Live Specifications:
From Requirements to Automated Tests and Back

Paul Gerrard
paul@gerrardconsulting.com

gerrardconsulting.com
We believe there are better ways to build software
Agenda

• From Waterfall to Continuous Delivery
• Definition and assurance
• Traceability achieved in-flight
• The end of manual feature checking
• Traceability and impact analysis
• Summary
From Waterfall to Agile to Continuous Delivery

Testing isn’t dead, it’s redistributed
Towards Live Specifications

• I want to take you on a journey; a thought process
  – From Waterfall to Continuous Delivery
  – From dead documentation to LiveSpecs using the ideas of Behaviour-Driven Development

• Won’t discuss BDD tools

• Will describe some the beneficial changes a traditional development organisation can make

• We’ll use several Agile ideas, but in a pragmatic way to achieve our goals

• Testing is changing – for the better.
A methodological shift...

- A methodological shift from staged to iterative and now ‘continuous’
- Just as we get used to Agile, we now need to consider continuous ‘Specification by Example’ (SbE) approaches (also see TDD, ATDD, BDD)
- SbE promotes a continual process of specification, exampleing, test-first, and continuous integration
- Did you know, in Malaysia, they are skipping Agile?
Businesses are pragmatic – but always want MORE

• Companies aren’t interested in BDD; they’re interested in delivery
• Continuous delivery sounds even better
• BDD might help them achieve it, but it has to be more than a ‘plug-in’ to waterfall
• How can non-agile companies ever hope to achieve continuous delivery?
Put it another way

• If a company has a trusted set of business rules defined in requirements**, how can delivery of a system to meet those requirements be achieved in a continuous way?

• How can larger requirements, evolved over weeks or months be channelled into teams doing continuous delivery?

** sources of knowledge that defines the system
Behaviour-Driven Development and Continuous Delivery
BDD isn’t about testing it’s about delivery and achievement

• BDD is growing popular and the transition from scenarios to test code is eyecatching
• BDD extends TDD upstream to stakeholders
• In principle, stakeholders define the scenarios
• Continuous testing tells you where you are

“Faster software delivery: from months to minutes”

dannorth.net
Behaviour-Driven Development

- Requirement
- Story (structured)
- BDD Tool
- Generated Test Code
- Test Fixtures
- Test Tool

+ Candidate Regression tests

Test Status
- Red
- Amber
- Green

Write the Code
Run the Test
Test Tool
Generated Test Code
BDD Tool
Story (structured)
Requirement

Intelligent Definition and Assurance
Slide 14
From traditional delivery…

… to Continuous Delivery
The Deployment Pipeline

- Automated Unit tests
- Automated Acceptance Tests
- Manual User Tests?
Deployment into Production is not the Same as ‘Going Live’

Release to production and test environments on-demand
Go Live weekly/monthly/quarterly…
It’s not about automation, it’s about trust

A Deployment Pipeline depends on a Trusted Requirements Pipeline
If a requirement drives today’s delivery it must be trusted

- Continuous delivery is a hungry beast that eats requirements
- Requirements must be trusted to be mature, complete and coherent enough to deliver the business value envisaged by stakeholders

TRUSTED, NOT PERFECT
Definition and Assurance

Bringing Analysis and Testing Together with BDD
Why do we separate analysis and assurance?

- Agile has shown (among other things) the tremendous value of fast feedback
- But why wait 4 weeks, 2 weeks?
- Why do we use developers to build software to prove that the requirements are poor?
- Let's validate those requirements and trust them.
Redistributed testing

Functional Requirements

Build (+little testing) but lots of rework

Testing (lots of checking + bugs)

Trusted Requirements + Stories

BDD/Test-Driven

Test

SAVED TIME
Stories and scenarios validate requirements

- Users often start the discussion with a story
- Analysts derive requirements by seeking patterns in stories
- Features and scenarios provide illustrations and provide feedback
- Stories validate requirements
DeFOSPAM

How to validate a requirement with business stories

(googel it)
Which and how many scenarios are enough?

- To understand feature scope?
- To get stakeholder to accept?
- To validate the requirement?
- To estimate work to build the feature?
- To unit test this feature?
- To system test this feature?

Scenarios are created to meet several goals
Traceability achieved in-flight

Traceability is a by-product of this way of working, not a difficult afterthought
(with the aid of some screen shots from businessstorymanager.com)
Goal and risk network

Diagram Key
- Goal Status: Planned, Amber, Red, Green, Completed, Abandoned
- Risk Status: Open, Closed, Improving, Worsening

Slide 30

Intelligent Definition and Assurance

Lucky Numbers Lottery Game (Bottom-Up)
A requirement

The calculator will accept the inputs: a number A, an operator O and another number, C.

The calculator will validate the numbers A and B as numbers in the range -1000,000,000 to + 1000,000,000.

The operator maybe one of the following: "+" (plus), ".-" (minus), "*" (multiply) or "/" (divide).

The calculator will perform the calculation according to standard arithmetical rules as follows:

- A + B will add the values of A and B.
- A - B will subtract the value of B from A.
- A * B will multiply A by B.
- A / B will divide A by B.

5. The calculator will print the result as a real number with up to 20 significant digits.
6. Errorous in put will be handled as follows:
   - Invalid values for A, the message "A is not a number" or "A is out of range" as appropriate.
   - Invalid values for B, the message "B is not a number" or "B is out of range" as appropriate.
   - Invalid values for O, the message "Invalid operator" will be displayed.

7. Blank or null values for A, B or C will terminate the program.
4. The calculator will perform the calculation according to standard arithmetical rules as follows:
   - A + B will [add] the values of A and B
   - A - B will [subtract] the value of B from A
   - A * B will [multiply] A by B
   - A / B will [divide] A by B
5. The calculator will print the result as a real number with up to 20 significant digits.
6. Error requests in the calculator will be handled as follows:
   -_invalid values for A, the message "A is not a number" or "A is out of range" as appropriate
   -_invalid values for B, the message "B is not a number" or "B is out of range" as appropriate
   -Invalid values for C, the message "Invalid operator" will be displayed
7. Blank or null values for A, B, or C will terminate the program.

<table>
<thead>
<tr>
<th>Story/94 Feature:</th>
<th>Priority: Must Have</th>
<th>Assigned: Paul Gerrard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculator2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>As a Any ordinary user of the system</td>
<td>Perform a Calculation</td>
<td>Achieve a Numerical Result</td>
</tr>
</tbody>
</table>

**Scenario/111 Perform Calculation**

- given first=firstnumber<firstnumber> and operator=operator<operator> and second=secondnumber<secondnumber>
- When I press the return key Then the message <result> is displayed and the calculator exits

<table>
<thead>
<tr>
<th>firstnumber</th>
<th>operator</th>
<th>secondnumber</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>Ending</td>
</tr>
<tr>
<td>1</td>
<td>+</td>
<td>1</td>
<td>Ending</td>
</tr>
<tr>
<td>abc</td>
<td>+</td>
<td>1</td>
<td>A is not a number</td>
</tr>
<tr>
<td>1</td>
<td>+</td>
<td>pqr</td>
<td>B is not a number</td>
</tr>
<tr>
<td>1</td>
<td>×</td>
<td>1</td>
<td>Invalid operator</td>
</tr>
<tr>
<td>-1,000,000,000.000001</td>
<td>+</td>
<td>1</td>
<td>A out of range</td>
</tr>
<tr>
<td>10000000000.000001</td>
<td>+</td>
<td>-10000000000.000001</td>
<td>A out of range</td>
</tr>
<tr>
<td>1</td>
<td>+</td>
<td>10000000000.000001</td>
<td>B out of range</td>
</tr>
<tr>
<td>1</td>
<td>+</td>
<td>-10000000000.000001</td>
<td>B out of range</td>
</tr>
<tr>
<td>10000000000.000001</td>
<td>+</td>
<td>-10000000000.000001</td>
<td>-20000000000.0</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>*</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>/</td>
<td>1</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Dictionary index references

<table>
<thead>
<tr>
<th>Reference</th>
<th>Ref Type</th>
<th>In Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>Proposed Business Term</td>
<td>requirement/196/content</td>
</tr>
<tr>
<td>divide</td>
<td>Proposed Business Term</td>
<td>requirement/196/content</td>
</tr>
<tr>
<td>multiply</td>
<td>Proposed Business Term</td>
<td>requirement/196/content</td>
</tr>
<tr>
<td>subtract</td>
<td>Proposed Business Term</td>
<td>requirement/196/content</td>
</tr>
<tr>
<td>number</td>
<td>Glossary Term</td>
<td>requirement/196/title</td>
</tr>
<tr>
<td>number</td>
<td>Glossary Term</td>
<td>requirement/196/content</td>
</tr>
<tr>
<td>Operator</td>
<td>Glossary Term</td>
<td>requirement/196/content</td>
</tr>
<tr>
<td>significant digits</td>
<td>Glossary Term</td>
<td>requirement/196/content</td>
</tr>
</tbody>
</table>

Library References for this Record

- **Process Image**: Title, Version 1, File type image (jpg), Download, Remove

Add a Reference to an Existing Document  Add New Document to the Library
Requirement, feature, scenario traceability

Requirement/Feature/Scenario hierarchy

Feature/Scenario Detail
So what?

• We have the full business hierarchy:
  • Business goals (and associated risks)
    – Requirements
      • Features
        – Scenarios – and therefore TESTS
  • We can report test coverage at any of these levels
  • We can also report coverage against any entry in the glossary (Ubiquitous Language).
The End of Manual Feature Checking

Let the tools do it all and free the testers to do exploration
Fully automated feature checking

- Test harness code generation is just a report:
  - A unit test framework xUnit (Java, C++, Python…)
  - A ‘BDD’ tool e.g. cucumber, fitnesse, specflow…
  - A framework like Robot Framework…
  - Some proprietary tools
- The fixture code connecting the test harness to the system under test needs to be written
- If our story scenarios can be used to generate a test harness script (for our favourite test tool)…
- …then all story-based checks can be run automatically – on demand and in the cloud.
The feature test feedback loop

- Goal
- Risks
- Requirement
- Scenarios
- Test Log
- Story/Feature
- Examples

Scope of the Dictionary

Intelligent Definition and Assurance
Traceability and impact analysis
Requirement, feature, scenario traceability
Business impact analysis becomes viable

• We have traceability from goals, risks through requirements stories and tests
• With continuous delivery, these artefacts are always in synchronisation
• We can trust our analyses much more
• Most projects that fail should never have been started in the first place

*Lean starts with choosing the right projects in the first place.*
Summary
Agile Business Analysis and Testing become One

Test strategy and requirements validation are part of analysis
Requirements pipeline and tests are goal- and risk-based

We deliver software against goals
We test to address risk and measure achievement
Testers work closer with BAs or even become one
Agile Governance comes naturally

Traceability + Project/Iteration Management + Change History = Governance
Close

• We have an opportunity to transform the way we deliver with good governance
• Requirements+stories define the need
• Development teams have dramatically better input from their business analysts
• Tests are derived from specs and give rapid feedback, so specs are accurate (for all time)
• Automation helps, but it’s the change of mindset that matters.
Thank-You

Our Thinking: businessstorymethod.com

Our Technology: businessstorymanager.com
Live Specifications: From Requirements to Automated Tests and Back

Paul Gerrard
paul@gerrardconsulting.com

gerrardconsulting.com

business story manager

gerrardconsulting.com

The Business Story Pocketbook
Paul Gerrard

The Tester's Pocketbook
Paul Gerrard