Agile Regression Testing Using Record & Playback

Gerard Meszaros [1], Ralph Bohnet [2], Jennitta Andrea[3]

ClearStream Consulting
Suite 3710, 205 Fifth Ave SW
Calgary, AB
T2P 2V7 Canada
[1] gerard@clrstream.com
[2] ralph@clrstream.com
[3] jennitta@clrstream.com

Abstract. There are times when it is not practical to hand-script automated tests for an existing system before one starts to modify it. In these circumstances, the use of "record & playback" testing may be a viable alternative to hand-writing all the tests. This paper describes experiences using this approach and summarizes key learnings applicable to other projects.

1 Introduction

Scripting tests by hand as required by JUnit or it's siblings is hard work and requires special skills to write tests. Writing functional (or acceptance) tests using JUnit is particularly hard because of all the data requirements. One possible alternative is the FIT frameworks [1] but these still require someone to develop utility code to act as "glue" between the testing framework and the system under test (SUT). Each of these approaches requires that the system provide an interface by which the tests can be conducted.

1.1 Catch-22 of XUnit-Based Testing

On several recent agile projects, we found ourselves needing to modify an existing system that had no automated tests. In one case, the manual retest effort was expected to involve several person-years of effort and many months of elapsed time. Hand-scripting JUnit (or equivalent) tests was considered too difficult because the systems were not designed for testability. (e.g. business logic embedded in the UI; no access to an application API; no means of controlling all the test setup (such as "stubbing")). Refactoring the system for testability so that tests could be written was considered too risky without having automated regression testing to verify that the refactoring had not introduced problems. And even if we had done it, we were concerned that we could not hand-script all the necessary tests, complete with their expected outcomes, in the time and resource budget available to us.

1.2 Looking for Alternatives to XUnit

This led us to investigate all possible alternatives to XUnit style testing. Most of this effort focused on "record & playback" (R&PB) styles of test creation, an approach that involved recording functional tests on the system before we made the changes and regression testing the refactored system by playing back these tests. The tests verified the overall functionality of the system and, in particular, much of the business logic it contained. Once the tests were recorded and verified (successfully played back on the original system), we could then start refactoring the system to improve its design. We felt that this would allow us to quickly record a number of tests that we could play back at will. Since we had an existing version of the system to use as a "gold standard", we could leave the effort of defining the expected outcomes to the record and playback framework.

This paper describes the options we considered, their advantages and disadvantages and how we ended up regression testing the system. The work also lead to an understanding of where R&PB can be used in a more general context than the specific projects with which we were dealing.

2 Issues with R&PB Test Automation

R&PB style testing predates XUnit-style testing by many decades. Test automation folklore is rich with horror stories of failed attempts to automate testing. This paper describes critical success factors for making this style of testing work, what to avoid, and best practices in R&PB test automation.

The "robot user" approach to test automation had received enough bad publicity in past attempts at test automation that we found it to be a hard "sell". We had to convince our sponsors that "this time it would be different" because we understood the limitations of the approach and that we had a way to avoid the pitfalls.

2.1 The "Fragile Test" Problem

Test automation using commercial R&PB or "robot user" tools has a bad reputation amongst early users of these tools. Tests automated using this approach often fail for seemingly trivial reasons. It is important to understand the limitations of this approach to testing to avoid falling victim to the common pitfalls. These include *Behavior Sensitivity, Interface Sensitivity, Data Sensitivity* and *Context Sensitivity.*

Behavior Sensitivity
If the behavior of the system is changed (e.g. the requirements are changed and the system is modified to meet the new requirements), any tests that exercise the modified functionality will most likely fail when replayed. This is a basic reality of testing regardless of the test automation approach used.

Interface Sensitivity
Commercial R&PB ("robot user") test tools typically interact with the system via the user interface. Even minor changes to the interface can cause tests to fail even though a human user would say the test should still pass. This is partly what gave test automation tools a bad name in the past decade.

Data Sensitivity
All tests assume some starting point; these are often called the "pre-conditions" or "before picture" of the test. Most commonly, this is defined in terms of data that is already in the system. If the data changes, the tests may fail unless great effort has been expended to make the tests insensitive to the data being used. More recent versions of the test automation tools provide mechanisms that can be used to make tests less sensitive. This has added a lot of complexity to these tools and, as a result, they often fail to live up to their promises. This has likely contributed to the bad reputation they have received.

Context Sensitivity
The behavior of the system may be affected by the state of things outside the system. This could include the states of devices (e.g. printers, servers) other applications, or even the system clock. E.g. the time and/or date of test.

2.2 Agile Project Issues

There are other issues with R&PB testing that are specific to an agile project environment (especially eXtreme Programming.)

Not Test-First
Many agilists (especially advocates of eXtreme Programming) would argue that R&PB test automation completely undermines the notion of automating acceptance tests before the functionality is built because it requires the SUT to exist before the tests can be recorded.

3 Understanding Test Automation Choices

As part of our analysis of the choices available to us, we came up with a way of classifying the approaches to test automation. This helped us better understand why certain approaches worked better in some circumstances than others.

3.1 Approaches to Test Automation

Classifying Approaches to Test Automation
There is more than one way to automate tests. The approaches can be classified using a 3 dimensional grid. The three dimensions are:

- Granularity of SUT. The SUT can be a single unit (module, class or even method), a component, or the entire system.
- Test Creation Approach. The two main options are "Record & Playback" (R&PB) and hand-scripted tests. [1]
- Test Interface. The two main options are testing via the user interface or testing via an internal software interface or API.

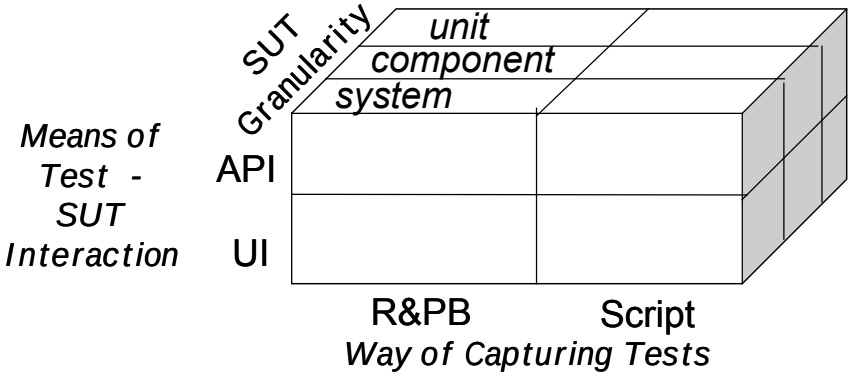

Fig. 1. The three dimensions of test automation

3.2 Common Combinations

While there are 2x2x3 possible combinations, it is possible to understand the primary differences between the approaches by looking at the front face of the cube. Some of the 2x2 combinations are applicable to all levels of granularity while others are primarily used for system testing.

Hand-Scripted Quadrants
The upper right quadrant of the front face of the cube is "modern XUnit". It involves hand-scripting tests that exercise the system at all 3 levels of granularity (system,

[1] There is a third approach: the generation of tests from semi-formal requirements specification. However, the authors do not feel qualified to comment on the relative merits of this approach.

component or unit) via internal interfaces. A good example of this is unit tests automated using JUnit.

A variation on "modern XUnit" is "Scripted UI Tests" with the most common examples being the use of HttpUnit, JfcUnit or similar tools to hand-script tests using the user interface. (It is also possible to hand-script tests using commercial "Robot User" tools.) These would fit into the bottom right quadrant. Where the entire system is being tested, this would be at the system test level of granularity. They could also be used to test just the user interface component of the system (or possibly even some UI units such as custom widgets) but this would require stubbing out the actual system behind the UI.

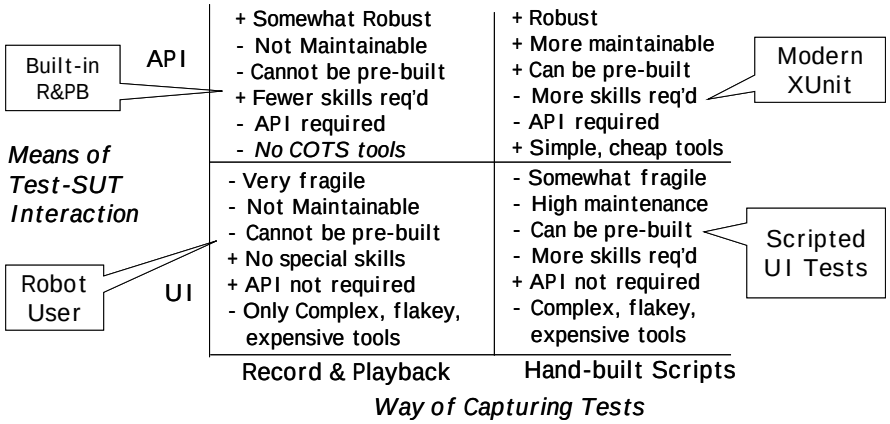

Fig. 2. The four test automation quadrants

Record & Playback Quadrants

The bottom left quadrant is "Robot User" This involves recording tests that interact with the system via the User Interface and is the approach employed by most commercial test automation tools. It applies primarily to System testing. This approach is primarily focused on testing the entire system, but like "scripted UI Tests", could be applied to the UI components or units if the rest of the system can be stubbed out.

The top left quadrant is not well populated with commercial tools but is a feasible option when building R&PB into the application itself. It involves creating a record & playback API somewhere behind the user interface. This is then used to record everything that affects the system state into a file that can later be used for input.

4 Implementing R&PB Test Automation

Record and Playback test automation can be implemented using either commercial tools or by building a record and playback capability into the application.

4.1 Using Commercial R&PB Tools

Commercial R&PB testing tools can be used in several ways. Most commonly, they are used to test an entire system including the business logic and the presentation logic.

Testing User Interface Behavior

Testing of user interfaces is one area in which commercial "robot user" tools can be used to good effect. The key is to make the system deterministic enough from a business logic perspective so that the UI tests can focus on verifying UI behavior without having to deal with variations in behavior caused by differences in the context (data, time/date, etc.)

This can be done in two ways:

1. Define a set of test data that can be frozen for the life of the tests. You may need to stub out any interfaces to other systems (or components such as the system clock) to ensure complete determinism.
2. Configure the user interface component to use a dummy version of the business logic component of the application . This "mock application" can be programmed (hard-coded or data driven) to return canned answers to requests. This ensures complete determinism of the "business logic" and allows the tests to focus on changes in UI behavior in response to the canned responses that are returned.

Testing Business Logic

This is probably the weakest usage of R&PB test automation even though many improvements have been made in the way commercial R&PB tools record tests. Either the "hand-scripted via API" or FIT approach would likely be a better long-term option.

If you do choose to use "robot users" to test the business logic, make sure you freeze the test data to eliminate a very common source of false test failures. Also, ensure that the application user interface is stable and that the functionality is not going to change between when you record the tests and when you plan to re-run them.

4.2 Building R&PB into an Application

Commercial "robot user" tools are not the only way to do R&PB style test automation. Depending on the architecture of the system under test, there are several ways to build R&PB right into the application. This is the way we chose to automate the functional tests on several recent projects.

The main advantage of this approach is that it can be applied to any system regardless of the technology of the user interface. It is also more robust than most commercial robot user tools because one source of potential failures – loss of synchronization between the test tool process and the SUT process – is eliminated by virtue of R&PB being built into the application. In effect, it moves the test automation

approach from the lower left quadrant ("robot user") to the upper left quadrant ("R&PB via API").

R&PB Decorator Between UI and Service Facade

Where the system consists of a cleanly separated UI component and a business logic component accessed via a service façade, the system can be configured to place a Recording Decorator between the two components. It records each request passed into the service component and the response that came back. A leading candidate file format for recording the interactions is XML.

Playback is accomplished by building a test driver that calls the service facade with the recorded requests and compares the actual responses with the previously recorded responses. It is quite simple to build a test driver that reads the XML containing the sets of `<request>` and `<expectedresponse>` elements. The user interface component is often omitted during test playback but in some cases it may be present so that test progress can be monitored.

A component container (such as an EJB application server) is well positioned to provide the capability to record the requests being passed to the managed component. Too bad that most do not yet provide this capability.

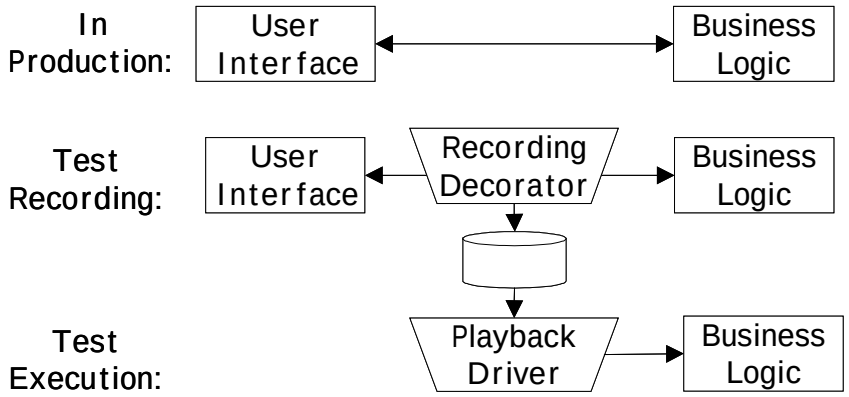

Fig. 3. Record and Playback using a Recording Decorator

Building R&PB into the UI of the application

If the application is not cleanly separated into a User Interface component and a service façade component, it may be possible to build the R&PB capability right into the User Interface. (See Figure 4) This is most cost-effective when it can done by building R&PB into a generic driver so that one doesn't need to sprinkle R&PB hooks throughout the system.

Example 1: A servlet-based application

We used a Transform View architecture in which a servlet calls business methods on a service façade and then invokes an XLST transform on the returned XML. We

placed recording hooks into the servlet to record the user's request (URL plus parameters), resulting XML and the resulting HTML after XLST transformation. A test menu was added to the UI to allow recording to be turned on and off.

Test execution (playback) was done by building a very simple JUnit test that submitted the recorded URLs using HttpUnit and compared the returned HTML with the recorded HTML. To avoid the messiness of manually locating differences between the two HTML strings, we wrote a *custom assertion* (a special version of AssertEquals) that would diagnose the problem and report the location where the two strings differed.

We were also able to easily build unit tests for our XSL transforms by passing the recorded XML and the XSL to the transforming and comparing the resulting HTML with the previously recorded HTML.

Example 2: A project re-engineering a "safety-involved" system
The system contains complex business rules that were not fully understood by the business. We needed to verify that the re-engineered system implemented the rules correctly. Unfortunately, the user interface was tightly coupled to the rules logic so it was not possible to use scripted tests via an API. We placed R&PB hooks into the generic screen I/O utilities whenever possible but we also needed to place hooks in a number of other places in the UI code. By recording on the old system and playing back on the new, we were able to quickly identify any differences in system behavior.

4.2 Test-First Development with R&PB

While at first glance, the title of this section may appear to be an oxymoron, it *is possible* to build playback tests before the system is built. As long as the form of the recorded interaction is human readable, it can also be human writable. (This is one of the key advantages of using XML for recording the interactions.)

Many of the commercial "robot user" tools do record the interactions in a human readable form. Some generate completely proprietary test scripting languages while others record tests in "standard" scripting languages such as VbScript or JavaScript. You can record a few tests using the tool of choice to see how certain types of interactions are done. Then you can start scripting tests based on what you have learned. One key advantage of doing this is that you can make sure the tests are less sensitive by using the "right" approach (e.g. using the *title* of a dialog box rather than the *ID*.)

The FIT approach to testing is an example of how a test might be "pre-recorded". The FIT framework could be modified to generate scripts that are runnable in your *robot user* tool, thus moving the effort of understanding how to interact with the system into test code generation framework.

When recording tests as XML, consider creating XSL style sheets that can transform the XML into HTML FIT tables. This would make the tests easier to read (no XML) and allow users to run the tests easily from a website.

4.3 Critical Success Factors

So, assuming you have decided to give *robot user* testing tools a second chance, what features do you need to look for in the testing tool? And what techniques do you need to apply to system development and test automation to be successful?

Designing the system for context independence.
You must be able to configure the system with a known starting point consisting of both data and the system date.

Tool Provides Means to Initialize System
Tests must be able start up the system with the known starting point.

Functionality Stability
R&PB testing can only be used to good effect when a significant portion of the applications functionality is expected to be unaffected by the next release. Any tests that encounter modified functionality must be rerecorded as the functionality is verified manually.

User Interface Insensitivity
It must be possible to record tests in a way that changes to the UI that do not affect the business logic do not cause tests to fail. (There may be other tests recorded that verify that the UI behavior has not changed and these will need to be sensitive to this kind of change.)

Separation of Tests for UI and Business Logic
All tests that verify business logic should be recorded in a UI insensitive way. A separate set of tests (either manual or automated) should be used to verify the UI has not changed. It can be useful to have different sets of tests with different sensitivity as these can be used to do "defect triangulation" (narrowing down where the defect is located.)

Limited Lifetime
Recognize that *robot user* tests will have a limited lifetime. They will not survive certain kinds of changes to the user interface or the business logic inside the system. Have a strategy for managing the tests that allows you to identify the those tests that will be impacted and which would need to be either discarded, rerecorded or superceded by newly scripted tests. One good way of doing this is to cross-reference the tests with the requirements by using a test management tool such as Test Director.

5 Applicability

Record and Playback testing should be considered when:

- You need to refactor a legacy system to make it amenable to XUnit-style hand-scripted tests and you feel it is to risky to do so without having regression tests.
- You cannot afford the time or cost of hand-scripting tests
- You do not have the programming skills required to hand-script the tests.

Record and Playback testing should be avoided when:

- You cannot fix the behavior of the system by freezing/snapshot the data on which the system will operate.
- The behavior of the system is expected to change significantly between when the tests can be recorded and when they will be played back.
- If you want to use the automated tests as a specification and there is no existing system that can be used for recording the tests.

6 Conclusion

Sometimes, R&PB testing is your only viable option given various project constraints. E.g. When dealing with legacy systems that do not have automated tests, Record & Playback style testing is a cost effective way to create regression tests that can be used to verify that design changes to the system do not introduce defects.

R&PB testing tools and techniques have matured significantly over the years and can now avoid many of the potential pitfalls when used properly.

When commercial R&PB test automation tools are unavailable, too costly, or too undependable, it is feasible to build the R&PB capability right into the system under test.

7 Acknowledgements

We would like to thank the many clients who gave us the opportunities to gain the experiences described in this paper.

References

1. Cunningham, Ward. FIT: *Functional Integrated Test.* http://fit.c2.com.
2. Fowler, Martin. *Patterns of Enterprise Application Architectures*.
3. Gamma, Eric et al *Design Patterns, Elements of Reusable Architecture*
4. JUnit testing framework: http://JUnit.org
5. HttpUnit and JfcUnit user interface testing frameworks: http://JUnit.org
6. Mercury Interactive's WinRunner functional testing tool and TestDirector test manager: http://www-svca.mercuryinteractive.com/products
7. Compuware's QaRun functional testing tool: http://www.compuware.com/products/qacenter/qarun/
8. Rational Software's Rational Robot functional testing tool: http://www.rational.com/products/robot