Multiple Perspectives on Executable Acceptance Test-Driven Development

Grigori Melnik, Frank Maurer

Department of Computer Science, University of Calgary, Canada
{melnik, maurer}@cpsc.ucalgary.ca

Abstract. This descriptive case study is about the dynamics of a software engineering team using executable acceptance test-driven development in a real world project. The experiences of a customer, a developer, and a tester were discussed. The observed consensus among multiple stakeholders speaks of the effectiveness of the practice in the given context.

1 Introduction

Acceptance testing is an important practice regardless the type of the process followed. Acceptance testing is conducted (preferably by the customer) to determine whether or not a system satisfies its acceptance criteria. The objective is to provide confidence that the delivered system meets the business needs of the customer.

In the agile world, the iterative nature of the processes dictates automation of the acceptance tests (i.e. producing "executable acceptance tests") as manual regression testing at the customer level is too time consuming to be practical and feasible given the short timeframes of agile iterations. Furthermore, eXtreme Programming (XP) and Industrial XP advocate writing these tests in the test-first/TDD fashion. As a result, "*executable acceptance test-driven development*" (EATDD), or "*story-test driven development*" as it is called sometimes, makes it possible to formalize the expectation of the customer into an executable and readable contract that programmers follow in order to produce and finalize a working system [4].

This investigation set out to characterize and validate, in a given context, the main proposition: *executable acceptance testing is an effective tool for communicating, clarifying, and validating business requirements on a software project*. Even though we did not impose our definition of "effectiveness" on the respondents, we recognized that an effective practice and tool must address several aspects of communication. Those aspects include clarity of requirements, ability to deal with complex, end-to-end scenarios, ease-of-learning and ease-of-use. Therefore, we structured the interview guides accordingly to unveil these facets.

2 Context

The team was collocated and consisted of an on-site full-time product manager who was a domain expert (the "Customer"), a project manager, 10–12 developers, 1–4 QA engineers, and one consultant who introduced the methodology, the practices, and who also performed specialized work. All development was done in-house. The team

worked on a single greenfield project (no project switching) – the implementation of an EDI transaction platform to allow users to define business rules around delivery of certain critical documents (for example, purchase orders) via a Web interface, to execute those rules, and to notify some parties who needed to be notified. It is important to stress that this was not a simple rule engine. To give the reader a sense of the project caliber, there were about 7,000 acceptance tests.

The team adopted XP (with a coach) and diligently carried out all practices of XP. The iterations were two weeks long. The project lasted 10 months, and despite some difficulties and growing pains, it was successful – the team was able to release a high-quality, feature-complete application on time (as unanimously recognized by all respondents – the Customer, the Tester, and the Developer). In addition, the marketing was satisfied and accepted the system, the existing clients of this vendor were happy with the product and the vendor even managed to sign up new clients.

Three members of the team took part in this study: (1) *the Customer*, whose job was to identify a high-value set of user stories for each iteration and do what was necessary to help the developers understand and implement these stories; (2) a lead QA engineer ("*the Tester*"), whose job was to review acceptance tests specified by the Customer, suggest new scenarios, find problems, and in any other way help the team to understand what was going on; (3) a lead developer ("*the Developer*"), whose job was to implement the system that met business requirements of the Customer.

It is important to note that the Customer had an information systems background. While he was not a software developer, he was performing a job similar to what a business analyst would do. Therefore, this report does not make any speculative generalizations on whether a non-technical customer would be as capable as the one we interviewed. In fact, the chances are likely that it would not be the case.

3 Discussion

3.1 Learning the practice

The team adopted executable acceptance test-driven development and the FIT framework [2] with no prior experience in the practice. A 4-hour intro was given to all team members. During the first iteration, the consultant assisted the team with writing user stories and acceptance tests. After that, the team felt comfortable specifying their business rules in the form of acceptance tests. According to all three respondents, learning the technique and the framework was easy. According to the Tester, after a couple of days of "playing" with FIT, the team could operate the framework and write basic test scenarios. Everyone on the QA team caught on within a week and was able to get on their own with the job of writing and running acceptance tests. The learning curve was quite short. The Tester enthusiastically noted that "FIT is simple!"

3.2 Reflection on the practice

Because of the inherent complexity of the domain, for each iteration meeting, the Customer would prepare an "info-sheet" – a short (one- to three-page), informally-

written document with plenty of diagrams, callouts, and, most importantly, mock screen shots. It was meant to describe characteristics, behavior, and logic around a coherent set of features. It was not meant to be an authoritative specification and no official signoffs were used. The iteration planning meeting would involve the following: (1) Discuss the info sheet and talk about functionality; (2) Resolve any general questions about functionality; (3) Define a user story; (4) Define acceptance tests (criteria) for that story; (5) Repeat 3, 4. Notice, defining acceptance tests did not mean coding them in FIT. Initial "sketching" of the test was done at the back of an index card. When the list of possible test cases got longer, the testers suggested recording them in a spreadsheet – "something that we could later go back to". Later, either the Customer or the Tester would create an actual FIT table. The Customer explains: "We defined all requirements in general groups. I went into the planning meetings with well-described 'featurelets' and came out with stories and ideally acceptance tests." An example of a story could be "Rule: deadline dates are all treated as Eastern time". The team then identified all places in the system where the time was relevant (the UI, database, email, etc.) They stopped – that was enough to write an acceptance test. The developers could start their work based on the user story and acceptance test summary. They would have the detailed tests before their implementation was completed. This story-by-story procedure, including the invention of the info-sheets, matches the pattern of specifying business rules with executable acceptance tests the investigators expected. Importantly, the test-first paradigm of development was truly adopted and followed throughout the project.

We pursed the line of inquiry to understand why the info-sheets were necessary. The Customer aimed the info-sheets "at where they needed to be – to communicate the context to developers". Stories were isolated and stand alone. So the team talked about the stories, but "stories and talking are not great for communicating the details that should persist". The info-sheets would help to answer the questions why the developer was doing this or that and what this piece connects to. This is illustrative of a common concept that the customer needs to come to the iteration planning meetings prepared and have a very concrete understanding of what he wants from the upcoming iteration. This understanding can be documented upfront (if the customer thinks that this is beneficial – which was the case with this project and this customer).

Testers and the Customer paired up often with developers when specifying test scenario details (what data to use, what actions to execute, etc.) "Sitting down with the developers and giving feedback to them – they didn't need much more than that". Everybody agreed that "it was very interactive between the developers, the QA, the Customer – everyone!" The Tester pointed out that the "open space led a lot to XP thinking and very open communication. Everybody knew what everybody else was doing". It is worth reminding that the team size was ideal for this type of the process (13 – 18 people) and in a different setting (larger team or non-collocated team), the results may have been different.

While working on a story, the team may have realized that they had missed several cases. In this event, additional acceptance tests would be written. This occurred 30-50% of the time. The phenomenon can be explained by the nature of continuous learning about the domain and the system through testing (this aspect of continuous learning is emphasized by the thought leaders and practitioners of the context-based school of testing, and exploratory testing, in particular [1]). During iteration planning,

one often cannot think of all acceptance test scenarios, but as the person dives in the story implementation, other things become apparent and new scenarios are added.

3.3 Acceptance test authoring

All acceptance criteria were specified by the Customer and the QA. When it came to actual authoring of tests in the form of FIT tables, about 40% of all FIT test pages were written by the Customer, 30% by developers, and the remaining 30% by testers (based on the estimates provided by the Customer and the Tester). The Customer found that "in practice, it was best if the Customer wrote acceptance tests. This is related to the fact that going from a general description to a test has some fluidity in interpretation." Because of the domain complexity, the customer either had to communicate in greater details what the test should be and then review it, or simply do it himself. The Tester reported specifying acceptance tests in pairs with the actual developers of a story or with the Customer. If it was with the Developer, the acceptance tests would be reviewed by the Customer in an informal review session (that usually took no more than 10 minutes and was done on the fly). This was possible due to team collocation and informal communication flow.

The Developer indicated that for negative tests, they wrote sophisticated error messages (and comprehensive checks) to convey the meaning of what may have caused that error. Moreover, the developer went beyond functional tests in FIT. They extended the FIT framework to capture runtimes and do basic load testing.

3.4 Challenges in specifying requirements in the form of acceptance tests

Several experts in the industry question the expectation of the agile teams for the customer to write acceptance tests (see, for example, [5]). Therefore, the customer's opinion of the difficulty of specifying executable acceptance tests was especially important to this investigation. The Customer testifies: "[It was] not particularly hard… Because we were all there (developers, testers, and I [the Customer]) talking about the story. So, the acceptance test was a natural segway." Apparently, the difficulty was not the practice itself, but the discipline of doing it. "Once functionality was discussed and the stories were defined, the team wanted to be done. Forcing ourselves to think in detail about what tests needed to be performed and what the logic of those test scenarios should be, was hard". Devoting proper attention to the tests at the beginning was something the team had to work on. This question of discipline was intriguing, so the researchers pursed the line of questioning further. The Customer recognized that putting off writing an acceptance test was "a dangerous thing" (even if it did not happen frequently). He paraphrased from the book "Zen and the Art of System Analysis" by Patrick McDermott [3]: "We delay things because they are either difficult or unpleasant. Difficult things become easier over time, and unpleasant thing become more so". The question was whether the team was postponing writing the acceptance tests because they were "difficult" or because they were "unpleasant". It turned out that it was usually because of the "unpleasant" aspect. The Customer explained: "It was complicated stuff to test, and the thought of diving into that

complexity, just when we thought we were done, was unpleasant." The team did realize that it had to put discipline into this.

All in all, both the Customer and the Tester were quite enthusiastic about EATDD and, specifically, FIT. The following testimony of the Customer illustrates one of the reasons for this enthusiasm: "FIT is definitely more accessible and I could write FIT tests. That was huge!" Doing so helped the Customer and the team to discover a lot of missing pieces or inconsistencies in a story. The FIT tests were concrete.

4 Conclusions

The Customer and the Tester decisively recognized the effectiveness of the executable acceptance test-driven development for specifying and communicating functional business requirements. In his own characterization, the Customer "was happy". The Tester enthusiastically declared "It [EATDD] made the whole testing process more focused. It made it more unified – everybody agreed on the tests – it was the same tests running over and over again. It made our code a lot cleaner. When we found bugs in our system, we would go and update our FIT tables related to that particular function, so that we could catch it the next time it [the bug] transpires… It was just a good, fresh, new way to run the testing process. The other thing that I loved about it is, when you found a defect and you wrote a test around it, if it was a quality test, it didn't happen again – it was caught right away. Obviously, it made my job [as a QA] much easier and made the code a lot better."

Furthermore, the Customer did an internal survey of the team and found that the developers felt that the info-sheets together with iteration planning meeting were quite effective. As mentioned earlier, the developers may have been less enthusiastic about FIT from time to time as they deemed writing acceptance tests in FIT required more effort than implementing them in JUnit. However, there was no argument about the value of FIT tests from the perspective of making the tests "as English as possible" (i.e. readable and intuitive). This is remarkable as it clearly demonstrates the consensus among all three interviewees – the Customer, the Developer, and the Tester – on the value and effectiveness of executable acceptance testing.

References

[1] Bach, J. "Exploratory Testing Explained". Online: http://www.satisfice.com/articles/et-article.pdf

[2] Fit: Framework for Integrated Test. Online: http://fit.c2.com/

[3] McDermott, P. Zen and the Art of Systems Analysis: Meditations on Computer Systems Development, 2/e. Writers Club Press, Lincoln, NE: 3, 2003.

[4] Reppert, T. "Don't Just Break Software, Make Software: How Story-Test-Driven-Development is Changing the Way QA, Customers, and Developers Work". *Better Software*, 6(6): 18–23, 2004.

[5] Sepulveda, C. "XP and Customer Tests: Is It Fair?" Online: http://christiansepulveda.com/blog/archives/cat_software_development.html