Communicating Domain Knowledge in Executable Acceptance Test Driven Development

Shelly Park and Frank Maurer

University of Calgary, Department of Computer Science
2500 University Drive NW, Calgary, Alberta, Canada, T2N 1N4
{sshpark,fmaurer}@ucalgary.ca

Abstract. We present results of a case study looking at how domain knowledge is communicated to developers using executable acceptance test driven development at a large software development company. We collected and analyzed qualitative data on a large software development team's testing practices and their use of a custom-built executable acceptance testing tool. Our findings suggest that executable acceptance tests (1) helps communicate domain knowledge required to write software and (2) can help software developers to communicate the status of the software implementation better. In addition to presenting these findings, we discuss several human aspects involved in facilitating executable acceptance test driven development.

Keywords: Executable Acceptance Test Driven Development, Requirements, Ubiquitous language, Domain knowledge, Knowledge management.

1 Introduction

The purpose of the Executable Acceptance Test Driven Development (EATDD) is to facilitate better communication between the customers, domain experts and the development team by mandating that the requirements must be specified in form of executable acceptance tests. This general idea is currently called many names: functional tests [1], customer tests [2], specification by example [3] and scenario tests [4] among many. Recently, EATDD is becoming very popular in the agile software engineering community.

What is distinct about the EATDD process is that requirements are specified in executable acceptance tests by the customer instead of using natural language. An executable test either succeeds or fails, i.e. there is no ambiguity. Any specification must be understandable and writable by domain experts as well as the development team members. Evans stated that finding a common communication language or a Ubiquitous Language [5] is important in communicating the correct requirements between customers and the developers.

To pursue a better understanding of the functional requirements specification process in EATDD in terms of how the domain knowledge is communicated across different stakeholders, we performed a detailed case study at CGI [6], a large international information technology and business process service firm with offices located in

P. Abrahamsson, M. Marchesi, and F. Maurer (Eds.): XP 2009, LNBIP 31, pp. 23–32, 2009.

16 countries. The software developers at CGI in Calgary office were given a challenge to build oil & gas production accounting software by their clients. Production accounting deals with accounting involved in acquisition, exploration, development and production of oil and gas reserves. The production accounting domain knowledge is very complex and no single developer can understand the client's domain in its entirety. The development team follows Agile development practices and started to use executable acceptance test driven development (EATDD) based on its perceived usefulness for communicating requirements between the developers and the business representatives. The team built a custom in-house testing tool to facilitate their requirements elicitation and acceptance testing process. The purpose of our case study is to understand how the custom testing tool helped establish a Ubiquitous language between the business domain experts and the developers. This is a qualitative research paper that looks at how the tools facilitated the communication of complex knowledge to the developers.

The primary audience for our research paper is researchers in EATDD field in Agile development environments. In addition, the broader audiences include industry practitioners who are interested in using EATDD. This paper provides a first in-depth longitudinal case study on the EATDD practice. The project under study is running for four years. We collected data about the software development project and corroborated our findings with interviews and direct observations. We investigated their in-house custom executable acceptance testing tool and its impact on the development *process.*

Section 2 presents related research on EATDD. Section 3 presents a research methodology and our research design. Section 4 provides background information about the development project. Section 5 presents our observation. Section 6 discusses the significance of our research and section 7 presents the threats to validity in our research design. Section 8 concludes the paper with some final thoughts on our case study and presents the next step in our research.

2 Related Work

EATDD is inspired by test-driven development (TDD) [1]. Similar to TDD, EATDD asks for requirements as a set of acceptance tests that can test for functional software requirements. What is distinct about this process is that requirements are specified in executable acceptance testing form by the customer instead of in a natural language. The specification must be writable and readable by customers and the development team members. The practice became particularly popular in the Agile development community, especially with the introduction of Cunningham's testing framework called Fit [7] and their subsequent add-ons such as Fitlibrary[8] and Fitnesse [9]. Similar testing frameworks are also becoming available in the market, e.g. Greenpepper [10]

Melnik et al. conducted a study on how students perceived and used Fit testing framework for specifying software requirements [11]. Their result shows that despite the students' objection to writing the executable acceptance tests in the beginning, all of the students were able to convey the functional requirements completely. At the end of the course, 80% of the students said they didn't want to specify all of the requirements in prose and would prefer a tool like Fit. Read et al. found that about 2/3 of the students said they will use Fit for acceptance tests in the future [12]. Sauvé et al. also corroborated the finding that students were able to improve their communication through their custom testing tool called EasyAccept [13].

Melnik et al. also conducted a study with industry practitioners. They found that customers and testers "recognized the effectiveness of the executable acceptance test-driven development for specifying and communicating functional business requirements", but the developers preferred unit tests over Fit tests. Ricca et al. found that Fit tables can help clarify a set of change requirements in the software with improved correctness with no significant impact on time. However, they found that the benefits of Fit vary by developers' experience [14].

While these studies suggest that executable acceptance tests can improve communication about functional software requirements, the empirical basis is still limited. The existing research work still doesn't explain why some people are very enthusiastic about it and some are very hesitant. We also need to find out how executable acceptance tests should be specified and what additional purpose these tests serve.

3 Research Approach

In this section, we describe our research methodology and our research design. Human factors are difficult to measure because it is impossible to measure the software engineering process independently of the practitioners. When we investigate a real-life software development project, we can't always measure, control or identify all of the factors that influence the development process [15]. Therefore, to facilitate our qualitative research in such settings, we decided to leverage a case study research strategy, which is a set of research procedures that are designed to facilitate research where researchers are interested in "how" and "why" of the phenomena under study but has very little control over the behavioral events. The purpose of using a case study method is to deliberately uncover the contextual conditions that led to the phenomenon "believing that they might be highly pertinent to [the] phenomenon" under study. In a software engineering context, Fenton et al. [16] label case studies as "research into the typical". Case studies can help discover the relationship between humans and the environmental context.

The goal of the case study research strategy is to contribute a hypothesis about the phenomenon under study mainly through *analytical generalization* rather than *statistical generalization* [17]. It means that although we cannot make any definite conclusion based on one case study, a single case study can provide very rich insights into the problem under study, especially when the evidence is consistent with previously existing research work.

Our case study is done at CGI [6]. While we were collaborating with CGI for an extended period of time [18], the fieldwork for the current study was done over two additional days during which we conducted in depth interviews with members of the CGI team. Our results are based on newly collected data as well as insights from previous collaboration. The new data represents over four years of development practice. We interviewed three additional software developers and one project manager for detailed data collection. In addition, we interacted with five additional software developers to corroborate our findings. We did not record the interviews, because recording the interviews was seen as to be too intrusive in the company environment. However, any interesting remarks made by the developers were carefully written down during the interviews along with any observation we made. The length of the interview varied

greatly depending on how much information the developers were providing for our research. We also participated in one daily scrum meeting and observed three other daily scrum meetings. Our empirical data also includes direct observation of a developer who was engaged in a debugging process of a failing executable acceptance test, which lasted about one hour. We wanted to understand how acceptance tests were used to fix the failing code.

Our analysis of the collected data involved open coding and code categorization using our field notes [19]. During open coding, we identified a set of codes that could provide most insights into the data from our field notes. Then we categorized the codes to determine the relationships among the identified codes and a list of themes was generated.

4 The PAS Project

A case study assumes that contextual condition is important in the phenomenon under study. Therefore, in this section, we are going to briefly describe the PAS development project. PAS is production accounting software for the petroleum industry. The purpose of oil and gas production accounting software is to keep track of oil and gas production and calculate various capital interests invested in the oil and gas wells. CGI already had an existing production accounting software called CGI Triangle, but the system needed to be rewritten due to the obsolescence of technologies used for its development.

The CGI PAS project was sponsored by four of the largest oil and gas companies in Canada. An oil and gas production accounting system is an extremely critical software system for oil and gas exploration companies, because engineers and production accountants not only have to keep track of their oil and gas productions, but they need to be able to calculate tax and various interests[1] being represented in their oil and gas wells. Because each country and province has their own unique set of regulations on tax and interest calculations, it is important for the oil and gas companies to use software that reflects the regulations for the political jurisdiction where the well exists. The amount of engineering and accounting knowledge involved in the petroleum industry in order to build PAS is so complex that it is absolutely impossible to build the software without having someone with the expertise who can lay out the information properly to the software developers. While all software development requires business domain experts, the problem with PAS is that production accountants need years of training before they understand the domain. The knowledge is not easy to be picked up by developers on the side. The team also needs someone who can keep track of the changing set of government regulations in the oil and gas industry.

The number of software developers in PAS project fluctuated over the last four years of its development; therefore, the amount of knowledge about the project development within the team fluctuated. At the time of the fieldwork, PAS is already deployed and operational in the client's work environment. Our contacts in the company told us that there are about 80 software developers, testers and clerks at the time of the fieldwork. For an Agile development team, it is quite a large team. The team is split up into several subteams (one for each major component).

[1] Interest: how costs and revenues are shared by stakeholders.

The development area is a large open space. Each subteam had, what they called, a SPOC (Single point of contact) and an SME (subject matter expert). A SPOC communicates the progress of the team and addresses any concerns that other teams might have on the component they are building. The SME is the person who has the domain expertise to define the requirements, answer any domain-related questions from the developers and test the end products to ensure that the requirements were correctly understood and implemented by the developers.

Each subteam had 2 to 8 developers and they stay as a team only for the duration of building the specific business component. Each team holds a daily scrum meeting in the morning. Due to the size of the team, each team had separate scrum meetings and each team sent a team representative to inter-team daily scrum meetings. In addition, they kept track of bug lists using Jira [20]. There were 927 'GUI Smoketests' that test the user interface layer, 93 report regression tests and numerous unit tests and other types of tests that we did not look into carefully for this research.

5 Observation

In this section, we present our observations by describing CGI's executable acceptance test-driven development process. We first describe our observation and then we summarize the implications of our findings.

5.1 Choose the Requirements Specification Tool from the Customer's Domain

The requirements specification is defined using the language and formats of the business domain. This can reduce the extra overhead of learning the specification tool by the domain experts (customer representatives). Our case study shows that the domain experts chose Microsoft Excel as their requirements specification tool, which is a standard tool for communicating production accounting data in the oil and gas industry. The standards for the data formatting are regulated by the provincial energy regulatory boards [21] and production accountants generally use Microsoft Excel to facilitate their business communication. Although production accountants are not required to use Microsoft Excel, it is a common practice to do so in the oil and gas industry.

Microsoft Excel became a preferred tool for requirements specification, because Excel was familiar to the domain experts. Excel has features such as VB macro programming and pivot tables. The domain experts understood and used these Microsoft Excel features proficiently. An acceptance test file has 12 macros, which are used to create these table templates and help with the test automation process.

One Excel file contained many requirements and the developers considered one Excel file as one acceptance test. Each acceptance test is composed of many calculation worksheets, which are represented using Excel worksheets. Each worksheet can contain 20 to 650 rows of test data and about 3 to 30 columns. Each row can return more than one expected output result.

There are a total of 17 business components in the PAS project. Each business component had multiple acceptance tests and each acceptance tests had multiple worksheets (or the developers called them 'views'). There are a total of 321 acceptance tests for the PAS project. The domain experts were able to write and maintain these tests whenever a feature is added or changed in the software.

The use of Excel was motivated by the ease of transferring and codifying the domain knowledge via this medium by the domain experts, especially because Excel is the conventional tool in the business domain. It is easier to codify tacit knowledge to explicit knowledge if the tool is already utilized to facilitate communication in the customer's domain. In addition, (1) domain experts who are familiar with the specification tool are more likely to create and maintain acceptance tests for the developers and (2) the tool can communicate the domain knowledge best because it can represent the domain data appropriately. What makes Microsoft Excel interesting is that this tool is universally well known and easy to learn by both the customers and developers. Finding a common tool that can be used and understood by business experts as well as the development team is a crucial and important condition for a successful EATDD process. Excel not only allowed the production accountants to leverage their existing computer skills for writing the requirements, but it also provided the contents in a form that developers can easily turn into acceptance tests.

5.2 Communicating the Business Domain Knowledge

In this section, we are going to analyze the kind of knowledge their acceptance tests are conveying to the developers. Due to the page limitation, Table 1 shows only a very small snapshot of a large executable acceptance test that has 644 rows and 14 columns, which is a typical example set of scenarios required to test for real-life production accounting. Table 1 is testing for a contract allocation. The last three columns show the sum of the expected share of the energy for everyone who has a stake in the reserves profit. We found that most of the tests are transaction style calculations. Generally the tests identify *where (a physical entity* such as reserves or a facility), *who* the transaction is for, *what* values should be assigned and optionally *when* the transaction occurs. The format is typical of any production accounting spreadsheets. The test data is created by the domain experts using similar production data, but the data is so realistic that it could be the real production accounting data.

Table 1. An Example Snapshot of Executable Acceptance Test Definition

Fac ID	Cont Name	CParty Name	Prod Code	Cont Type	Settle-ment	Sum of VOL	Sum of PRICE	Sum of VAL
1000405 00722W 500	FH roy-alty C5	Farmer Cassie	C5-SP	Royalty	Cash	2.76	339.98	938.37
					Cash	2.76	339.98	938.37
	FH TIK on gas – 100%	Kathy and Co.	GAS	Royalty	TIK	2.2	235.69	518.51
					TIK	2.2	235.69	518.51

The domain experts also provided workflow diagrams that are not executable, but they complement the executable specification. They were designed to inform the developers about the business workflow for the acceptance test. Figure 1 shows a

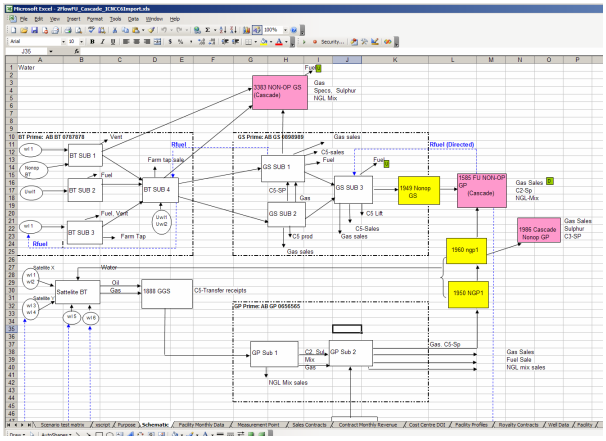

Fig. 1. A diagram explaining the business process involved in a battery facility

workflow diagram of a battery facility[2]. Our analysis shows that the domain experts need to provide two types of documents: testable requirements specifications and an overview document to put the specification into context. The overview document can be used by the developers to point at something to ask for more information about the domain.

5.3 Making the Requirements Specification Executable

The testing framework is based on Excel, Ruby and JUnit. Ruby is used to automate the user interface layer. JUnit is used to execute the script file and to compare the test output values. Executing one acceptance test can take up to 1 hour or more, because it simulates real-life production data and calculations and it runs against the UI layer. A production data file contains months or even years of production data. The largest Excel file is about 32 megabytes in size.

Figure 2 shows a time-series analysis graph of percentages of succeeding acceptance tests captured at the end of each sprint. Notice that over time, the developers became much more conscience about passing the acceptance tests. But we cannot make a definite conclusion about the quality of the software from the graph.

We wanted to get better understanding whether the graph reflected the amount of domain knowledge that was transferred to the developers. We asked the developers to explain specific parts of the acceptance test that were failing. They explained it in terms of how they can fix the problem technically, but they could not explain the domain knowledge behind these tests by putting it into context of the industry. However, they knew enough to explain why the test would fail. Based on our interview data, we can assume that the team's understanding of the business domain is fragmented across many developers. It also meant the executable acceptance tests provide enough information for the developers to implement the code even if they have limited understanding about the domain in which it will be used. The regression tests using executable

[2] Battery facility: a plant where raw petroleum is separated into different types of hydrocarbons.

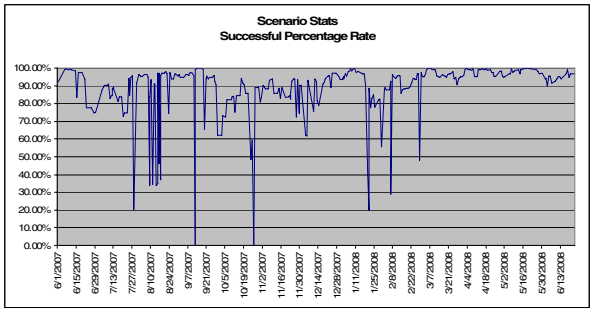

Fig. 2. A time-series graph showing the percentage of acceptance tests succeeding at the end of each sprint. 0% success rate was due to a test automation problem at the time rather than any serious software malfunction.

acceptance tests were important because they highlighted this knowledge gap among the developers. It made the knowledge gap transparent to everyone. Therefore, they knew who and when to seek out help when these tests failed. Unlike unit tests, failure in executable acceptance tests meant they misunderstood domain knowledge, thus it signaled possible problems in delivering business value to the customer.

They did not need in-depth production accounting training to fix the software problem, because executable acceptance tests usually gave enough information to identify the problematic code and also provided the expected answer. However, a more in-depth empirical study is required to understand the developer's cognitive processes involved in going from failing executable acceptance tests to fixing the code.

6 Discussion

We categorize our findings into three categories: *communication, medium* and *context.* We discovered that the purpose of having executable acceptance tests is to *communicate* the domain knowledge to the software developers. The acceptance tests were a feedback system for the developers to confirm their understanding about the requirements and the business domain. Previous research also validates our finding [23, 24]. More than any other types of development artifacts, executable acceptance tests are utilized to fill the knowledge gap between the domain experts and the software developers. The failing tests from the automated regression tests are a good starting point for developers to question their understanding about the domain. Without such feedback system, the developers would not know how to validate their understanding about the requirements. However, no one previously looked at how specific types of domain knowledge are written in executable acceptance tests.

The *medium* used for acceptance test specification is important for successful EATDD process. Previous papers found that tools are important [24]. However, we discovered that tools are important for the domain experts more than the developers. The team discovered that production accounting knowledge is very well organized using Microsoft Excel. We hypothesize that Microsoft Excel made the test specification easier for the domain experts. By giving more power to the domain experts, the

developers were able to gain valuable acceptance tests that became critical artifacts for their success. We believe that directly utilizing the language and formalism of the business domain (instead of development oriented languages and formalisms) will improve communication between the business side and the development side of a software project.

We also discovered that the *context* is also important. First, we discovered that the acceptance tests not only provided testable example data, but we also need to provide overview documentation about the domain knowledge and business workflow diagrams. The extra information helped communicate the necessary domain knowledge needed to understand the acceptance test specifications.

7 Threats to Validity

For external validity, we believe our case study can be generalized to very large software development projects where the software developers do not have complete understanding of the domain knowledge. However, our case study only supports transactional style domain data. For threats to internal validity, the observation was collected and analyzed mainly by the first author, which may have introduced some unintended bias. Our interview data also mostly reflect the perspectives of the developers. To ensure construct validity, we performed a detailed inspection of the tool and the requirements. We are confident about our findings, because our observation and test data represents over four years of development practice.

8 Conclusion

Our case study suggests that a successful software development process is one that allows the developers to quickly verify and validate their understanding about the domain knowledge as well as the software requirements. However, in order to do so, the software developers require a reason to believe that their domain understanding needs further inquiry. An automated feedback system, especially using executable acceptance tests, allows the developers to ask the right question to fulfill their task.

References

1. Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley, Reading (1999)
2. Jeffries, R.: What is XP?, `http://xprogramming.com/xpmag/acsFirstAcceptanceTest.htm`
3. Fowler, M.: Specification by Example
4. `http://www.martinfowler.com/bliki/SpecificationByExample.html`
5. Kaner, C.: Cem Kaner on Scenario Testing: The Power of 'What-If' and Nine Ways to Fuel Your Imagination. Better Software 5(5), 16–22 (2003)
6. Evans, E.: Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, Reading (2003)
7. CGI, `http://www.cgi.com`

8. Fit, http://fit.c2.com
9. FitLibrary, http://sourceforge.net/projects/fitlibrary
10. Fitnesse, http://fitnesse.org
11. Greenpepper, http://www.greenpeppersoftware.com/
12. Melnik, G., Maurer, F.: The Practice of Specifying Requirements Using Executable Acceptance Tests in Computer Science Courses. In: Proc. of 20th OOPSLA 2005, San Diego, USA, October 16-20, 2008 (2005)
13. Read, K., Melnik, G., Maurer, F.: Student Experiences with Executable Acceptance Testing. In: Proc. of Agile Conference 2005, July 24-29, pp. 312–317. IEEE Press, Los Alamitos (2005)
14. Sauvé, J.P., Neto, O.L.: Teaching software development with ATDD and EasyAccept. In: Proc. of SIGCSE tech. symp. on CPSC Education, Portland, USA, pp. 542–546 (2008)
15. Ricca, F., Penta, M., Torchiano, M., Tonella, P., Ceccato, M., Visaggio, C.: Are Fit tables really talking?: A series of experiments to understand whether fit tables are useful during evolution tasks. In: Proc. of the 30th ICSE 2008, Leipzig, Germany, pp. 361–370 (2008)
16. Juristo, N., Moreno, A.: Basics of Software Engineering Experimentation. Kluwer Academic Publishers, Dordrecht (2001)
17. Fenton, N., Pfleeger, S.: Software Metrics: A rigorous and practical approach, 2nd edn. PWS Publishing Company (1996)
18. Yin, R.: Case Study Research: Design and Metrics. Sage Publications, Thousand Oaks (2003)
19. Grewal, H., Maurer, F.: Scaling Agile Methodologies for Developing a Production Accounting System for the Oil & Gas Industry. In: Proc. of Agile (2007)
20. Strauss, A., Corbin, J.: Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory. Sage Publications, Thousand Oaks (1998)
21. Jira, http://www.jira.com/
22. Energy Resources Conservation Board, http://www.ercb.ca/
23. Melnik, G.: Empirical Analysis of Executable Acceptance Test Driven Development, Ph.D Thesis, University of Calgary, Department of Computer Science (August 2007)
24. Melnik, G., Maurer, F., Chiasson, M.: Executable acceptance tests for communicating business requirements: customer perspective. In: Agile 2006, Minneapolis, MN (2006)