Efficient Test Automation on an Agile Project

Presentation for UCAAT, October 2013
Lukasz Grabinski & Jackie McDougall
The Client & the Project

- **Business Background**
  - Our client provides financial support to students, providing loans and non-repayable grants for living, studying and tuition costs.
  - Smooth on-line loan application process is essential.
    - aligned with the Government’s ‘**Digital by Default**’ strategy
    - a positive experience for students
    - process of managing loans is extremely complex

- **Project Background**
  - existing web portal was confusing for customers, with each loan application on average resulting in 3.6 calls to the call centre for additional support.
  - cost of avoidable contact was £2.9 million per year
  - customer satisfaction was measured at 64% dissatisfied.
  - move towards modern service provision via the development of a new customer web portal.
  - aim is to drive traffic away from the call centre towards fully capturing applications on the web.
Implementing Automation?

Keep fixing it.

Define a lot of manual test procedures.

Purchase an expensive GUI test execution tool.

Hire an automation team to automate each procedure.

Build a comprehensive test library and framework.

Keep fixing it.
Implementing Automation

Processes

People

Tools

Sopra Group – Test Services
Application - overview

- Web portal to create, manage, submit and track application with captured customer data
- Multiple screens
- Many paths throughout the application process
- Various data capture – from simple Yes/No to complex recursive data objects
- Integration with multiple legacy systems through web services
- High focus on the usability and user experience aspects
Tools, tools, tools

- Gherkin
- Cucumber
- Pickles
- Java
- Selenium
- SVN
- BDD
- Agile
- Jenkins CI

Diagram shows a network of tools and technologies including Eclipse, Gherkin, Cucumber, Pickles, Java, Selenium, SVN, BDD, Agile, and Jenkins CI.
Evolution: DSL - your friend or enemy?

Before: No upfront DSL design led to over 600 step definitions, causing:
- Minimal reuse of the existing steps/code
- Lack of clear understanding what step does and how
- No practical use of the tests as documentation of system to business
- High cost of step implementation
- Difficult maintenance and increasing technical debt in the test code

After: Core of ~30 designed, parameterised steps used in 95% of the tests
- Easy test creation – using steps as templates with parameters published in the project wiki
- Clear understanding what to expect from the step
- Tests useful for the analysts, testers, developers and business
- High reusability
- Test automation effort reduced several times over
- Allow to use defined (business journeys) or explicit data (component/system tests)
- Limited number of additional, component test focused steps
DSL – Examples:

- **Before:**
  - “I click Next button”
  - “Button Yes has been clicked”
  - “I have clicked Save button”
  - “I use the previous page link”

- **After:**
  - “I click the (.*) ”
  - All available buttons and links published on wiki
  - New elements easy to add to the mapping table (abstraction layer)
Evolution: Data – drives tests or you crazy?

- **Before:** No test data design or approach, causing:
  - Complex and difficult to understand scenarios
  - High duplication of steps in test scenarios
  - Difficult test data management
  - Reduced coverage of tests

- **After:** Test data designed and stored as “persona” concept
  - Persona’s data leads to user story or specific test path with desired data
  - Short and concise scenario – 2 steps to get to any point in the application process
  - Easy data management
  - Higher coverage at lower cost
  - Faster test execution – ability to create application with required data through web services allow direct jump to page directly rather than using Selenium
Data – Examples:

**Before:**
- “I login as user JOHN SMITH”
- “I answer X for the first question”
- “I enter A data”
- “I answer Y for the second question”
- “I enter B data”
- “I answer Z for the second question”
- “I enter C data”
- “I click Next button”
- “I am navigated to the next page”
- “My first question data is A”
- “My second question data is B”
- “My third question data is C”

**After:**
- “I am logged in persona JOHN SMITH on page X”
- “I have completed page Y until and including question Z”
- “My first page data is persisted”
Evolution: “Id”entify your page elements

- **Before:** No abstraction from maze HTML ids, causing:
  - Difficult test creation
  - Confusing test scenarios and thus system documentation
  - More complex and less readable tests

- **After:** Mapping abstraction layer – from HTML id (part id) to a name
  - Meaningful name of the component – be it a button, field or an error message
  - Clear to understand tests and thus system documentation
  - Easy to manage and update
  - Single place – no confusion where to look for
Evolution: Structure your tests

**Before:** No clear structure and purpose for the tests, causing:

- Difficult test management
- Duplication of scenarios across tests
- Missed crucial scenarios
- Tests as documentation difficult to use by business

**After:** Split into “Journey”, “Page” and “Component” tests.

- “Journey” tests are user story related scenarios - UAT if you like - taking persona for a journey through the full or part of the application process
- “Page” tests are classed as system tests, providing more detailed coverage for the specific page, business logic or data handling
- “Component” tests are focused on specific components of the application – such as numeric data capture field or address capture, providing most detailed coverage
- Clear view what tests are required and what level of coverage are to be achieved
- Easier test scenarios / execution management and partitioning
Making the Process Work

0 - At start of project build the skeleton automation framework
   1 - Depending on the project - either BA prepares the gherkins as the base stories or tester prepares the drafts based on stories; but good agile practice is to collaborate & talk to each other often (not as a separate task)
   2 - 99% of time it’s more practical to build code first, automate tests later - with an overlap; automate tests sometimes could start when build of code starts, sometimes later

3 - Cucumber/Java is what we applied. You could use alternative tools like Twist, Cucumber & Ruby, Capybara, C# etc.

4 - After 3 amigos, you can tag the tests with appropriate annotation, and have them executed on the CI in a separate job (for example “Work In progress”), so from a progress perspective it is clear how much work is still to be completed in-sprint.
Thank You