Automated Acceptance Testing Using Fit

Geir Kjetil Hanssen  
NTNU IDI/SINTEF ICT  
ghanssen@sintef.no

Børge Haugset  
SINTEF ICT  
hau@sintef.no

Abstract

Automated acceptance testing is a new and promising agile testing approach. Fit is the most established technical framework for specifying and executing acceptance tests which, ideally, lets the users express requirements in the form of acceptance tests. We performed an industrial case study to learn more on the costs and benefits of Fit tests. We learned that Fit tests may improve important parts of an agile development process but there is still a need for further research and improvements.

1. Introduction

Testing in agile software development [1] is a fundamental practice to enable visibility of progress and to enhance communication and feedback within the development team and ultimately with the customer. Agile development is a highly user-centric and change-driven process and it is vital that new and changing requirements are backed up with tests that can be defined quickly and executed frequently to have a constant view of the state of the evolving software product. So far unit testing and test-driven development are the most known, practiced and researched approaches to agile testing [2, 3]. However these are testing approaches that focus on the functional and technical level. These tests are defined by developers - for developers and give no direct value to customers or users which are central stakeholders in any agile development project. Over the past few years a complimentary 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 team paired with some expected result – an acceptance test.

Whilst unit testing is about testing low level units such as methods, acceptance tests integrate at a higher level, testing business logic. In this way a customer or potential user can relate to tests and it can be an efficient way of communicating precise requirements from customer to developers. Performing acceptance testing in a manual way will in most cases be tedious, expensive and time consuming. This does not fit the paradigm of agile development which relies on instant feedback and short development cycles. Automation of acceptance tests may thus be seen as a promising initiative to ease and speed up 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, but on a higher level, understandable to users. Unlike most traditional testing approaches where testing is postponed to later phases, agile testing strategies, including AAT enables tests to be defined iteratively just as requirements are defined or modified in iterations. One of the most prominent solutions for AAT in agile development is the Framework for integrated tests (Fit) [4] which provides a simple mechanism for documenting requirements as acceptance tests. Tests work the code through fixtures that integrate with business logic and can be used for regressive testing of the growing software product.

Fit is, according to [4], supposed to solve three main problems in software development: 1) Communication: Ordinary written requirements can be ambiguous, prone to misunderstandings and lead to wrong design - Fit tests tend to improve the communication of what a software product should do through concrete tests that are based on customer-defined examples from the business domain. 2) Agility: Fit tests are aimed at supporting frequent changes; tests ensure that any changes in the software do not break previously satisfied requirements – thus enabling agility. 3) Balance: Fit tests are aimed at helping a development team to spend less time on gaining balance with fixing problems by reducing the number and severity of problems, catching them early, and making sure they don’t return.

The aim of this paper is to address these potential high-level benefits as well as more detailed aspects
through an industrial case study. Can we – from practice – see these effects? In a case-study, we have interviewed four professional developers from an industrial Scrum development project [5] applying Fit tests to learn from their experiences. We build this work on a thorough review of similar experience reports on the use of AAT and Fit. We continue this paper by first describing Fit in section 2, followed up by an overview of the current state of research on Fit and automated acceptance testing in section 3. We then describe the study context, our research method for the case study and its limitations - see section 4. In section 5 we report the results from the case study. In section 6 we discuss our findings and relate them to the intentions of Fit. Section 7 summarizes and concludes our work and points to future research.

2. Automated acceptance testing using Fit

Fit test descriptions are organized in tables. A simple example can be seen in Figure 1. This test describes a system doing division on numbers. When running the tests, the input ‘numerator’ divided with another input ‘denominator’ should give the result written in ‘quotient’? Fit isn’t automatically connecting these tables to the underlying business code it is verifying. This connection is made by using fixtures, which are different ways of reading the tables and translating them to calls in the business code. Several fixture types are included in the tool, but they can also be made from scratch. Three types of fixtures are:

ColumnFixture, for testing calculations: Testing according to business rules. Certain input values shall result in certain output values. Figure 1 is an example of this type. ActionFixture for testing actions: The test defines a series of actions such as acting out calls for buttons pressed, values entered into input fields, etc. The running test reports if the application gives the appropriate results. (Actions are start, enter, press, and check). RowFixture for testing lists and other collections: A set of input or actions that constitute a single group and need to be tested together. The list of elements does not necessarily need to return correct values in any particular order.

The Fit tests can be run at any moment after they are created, even if the underlying code hasn’t yet been created. The result of running a Fit test varies accordingly, see Figure 1. A green result (grey in this figure) means the test has returned the desired value, and has passed (the two topmost result rows in the figure). A red test result (dark grey here) means the test failed (the bottom row), while yellow means part of the test or something else went wrong. A grey test result shows that the test hasn’t been processed. Here we see that the division code returns correct values for the two first pairs of input values, as the output value fields are colored green. The third division result is red, indicating a test failure. The expected result is shown together with the actual result from the call. The depicted example is a very simple one. The actual Fit tests in a working condition could be much larger. It shows, however, that the direct system requirements of the wanted procedure are possible to write in a table format, hopefully easy to understand for users, in contrast to unit tests which normally only is meaningful to developers.

Since the Fit tests themselves are made in a simple table format, many tools can be used to create them. This includes Microsoft Excel, HTML and a wiki. FitNesse is a software testing tool, a wiki and a web server combined. Since it is a wiki, adding and maintaining tests is easy. It is based on the Fit framework. FitNesse is available for free at www.fitnesse.org. While FitNesse is written in java, versions for other programming languages such as C++, Python, Ruby, Delphi and C# exist. Note that the developers of FitNesse express that FitNesse should be used in addition to unit tests, and not instead of them.

3. State of the research

Fit seems to be an easy tool to learn. The average introduction of the basics is a 3-4 hour crash course plus relevant readings. Melnik et al. report that 90% of technology students managed to deliver their Fit test assignments after a brief introduction [6]. Unfortunately there isn’t much clear evidence about non-technical users. Business graduates (acting as clients/users) found it hard to learn, but eventually produced good specifications with ease [7]. The teams mixing business and computer graduates were

<table>
<thead>
<tr>
<th>Division</th>
</tr>
</thead>
<tbody>
<tr>
<td>numerator</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>12.6</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Figure 1. Fit test table execution

---

1 http://fitnesse.org/FitNesse.AcceptanceTests
no better at describing a good quality specification than pure computer graduate teams. One company describes successfully adopting and using Fit tests [8]. However, this specific customer had an information systems background. When asked whether they would use Fit again, two studies show positive responses [7, 9]. Findings show that there was no correlation between students’ own perception of the value of the tool and the quality of the requirements specification they produced [7].

The general impression is that Fit is a good tool for specifying requirements. Melnik and Maurer found that 80% of students in an experiment preferred requirements specified in Fit as opposed to prose [10]. Similarly, Read et al. noted that the average student in his experiment felt Fit to be adequately appropriate for defining functional requirements [9]. The same students rated writing Fit tests as just as difficult as writing JUnit tests and requirements specifications in prose. 43% of them found the Fit specifications good enough to implement a web service. 26% managed but had problems. In a small survey of test notation tools by Geras et al., Fit scored best on ‘ease of use’ (together with scripts and XML) [11].

Interestingly, in an industrial case study Prashant et al. found that customers don’t always want to write the tests [12]. 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 (diagrams, mock-ups and call-outs) that the customer brings to each iteration meeting [8]. This made it easier for the customer 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 tabular structure of the Fit tests was seen as superior to XML for writing tests [13]. Being able to write test code needs to be easier than writing normal code [14].

Writing acceptance tests has at times been given lower priority, because of the complexity [8]. 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. Fit has been attributed to help 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 that Fit reduces noise, over-specification, and ambiguity [6].

One effect when projects grow large is the growing need to organize the specifications. Prashant et al. report that one company started storing tests in different directories according to their lifecycle status [12]. They also think their successful refactoring of a large application part was due to their extensive readable Fit test coverage. The use of semi-automated Fit test creators has been suggested to help with these problems [15].

Melnik et al. 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 [7]. They think this is an issue which requires further study.

One problem in using tools for development lies in letting the tool drive the collaboration and not vice versa. Interestingly, Prashant et al. noted in their study that one group claimed Fit initially hindered communication [12]. They focused too much on preparing ‘syntactically correct Fit documents’ for the development team, and didn’t focus on developing the most appropriate specification. 41% of student groups developing software using Fit had no contact with the test suite development team during development [9]. The Fit suite summary has by some student groups been used as a project dashboard [16], this could also be used by project customers as a feature to keep track of project status.

According to Prashant et al., the Fit specifications should be the main definition of the specification, and should be integrated in the continuous build process [12]. Abstraction should not come in the way of readability. Organizing a team is always a struggle. One team ended up with so many Fit documents that they created a special team to deal only with this [12]. This increased the Fit development, and enabled analysts to effectively write detailed Fit documents. The result was that developers ended up 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 a project, yet still succeeded. This they claimed to be 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.

Coding to make an acceptance test pass can take a lot more time than for unit tests [17]. One problem arises when trying to separate between ‘unimplemented failures’ and (the more important) regression failures. A possible solution made by Deng et al. was to create an Eclipse add-on,
FitClipse, which remembers 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 they warn about not thinking outside the box [10]. 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 [11]. This included performance, security, and usability.

4. Case study context, method and limitations

To build on the knowledge base on Fit and AAT we studied a software development project at a medium sized Norwegian consultancy house that applied both Scrum and the Fit acceptance-testing framework. The case project is an extension of a previous delivery starting in 2003 and was still ongoing at the time of the study. The team varies from 5 to 10 persons. The customer consecutively pays for resources spent (time and materials). The system being maintained is part of a larger system of subsystems delivered by other suppliers. The development team adopted Fit about a year before the study took place. Fit tests were mainly used to test interfaces to external components. To collect data from the case project we interviewed four developers about their experience in using Fit tests. Each interview was made on-site lasting for 30-40 minutes and was done semi-structured (using an interview guide) to allow the respondents to reflect. The interview guide did not contain simple questions of the yes/no type but a set of topics for reflection based on findings from existing literature on acceptance testing (documented in the state of the research section). The interviews were recorded and transcribed by a professional language service company. The resulting four transcripts were analyzed according to the principles of constant comparison [18] meaning that all transcripts were analyzed to identify common feedback and experience. In the analysis we followed the structure of the interview guide to identify conformance or divergence in the respondents’ replies as well as looking for topics of relevance not covered by the interview guide but that emerged through the respondents’ reflections. The summary of the interviews is reported in the following chapter.

This study design has some limitations. Firstly, we have studied only one project and interviewed four developers. Thus we are not able to generalize our conclusions, however – studies of automated acceptance testing and the use of a framework such as Fit are few and the practice is relatively new. Thus we believe that our findings and conclusions are of practical importance to other similar software organizations considering automated acceptance testing and other researchers addressing similar topics. Secondly, our data and analysis are strictly qualitative focusing on practitioners’ subjective experience in a real industrial setting, and not on quantitative metrics such as quality and productivity. We believe that, in such an early phase of the introduction of a new agile technique and tool, it is important to investigate the usefulness as seen from the practitioners’ perspective. If experiences are found to be generally positive with potential improvement issues, we believe this and other similar studies can prepare for more quantitative studies.

5. Results

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

5.1 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. The developers had worked so long within the domain that they felt they at times had better domain knowledge 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. Each developer decided if tests were needed and flowingly had the responsibility of writing tests for the feature he/or she worked on. Sometimes this was based on consulting others in the team. The project appointed a person 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 are considered to be not worthwhile to test this way.
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. 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 project gave the developers a one-day introduction in the form of a workshop.

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 synchronized. Besides development of code, the tests were also modified either as an outcome of test results or that a customer reported an error. The developers 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 are stable and need less updates of tests, while code and tests closer to the GUI tends to be more unstable and cost more in terms of maintenance.

5.2 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 on 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 current status. Tests were easier to grasp than the respective code, and acted as spreaders of competence. 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 – they perform their own testing.

Tests improving understanding of the system. The 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 for communicating with the customers they were not used for documentation of requirements.

Process control. The use of acceptance tests seems to contribute to the visibility and process control of Scrum. 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 improve 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.

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. The customer had for historical reasons not paid much attention to documentation.

Fixture/test fatness/complexity. Fixtures and tests grew large, typically resulting in many columns in Fit. They can be difficult to follow and thus sometimes large tests are broken down into several simpler ones. In general the developers try to keep the tests simple in the first place.
5.3 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 to 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
− In cases where external systems are unavailable at development time, acceptance tests acts 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 see that tests themselves acts as documentation for the developers preventing later changes from breaking the tests.

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 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 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 the 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 on 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.

6. Discussion

We started out by referring to some intended effects of using Fit tests. Can we find any support for these from our and others studies?

About communication: We see clearly from our case that Fit-tests were not used for communication between users and developers. Development team members wrote these tests themselves based on their experience with the application domain and of course based on other communication with the customer side. Other studies are inconclusive on this issue. Gandhi et al. [12] commented that customers may not want to express tests in this way. Melnik and Maurer point out that extra means of communication (infosheets) are established [8]. This may indicate that expressing requirements in the Fit table format still is too restrictive. This is aligned with the media richness theory [19] that explains an increase in communication efficiency with an increase in media richness. Another aspect of communication is the internal team communication. Here we see from both our case study and other relevant studies that having features of the software product covered by tests leads to higher confidence in the development team. As the code is "protected" by tests and thus safer to alter, developers claim that is also easier to share competence in the team.
About agility: Using Fit tests is supposed to support design changes when the needs of the business change. In our case, at the start developers tended to match all features with proper Fit tests. Demanding on resources, the practice changed to only covering the more complex features with tests. Tests were written when the need occurred, as defined by the developers. In this sense writing Fit tests seems to enhance agility – over time the test suite grew and formed a safety net that affected how flexible the developers could be with respect to quickly altering design and code. Developers would instantly see if changes negatively affected already completed features.

About balance: Using Fit is supposed to reduce time to gain balance by quicker recovery from problems. We got clear feedback from the interviewed developers that their use of Fit helped them find more errors in less time than with manual acceptance testing. This freed up time to spend on developing new features.

Further on, our findings show that very short and simple courses seem sufficient to learn the basics of Fit – 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 – aiming for 100% test coverage might not be relevant. The project we studied did not engage customers in writing tests, however Read et al. [9] indicate that persons without an IT background may find it harder to grasp. 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. We know of only one study report that discusses quality of AAT in relation to quality of implementation. Melnik et al. found that there was no such correlation [7] and state that further research is needed. Writing acceptance tests may make developers reflect on the design and system behavior in advance of programming. A similar effect, general to up-front test definition, can be seen from studies of test-driven development [2, 3]. 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 Reads student experiment that developers discovered both inserted errors and even non-intentional errors [9]. 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. Hence we see that the use of acceptance tests may improve communication and cooperation in the development team. This is in contrast to the experience in one of the other studies. Prashant 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 [12]. This seems to be a balancing act; benefit should be weighted against cost. This balance deserves further research.

Only a few studies addresses 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 [8]. As a contrast, Prashant et al. observed, like us, that customers preferred old-fashioned requirements descriptions in plain text [12]; so this matter is not fully investigated. 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 it synchronized with the code. Deng also underlines this [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 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, some even commented that it was fun. Other studies also report this opinion amongst developers [7, 9], similarly we would also like to see studies of customer opinions.
7. Conclusions and further work

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 tests for communicating better with the customers, something we will aim for in later studies. One of the most promising features of this testing strategy is to improve the link between customers and developers, both for specifying requirements and for verifying results. Yet we see that this is still a challenge even when utilizing an infrastructure like FitNesse. One step forward to realize the full potential would be to better understand why this interaction is hard to achieve. We suspect that this type of frameworks must be supported by an appropriate work process or model; a tool by itself may not be sufficient. We would also like to see more quantitative studies investigating process and product effects such as process control and product quality.

8. Acknowledgements

We would like to sincerely thank the case company for giving us access to interview their developers. This work was partly funded by the Research Council of Norway under grant #179851/140.

9. References


