Enterprise Tester – A Model Driven Testing Project

Bryce Day
Catch Limited
Auckland, New Zealand
Email: bryce.day@catchlimited.com

Christof Lutteroth
Department of Computer Science
University of Auckland
Auckland, New Zealand
Email: lutteroth@cs.auckland.ac.nz

***Abstract*—We detail the theory and development of Enterprise Tester, a web based test management tool that utilizes UML analysis specifications to automatically generate test cases at the system, integration and user acceptance testing levels. Enterprise Tester was designed to streamline the process between the analysis and test teams. During its development, a number of challenges had to be faced regarding support for other vendors' tools, numerous interpretations of the UML standard, and full round trip support for test cases that trace back into the analysis model. The paper discusses requirements, challenges and results of the Enterprise Tester tool, and outline some future work.**

***Keywords*-Testing, UML, models, test cases, tool integration**

I. Introduction

More often than not software development teams are considered an organizational type of their own. Current software development teams tend to work using various methodologies to deliver an end result. Rarely are they compared to standard manufacturing organizations, though on closer inspection they have many similar traits. Both types of organizations try to be as efficient as possible in the production of goods they produce, minimize rework, test for quality of output and try to standardize components across multiple products within the product range produced. A typical manufacturing organization is physical in nature, and it tends to be relatively simple to see inefficiencies in the production line. Software development organizations deal in intellectual property and have a production line made of people, processes and other software applications, making it much harder to see inefficiencies in the Software Development Life Cycle (the production line).

This paper discusses a project developing a tool for software testing, Enterprise Tester. The project was undertaken by the New Zealand ICT consulting company Catch Limited. Enterprise Tester is a web based test management solution that allows organizations to create, manage and execute test cases within a web based environment. It allows software development organizations to implement an end-to-end solution that streamlines the development process, at a fraction of the price of other comparable solutions in the market place.

One of the main functions of the Enterprise Tester is to extract use case specifications from the analysis phase to automatically create test cases for the testing phase. This avoids much of the duplicated work that had to be done when defining test cases for an application. The tool also integrates with an issue tracking system in order to manage the defects discovered while testing. Use cases from requirements specifications are related to test cases and tracked issues. As a result, the tool is able to provide traceability between issues and requirements, helping to prioritize defects and software development efforts.

Section II introduces the company, Catch Limited. Section III discusses the main requirements for the Enterprise Tester tool. Section IV discusses the UML Testing Profile, explains why it has not been used for Enterprise Tester, and how Enterprise Tester is different from other popular testing tools. Section V introduces the model underlying Enterprise Tester, and Section VI describes the tool itself. Section VII outlines its implementation, and Section VIII gives an evaluation of the tool's success so far. Section IX points out some future work, and Section X concludes the paper.

II. About Catch Limited

Catch Limited is a New Zealand based business and ICT consulting company whose focus lies toward delivering both outstanding service and excellent value to its clients. It has seen the business significantly expand its revenue and services in the years since it started out in July of 2004. Catch provides services with a business focus primarily on business analysis, project management, quality assurance, training, mentoring, and the development of products to improve organizational efficiency. Catch works with other ICT companies or companies with an ICT function to assist them in becoming more efficient, e.g. by implementing and improving tool support, or by training and mentoring them to use tools and methods such as Object Orientation (OO), Unified Modeling Language (UML), Business Process Modeling Notation (BPMN) [1], and The Open Group Architectural Framework (TOGAF) [2].

The organization has seen significant growth even over this short period of time. By delivering a high quality of service, adapting to clients needs and harnessing relationships with other leading ICT companies, Catch to date has achieved a doubling of turnover in each year of operation.

Some of the organizational milestones worth mentioning are the organizations placement in the Deloittes FAST 50 programme in 2008 and 2009 [3], which is a measure of the fastest growing companies in the region. Catch Limited placed 15th and 31st respectively in the country, as well as making it into the Asia Pacific FAST 500 rankings for the past three years. This growth is extremely unusual for a consulting organization, since product or recruitment organizations typically grow at this pace, and is testament to the planning, processes, and systems that have been implemented. Other notable achievements include the opening of a second branch in Wellington, besides the main branch in Auckland, the 2009 Jolt productivity award received for the rapid prototyping product Screen Architect, and its recent inclusion in the MIS Australia Strategic 100 and New Zealand CIO Strategic 100 as one of ten "Rising Stars" across the Asia Pacific region.

The organization has a five-year business plan, which aims to continue this rapid growth over the coming financial years. Some of the aims include the further growth of the Auckland and Wellington offices in terms of staff, as well as further expansion into the Sydney region, where they are currently working with a number of major financial organizations. To achieve further growth, consultancy operations had to be made more efficient.

Right from its inception, Catch has aimed to be a global organization and as such global standards and methodologies have been adopted throughout the organization. All software development is specified using Sparx Systems' Enterprise Architect CASE tool [4], and utilizes UML and BPMN in a central repository of projects. Over the years Catch has become very efficient in producing UML specifications for projects the organization has been involved in. These specification documents are read and referenced particularly during handover to subsequent teams within the development process. However, it has become increasingly apparent that very little reuse of the models occurs, specifically in the testing of developed software.

III. Requirements

With an increasing volume of testing work, Catch realized that the organization needed a test management tool. Roughly 15% of the time of a project was spent on testing. There is a strong trend towards automated testing, but Catch chose to invest in the creation of a test management tool without a focus on test automation for the following reason: 80% of the time spent for testing was spent on the specification of test cases, while only 20% was spent on performing the testing itself. Automated testing would only address the 20%, which is relatively unskilled and inexpensive anyway. Therefore the main objective of the new tool was to optimize the specification and management of test cases, and integrate this into an organizational workflow. Additionally, automated testing requires specialist skills and is more suited towards large projects that require substantial amounts of regression testing.

Previously there would be a duplication of work in the software development process. Enterprise Architect [4], a popular CASE tool by Sparx Systems, would be used for analysis, i.e. for specifying requirements and associated use cases. This would be done using UML diagrams and textual documents. At a later stage, test cases would be created from these specifications in a semi-manual process. Although the use cases provided most of the necessary data for testing, additional efforts would have to be spent, with much of the relationships between requirements, use cases and test cases becoming hard to trace.

The tool should be web based because this would enable easy access from everywhere, and circumvent the common problems of having to install a new application on each staff members client machine. This would fit well into the organization's infrastructure, as Catch is aiming to become a global organization with capabilities for distributed operation and development. Other tools that Catch is using, such as the JIRA issue tracker [5], are also web based.

After some investigation we were unable to find any test management tools that were web based and integrated with Enterprise Architect. Integration with Enterprise Architect was key for us as an organization since our standard for analysis was to use this tool. We then looked at the buy and modify options, i.e. extending an existing tool with the functionality we required, versus the option to build our own testing tool. What we found is that the space of test management tools was not very crowded. Few tools are available besides the well know HP Quality Center and Rational tools, both of which we considered too expensive compared to Sparx Systems and Atlassian pricing.

In order to relate issues back to use cases and requirements, it was important to integrate the test management tool with an issue tracking system. Such tracability would be useful for prioritizing issues and guiding software development and maintenance, since requirements are usually easier to prioritize than the issues themselves. What we quickly realized from our analysis was that incident management tools were quite common, and that we should not attempt to develop any type of incident management tool. After further investigation, we decided that Atlassian had a well regarded set of tools. Furthermore, we found that as an organization they aligned with our values and with the Sparx Systems values and philosophy, making them a great fit as a partner. These values are to provide tools of high quality at an affordable price, improving those tools continuously with short development cycles, and being very customer focused.

So the decision was made to develop our own web-based test management solution, which would integrate with Enterprise Architect as well as Atlassians JIRA incident management system. This would not only allow Catch to become more efficient, but also allows us to build a product

and grow a new global revenue stream.

IV. Related Work

A UML Testing Profile [6] was released July 2005. It looks to add a methodology for specifying software test cases to the UML standard. This profile, while useful in defining terms and structuring automated testing within the UML format, seems to add additional work to the tester's role. An organization would require the test analyst to be upskilled in UML in general and the UML Testing Profile in particular prior to them making use of it.

Rightly or wrongly we believed that testers working with the UML Testing Profile did not achieve the efficiency gains we were looking for. The profile seems to target automated testing, in particular unit testing, which is not the problem that Catch wanted to address. As mentioned earlier, Catch was trying to optimize the specification and management of test cases primarily at the system, integration and user acceptance test levels.

While Catch is using UML extensively, we were looking for a simple solution that would use the existing UML constructs, rather than having to integrate a completely new layer of complexity. Compared to the status quo, we believed that this profile adds additional work to the testing role, and would cost the project and organization more. For the specification of test cases that would be executed manually, the existing UML constructs as used in the requirements and use case specifications seemed sufficient.

Our aim was to streamline testing and analysis and to facilitate information sharing, as well as minimizing the documentation and set-up the tester was required to perform. The simplest way we saw of doing this was to achieve the following: first, allowing the automatic reuse of the analysis specifications such as use cases and scenarios; second, creating an application that could manage test cases systematically; third, integrating the tools used for requirements specification, test case management and issue tracking.

Regarding the management of test cases, it is important to understand the difference to a purely document based approach. Previously, test case specifications were created in the form of individual documents for each project. This meant that a test case occurring in different contexts or projects was copied and pasted, which caused inherent problems in the maintenance of such documents. If a test case was changed, all of its occurrences in all the documents had to be changed – a process that is cumbersome and error prone. Enterprise Tester was designed to support inclusion of test cases by reference, also across projects. For example, most web applications would use the same test cases for testing a login function with the process being the same the data may be different.

In the following, we will contrast unit testing, which is not addressed by Enterprise Tester, with system level testing, which is what the tool was created for. This will help to clarify the differences to various other testing tools.

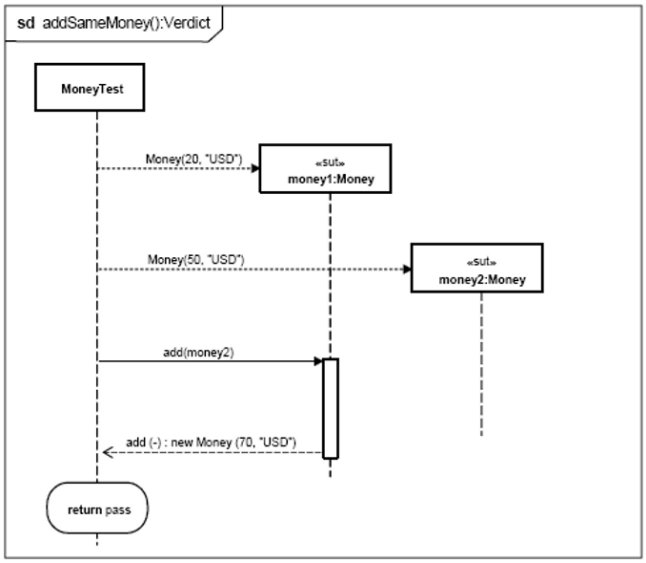

Figure 1. Sequence Diagram Example from the UML Testing Profile.

A. Unit Testing

Unit testing is designed to test the application at a code level. Each line of code that is written could have a corresponding test associated with it. Typically unit testing is performed by the developer during the development phase of the project, and allows the developer a level of certainty over the quality of the code they are producing. In recent times, the trend is to move into test driven development, where automated unit tests are developed around functions of the code while the developer is creating these functions. This way, they can run these tests to check any new or altered code and verify that the changes have no detrimental effect on the existing code base.

The more common tools used for unit testing are JUnit and NUnit, which allow Java and .NET applications to be tested. While not widespread, there are fairly well known translations between UML and JUnit / NUnit testing frameworks, which in the most part utilize Sequence Diagrams as models to generate unit test code. Such translations between models and code are used in Model Driven Development (MDD), where this translation can be performed automatically by tools.

The sequence diagram in Figure 1 shows a test case that checks if two monetary amounts, 20 USD and 50 USD, add up to 70 USD after the method `add` is used on them. The diagram describes a sequence of messages that are passed from one person or object to another, and allows a modeller to specify the operations that occur. As can be seen in the example, sequence diagrams are often very close to source code and contain a lot of implementation details.

Such details typically cannot be specified by an analyst, but may instead require a developer.

This form of Model Driven Testing (MDT) [7] works fairly well for people well versed in UML Sequence Diagrams, which typically developers are. But testers for the most part do not have an in depth understanding of UML. If they have worked with UML, they do not typically model in it on a daily basis, so putting a correctly formed Sequence Diagram together is not second nature. As a result, using UML Sequence Diagrams makes the familiar and often very simple task of testing an application much more complex for testers, and requires them to learn UML.

B. System Level Testing

System testing sees the testing of the system from end-to-end, i.e. not only testing individual functions that make up the different components but also testing the integration between components. This is the typical form of testing that a tester would perform. In most cases, they will treat the internals of the system being tested as an unknown (black box testing), and aim to perform tests that look for an expected result across a module or the entire system. This form of testing allows the application as a whole to be tested, verifying that the output is what the stakeholder expects. In most cases, for performing the actual testing relatively unskilled resources can be utilized.

This is the type of testing addressed by the Enterprise Tester tool. Compared to tools for unit testing, the market for this type of tool is only sparsely populated. Enterprise Tester focuses on tool and workflow integration, optimizing the specification and management of test cases. Automated test case execution is planned, but currently not yet supported. Note that test automation on the system level is harder than for unit testing. This is because system level testing usually needs to operate through the user interface rather than a lower level API. Naturally, getting a program to simulate a user's actions is harder than calling API functions.

V. Testing Model

The challenge with system testing is finding the skilled resource that can develop the test cases to then be executed. Since analysts must develop specifications that typically include use cases, this is the ideal place to "pull" the information from. This would then streamline the analysis-test phase integration without adding any additional overhead to the UML model, or cost in the form of higher skilled testing resources.

We believe the solution is to utilize the use cases and the information they contain that were specified by the analysts. A use case in simple terms is a function to be performed, which has a specified goal and defined steps in which to achieve a certain outcome. A use case may have several scenarios, which are paths through that specific use case, commonly referred to as user stories in agile development.

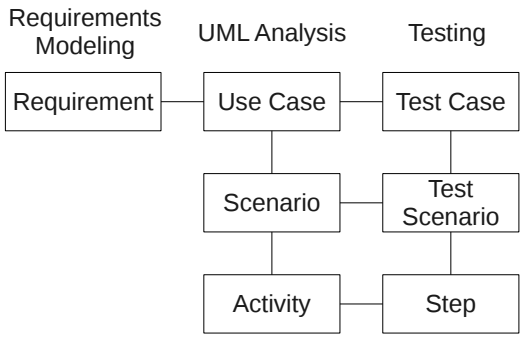

Figure 2. Relationships between requirements modeling, UML analysis and testing as utilized in Enterprise Tester.

For example, a Login use case would have at least two scenarios: one in which the login is successful, and one in which it fails.

Testers can reuse approved use cases and associated scenarios as test cases. To this end, Enterprise Tester was developed to allow the automatic development of these test cases from the UML analysis model that contains the use cases and scenarios. Figure 2 roughly outlines the relationships between requirements, UML analysis models, and the testing framework, which allow integration between analysis and testing in Enterprise Tester.

In UML analysis models, use cases consist of a number of scenarios, which in turn consist of several activities. Enterprise Tester associates corresponding test cases to the use cases, with test scenarios corresponding to the different use case scenarios. The activities in the UML analysis models correspond to steps that are performed during testing. Because use cases are mapped to requirements, test cases can eventually be traced back to the requirements that are tested.

VI. Using Enterprise Tester

During the project we found that just creating a web based test management tool was not enough. The application needed to be simple to use, so particular attention was given to the user interface. The key component of the user interface is a tree control, the Explorer Tree, that allows the user to perform all the common functions. For simplicity, no menu bar was added. The Explorer Tree can be used to navigate the items associated with a project, as well as manage different project contexts.

Historically, other test management tools have the user select a context (typically a project), and then allow the user to work within that context. Because one of our aims was to bring a OO approach to testing, we wanted to be able to reuse elements not only within a project but across projects. Because the Explorer Tree contains all projects, it allows users to move or copy elements across different projects. The

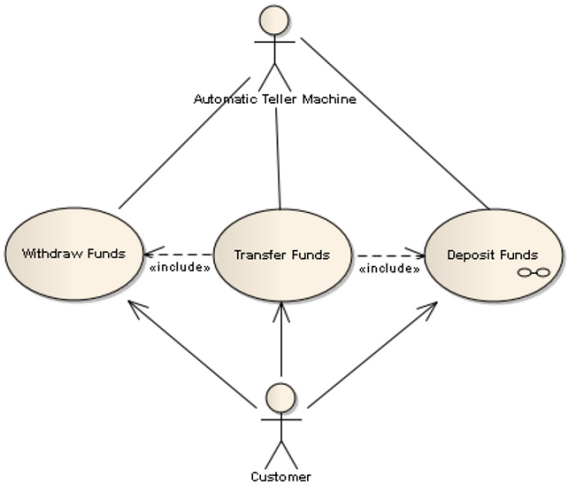

Figure 4. Example of a Use Case Model developed in Sparx Systems Enterprise Architect.

tree structure allows users to perform complex tasks with simple mouse operations such as dragging and dropping. For example, a test case from the library area of the tree can be dragged into the execution area of the tree to automatically create a test execution script.

Figure 3 shows a screenshot of Enterprise Tester, which contains the Explorer Tree on the left side and a User Dashboard with various testing related data on the right. The User Dashboard allows users to create a custom overview of the testing. It contains various user generated reports that are updated each time the dashboard is refreshed. As can be seen in the figure, Enterprise Tester has four key areas within the Explorer Tree:

1) **Requirements**: This is where requirements are stored; it allows users to associate the requirements with test cases.
2) **Templates**: This is the library where test cases are created and stored as templates for use in different execution sets.
3) **Execution Sets**: This is where a test case is executed and execution results recorded.
4) **Incidents**: This is where all incidents are recorded, allowing a user to see the status of the incidents.

Let us first look at the specifications Enterprise Tester imports from the Enterprise Architect tool. A basic Use Case model in Enterprise Architect is shown in Figure 4. The model outlines the use cases for an Automatic Teller Machine, and includes the use cases Withdraw Funds, Deposit Funds and Transfer Funds. Each of these use cases contain several scenarios. For example, the scenario Successful Cash Withdrawal, with a list of all scenarios of use case Withdraw Funds, is shown in Figure 5.

The bottom panel contains a list of all the scenarios,

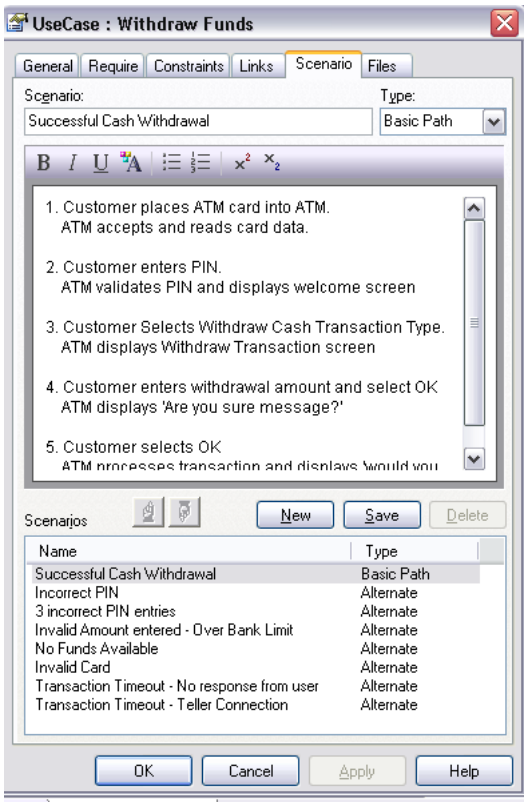

Figure 5. Successful Cash Withdrawal Scenario of the Withdraw Funds Use Case in Enterprise Architect.

while the top panel displays the description of the selected scenario. One of the challenges we have is to take the free form text contained in the description area and apply a structure to it. In the current v1.8 release of Enterprise Tester, we allow this information to be imported, but we do not attempt to manipulate it in any structured way. In version 1.9 of Enterprise Tester we integrate with the newly release version 8.0 of Enterprise Architect that includes an enhancement to allow the creation of structured use cases.

A simple testing overview is readily accessible by double clicking on any execution package. This provides a snapshot of the testing process to date, with time estimates and test case execution summary reports. An example of such a report is shown in Figure 6. The screenshot shows the overall estimated testing time, the time taken to date, and the remaining testing time in the top left panel. The top right hand panel shows the status of the test execution, i.e. the number of failed tests, passed tests, blocked tests etc. The bottom panel shows a list of the test case execution scripts that are contained under the selected execution package.

VII. Implementation

The project was undertaken initially by a core team of 5 over a 6 month period, and has continued at a reduced rate for another 6 months. The initial development was focused

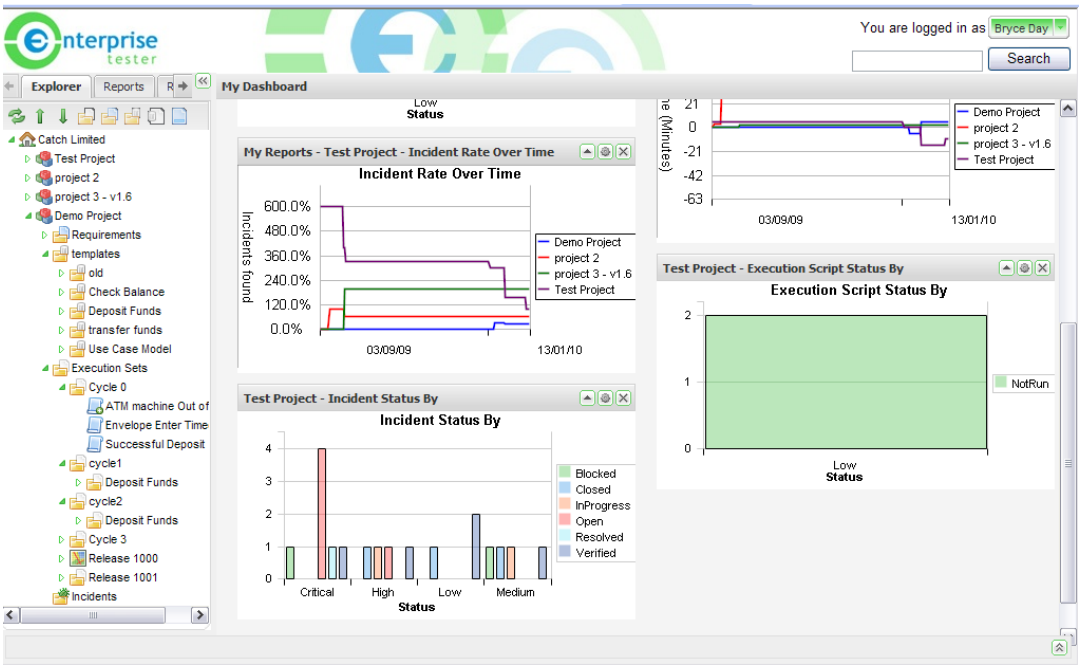

Figure 3. The Enterprise Tester Explorer Tree and User Dashboard.

on creating the correct architectural framework and a simple to use user interface, while implementing core test management features into the product. After this initial period of development, customer feedback was sought, building a community around the tool and allowing customers to submit suggestions. This helped to guide the project strategically, and to identify feature requests that were of high priority to our customer's test practices. We tempered customer enhancements with a product roadmap that identified core functionality that the team aimed to implement. This allowed us to continuely improve the product while not getting pushed too far off core product features by very specialist customer requests.

We implemented Enterprise Tester using AJAX with a .NET backend, and utilized nHibernate to interface to various database systems. Our initial database implementation was based on MS SQL, but subsequently we have added support for Oracle, PostgreSQL and MySQL. Additional database support has not been too difficult using nHibernate, since it abstracts away most of the database specific implementation details, allowing support to be added with minimal time and effort.

An overview of the architecture is given in Figure 7. It was designed with simplicity in mind, avoiding unnecessary complexities and reusing as many existing components as possible. Furthermore, the architecture supports tool extensions through plug-ins. Like this, many functions can be modularized as plug-ins, instead of having to integrate them into the application core. A number of plug-ins were implemented that are shipped with the core tool.

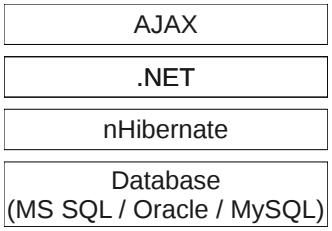

Figure 7. Architecture with the key technologies used in Enterprise Tester.

Enterprise Tester ships with a defect tracker, JIRA and Enterprise Architect plug-ins. In the new version v1.9 it will also include an Atlassian Crowd plug-in and an LDAP plug-in. The application was built to utilise plug-ins also for the core functionality. In the future, the community will be able to develop additional plug-ins of their own to extend Enterprise Tester.

VIII. Evaluation

Enterprise Tester is now used in seven countries by organizations of varying types and sizes. Enterprise Tester has Enterprise, Corporate, Professional and Personal licenses available for sale, which cover different implementation sizes. To date all license types have been sold. Typical users are test analysts, and we have installations in legal, health, scientific and software development verticals, to name a few.

All users utilize the tool to manage their testing process, while a number also take advantage of the integration with Sparx Systems Enterprise Architect. All of them have plans to move towards full end-to-end integration using

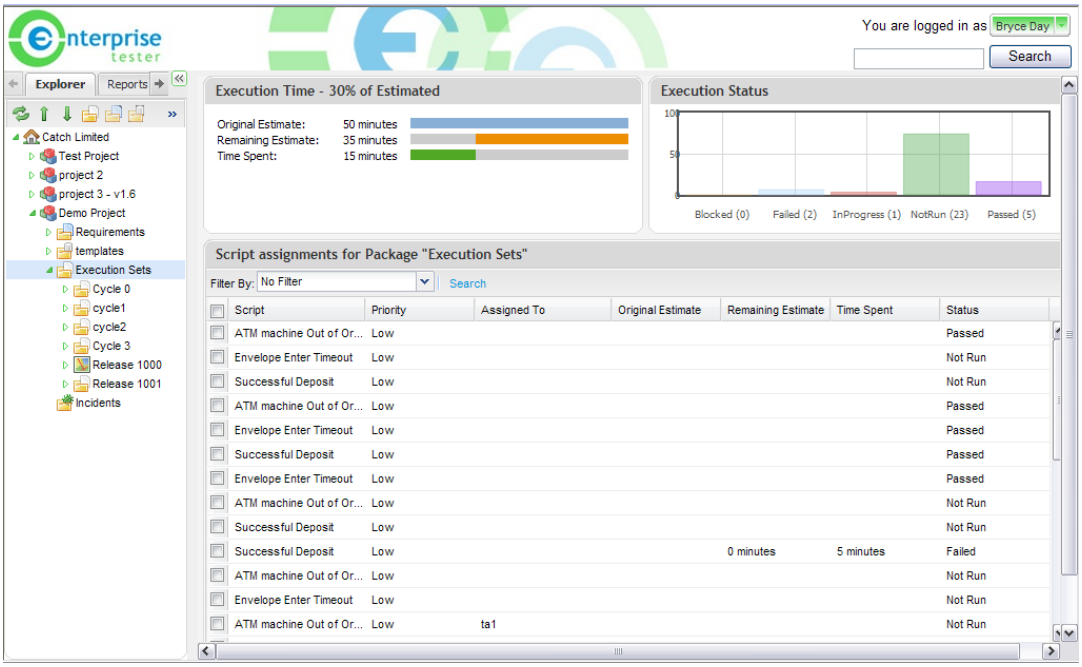

Figure 6. Enterprise Tester testing overview.

Enterprise Architect, Enterprise Tester and JIRA. Project team sizes and usage patterns vary with the customers using Enterprise Tester. One customer has over 20 projects running concurrently, while another has a single testing team running a single project. The tool scales well, and being web based, organizations that have offices around the globe are typically drawn to it.

Since the release in March 2009, we are constantly receiving feedback from customers, and aim to cultivate a community around the tool. We have recently released a forum to allow users to ask questions and discuss usage patterns. We have implemented requested fixes and enhancements typically within a couple of months from when we received a request. Only larger and more complex enhancements are scheduled as part of our product roadmap for future releases. In February we will be releasing version 1.9 of the tool, with each previous release including a mix of new features, enhancements and bug fixes. All of these previous releases have improved the tool, and we aim to continue this monthly incremental improvement in the foreseeable future. We currently have monthly releases planned through until mid 2011.

Enterprise Tester up to this point has not been customized for individual customers. We allow values for pick lists to be configured, but any feature or enhancement developed is included in the application as a whole. In addition to pick list customization, we have developed the application in such a way that it utilizes a plug-in framework. The application has core functionality, and additional plug-ins can be installed to extend the application features. Currently we have not released a Software Development Kit (SDK) for Enterprise Tester, but anyone is able to develop their own extensions to it, and the team at Catch are keen to assist. We believe the community will start to develop plug-ins in the future, initially to allow migration from current test management tools to Enterprise Tester, and then maybe hooks to automated testing tools.

Typically customers pick up the tool fairly quickly if they are used to using Enterprise Architect. If they are not, then in most cases a single one hour demonstration is enough to show them through the basics of the application, and get them started on how to use it. One of the team's aims was to make the application so simple to use that anyone would be able to use it without training.

We have had a number of encouraging customer comments around the tool, typically along the lines of "Enterprise Tester has more functionality than other tools we have seen", "the team are really excited about implementing Enterprise Tester, they cant wait to get their hands on it" or "we really like the user interface". Most of the improvements refer to enhancement requests around specific items that a customer would like to have implemented, which we add to the roadmap for future implmentation.

On the whole, the outcome of the development has been extremely positive, with the team seeing a decrease in test script creation time at a minimum of 25% and up to 75%, depending on the detail added to the analysis model by the analyst. This range of outcomes, while positive, is also one of the issues faced. Enterprise Tester integrates with Sparx Systems' Enterprise Architect, which is a leading UML tool.

However, like others in the field it implements UML, but does not guide the user as to what methodology to use. We expect that with the release of Enterprise Architect version 8.0, which includes structured use cases, this will improve to approximately 80%.

The approach Catch has taken allows users to implement the tool in many different ways to suit their custom SDLC process. But conversely, these different implementations do not allow a consistent outcome to be achieved, and hence consistent analysis results. A structured approach to analysis, especially use cases and their associated scenarios, would allow consistent high quality analysis to be performed, and consistent reduction in test script time to be achieved across projects and organizations. Catch has been working with Sparx Systems to implement the functionality required for this and it is due in version 8.0 of Enterprise Architect.

IX. Future Directions

Over the foreseeable future, the team plans to extend the tool to include advanced functionality, in order to make it one of the top Test Management Tools globally. These features include version control and automated testing. Full round-trip synchronization of use cases and test cases between Enterprise Architect and Enterprise Tester are also planned. There are a number of other enhancements, such as the SDK release and custom fields, that will be implemented over the next 12 months. We will be directed by customer feedback as to the priority of some of these enhancements and will look to customers for feedback on the implementation of features requested prior to their full public release.

Version Control would allow customers to take baselines of test cases, and version them as part of a package that includes code, test cases and their UML models. This will allow customers to roll forward and backward between different baselines. A number of customers have requested this enhancement, with Scotia Bank in Canada requesting it prior to April 2010 and the team aims to implement this in March 2010. The integration with tools for automated testing would allow automated test scripts to be executed from within Enterprise Tester, and the results passed back, to be collated with the manual results within the tool.

X. Conclusion

In summary, Enterprise Tester has been successfully developed and released on the global stage. Its ability to integrate with both Sparx Systems' Enterprise Architect and Atlassian's JIRA to provide an end-to-end solution is a world's first. The aligned pricing philosophy of all three organizations, i.e. Sparx, Atlassian and Catch, brings Enterprise Tester in reach of all small, medium and large organizations.

Enterprise Tester utilizes the requirements, use cases, and scenario information from UML models to auto generate test cases, saving the test analyst an estimated minimum of 25% of their time and equates to a minimum of 4% of the total project cost. It also enables tracing from issues and test cases back to requirements, thus helping to guide software development efforts. The aim is to continue to improve the time savings with the monthly releases of new versions of Enterprise Tester, and give smaller organisations the ability to implement an end-to-end solution that rivals the current market leaders at a much lower cost.

References

[1] Object Management Group, "Business Process Modeling Notation Specification," OMG Final Adopted Specification, February 2006.

[2] The Open Group, "TOGAF Version 9 - The Open Group Architecture Framework," Document G091, 2009.

[3] Deloitte, "Deloitte/unlimited fast 50 index," 2008, available from: http://www.deloitte.com/.

[4] Sparx Systems, "Enterprise Architect," http://www.sparxsystems.com/products/ea/.

[5] Atlassian, "JIRA Issue Tracking Tool," http://www.atlassian.com/software/jira/.

[6] Object Management Group, "UML Testing Profile Version 1.0," July 2005.

[7] P. Baker, Z. Dai, J. Grabowski, O. Haugen, I. Schieferdecker, and C. Williams, *Model-driven testing: Using the UML testing profile*. Springer, 2007.