The Requirements Abstraction in User Stories and Executable Acceptance Tests

Shelly Park, Frank Maurer
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada
{parksh, maurer}@cpsc.ucalgary.ca

Abstract

Specifying requirements is a complex task because information can be represented at varying levels of abstraction. This paper looks at two types of requirements abstractions: user story and executable acceptance test. We present how user story and executable acceptance tests are integrated in our project planning tool called Agile Planner with Fit and we argue that translation between information abstractions is an important feature in planning tools.

1. Introduction

Agile project planning is a collaborative process involving customers, development team members and other project stakeholders to discuss the requirements for the upcoming iteration, estimate the completion time, determine resource allocations and negotiate the feature priorities. In card-based planning, the customer and developers can represent the requirements with index cards whereby each index card contains one user story and a user story describes a functional or non-functional feature in the software from the customer's point of view. The story cards are used to estimate the development time, to assign team members to tasks and to allocate proper resources to each task. The story cards are important because they are the highest level of requirement abstractions from a business perspective and should be easy to understand by everyone.

Although story cards capture the requirements from business perspective, the developers and testers may require more detailed requirements description in order to effectively test and develop the software based on the proper understanding of the domain. In executable acceptance test-driven development (EATDD), the requirements are made explicit in form of tests and correctness and completeness of the software can be verified automatically through successful passing of the tests. Executable specification can help clarify ambiguities in the requirements specifications because the requirements are testable with concrete examples for one of many reasons. However, it is also rare that the acceptance tests are a complete representation of all acceptance criteria. The final acceptance must come from business representatives based on a manual test, which also can result in adding previously unspecified acceptance criteria to the acceptance tests.

Specifying requirements is a complex task because information can be represented at varying levels of abstraction ranging anywhere from vague wish lists to user stories and/or executable acceptance tests. In addition, different project stakeholders require different levels of details in the requirements specifications as well as different types of information. A lack of requirements or missing requirements can be frustrating to the developers and testers, possibly leading to miscommunications. The requirements specification method should help the users to avoid the seven sins of requirements specifications, which are noise, silence, over-specification, contradiction, ambiguity, forward reference and wishful thinking [1].

Requirements specification is rarely a one-step process. The requirements provided by the customer may have a general idea about what the customer want in the software, but it may not contain enough details for the testers or the developers to write good code based on them. In many cases, the business analysts and/or Quality Assurance Engineers (QAs) may need to work with the customers to obtain more details or clarifications.

In this paper, we look at two types of requirements abstractions: user story and executable acceptance test. We argue that translation between information abstractions is an important feature in planning tools and argue its importance based on two types of requirements modeling techniques we have chosen. We present how the two information abstractions, user story and executable acceptance tests are integrated in

our project planning tool called Agile Planner with Fit [2]. In section 2, we present the role of abstraction and what user stories and executable acceptance test specifications are. Section 3 discusses the design details and section 4 presents the analysis and section 5 presents the future work.

2. Background

Kramer and Hazzan argue that some software engineers can produce more clear and elegant software design because they have better abstraction skills [3]. Kramer and Hazzen define abstraction as "a cognitive means in order to overcome complexity at a specific stage of a problem solving situation…we concentrate on the essential features of our subject of thought, and ignore irrelevant details" [3]. Abstraction is important in solving complex problems because the problem can be solved using conceptual ideas by removing unnecessary details. The problem may be a lot easier to solve if the problem is presented at a different level of abstraction as all humans have a limited cognitive ability to handle multiple chunks of knowledge at the same time [4].

Abstraction is important in requirements specification because transferring the domain knowledge to the development team is a difficult task. The complexity of the problem is much more apparent to the development team if the requirements are presented at an appropriate level of abstraction. We believe that presenting the requirements at correct level of abstraction can help identify missing or incorrect requirements quickly, because the depth and breadth of the problem are much more apparent to readers. The team members can miss out on important information from the customer if the domain knowledge is presented at wrong level of abstraction; either missing the big concepts or missing the small important details. Additionally, the most difficult part of abstraction can be deciding which level of abstraction to use for the information that he/she is trying to present. Sometimes the skill can only come from years of experience as well as a good understanding of the background/domain knowledge that the receiver of this information brings to the development effort.

The research problem is to discover the role of abstraction in requirements specification and the necessary translation techniques required to effectively present the information using two or more different requirements abstraction techniques. In this paper, we are going to take two specification techniques: user stories and executable acceptance tests. It is important to note that neither technique actually defines a specific abstraction level, but the differences in these two techniques can actually complement each other well in presenting requirements in different abstraction level.

User stories are "high-level definition of a requirement with just enough information" to produce effort estimation and to remind everyone about the conversation with the customer [5]. User stories are even more high-level requirements definition than use cases [5]. A good scope of a user story is a task that is small enough to get reasonably accurate time estimation and to be able to finish the implemented within the iteration. The reason for using an index card to represent a user story is to provide a physical representation of the task and model the relationship between user stories using physical proximities of the cards and spatial coordination of the cards, such as clustering the cards based on sub-projects or stacking the cards based on the priorities. The use of index card is encouraged because it can convey workflow and task or component dependency information without requiring high-tech equipments or special software engineering training. Due to space constraints, index cards also do not allow all the required details to be specified on the card, which has two benefits. First, it encourages, even enforces, the developers to talk to the business stakeholders before they start working on the design and implementation of the feature. Second, having vague requirements in the beginning is actually good for the development efforts as getting the detailed requirements at the beginning are often unrealistic and it can encourage lean approaches to requirements decision making.

In addition to a short description of the user story, the card contains time estimation to complete the task, the person(s) assigned to the story card, a priority ranking and any other pertinent information that are useful to the team. Once the story cards are written up, the developers post the cards on a board that are divided into sections, which can be divided by projects, iterations, progress or any other type of categorization that the team finds to be useful. The story cards can be re-organized under different sections during daily scrum meetings [6] to reflect its status.

Executable Acceptance testing is based on the concept of test-driven development. The requirements are written in form of tests and the developers write the test code that verifies the correctness of the implementation against the specification. A well-written executable acceptance tests can provide concrete examples about the requirements and can be used as live documentation about the state of the software implementation. It is important to note that executable acceptance tests can be as equally vague as

the user stories. Only the fixtures that hooks up the tests to the code makes the requirements unambiguous as we can clearly see which part of the code is hooked up to the behavior specified in the test. However, if the customer leaves the test very vague, there is a risk that the meaning of the test is left up for interpretation by the developers.

Currently, one of the most popular executable acceptance testing tools is Fit [7] and its derivatives such as Fitnesse [8]. In Fit, a specification is written in a tabular format and the software implementation is hooked up to the tests using a fixture. The acceptance tests are useful because they define the criteria for the acceptable final system from the customer's point of view in a testable way.

3. Implementation

The implementation of Agile Planner with Fit [2] (Figure 1) is built based on two plugins: Agile Planner [10] and Fitclipse [9]. Agile Planner is a card-based planning tool. The virtual index cards can be manipulated much like physical index cards, such as stacking up the cards, enter the user story, lay them out on a canvas and spatially organize the cards. Multiple users in different locations can manipulate the cards simultaneously using telepointers, which are mouse pointers from different locations. The cards can be encapsulated inside a larger canvas called i*teration* and the user can create as many iteration canvases as needed to categorize the cards as the user see fit. Agile Planner is an Eclipse plugin, which allows the users to perform the project planning within the Eclipse IDE environment and keep the project plan in the same location as the code.

Fitclipse [9] is used to implement the executable acceptance testing part. Fitclipse is an Eclipse plugin that extends Fit [7] and FitLibrary [11]. In addition to the functionalities that come with Fit and FitLibrary, Fitclipse also keeps track of the history of the test results, enables multi-modal test execution and adds acceptance test refactoring capability. Fitclipse is also an Eclipse plugin, which allows the developers and testers to keep the acceptance tests in the same project location as the code.

Integrating these two tools faces two conceptual challenges. The first problem is information visualization between story cards and the acceptance tests. In order to accommodate the spatial constraint of the story card, we decided to represent the acceptance tests at the back of the index card. The virtual story cards can be flipped virtually and the back of the card is associated with the executable acceptance tests. By clicking on the edit button, the user is taken to the Fitclipse plugin view where the user can manipulate the acceptance tests.

Second, the state of the software development must be apparent no matter which abstraction level is chosen. For example, if the user story is half done (i.e. half of its acceptance tests are passing), that information should be readily available to the users. Currently, the user can flip the card and get the result of executable acceptance tests from the FitClipse view, although we plan to give richer visual representation in the future. For example, the user story could show a percentage complete indicator based on the state of all acceptance tests.

The approach of mapping the requirements to different levels of requirements abstraction is an innovative approach to looking at how requirements can be communicated, because executable acceptance tests can provide concrete, measurable development progress no matter which level of abstraction the customer decides to look at. The number of passing executable acceptance tests can be mapped easily to user stories and the customer can choose to view the progress of the project from whichever requirements abstraction they prefer to look at. Figure 1 shows Agile Planner with Fit (APF) with story cards. Figure 2 shows the story card being flipped in order to link to the executable acceptance tests view. Figure 3 is the Fitclipse view in order to manipulate the tests in more detail.

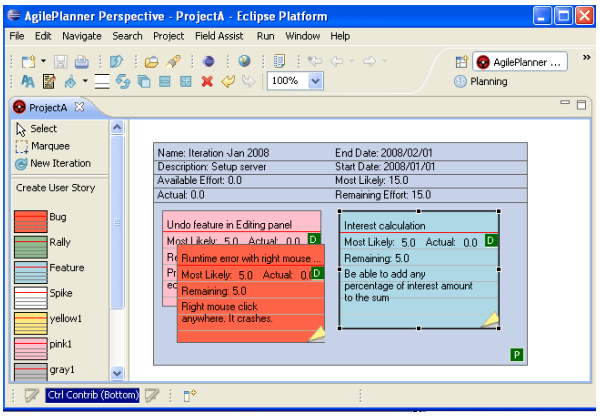

Figure 1: APF with the front view of the story cards

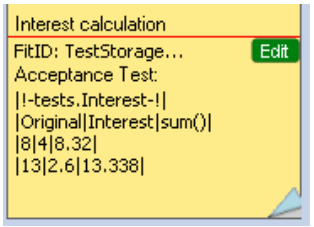

Figure 2: A story card flipped to the backside

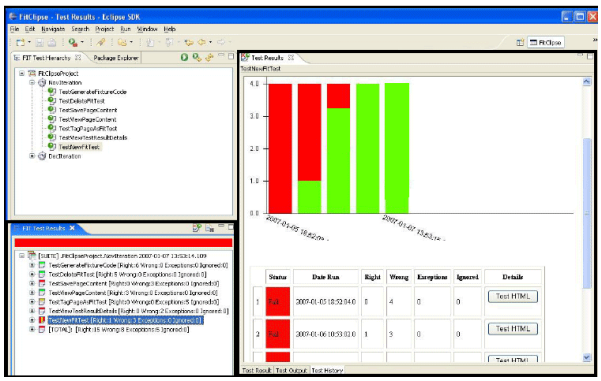

Figure 3: APF in Fitclipse view showing the Fit tests

In the next section, we examine the current state on this project and what we aim to achieve for the project ahead.

4. Analysis

The aim of integrating user stories with acceptance test is to capture the same kinds of information no matter which requirements abstraction is used. The benefit of executable acceptance testing is that the requirements are specified in a testable way, thus the project progress is readily available by analyzing the number of passing tests. The benefit of user stories is that the user stories can easily convey the big picture of the project. By linking the two requirements descriptions, the benefit of executable acceptance tests can be transferred to user stories and vice versa.

Story cards are in the risk of being outdated and being forgotten after the initial requirements analysis, much like a requirements document in traditional software engineering processes. One of the reasons why these artifacts from an initial requirements analysis become irrelevant and forgotten is that these artifacts do not easily evolve with the software evolution due to lack of linkable traceability between the code and the requirements artifact. A way to make the user story relevant for the developer all throughout the lifecycle of the project is to associate the code with executable acceptance tests and allow the test definition to evolve with the software implementation. APF allows the linking for both types of requirements specifications. The requirements gathering process naturally lends themselves well for having these two types of specification to work together well.

5. Future Work

The challenge with the spatial constraint of the story cards will be improved as a fit test can be very long and wide and many fit tests can be associated with a story card. In addition, the mapping will become many-to-many mapping between user stories and fit tests. The number of passing/failing tests, history of failing tests and the percentage finished on a user story will become readily available from the story card without flipping to the Fitclipse view.

We will be performing user evaluation with the tool to achieve better understanding about how people use the requirements techniques based on the abstraction needs. The user evaluation will give better insight into the way people formulate the requirements and how the information is presented. We will perform an empirical evaluation on the way people specify and change requirements specification based on the features that are available in the tool and discover how abstraction information is used and manipulated through APF as the requirements evolve.

6. Conclusion

In this paper, we discussed the importance of requirements abstraction in communicating and formulating the requirements in order to understand the problem better. Agile Planner for Fit tries to solve the problem by associating two types of requirements specifications, user story and executable acceptance test, to achieve the benefits from both types of specifications.

7. References

[1] Meyer, B., On Formalism in Specifications, *IEEE Software*, 2(1), 6-26, 1985

[2] Agile Planner with Fit, http://mase.svn.sourceforge.net/viewvc/mase/AgilePlannerFit

[3] Kramer, J., Hazzan, O., The Role of Abstraction in Software Engineering, *Proc. International Workshop on Role of abstraction in software engineering*, pg 28, 2006,

[4a] Miller, G. The magical number seven, plus or minus two: Some limits on our capacity for processing information, Psychological Review (62), 81-97, 1956

[5] Ambler, S. The Object Primer: Agile Model-Driven Development with UML 2.0, Cambridge University Press, 2004

[6] Schwaber, K., Agile Project Management with Scrum, Microsoft Press, 2004

[7] Fit, http://fit.c2.com/

[8] Fitnesse, http://fitnesse.org/

[9] Fitclipse, http://sourceforge.net/projects/fitclipse/

[10] Distributed Agile Planner http://ebe.cpsc.ucalgary.ca/index.php/AgilePlanning/AgilePlanner

[11] FitLibrary, http://sourceforge.net/projects/fitlibrary