Test Automation for Legacy Systems

Rick Mugridge

Rimu Research Ltd

rick@rimuresearch.com

Test Automation for Legacy Systems

- An Agile perspective
- Why test automation?
- Legacy systems
- What makes testing difficult?
- Test automation
- SpecifyByExample

An Agile Perspective

- Release regularly, using short iterations
  - So agile is like maintenance phase
- Develop stories: slices of functionality
- Evolve design and code
- Drive with tests: TDD
  - So code is written to be testable
- Continuous integration

An Agile Perspective: For Testers

- Code is unit tested, with high coverage
- So testing can focus on integration and end-to-end
- As slices of functionality are completed, they can be tested in parallel with ongoing development
  - Avoids big batches

An Agile Perspective: For Testers

- Systems are architected from the beginning to support testing
- Clean modularity/independence, configuration
  - Fewer combinations as + instead of *
    - 30 + 40 + 50 + 60 (+ 5) = 185
    - 30 * 40 * 50 * 60 = 3,600,000

An Agile Perspective: For Testers

- Developers make changes to better enable testing. Egs:
  - ids added to HTML for Selenium locators: simpler and more robust
  - Can set date/time to test time
  - Mock a web service to respond with unusual errors

Why Use Test Automation?

- Systems evolve
- A change in one part can easily cause a subtle break elsewhere
- Test automation ensures:
  - New functionality works as expected
  - Existing functionality still works

Why Use Test Automation?

- The longer the delay in getting feedback:
  - the more expensive it is to fix
  - the more it delays the release
  - the more other change has been made to confuse the issue
- Automation speeds up the feedback
- And it allows more time for exploratory testing where it can be most effective

What about a Legacy System?

- A legacy system has no (or insufficient) automated tests
- It's hard to evolve, because
  - Feedback is too slow, because
    - Manual testing is too slow and not comprehensive enough
- So systems are simply added to
  - They turn into a big mess and everyone is afraid to make serious changes

What makes any Testing difficult? System Issues

- What is *really* supposed to happen?
- What data can I set up? Or do I have to depend on existing data?
- How do I know that the right thing has happened? Eg, email has been sent.
- How do I trigger particular behaviour? Eg, an odd error condition.
- How to cover all the combinations?

What makes any Testing difficult? System Time Issues

- Who else is using (and changing) the test systems at the same time?
- What about things that happen at a scheduled time? Eg, end-of-month
- What about things that happen with a delay of varying duration? Eg, ajax

What makes any Testing difficult? Focus Issues

- What has changed?
  - What do we need to test again?
  - What can we safely ignore?
- How can we use our time wisely?

Tackling a Legacy System: Incremental

- Add test automation incrementally where it can be most effective
  - For new functionality
  - For unstable, error-probe existing functionality
- Recording techniques can be used here
  - Eg, Selenium IDE, SoapUI
  - But...

Tackling a Legacy System: Catch 22

- Catch 22:
  - Test automation can be greatly enabled by changes to the code
    - Eg, modularity of subsystems
  - But changing the code needs test automation to avoid introducing bugs
- This also applies to developers as they add unit tests

Tackling a Legacy System: Catch 22

- Solution: Step by Step
  - Patience
  - Tests have to be end-to-end, so slow
  - Once have better coverage, can make larger changes to code
  - Over time, the tests and the code evolve so that both get better
    - So quicker to make changes reliably

Tackling a Legacy System:

- We'll look at two issues:
  - Test Evolution
  - Making Sense

(These issues often apply to non-legacy systems as well)

Tackling a Legacy System: Test Evolution

- Existing tests will need to be revised:
  - The UI has changed
    - Eg, a web page has been restructured
  - A business rule has changed
    - Eg, the rule has to take into account a new type of Customer
  - The database schema has changed
    - A Customer can now have several email addresses

Tackling a Legacy System: Test Evolution

- With long verbose, repetitive tests:
- A simple change in the web page requires that all tests that have details of that page have to be changed
- There may be hundreds of them
- Recording techniques make this worse
- There are various problems with this...

Tackling a Legacy System: Test Evolution: Problems

- Making all the changes is boring to do, and thus error prone
- It's not easy to find all tests to change
- It's easy to miss some cases and have incorrect, failing tests
- It's easy to end up with test suites that don't all pass
  - So have to manually check the failures

Tackling a Legacy System: Making Sense

- Often, the major problem is in knowing, clearly, what the system is supposed to do
  - What are the business rules?
    - How are discounts calculated?
  - What are the business constraints?
    - Do the rights of this user allow them to access this page?
  - What are the business objects?
  - What are the business processes?
- This is the perspective of the Business Analyst

Tackling a Legacy System: Making Sense

- When recording or writing automated tests, we can easily end up with:
  - Long, verbose, detailed, incomprehensible scripts
- That's because we include all the details:
  - What's the locator for that HTML element?
  - What's the XML structure look like?
  - What's the relational table structure, names, etc?
- That's because we've focussed on getting a test script together, rather than a business-level specification

Tackling a Legacy System: Solution: Abstraction

- The problems of test evolution and making sense can be solved by using abstraction
- Abstraction:
  - Removing irrelevant detail to focus on what's important
- We aim to have tests that:
  - are short, precise, avoiding unnecessary detail
  - clearly focus on one issue - eg, a rule, constraint or process
  - use the language of the business domain

Tackling a Legacy System: Abstraction

- With abstraction:
- It's easier to see if an important test case is missing
  - Eg, due to collecting test cases for a rule into a table or two
- It's easier to adapt to changes in the system
  - The details of the web page, the web services, the database, etc are encoded in one place
  - So there's only one change that's needed (usually)

Tackling a Legacy System: SpecifyByExample

- A tool to help with the abstraction process
- Open source
- Developing it as part of a research project involving:
  - University of Auckland, University of Waikato, Canterbury University and Rimu Research Ltd

Tackling a Legacy System: SpecifyByExample

- Works with FitLibrary tables
- FitLibrary's *defined actions* are an abstraction mechanism
  - They are procedures/functions with parameters
  - They are expressed in table form

SpecifyByExample

- *SpecifyByExample* processes the tests in a suite to see if:
  - Existing *defined actions* can be applied
  - New *defined actions* can be extracted, by looking for commonality in the tables across multiple tests
- The user is shown potential abstractions. They:
  - Choose which ones to apply automatically
  - Provide suitable names for any new *defined actions* and their parameters

SpecifyByExample

- *SpecifyByExample* works by looking for patterns of redundancy
- If two sequences of tables are similar, but have a small percentage of differences, that might be a useful abstraction
- But only a person who has some familiarity with the system can tell
  - One abstraction may make sense because it is related to a standard part of a process
  - Another abstraction may be a coincidence

Test Automation for Legacy Systems

- Small batches
- Fast feedback
- Meaningful, focussed test automation
- Importance of abstraction
- SpecifyByExample