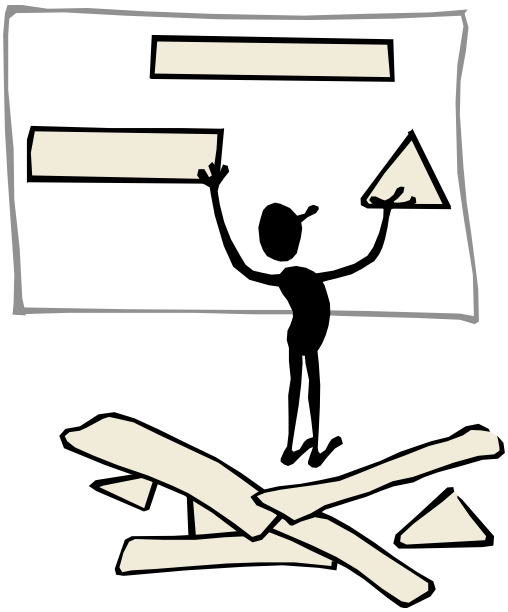
COMP33711

Agile Software Engineering

(Agile Methods Theme)

Acceptance Test Driven Development
An Overview

2013/2014 Academic Session



Suzanne M. Embury
Room KB 2.105
School of Computer Science
University of Manchester
Oxford Road
Manchester M13 9PL
U.K.

Telephone: (0161) 275 6128
Fax: (0161) 275 6236
E-Mail: S.Embury@cs.manchester.ac.uk

Acceptance Test Driven Development: Overview

Suzanne M. Embury

December 2013

Acceptance test driven development (ATDD) is a systematic process for moving from the specification for a piece of software (expressed in the form of automatable acceptance tests) to an implementation for that specification. The aim is to write code that conforms closely to this specification, and to write only such code. That is, we aim to avoid writing code that contains bugs, as well as avoiding writing code that isn't needed to meet the specification.

ATDD combines well with the practice of specification-by-example (indeed, for some authors, these terms are more or less synonymous), since the latter provides the acceptance tests which are needed for the former. In this document, we give a brief outline of the combined process, starting from stories, and ending with an implementation that satisfies the tests derived from them.

From User Stories to an Example-Based Specification

Acceptance tests (i.e., examples) can be created for a user story at any time over the course of its existence, although the most common approach is to create acceptance tests for stories in batches, when they are selected for implementation, or are reliably close to being so selected. For example, an example-writing workshop might be organised at the beginning of a release, involving the customer and other members of the team, to write the core sets of acceptance tests for each of the stories in the release. Alternatively, the tests might be written near to the beginning of an iteration, focussing only on the stories that have been selected for implementation in that iteration.

Thus, stories which are placed in the "in progress" column of a task board will normally have a (possibly complete) set of acceptance tests already written for them, when they are selected by developers for implementation. However, as we have mentioned, new tests may be added at any time, whenever a customer encounters a new scenario or a tester thinks of a new edge case, for example, even long after the team has considered the story to be "done done".

When using specification-by-example, requirements change may necessitate the addition of new acceptance tests, as well as modifications to the existing ones.

From an Example-Based Specification to Automated Acceptance Tests

Once acceptance tests have been written for a user story, they can be selected for automation (i.e., a "test automation" task can be added to the task board for that story, once it is selected for an iteration). Ideally, a framework will have been used to write the acceptance tests in a form that is amenable for direct automation (using FitNesse tables, or cucumber scenarios, for example) rather than being written in natural language. This means we have just one version of the tests that everyone works with (as opposed to having an English document containing tests and a suite of JUnit code

containing the—hopefully—equivalent tests, for instance), and so avoid transcription errors and problems of inconsistency between the two representations.

In FitNesse terms, the process of test automation involves moving from having a set of acceptance test assertions that are all coloured yellow when run, to having assertions that are coloured either red or green when run. Of course, most of the acceptance tests will be red, initially; only the small part of the desired behaviour that happens to be satisfied by the stub production code we create when automating the tests will be green.

Since we create fixtures for a type of acceptance test in most ATDD tools, the automation work needed for one acceptance test case will usually cause other tests cases to be automated too. This means that some new tests that are added (as described in the previous section) may already be automated when they are created, and may not require further attention from the team in this regard.

From Automated Acceptance Tests to A Compliant Implementation

At this point in the process, we should have a set of acceptance tests for the currently selected user stories, which are all either red or green when executed. A developer pair selects a story to work on, and focuses their attention on the specific acceptance tests for this story, while of course taking care not to break any acceptance tests that currently pass (or at least, not to break them for very long).

In larger teams, the developer pair will start by creating a separate branch in the version control system in which they can work on the new story, independently of the work of other pairs.

They then begin the ATDD process, which in outline is as follows. The pair chooses a failing acceptance test to work on. From within this test, they choose a failing assertion to work on (usually, but not necessarily, the first failing assertion they encounter when they run the test). The pair's task is now to make this one assertion pass, without causing any regression in the implementation of any of the stories completed so far; that is, once an acceptance test has turned from red to green, we try to keep it green when adding new code to the system. (It's obviously fine for new tests to be red when added, as that just means we haven't got around the implementing that part of the behaviour yet.)

To make the failing acceptance test assertion pass, the pair must make changes to production code. But, since they are using TDD, they cannot make any change to production code without first having a failing unit test. So, they now begin to use the TDD red-green-green cycle, until they have made the selected assertion pass. Sometimes this involves writing a single unit test corresponding to the failed assertion, but often multiple units of code are involved in the behaviour required by the assertion, and multiple unit tests will have to be built, covering all affected classes, until the behaviour specified by the assertion is also fully covered by the unit tests.

When all the acceptance tests for the selected story pass, and no regression of the other acceptance tests has occurred, then the pair can merge their branch with the main code branch. They may have to deal with any conflicts that arise, if another pair has been working on the same part of the code and has made changes that overlap with their own. Once the conflicts are resolved, and all tests pass, the merge can be completed and the new version of the system can be committed.