Talented Together

HOW TO IMPLEMENT EFFICIENT TEST AUTOMATION IN AN AGILE PROJECT

Agile Business Conference, October 2014
Lukasz Grabinski & John O'Hare

sopra

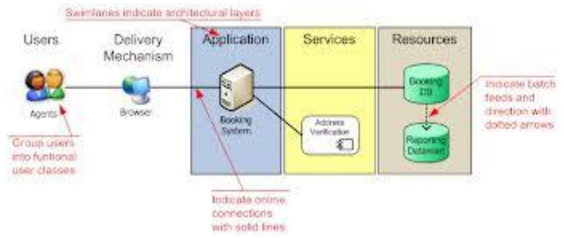
CONTENTS

THE CLIENT & THE PROJECT

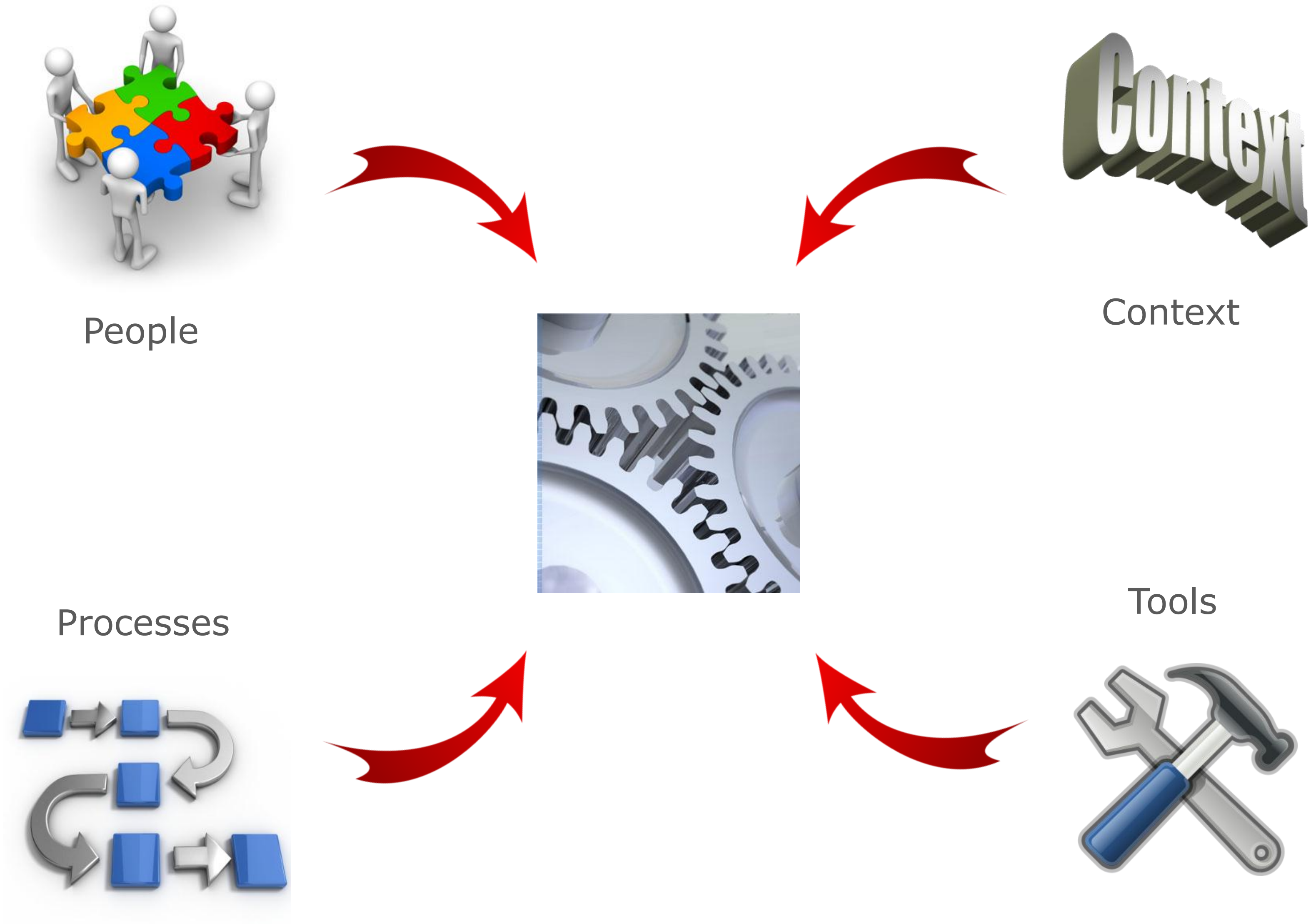
- ## Business Background
  - Our client provides financial support to students, providing loans and non-repayable grants for living, studying and tuition costs.
  - Smooth on-line loan application process is essential:
    - Aligned with the Government's 'Digital by Default' strategy.
    - Positive 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.

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


Swimlanes indicate architectural layers
Users
Delivery Mechanism
Application
Services
Resources
Agents
Browser
Booking System
Address Verification
Booking DB
Reporting Datamart
Group users into funtional user classes
Indicate online connections with solid lines
Indicate batch feeds and direction with dotted arrows

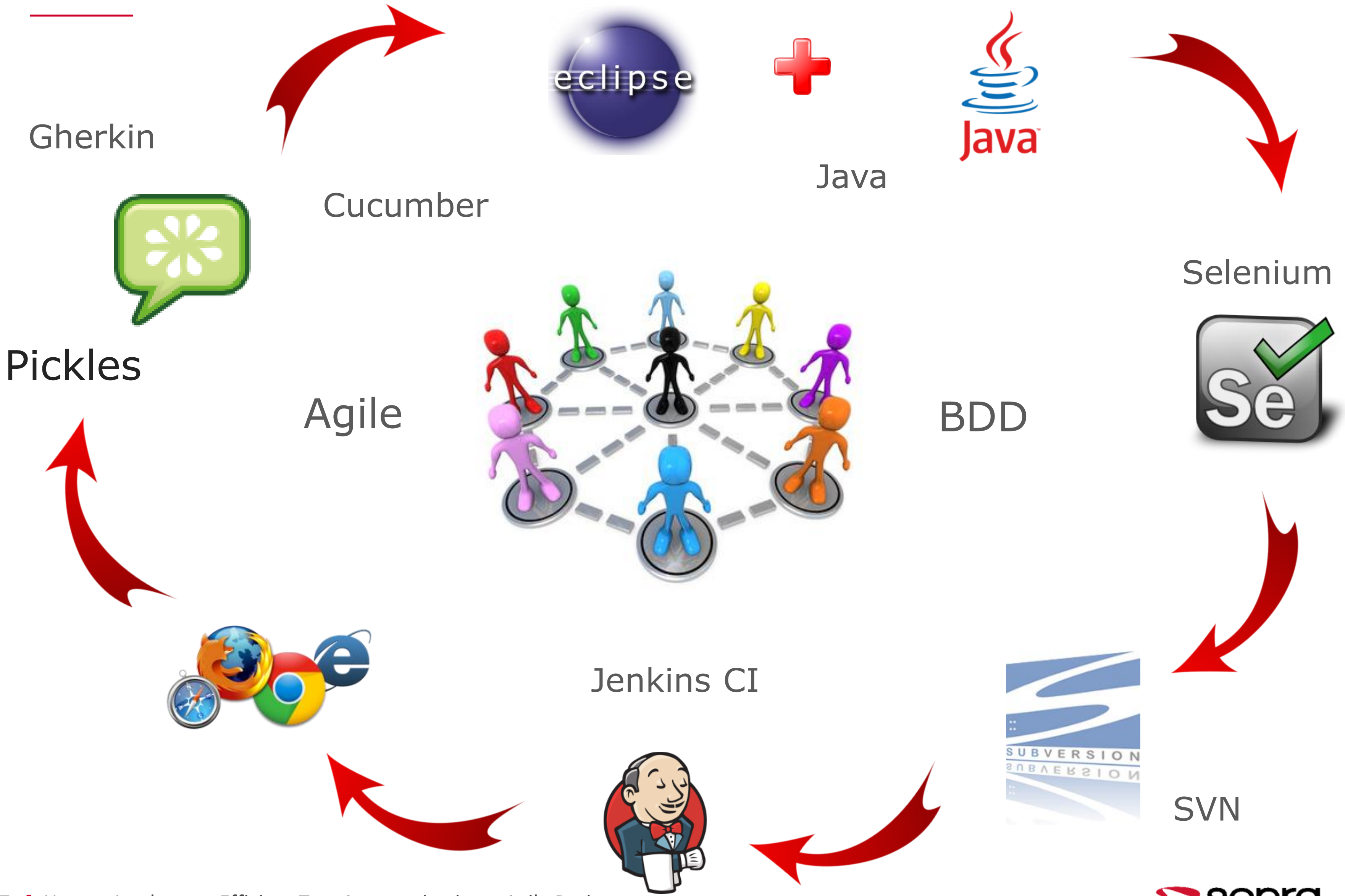
IMPLEMENTING SUCCESSFUL AUTOMATION



PLANNING AND ARCHITECTURE

- Test Automation is software too:
  - **Set clear objectives:**
    - How much do you want to automate? API ? Front end? Full end to end?
    - How is it going to compliment other testing areas like unit tests, manual exploratory testing?
    - What about the level of component/system/integration automation
  - **Consider project aspects:**
    - Profile of your team - especially developers and testers,
    - Projects aspects : Is it front end heavy? Complex business rules?
    - Timescales, environments, etc...
  - **Design:**
    - Framework does not mean complex and high up-front cost, it means fit for purpose yet flexible design,
    - Think about users - test automation should focus on the most repetitive tasks and give testers more time to design tests/exploratory testing,
    - How are you going to manage test data?

TOOLS, TOOLS, TOOLS



sopra

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 click Next button"
  - "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

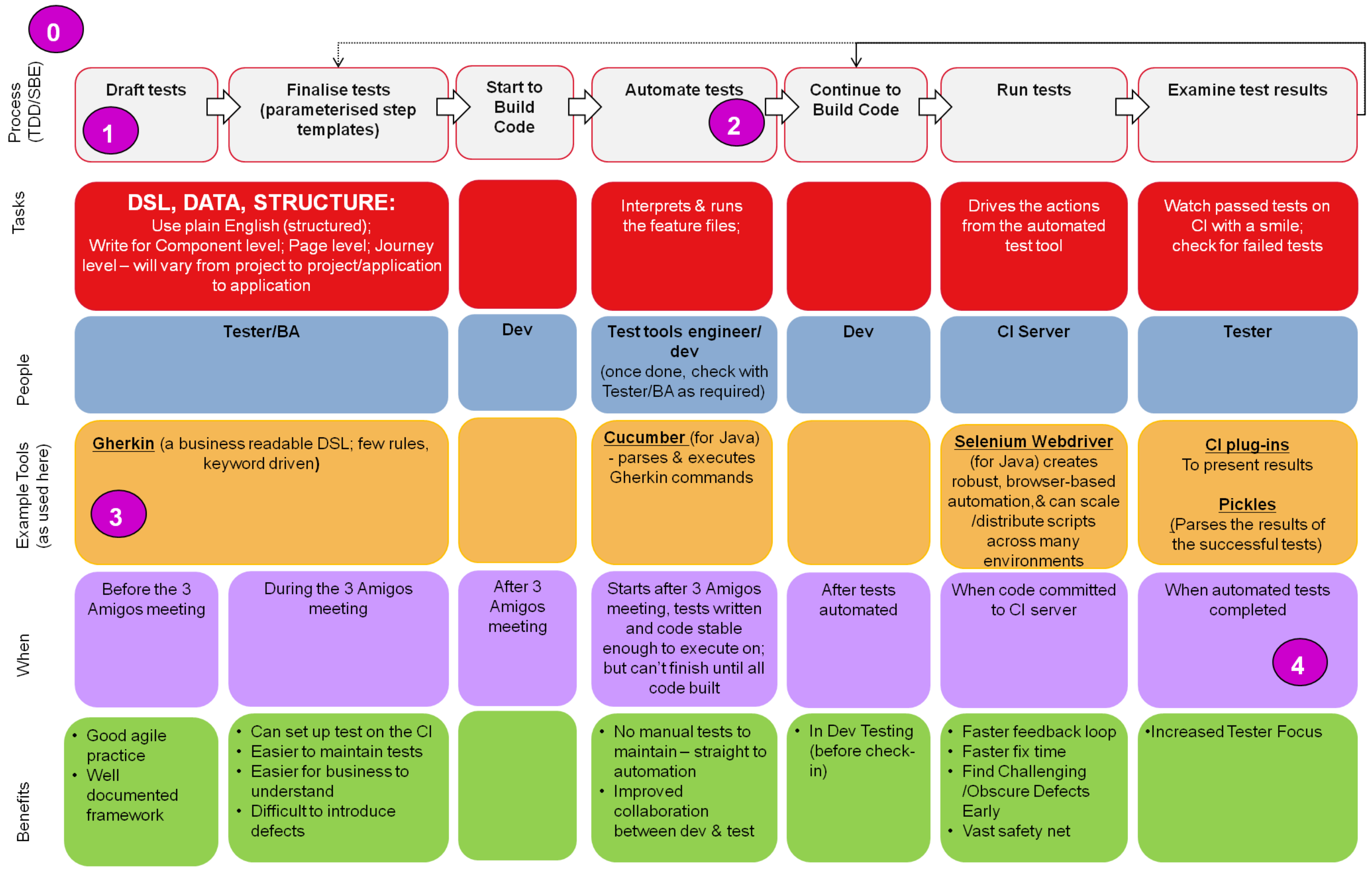
- **Before: No abstraction from maze HTML ids, causing:**
  - Difficult test creation
  - Confusing test scenarios and thus system documentation
  - More complex and less readable tests
- **After: Mapping abstraction layer – from HTML id (part id) to a name**
  - Meaningful name of the component – be 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

sopra

MAKING THE PROCESS WORK



0

Process (TDD/SBE)	Draft tests 1	Finalise tests (parameterised step templates)	Start to Build Code	Automate tests 2	Continue to Build Code	Run tests	Examine test results
Tasks	**DSL, DATA, STRUCTURE:** Use plain English (structured); Write for Component level; Page level; Journey level – will vary from project to project/application to application			Interprets & runs the feature files;		Drives the actions from the automated test tool	Watch passed tests on CI with a smile; check for failed tests
People	**Tester/BA**		**Dev**	**Test tools engineer/ dev** (once done, check with Tester/BA as required)	**Dev**	**CI Server**	**Tester**
Example Tools (as used here)	**Gherkin** (a business readable DSL; few rules, keyword driven) 3			**Cucumber** (for Java) - parses & executes Gherkin commands		**Selenium Webdriver** (for Java) creates robust, browser-based automation,& can scale /distribute scripts across many environments	**CI plug-ins** To present results **Pickles** (Parses the results of the successful tests)
When	Before the 3 Amigos meeting	During the 3 Amigos meeting	After 3 Amigos meeting	Starts after 3 Amigos meeting, tests written and code stable enough to execute on; but can't finish until all code built	After tests automated	When code committed to CI server	When automated tests completed 4
Benefits	• Good agile practice • Well documented framework	• Can set up test on the CI • Easier to maintain tests • Easier for business to understand • Difficult to introduce defects		• No manual tests to maintain – straight to automation • Improved collaboration between dev & test	• In Dev Testing (before check-in)	• Faster feedback loop • Faster fix time • Find Challenging /Obscure Defects Early • Vast safety net	•Increased Tester Focus

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.

QUESTIONS / ANSWERS

CONTACTS

Team name

Lukasz Grabinski	+44 (0) 131 332 3311
John O’Hare	+44 (0) 131 332 3311

Sopra
Orchard Brae House
30 Queensferry Road
Edinburgh
EH4 2HS

www.sopra.com