Acceptance Test-Driven Development by annotation of existing documentation

David Connolly, Frank Keenan and Fergal Mc Caffery

Abstract

Testing is frequently reported as a crucial stage in the software development process. With traditional approaches acceptance testing is the last stage of the process before release. Acceptance Test Driven Development (ATDD) promotes the role of an expert customer in defining tests and uses tool support to automate and execute these tests. This paper outlines a tool, AnnoTestWeb/Run aimed at expert customers specifying acceptance tests through reuse of existing documentation. Also outlined is a planned evaluation that includes industrial collaboration aimed at considering the impact of this tool on reuse of existing documentation.

Keywords

ATDD, tool evaluation, industrial collaboration, annotations, acceptance tests

1 Introduction

A large part of software development expenditure is attributed to *testing*. Traditionally, customer involvement in plan-driven development acceptance testing, the process of testing functional requirements with “data supplied by the customer" [1], is in the final stages of the development process preformed long after the initial investigation has completed [2]. Many reports, however, highlight that costs can be reduced by detecting errors earlier in development [3]. Also supporting this, in many domains including medical device industry software is developed in a regulatory environment with a tendency for extensive documentation. In contrast, agile approaches require *constant* customer collaboration throughout development, with customer provision of acceptance tests being an important part of this role. Often, it is recommended that tests be identified before implementation commences. In eXtreme Programming (XP) [4], for example, acceptance tests are defined as a part of the User Stories practice and, as such, are written before coding of the story begins. The practice of ATDD permits software development to be “driven by the requirements" [5]. A key advantage of ATDD in its wider context is that it leverages existing agile infrastructure including continuous integration and test-first development. One challenge is that acceptance tests need to be written afresh despite the fact that business rules are often already documented in numerous formats. However, ATDD is currently not well supported with tools that enable reusing existing documents, without rewrites, to create executable tests. A challenge therefore, is to support a suitably informed expert in performing the agile *customer role*, including easily creating tests from existing material. However, successful identification of accurate acceptance tests in this manner is not necessarily straightforward.

2 Related Work

Many approaches to conducting acceptance testing exist. Some concentrate on acting as a “recording device” allowing user actions to be replayed against a system, checking for deviations. However, this approach is mainly limited to Graphical User Interface (GUI) testing of a specific version of a system, using a tool such as the Selenium IDE [7]. Tools for writing acceptance tests in a customer friendly format and appropriate for continuous integration exist. RSpec, for example, is a “Behaviour Driven Development framework for Ruby” [8]. It promotes a workflow that involves writing stories in a somewhat prescriptive natural language style and then manually translating these steps into Ruby. While the authors consider this approach interesting for new stories, it has limitations in dealing with preexisting documents. Other open source tools aimed at supporting ATDD exist including EasyAccept that supports both tabular and sequential styles [9]. Generally, the Framework for Integrated Tests (FIT) is the most widely accepted tool for managing acceptance tests in agile development and therefore practicing ATDD [10]. In FIT’s simplest workflow a user, places inputs and some expected output into a tabular format, a ColumnFixture [11]. The developer then writes code (fixtures) that executes this data against the systems production code. Other built-in fixture included in FIT include ActionFixtures for testing a “sequence of commands” and RowFixtures for “comparing test data to objects in the system” [11]. FitNesse is a Wiki framework developed to support FIT [12]. It facilitates the editing of FIT tables in a browser allowing non-programming experts to add content. While FIT tables can be written in any tool that can export HTML, such as Microsoft Excel, these generic tools do not have any authoring features directly supporting the task domain. Existing tools that support either FIT or FitNesse includes FitClipse. As FitClipse [13] builds on FitNesse tests are entered using its wiki syntax. Mugridge introduces a process based around a library of fixtures named FitLibrary, which improves FITs “business-level expressiveness” to emphasise a “domain-driven design approach” [14]. It supports a type of fixture, DoFixtures, which is easy to read. Commercial software also supports such a workflow, with GreenPepper [15] supporting “executable specifications” while providing an expressive library of table types. However, none of these tools are focused on reusing existing documentation, so unlike the proposed approach these approaches require re-writes of content. In the requirements authoring process, Melnik and Maurer found that the use of FIT helped students to “learn how express requirements in a precise, unequivocal manner” [16]. In a number of experiments aimed at evaluating the impact of FIT tables on the implementation of change requests Ricca et al. [17], found improvement in the correctness of code produced and a greater impact on experienced students.

3 Tool Description and Workflow

AnnoTestWeb/Run is a browser-based tool built using the Google Web Toolkit and CouchDB. Creation of tests involves using annotations that describe different elements of an acceptance test. A metadata system provides extra detail to annotations for example label, data types. It features a simplified interface and workflow, partially because it is aimed at all members of agile teams, including non-developers. Basic conveniences are also provided, such as automatic saving and synchronization of the document and safe copy and pasting of annotated text between windows of the tool and through email applications through the use of an html-safe micro-format using class attributes.

The annotations are based on elements of good acceptance tests [6]. Three types for elements are defined, each given a colour coding and a greyscale safe symbol:

- Act (From Actors / Actions) are where inputs and outputs are used and expected or events
- Data (From Examples) concerns input data to the system under test
- Results (From Observables Results) concernes expected outputs from the system under test
- N.B. A fourth element “pre-conditions” of [6] is implemented using the order of named tests in a document rather than explicity defined annotations.

Data and Result Annotations have additional meta-data to support model:

- Label (this manditory metadata groups annotations so a named test might contain multiple runs)
- Value (usually the annotated text is the value, optionally it can be overrided)
- Data Types (to provide type conversion when passing the value of the annotation between the tool and the system under test)

The tests created by annotating existing text can be edited in a tabular style equivalent to FIT Tables, from within the tool. The basic user interface of the tool contains a toolbar allowing quick application of annotations, navigation between documents (using URLs) and allows management of named tests created from the document with a tabbed area. The UI with a simple test describing the querying of a timetabling application is displayed in Figure 1 below:

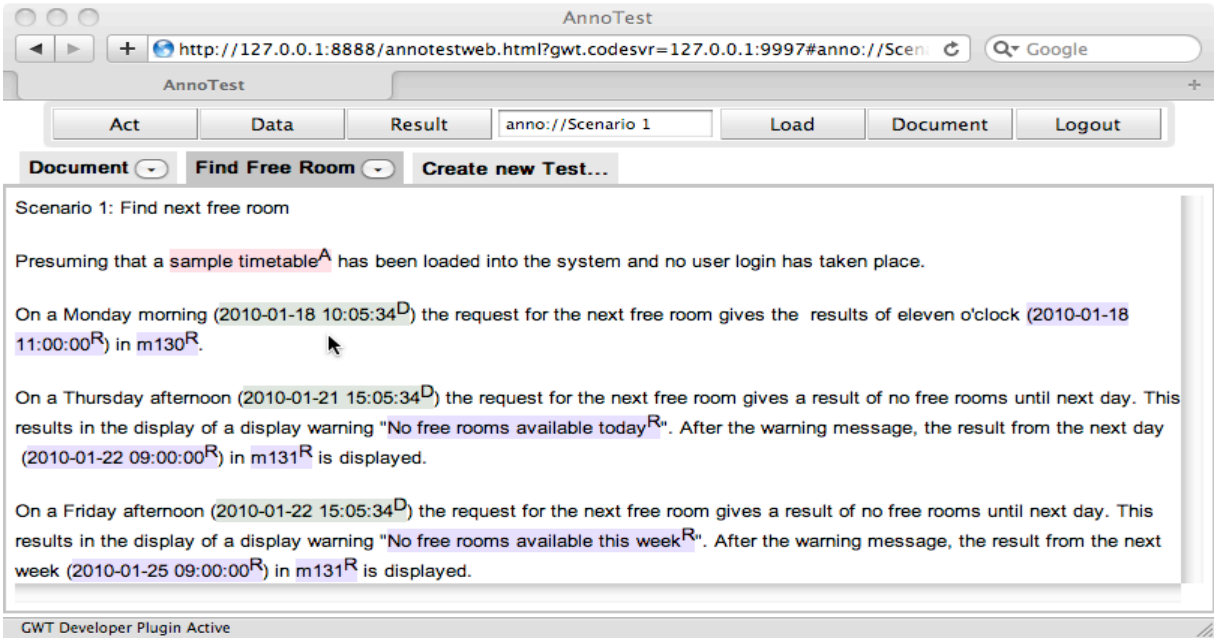

Figure 1: AnnoTestWeb/Run interface with a test ‘Find Free Room’ displayed

The tool is aimed at the person performing an agile customer role or a domain expert, with appropriate support from the software development team either with static review of tests or with direct pair participation. Their workflow for creating a test is simple. First, an appropriate name is chosen and appropriate text is identified then copied and pasted into the tool's rich text editor. Second, a named test is created. Third, highlighting text, selecting the type required and then providing appropriate metadata apply annotations. A fourth optional step is to customise the test with the Tabular editor for example where the test deviates from the natural order of the document. This tabular editor produces simple html tables appropriate for writing FIT fixtures, however a simplified interface for executing tests that integrates results with the annotated document is also included.

For example, sample text is loaded in a document named "Scenario 1" in Figure 1. A named test has been created, "Find Free Room", several annotations have been applied. Here the "Act" annotation is applied to "Sample Table". This is a reference to a class in the system under test, but the annotation type is also appropriate for references to standalone events or actors. The "Data" annotation is applied to inputs to the system; here a date "2010-01-18 10:05:34" has been annotated. The "Result" annotation is applied to the expected outputs of the system; here a free time "2010-01-18 11:00:00" and a free room "m130" have been annotated. As noted previously, metadata can be provided; here a label "freeRoom" has been applied to the later "Result" annotation, as visible in Figure 2 below:

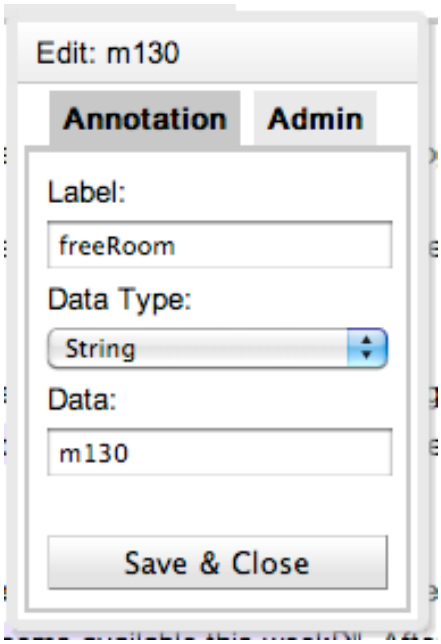

Figure 2: Metadata Editor displaying metadata of an annotation.

Delivering the data present in the annotated document and reporting of test results generated by program execution is designed to occur in a simplified fashion through a JavaScript Object Notation (JSON) API. This means that tests can execute in a wide variety of programming languages and environments. A library to speed up implementation of tests is also provided for Java and C#. This library has also been used to provide for input of manual steps that can occur test execution; this is in consideration of the combined software and hardware environment of our industrial partner in the next section.

The visualisation of the results of tests takes the form of these styles:

- Annotations (Simple pass or failure is visually displayed on Result annotations)
- Tabular (Actual value verses expected values are displayed in a table)
- UML (Actual value verses expected values are displayed in a clabject representation)

4 Industrial Evaulation Plan

The AnnoTestWeb/Run tool has been demonstrated to a domain expert and development team, to positive responses. The tool and the annotations in isolation have been tested experimentally with students, showing some benefits with annotations [18] and a favourable usability experience. Another prior empirical investigation showed annotations helped to reduce Errors in tests. Both these investigations considered similar metrics however the long running nature of the project will provide additional clarity.

The next step in the evaluation is to apply the tool in a long running industrial project. This evaluation will examine the use of the tool in a setting where existing documentation is available, including for example design documents and UML diagrams. At this planning stage the company will not be identified. The project background is of a new product containing both software and hardware components, developed using a plan-driven methodology. The project duration is of one year in total, with implementation at an advanced stage. The goal of the collaboration is to consider the tools impact on reusing existing documentation to generate acceptance tests, rather than in the context of acceptance driven development.

The following metrics will be gathered over the lifetime of the project:

- Errors:

  Gathered by recording mistakes in tests as they are corrected over the lifetime of project.

- Correct Elements / Missing Elements:

  Gathered by recording addition of annotation to tests over lifetime of project.

- Annotation Density:

  Gathered by comparing annotated text of tests to source documents.

The collection of these metrics will help to establish the impact of the tool and associated workflow on the testing process. Further as the acceptance tests created with the tool will not be the only aspect of testing on the project. This will allow detailed comparisons to be drawn between tests written by development team without the tool and tests written by a domain expert with assistance in the tool. This will include test quality metrics determined by the team and comparisons on time taken.

In addition structured interviews and questionnaires will be used to detail the experience of the team adopting the tool. A special focus in these supplementary efforts will be placed on human factors and usability of the tool.

5 Conclusions

This paper has presented a prototype tool and some of the background that led to its development. The work outlined in the previous section will be used to validate the tool in an industrial software development environment containing high quality existing documentation. Following this, improvements will be made to the tool based on the results of the case study.

The evaluation will be a prelude to a wider release of the prototype tool and further evaluation with teams using or adopting acceptance test driven development. Indeed the present industrial partner could be well placed to use the tool earlier in their processes. It is hoped that familiarity with the tool will put them in a position where adopting acceptance-test driven development is a logical conclusion of their document reuse.

6 Acknowledgments

This research is partially supported by Science Foundation Ireland through the Stokes Lectureship Programme, grant number 07/SK/I1299 and the Principal Investigator Programme (grant no. 08/IN.1/I2030)

7 Literature

1. Sommerville, I. Software Engineering, 8th edition, Addison-Wesley, pages 80-81, 2007.
2. Pressman, R. S. Software Engineering: A Practitioners Approach, European Adaption, 5th edition. McGraw-Hill, 2000.
3. Tassey, G. The economic impacts of inadequate infrastructure for software testing National Institute of Standards and Technology (NIST), May 2002.
4. Beck, K. and Andres C. Extreme Programming Explained: Embrace Change, 2nd edition, Addison Wesley, Boston, 2005.
5. Park, S.S. and Maurer, F. The benefits and challenges of executable acceptance testing, in APOS 08: Proceedings of the 2008 international workshop on Scrutinizing agile practices or shoot-out at the agile corral, ACM, New York, NY, USA, pages. 19-22, 2008
6. N. Jain, "Acceptance Test Driven Development". http://www.slideshare.net/nashjain/acceptance-test-driven-development-350264.
7. Kasatani, S. Selenium IDE. Website, URL last accessed 1st December 2008: http://seleniumhq.org/
8. RSpec Development Team. Website, URL last accessed 1st December 2008: http://rspec.info
9. Sauvé, J.P., Cirne, W., Osorinho and Coelho, R. EasyAccept Sourceforge Project. Website URL last accessed 3rd Dec 2008: http://easyaccept.sourceforge.net
10. Jain, N. Acceptance Test Driven Development. Presentation URL last accessed 30th Nov 2008. http://www.slideshare.net/nashjain/acceptance-testdriven-development-350264/
11. W. Cunningham, Framework for Integrated Test, September 2002. Website, URL last accessed 16th January 2009: http://fit.c2.com
12. FitNesse.org. Website, URL last accessed 7th February 2008: http://fitnesse.org
13. Deng, C., Wilson, P., and Maurer, F. "FitClipse: A FIT-based Eclipse plug-in for Executable Acceptance Test Driven Development". In XP 2007: Proceedings of the 8th International Conference on Agile Processes in Software Engineering and eXtreme Programming. 2007.
14. Mugridge, R. "Managing agile project requirements with storytest-driven development", IEEE Software, volume 25 (Jan.-Feb. 2008), pages 68-75, 2008.
15. Pyxis Technologies inc., GreenPepper Sofware, Website, URL last accessed 19th January 2009: http://www.greenpeppersoftware.com/confluence
16. Melnik, G. and Maurer, F. "The practice of specifying requirements using executable acceptance tests in computer science courses". In OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on OOPSLA, pages 365-370, 2005.
17. F. Ricca, M. D. Penta, M. Torchiano, P. Tonella, M. Ceccato, and C. A. Visaggio, "Are fit tables really talking?: a series of experiments to understand whether fit tables are useful during evolution tasks", in ICSE '08: Proceedings of the 30th international conference on Software engineering, 2008, pages 361-370.
18. Connolly, D., Keenan, F., and McCaffery, F. Developing acceptance tests from existing documentation using annotations: An experiment. In Automation of Software Test, 2009. AST '09. ICSE Workshop on (18-19 2009), pp. 123 –129.

Author CVs

David Connolly

David received his B.Sc. (Hons.) in Computing in Internet Technologies in 2007 from the Dundalk Institute of Technology. He is now undertaking research as part of a Ph.D. in the area of agile acceptance testing, at DkIT's Software Technology Research Centre (SToRC) under the guidance of Dr. Frank Keenan and Dr. Fergal Mc Caffery.

Frank Keenan

Frank lectures in Software Engineering in the Computing department at Dundalk Institute of Technology and is a member of SToRC. His current research interests include requirements engineering, agile development and context analysis. A particular interest is the transfer of knowledge gained to form innovative delivery approaches to subjects taught.

Fergal Mc Caffery

Dr Fergal Mc Caffery is a Lecturer with Dundalk Institute of Technology and a member of SToRC. He has been awarded Science Foundation Ireland funding through the Stokes Lectureship and Principal Investigator Programmes to research the area of software process improvement for the medical device domain.