

AutAT – An Eclipse Plugin for Automatic Acceptance Testing of Web Applications

Christian Schwarz
Bekk Consulting AS
PO Box 134 Sentrum
NO-0102 Oslo, Norway
+47 23 35 77 00

christian.schwarz@bekk.no

Stein Kåre Skytteren
Steria AS
PO Box 2
NO-0051 Oslo, Norway
+47 22 57 56 00

sks@steria.no

Trond Marius Øvstetun
Mesan AS
PO Box 1910 Vika
NO-0124 Oslo, Norway
+47 24 13 17 50

trond@ovstetun.no

ABSTRACT

In this paper we describe AutAT, an open source Eclipse plugin to better enable test driven development of web applications. AutAT lets non-technical people write acceptance tests (or functional tests) using a user-friendly graphical editor, and convert this visual representation of the tests into executable tests.

Categories and Subject Descriptors

D.2.5 [Software engineering]: Testing and Debugging – Testing tools.

D.2.1 [Software engineering]: Requirements/Specifications – Elicitation methods, tools.

D.2.9 [Software engineering]: Management – Life cycle, Software configuration management, Software quality assurance.

General Terms

Management, Documentation, Verification.

Keywords

Automatic acceptance test, web application, test driven development, change control.

1. INTRODUCTION

As web based systems get larger, more complex and embrace change, the need for continuous testing is increasing. Today, testing is often a manual activity creating large expenses if conducted too often.

XP (Extreme Programming) practice [1] states that acceptance testing should be automated so they can be run often and facilitate regression testing at low cost. The practice also states that it is the customer who should specify these acceptance tests and keep them updated as requirements change. Further, the tests should be used for test driven development.

In contrast, if this test-driven approach is not followed and automated testing scripts are created *after* a feature (or the XP

“user story”) is finished, the developer has lost a valuable source of information for implementing the feature. Also the test scripts can quickly get out of sync when the requirements change. The tests will then have to be updated when the problem surfaces, possibly an expensive activity, and often forgotten or down prioritized.

In addition, features may have been added which the automated test-tool may *not* be able to test efficiently. These features would in this case have to be tested manually, or be redesigned.

Both these problems would disappear if a change of requirements would be in form of (or accompanied by) a change in the system’s tests. The tests are then “a living requirements document” and ensure that all features are testable. The suite of tests should follow the system throughout its life cycle, even adding tests for reporting bugs.

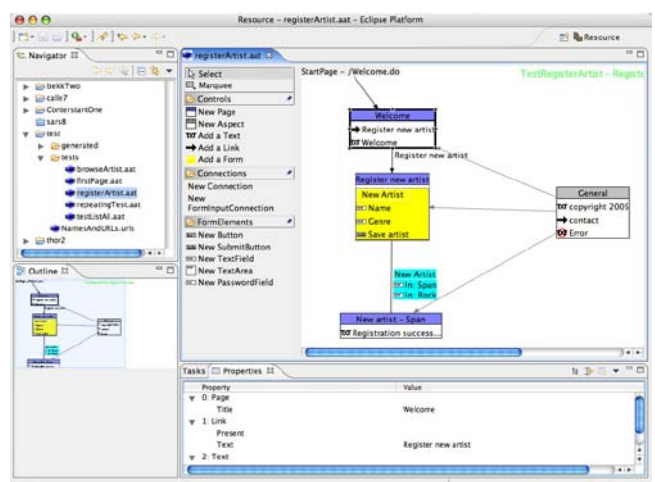
Change requests and bugs would thus get a comparable representation (in the form of tests that fail), and the customer can review test scores and decide which failed tests are of highest priority.

In order to enable this scenario, business-side people need a tool for specifying tests that is user-friendly and expressive. Existing frameworks and tools such as FIT and FitNesse support this process (see [2] and [3]), but are based on a textual format for specifying tests and require the customer to write assertions manually without rich editor support. This is what AutAT seeks to improve.

2. THE SOLUTION

While FIT and FitNesse are application agnostic and thus very general, we focus on web-applications. We have created an

Copyright is held by the author/owner(s).
OOPSLA’05, October 16–20, 2005, San Diego, California, USA.
ACM 1-59593-193-7/05/0010.



Eclipse plugin that makes it easier and more intuitive to specify acceptance tests for automated testing of web-applications.

AutAT is written using the Eclipse Graphical Editing Framework (GEF). It is an own perspective in Eclipse and has its own project type, with an initial wizard upon project creation.

The tests of a requirement (or user story) are written by specifying “workflow” of pages and input data. A page is a “box” in the GUI; three of them are seen in figure 1, left in the main view. For each page, it can be specified what content the page should have, including forms and links. Common content can be included in “aspects” (the right box in figure 1) that can be linked to multiple pages. A page can have multiple different successor pages, based on different form input. These different form input would be properties of the arrows that exit from the page.

The person writing the tests should specify enough different form input (and successor pages) to cover the main equivalence classes.

To verify that all specified input give an expected behavior, *executable* tests are generated based on the permutations of the workflow.

These tests can then be executed using an underlying test-engine, which is made interchangeable. Currently FIT + jWebUnit is used. This would normally be done as part of a continuous integration process or build script.

3. THE ARCHITECTURE

AutAT uses the standard Eclipse plugin development and GEF patterns. Figure 2 shows the main package structure of AutAT:

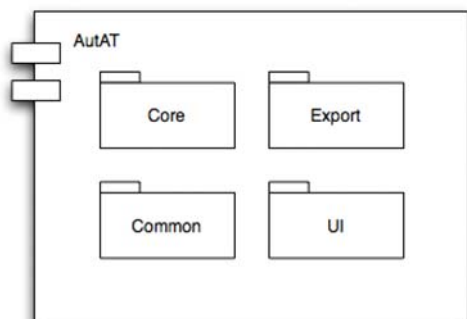


Figure 2: The main package structure of AutAT

The *Core* package is responsible for handling the tests in a project. These responsibilities include reading tests from files, converting them from XML to objects as defined in the AutAT common package, and performing the reverse operation to save them to files.

The *Exporters* package transforms a test from its object representation into some form that can be executed. The idea is to let several exporters co-exist, providing differing executable tests. The first implementation will be an exporter that will use the jWebUnit WebFixture extension to FIT, providing HTML files that FIT can process. JWebUnit is a java library simulating a web browser, providing ways to interact with pages and perform assertions.

The *Common* package contains the common classes used by the other packages. This includes the class model derived from the domain model.

The *UI* package manages the user interface and consists of four sub packages as shown in figure 3.

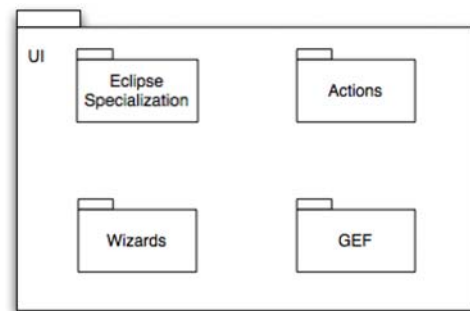


Figure 3: The structure of the UI package

The *Eclipse Specialization* sub package customizes the behavior of a project and the AutAT plugin. The *Actions* sub package contains actions that get triggered from the UI (e.g. exporting). The *Wizards* sub package creates new projects and new tests. The *GEF* sub package is the largest of them and contains the graphical editor. Its structure is closely linked to the Eclipse GEF patterns.

AutAT uses its own XML file format for storing the tests, and the AutAT graphical editor is an editor for these files. This XML representation enables use of Eclipse features such as “Local history” and version control software integration (Team API).

4. EVALUATION AND FUTURE WORK

AutAT is fully functional though there are some HTML and JavaScript limitations in jWebUnit.

AutAT can be downloaded at [4] and is the result of a master thesis [5] at the Norwegian University of Science and Technology (NTNU) in cooperation with Bekk Consulting AS, Norway. Developers around the world are welcome to contribute.

Further work includes exports to more test-engines, possibility to execute tests and see test statistics from within the AutAT perspective, and to extend the perspective to support more XP concepts.

5. ACKNOWLEDGMENTS

Our thanks go to Tor Stålhane of NTNU and Trond Arve Wasskog of Bekk Consulting AS for enabling the AutAT project to come to life.

6. REFERENCES

- [1] Extreme Programming: A gentle introduction, <http://www.extremeprogramming.org/>
- [2] FIT: Framework for Integrated Test, <http://fit.c2.com/>
- [3] FitNesse Acceptance Testing Framework, <http://fitnesse.org>
- [4] AutAT at BEKK Open Source Software, <http://boss.bekk.no/>
- [5] Skytteren, S.K. and Øvstetun, T.M., 2005. *AutAT: Automatic Acceptance Testing of Web Applications*. Unpublished MSc/MT Thesis, The Norwegian University of Science and Technology (NTNU).