there was no correlation between the students' own perception of the value of the tool and the quality of the requirements specification they produced [11]. It seems, in those authors' opinion, that the subjects are not aware of the quality of the approach used.

This does not necessarily mean that Fit is hard to master for non-technical persons; there is however not much empirical evidence for it being so.

3.2. Using Fit to create requirements specifications

In Fit the requirements specification is made in a tabular format. This table is connected to a set of fixtures to understand syntactically how the table is to be treated. Some insights have been made on how well Fit is able to specify such requirements, mostly via studies on student groups. The general impression is that Fit is a good tool for specifying requirements. Melnik and Maurer found that 80% of students in an experiment would (when asked afterwards) rather have requirements specified in Fit than in prose [14]. Similarly, Read et al. noted that the average student felt that Fit was adequately appropriate for defining functional requirements [13]. The same students rated writing Fit tests as just as difficult as writing JUnit[7] tests and requirements specifications in prose. When it came to creating the web service specified (the groups of students switched Fit tests among themselves) 43% of them felt they had been given good enough specifications to create the service, while 26% managed but had problems. The last 31% found the specifications to be insufficient. In a small survey of test notation tools by Geras et al., Fit scored best on 'ease of use' (together with scripts and XML) [15].

While Fit is intended to make the customer help write the tests, the customers don't always feel the same. In one project observed by Gandhi et al., the business analysts weren't interested in writing more

[7] JUnit is a framework for writing repeatable unit tests, see http://junit.sourceforge.net/.

than plain text, leaving it to the quality analysts to code the tests [16]. The authors suggest, in hindsight, that the Fit documents should rather be described as specifications. Likewise, Melnik and Maurer mention the use of previously-made info-sheets that the customer brings to each iteration meeting. These were internal documents containing diagrams, callouts and mock screen shots. They were not, however, to be treated as authoritative specifications [12]. According to the customer, these sheets made it easier to explain the context to the developers. In this particular project the customer ended up writing 40% of the tests, the developers and testers wrote 30% each. The customer felt that it was best to write these tests himself, as the domain was complex and that "…going from a general description to a test has some fluidity in interpretation." Likewise, Melnik describes another project where the tests were accompanied by embedded commentaries and occasional diagrams [17]. The tabular structure of the Fit tests was seen as superior to XML for writing tests for a chat server, especially because of its ease of changing the tests [18]. Whilst there is a growing amount of large IDE's for writing code, for instance IntelliJ[8] and Eclipse[9], there are no equal tools for testing. This is also acknowledged by Melnik, he concludes the support for refactoring is the weakest point [17]. And as Andrea mentions, being able to write test code needs to be easier than writing normal code [19].

Writing acceptance tests isn't always the most prioritized thing to do. One customer felt that it was easy to put off writing them because they were complex beasts, and "…the thought of diving into that complexity, just when we thought we were done, was unpleasant" [12]. It was unpleasant rather than difficult, though, and the authors nicely phrase the problem as "The difficulty was not the practice [of writing Fit tests] itself, but the discipline of doing it."

While finding the time (or inspiration) to write Fit tests might at times prove hard, there are some claims that doing just so leads to a better specification. The customer and the team in the said project say that using Fit led them to discovering a lot of missing pieces and inconsistencies in a story. [Ibid] This is potentially related to effects described by Melnik et al., where they claim Fit reduces noise, over-specification, and ambiguity [10].

3.3. Organizing specifications

One effect when projects grow large is the growing need to organize the specifications. Gandhi et al. report from a case study that as the amount of Fit tests in one company project expanded they started to keep them in different directories according to their lifecycle status [16]. This caused problems when people did not move documents properly. To solve this they developed a new Category Fixture that was able to sort documents even within the same folder. No matter if you chose to include numbers or not, they suggest being consistent in naming conventions. One article by Holmes and Kellogg about Selenium describes the same problems, wanting to run complete suites and at the same time grouping them in iteration, story and functionality set. Selenium supports multiple tags for one test, and that is reported to be a benefit [20]. The use of semi-automated Fit test creators has been suggested to help with these problems [21].

3.4. Effects of process and product quality

Melnik et al. studied the quality of acceptance tests and resulting implementation. They found no evidence to support that good quality acceptance test specifications made by a customer team resulted in better quality implementation by a developer team [11]. They think this is an issue which requires further study. In the study by Gandhi et al., one company reported having successfully refactored a large and important part of their application, and think it was successful due to their extensive coverage of readable Fit tests [16]. Likewise, changing a 'socket acceptance tester' from using XML to Fit tests was, according to developers, a success [18].

3.5. Communication

Being stated as one of the core benefits, it would be interesting to see what is found about the use of Fit for communication. One problem lies in letting the tool drive the collaboration and not vice versa. Interestingly, Gandhi et al. noted in their study that one group claimed that using Fit initially hindered communication [16]. They focused too much on preparing 'syntactically correct Fit documents' for the development team, and didn't focus on developing the most appropriate specification. Melnik reports about one team where the project manager said that "We value conversation more than the actual tool itself [Fit]" [17], p. 188, and that Fit helped them with this. Read et al. found in their study that 41% of student groups developing software had no contact with the

[8] http://www.jetbrains.com/idea/
[9] http://www.eclipse.org/

test suite development team during development [13]. In another study by the same authors the Fit suite summary was by some student groups being used as a project dashboard [22]. Since the tests were stored at a centralized server, project members and instructors alike could check the project progress whenever they wanted. It seems viable that also project customers could use such a feature to keep track of project status.

3.6. Fixture types

Fixtures are the 'code glue' that connects test tables with the business logic code and may vary; fat fixtures which contain a lot of the business logic itself, thin fixtures which delegate and point to business logic elsewhere and mock fixtures which simply return desired output. While implementing fat fixtures and mockups are sound ways to create passing ('green') code for later refactoring, thin fixtures provide the solution, among others for maintenance reasons. Student groups that started a project using a test-first design philosophy ended up with fatter fixtures than the groups that started with Fit in the third iteration [14, 22]. The authors mean the students were aware of this bad practice, but as there was no grade-wise reason for refactoring they did it the easy way. Gandhi et al. define 'complex fixtures' as concealing abstractions and concepts already present in the Fit document or the application code from each other. This happens when the fixture translates between languages from the document to the code, and should (as a result of 'smelling') be refactored [16].

3.7. Maintaining test specifications

According to Andrea, two (of several) key factors in a testing environment are easy and safe maintenance, and that there are easy ways to look up the right tests [19]. Gandhi et al. stress the importance of keeping the Fit specifications alive [16]. They need to be maintained constantly; the Fit document should be the main definition of the specification and should be kept free of duplication. Old Fit documents should be altered rather than new ones created if bugs occur or specifications change, and the Fit test suite should be integrated in the continuous build process. First they tried to solve the duplication problem by writing subroutines that could be called from within the Fit documents. This improved reading, but quickly became such a mess that the analysts were unable to read them. Abstraction should thus not come in the way of readability. Mugridge and Cunningham propose a tool where Fit tests are automatically created from transitions between nodes (system states) in a directed graph (the system spanned out) [23]. Likewise, Amyot suggests a tool for creating Fit tests from use case models which might be a viable approach [21].

3.8 Types and quality of tests

Negative tests are tests that are supposed to fail. Melnik et al. observed one group of students that on average wrote only 6% negative tests even while having learned of its importance [11]. In another study, developers told the authors they wrote negative tests, and included sophisticated error messages to explain what the errors consisted of. They also extended the Fit framework to catch runtimes and do basic load testing [12]. When it comes to deciding to write tests in suites or keeping single tests, Read et al. arranged a student experiment that showed no significant differences between groups focusing on either strategy during the ramp up phase [22]. In the following regression phase, when the tests had passed, students started using single tests a lot more than suite tests, especially when Fit was first introduced in later iterations.

3.9. Developer and team effects

Organizing a team is always a struggle. One team ended up with so many Fit documents that they created a special team who dealt only with this [16]. This increased the Fit development, and enabled analysts to effectively write Fit documents. These documents were very detailed. The result was that developers started writing code to make the tests pass rather than cooperating with the original customer. The company increased staffing from 6 to 24 developers in four weeks midways in the project. Doing this in a successful way was, according to the authors, partly because the Fit documents were readable, living documents that new developers could use to understand the domain. The large Fit test coverage also gave them the impression that unanticipated changes that introduced errors would be picked up by the system. The authors however call for caution about adding more developers without adding more analysts. For them, the same amount of analysts spent less time collaborating with developers, and more time specifying Fit documents. This resulted in more defects getting through to the application.

3.10. Developers and customers' use of Fit test results

In terms of reading the outcome of tests, Andrea mentions three goals for a new functional testing tool [19]. The system should allow for multiple readers of the tests such as customers, developers or testers. Secondly the tests should be readable in a plethora of formats, and the reader should be able to view the tests in the desired format regardless of its input format. Finally, there needs to be a view of how the different functional tests relate to each other, and a description of the business process they define. All tests must locatable by searching for names, functional areas, and metadata.

Melnik elaborates on the meaning of "completed". In a project he observed, the project manager describes how a passing acceptance test does not at all mean the task or story is completed [17]. Even when using Fit there are many other factors to attend to, such as measuring performance criteria and technical writing. With regard to Fit and viewing/organizing project status, one can say that there is still some way to go. We have previously in this article described the situation where one company had problems with sorting tests in folders. Further, there is currently no other way to view the tests than the table structure. There are some extensions for writing tests in other languages, for instance via use-case modeling, and this could give some readers another view on the structure [21].

3.11. Challenges regarding Fit (cost and restrictions)

Running test-driven development based on unit testing means defining small unit tests, code, and immediately seeing the result. A test defined at acceptance test level, however, can take a long time to code. According to a study by Deng et al., the average time to code a passing acceptance test was more than four hours, for some extending to several iterations [24]. The problem arises when trying to separate between 'unimplemented failures' and (the more important) regression failures. They have tried to solve this by creating an Eclipse addon, FitClipse, which knows if a test has previously passed or not. Another problem lies in the assumption that a passing test suite means the code is good. Melnik and Maurer touch briefly on the topic when they describe the "deceptive sense of security" students had when the tests passed, and warn about not thinking outside the box [14]. Melnik also mentions a team calling the collaborative writing of acceptance tests "bartering", and this team felt using Fit helped them implement features in an economically feasible yet still correct way [17]. Fit got the lowest score (2/12 points) of all test notation tools surveyed (albeit not a completed survey) when used for describing non-functional requirements [15]. This included performance, security, and usability. The opposite may also pose a problem. In a Selenium setting, small changes in the user interface (renaming a button) leads to failing the test even if the application (business logic) is intact [20].

4. Results from the case study

Our interview guide has three main sections and results are reported according to these; 1) how automated acceptance testing was used, 2) effects and results and 3) experience – looking into both positive and negative aspects.

4.1. About the use

Who writes the tests? Tests were written by the developers, no person had a dedicated responsibility for writing the acceptance tests and most noteworthy – no customers participated in expressing tests. They had worked so long in the area that they felt they at times had better knowledge of the field than the customers themselves. This has to be seen in relation to the team also using Fit tests mainly to test interfaces at a system services level. More specifically each developer herself decided if tests were needed and flowingly had the responsibility of writing tests for the feature currently being worked on. Sometimes this was based on consulting others in the team. One of the projects appointed a person being in charge of running the tests as well as reviewing them.

How were tests written? In the very start of using acceptance tests developers tended to define tests for most issues, also including simple ones. This has eventually changed over time, now only the most complex features are covered by acceptance tests – simple issues that one developer can have a complete overview of is considered not to be worthwhile to test this way. One of the projects involved development of a GUI-layer, and here acceptance tests were basically used to build a suite of regression tests for the interface (using Selenium). The other project used Fit to test at module or service interface level.

When were tests written? Tests were written when the need occurred, meaning that they did not define

any specific phase or point of time in the iterations to do so. Neither of the two projects used a test-driven design strategy. As the system under development grew the need and motivation to define acceptance tests also grew.

Training, support and introduction The introduction of this new practice was made as simple as possible; one expert external to the projects gave a one-day introduction in the form of a workshop to most developers in each project group.

Positive and negative tests In a few cases negative tests were written, meaning that a test is meant to provoke an error; this is used to ensure that the system handles errors as intended. One developer commented that it is hard enough to develop and maintain positive tests.

Maintaining tests / keeping tests and code in sync Besides development of code, the tests were also modified either as an outcome of test results or that a customer reported an error. Typically for the project using Selenium, a change in the GUI created a need to accordingly update the test as well. The project using Fit found that web-services were stable (to change) and thus the need to update these tests was low. In general there seemed to be a difference in code/test stability of core components or services that were stable and needed less updates of tests, while code and tests closer to the GUI tended to be more unstable and cost more in terms of maintenance.

4.2. About the effects, outcome and results

How use of AAT affects communication and cooperation with others (team and outside) Most developers reported clearly that having a part of the code covered by acceptance tests made it safer and thus more convenient to let others work in their code and vice versa. Similarly the acceptance tests increased clarity in the sense that all developers could see what the code is intended to do and the recent status. Tests were easier to grasp than the respective code. Also – in some cases where a developer sees that she will have to alter another developer's code in such a way that the test will break she gets an initiative to contact the original developer and discuss the issue. Except from rare cases, the customer does neither specify tests nor evaluate the results. This surprised us as AAT supposedly helps enhancing communication with customers by helping them write the tests in a format they understand. The reason for not doing so, the developers explained, is that developers often have a better total understanding of the system and problem domain and thus is in a better position to specify tests.

Tests improving understanding of the system The vast majority of developers expressed strongly that writing acceptance tests, preferably upfront, improved their understanding of the domain and the system under development, however it did not improve the understanding of the code itself. One developer also compared it to unit-testing and underlined that the clear separation of test and code in acceptance tests enabled him to get a better overview of what the system is intended to do as in contrast to how it does it, technically. Another developer commented that these effects are reduced when tests grow large.

Requirements management As the acceptance tests were not used as a communication medium with the customers they were not used for documentation of requirements. Customers preferred to express requirements in traditional ways and in meetings. Next, an internal transferred this input to acceptance tests, thus missing the potential benefit of having the customer doing this directly. As developers felt confident with the business, and naturally were more comfortable with the acceptance test tool itself, this appeared to be the most efficient practice.

Process control The use of acceptance tests seems to contribute to the visibility and process control of Scrum which both projects applied. Test results show graphically which modules that are finished.

Customer satisfaction We asked about the developer's subjective opinion on how the use of acceptance tests affects customer satisfaction. The general impression is that these tests help the developers to identify and correct more errors, and that fewer errors consequently give better customer satisfaction.

Product quality The developers share the opinion that their use of acceptance tests positively affects product quality in two related ways; first more errors are handled and secondly regressive testing frees time compared to manual testing. However, testing was usually not prioritized in extremely busy periods of the project. One experience from the use of Selenium, which tests the user interface, is that automated tests make it easier to test complex interfaces. This consequently enables more frequent testing than the alternative manual testing.

Documentation The growing suite of acceptance tests acts as a good extra documentation of the system,

yet on the behavioral level – such tests do, by concept, not indicate how functionality is implemented. The tests were not currently being used as documentation for the customer, but some developers stated that this would perhaps be something for later.

Fixture/test fatness/complexity Fixtures and tests grow large, typically resulting in many columns in Fit. They can be difficult to follow and thus sometimes large tests are broken down to several simpler ones. In general the developers try to keep the tests simple in the first place. When testing the GUI-layer, however, this can be hard as a series of interrelated screens typically contains a lot of logic resulting in large tests.

4.3. Experience

What does it solve? Each respondent was asked to summarize their positive experiences from the use of automated acceptance testing, both directly to themselves and for the total project. Top-rated issues were:

- It is safer to make changes in code
- We get fewer errors
- It is easier to share competence within the team due to perceived increased safety (of altering code)
- It reduces the need to do manual testing, which is a time-saver
- Compared to unit-tests, the acceptance tests give a better overview, it is also easier to add new tests
- Writing acceptance tests makes you think of what you are going to make before you do it. It also makes you find special cases that may be extra important to test properly
- Usually specifications from customers are poor so writing acceptance tests is in a way reflection on requirements
- It is very comforting for the developers to know that the system (or relevant parts of it) works
- Tests for GUI are especially useful as manual testing can be exhausting
- In cases where external systems are unavailable at development time, acceptance tests act as a good substitute.

Frequency of finding errors/issues Most errors are found shortly after the introduction of the case when the code is under development. However, change of test data in e.g. a database may also cause a test to fail, even if the code is unaltered. We also see that later changes to code may actually not cause a test to fail. This is because the test itself acts as documentation to the developer, who in turn is able to alter the code to not break the test.

Discovery of requirements that otherwise would have been missed Broken tests can also indicate disparities in the version control system, typically if someone has forgotten to update changes - tests may unveil not only code issues but integration issues as well. In general, acceptance tests may also help to reveal complex issues that by nature are unsuitable to time consuming manual testing.

Main negative experiences and problems When asked to summarize negative aspects of AAT, as applied in these projects, the following issues were reported:

- Having a test is by itself not a guarantee; it can be easy to fool oneself by writing poor tests
- One obvious cost of adopting any testing practice, including AAT, is the time used to define and maintain tests in synchronization with the code being tested. Particularly maintenance of tests is reported to require a lot of effort. This gets even more prominent when tests have hard-coded data for check of results
- Typically, when a developer is new to this testing strategy tests can easily grow very large – it takes experience to establish a proper test design
- Some developers found it to be hard to write proper tests due to many awkward rules in the domain and the business logic of the system being developed. Such tests easily get complicated with the effect that they are both hard to understand and modify.

Is it relevant to continue to use AAT, any modifications of practice? All eight developers responded positively when asked if they would like to continue to use AAT in new projects. They underlined that it would be a great advantage to start using the practice from the very start of the project. Some developers also commented the need to be deliberate on what to test and to test small parts of the system to avoid large tests that are cumbersome to handle.

5. Discussion

The discussion here compiles the results from the literature review and the case study. The intention of AAT and tools such as Fit or Selenium is among others to improve communication and collaboration, both internally in a software development team and with the customer. Expressing requirements as AAT is supposed to both make it easier for the customer-side to express their requirements and aid in knowing if the

system under development meets those needs. One aspect of establishing an efficient practice of AAT is the ease of learning and introduction. We see from our findings that very short and simple courses seem sufficient – also for non-technical customers. Even so, we found that it took some time and experience to be able to see which parts of the system that need test coverage. None of the projects in our case study engaged the customers in writing tests, however Read et al. [13] indicate that persons without an IT background may find it harder to grasp. The developers we interviewed expressed that they felt that they had a better understanding of the solution being developed and the related business area. This indicates that just giving the customer side the opportunity to express acceptance tests using tools such as Fit or Selenium is not enough, and we believe that this process must be managed better. This can for example be done by identifying the right customer representative for different parts of the solution and assisting them in the definition of tests. Thus it may not be an issue about the tool itself but about how the process is managed. One promising technique that could address this issue is storytest-driven development [4]. In 2007 Melnik defended his PhD thesis, focusing on empirical analyses of executable acceptance test driven development [17]. While this thesis was not found in our search, articles used in our summary found the basis for his thesis. He concludes with there being a correlation between automated (executable) acceptance-test driven development and the enhanced communication in software teams. We find no industrial experience on what it takes for non IT professionals, such as most customers are, to specify requirements using acceptance tests. Yet it is likely that introducing AAT to customers would require more extensive training and support.

The main impression from the literature review is that stating requirements in the form of acceptance tests generally works well. Melnik et al. found that 80% of the students in an experiment rated this way of expressing requirements higher than prose [14]. Correspondingly, our respondents experienced that writing acceptance tests improved their understanding of the domain and the system due to the separation of test and code. Yet we must underline that the customer did not express requirements this way – this task was handled by an internal in the project taking this responsibility on behalf of the customer *for testing*. Only one paper discusses quality of AAT in relation to quality of implementation. Melnik et al. found that there was no such correlation [11], and state that further research is needed. We found that it seems that writing acceptance tests make developers reflect on the design and system behavior in advance of programming. Developers also reported the tests to be valuable documentation of the intended behavior of the system, thus resembling unit tests but on a higher level.

One of the obvious reasons for any type of testing is to discover errors, non-conformance or flaws that need to be resolved. We see from Read‘s student experiment that developers discovered both inserted errors and even non-intentional errors [13]. Our interview respondents reported that the new testing practice helped them find and remove more errors than before. They also felt it made them able to see integration issues more clearly. One of the most evident findings in the interviews was that developers found it more convenient to share code with other developers for two reasons. First the tests worked as internal developer-documentation of the code, making it easier to understand what it is intended to solve. Secondly the tests made it safer to alter the code as it immediately became visible if some error was introduced. Hence we see that the use of acceptance tests may improve communication and cooperation in the development team. This is in contrast to experience in one of the other studies. Gandhi et al. observed that using acceptance tests (in the early phases) actually hindered communication as developers put too much effort into expressing correct Fit documents, on the expense of expressing only appropriate tests [16]. This seems to be a balancing act; benefit should be weighted against cost. This balance deserves further research.

Only a few of the analyzed articles addressed the customer communication aspect of AAT: Melnik and Maurer found that engaging customers with an information systems background resulted in a successful adoption of Fit [12]. As a contrast, Gandhi et al. observed, like us, that customers preferred old-fashioned requirements descriptions in plain text [16]; so this matter is not fully investigated. Not discussed in the literature, but obvious in our interviews is the security and confidence amongst the developers as a result of having the system covered by tests explaining intended behavior. This effect is just the same as found in studies of unit testing [2] as well as other studies in agile development in general [25]. The tests help to reduce the fear or resistance of changing the code even if this would be beneficial in development and maintenance of the code. While AAT shows to be beneficial we also find issues related to costs. Naturally, defining tests costs time and attention. Having established a test that is closely connected to the code also implies a need of maintaining the test to keep test and code synchronized. Deng also underlines this [24]. One team reported that using Fit helped them

have a conversation about the features, so they would be both economically feasible and still correct [17].

The respondents we interviewed reported maintenance to be a considerable cost, especially when the tests also contained hard-coded data for checking results. Another important issue found, that applies to testing in general, is that it is easy to write a test that completes successfully, yet it may still not cover the code properly. Thus, writing tests requires careful consideration or else they can give a false sense of security. None of the studies forming the base for this discussion report quantitative data documenting the pivot point: net benefit of AAT. We here rely on developers' preference of continuing to use AAT based on their experience as an indication of the feasibility of this testing practice. All of our respondents replied positively when asked about this. Other studies also report this opinion amongst developers [11, 13]; similarly we would also like to see studies of customer opinions.

6. Conclusions and further work

In total, from the reviewed literature and the complimentary case study we are able to draw some conclusions. First of all it is worth noticing that one of the main motives for applying AAT is poorly reflected by practice – based on experience it seems that it is somewhat inappropriate for customers to actively express requirements in the form of automated acceptance tests. This does not necessarily reflect the feasibility of acceptance testing; it can be an issue of how this is prioritized on the customer side. In most cases this is left to developers, and writing acceptance tests seems to be more a case of reflecting over requirements. While automation of acceptance tests holds a great promise for improving the efficiency of development, writing and not least maintaining such tests come with a cost. This makes it important to carefully consider the potential benefit against the cost in advance. Correspondingly having a test completing without errors can be deceptive, a test must truly implement the intention of the requirement(s) to be valuable otherwise it can create a false sense of control. Another observation seems to be that AAT helps to increase the level of confidence amongst developers – sharing code seems easier which again may improve collaboration.

Automated acceptance testing surely holds a great promise. The limited experience base tells us that some important gains have been achieved. Yet there is still a great need for investigating this testing practice further. We have pointed out some new topics and concepts that deserve closer investigation in later studies. Especially we would like to pursue the role of these types of test for communicating better with the customers, something we will aim for in our later projects. We would also like to se more quantitative studies investigating process and product effects such as process control and product quality. Lastly we would like to look at automated acceptance testing connected to a test-driven design method, as this might indeed prove beneficial.

7. Acknowledgements

We would like to sincerely thank the case company for giving us access to do interviews. This work was partly funded by the Research Council of Norway.
This work was partly funded by the Research Council of Norway under grant # 179851/I40.

References

1. Dybå, T. and Dingsøyr, T., *Empirical Studies of Agile Software Development: A Systematic Review.* Information and Software Technology, 2008. **In press**(doi:10.1016/j.infsof.2008.01.006).
2. Erdogmus, H. and Morisio, M., *On the Effectiveness of the Test-First Approach to Programming.* IEEE Transactions on Software Engineering, 2005(31): p. 3.
3. Müller, M. and Hagner, O., *Experiment about test-first programming.* Software, IEE Proceedings, 2002. **149**(5): p. 131-136.
4. Mugridge, R., *Managing Agile Project Requirements with Storytest-Driven Development.* IEEE Software, 2008. **25**(1): p. 68-75.
5. Mugridge, R. and Cunningham, W., *Fit for Developing Software: Framework for Integrated Tests*. 2005: Prentice Hall Professional Technical Reference.
6. Torraco, R.J., *Writing Integrative Literature Reviews: Guidelines and Examples.* Human Resource Development Review, 2005. **4**(3): p. 356-369.
7. Succi, G.M.M., *Proceedings of the first annual Conference on Extreme Programming and Flexible Processes in Software Engineering*, ed. G.M.M. Succi. 2000, Italy: Addison-Wesley.
8. Kitchenham, B., *Procedures for Performing Systematic Reviews*. 2007, Keele University and Empirical Software Engineering National ICT Australia Ltd.
9. Seaman, C.B., *Qualitative methods in empirical studies in software engineering.* IEEE Transactions on Software Engineering, 1999. **25**(4): p. 557-572.
10. Melnik, G., Read, K. et al., *Suitability of FIT User Acceptance Tests for Specifying Functional Requirements: Developer Perspective*, in proceedings of *XP/Agile Universe*. 2004
11. Melnik, G., Maurer, F. et al., *Executable acceptance tests for communicating business requirements: customer perspective*, in proceedings of *Agile Conference*. 2006
12. Melnik, G. and Maurer, F., *Multiple Perspectives on Executable Acceptance Test-Driven Development*, in proceedings of *Agile Conference*. 2007
13. Read, K., Melnik, G. et al., *Student experiences with executable acceptance testing*, in proceedings of *Agile Conference*. 2005
14. Melnik, G. and Maurer, F., *The practice of specifying requirements using executable acceptance tests in computer science courses*, in proceedings of *ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. 2005
15. Geras, A., Miller, J. et al., *A survey of test notations and tools for customer testing*, in proceedings of *Extreme Programming and Agile Processes in Software Engineering*. 2005
16. Gandhi, P., Haugen, N.C. et al., *Creating a living specification using FIT documents*, in proceedings of *Agile Conference*. 2005
17. Melnik, G., *Empirical Analyses of Executable Acceptance Test Driven Development*, in *PhD Thesis, Department of Computer Science*. 2007, University of Calgary: Calgary, Alberta. p. 190.
18. Mugridge, R. and Tempero, E., *Retrofitting an acceptance test framework for clarity*, in proceedings of *Agile Development Conference*. 2003
19. Andrea, J., *Envisioning the Next-Generation of Functional Testing Tools.* Software, IEEE, 2007. **24**(3): p. 58-66.
20. Holmes, A. and Kellogg, M., *Automating functional tests using Selenium*, in proceedings of *Agile Conference*. 2006
21. Amyot, D., Roy, J.F. et al., *UCM-driven testing of web applications*, in proceedings of *SDL Forum*. 2005
22. Read, K., Melnik, G. et al., *Examining usage patterns of the FIT acceptance testing framework*, in proceedings of *Extreme Programming and Agile Processes in Software Engineering*. 2005
23. Mugridge, R. and Cunningham, W., *Agile test composition*, in proceedings of *Extreme Programming and Agile Processes in Software Engineering*. 2005
24. Deng, C., Wilson, P. et al., *FitClipse: A Fit-Based Eclipse Plug-In for Executable Acceptance Test Driven Development*, in proceedings of *Extreme Programming and Agile Processes in Software Engineering*. 2007
25. Fægri, T.E. and Hanssen, G.K., *Collaboration and process fragility in evolutionarily product development.* IEEE Software, 2007. **24**(3).