

Automating the Automation - Harnessing the power of model based testing

James Walker – Manager, Software Engineering
Joshua Taylor – Principal, Product Owner



James Walker, PhD

Responsible for the engineering team
for CA Agile Requirements Designer.



Joshua Taylor

Product owner of CA Agile
Requirements Designer.

Agenda

- 1 **TRADITIONAL TESTING**
- 2 **THE SECRETS OF SUCCESSFUL AUTOMATION**
- 3 **MODEL BASED TESTING FOR AUTOMATING THE AUTOMATION**
- 4 **DEMO - HOW CA USE AGILE REQUIREMENTS DESIGNER**
- 5 **WRAP UP AND QUESTIONS**

Traditional Software Development

Poor Requirements

- A plethora of techniques exist, most are written in ambiguous natural language
- The requirements are “static” - they offer no way to derive tests directly from them...
- ... no way to update tests when the requirements change
 - this has to be done manually

Chapter 6: Requirements Analysis		99
6.1	Prioritize Requirements	99
6.2	Organize Requirements	103
6.3	Specify and Model Requirements	107
6.4	Define Assumptions and Constraints	111
6.5	Verify Requirements	114
6.6	Validate Requirements	117
Chapter 7: Solution Assessment & Validation		121
7.1	Assess Proposed Solution	121
7.2	Allocate Requirements	124
7.3	Assess Organizational Readiness	127
7.4	Define Transition Requirements	131
7.5	Validate Solution	134
7.6	Evaluate Solution Performance	137
Chapter 8: Underlying Competencies		141
8.1	Analytical Thinking and Problem Solving	141
8.2	Behavioral Characteristics	144
8.3	Business Knowledge	145
8.4	Communication Skills	148
8.5	Interaction Skills	150
8.6	Software Applications	152
Chapter 9: Techniques		155
9.1	Acceptance and Evaluation Criteria Definition	155
9.2	Benchmarking	156
9.3	Brainstorming	157
9.4	Business Rules Analysis	158
9.5	Data Dictionary and Glossary	160
9.6	Data Flow Diagrams	161
9.7	Data Modeling	163

The results are often poor

Summary

Ambiguity Category	Number of occurrences
Contradiction	7
Ambiguity of reference	10
Dangling Else (including improper use of conditionals)	9
I.E. versus E.G.	1
Omissions (Missing Clauses)	5
Omissions (Missing Definitions)	3
Completely Ambiguous	1
Ambiguous Logical Operators	2
Ambiguous Precedence Relationships	5
Incomplete Definition Completed Elsewhere	3

Total ambiguities found: 46

Over 50% of defects are introduced in the design phase

Manual Test Case Design

- Currently manual – a **time consuming, error-prone** process
- Is unsystematic, ad hoc, and has no real notion of “**coverage**”
- **Over-testing** and **under-testing** – 10-20% coverage with 4 times over-testing
- Poor requirements lead to **poor overall testing**, with testers having to fill in the gaps
- No linkage to **test data** – process is manual, painstaking and very time-consuming
- No flexibility for **change requests**: a critical weakness in an agile or continuous delivery environment. Changes take longer than the original requirement.

	📄	Step Name	Description	Expected Result
		Step 1	Enter valid username	Username is shown on login panel input
		Step 2	Enter valid password	Password is entered but masked on the display
		Step 3	Click login	The user is forwarded to the welcome page

Poor Testing -> Automated testing

Automated testing: Manual script generation

- Automated testing frameworks are heavily scripted
- Script generation is usually done manually
- As well as the maintenance of scripts

- Alternative solutions use:
- Record playback
- Script-less automation frameworks (keyword)



But you're still doing manual test case design!

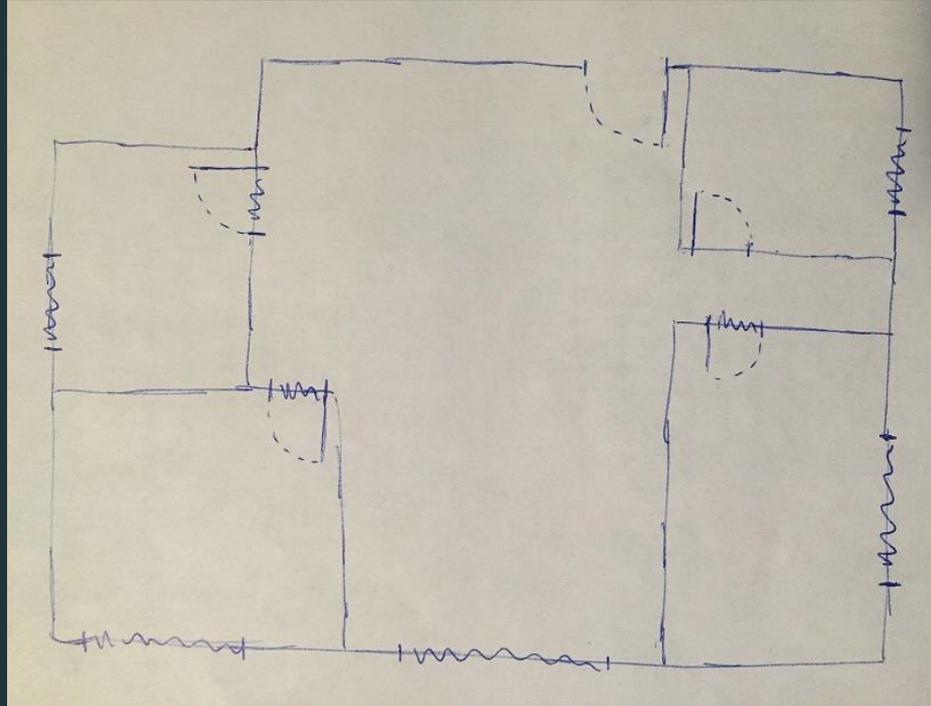
Fundamentals of Successful Automation

Fundamentals of Successful Test Automation

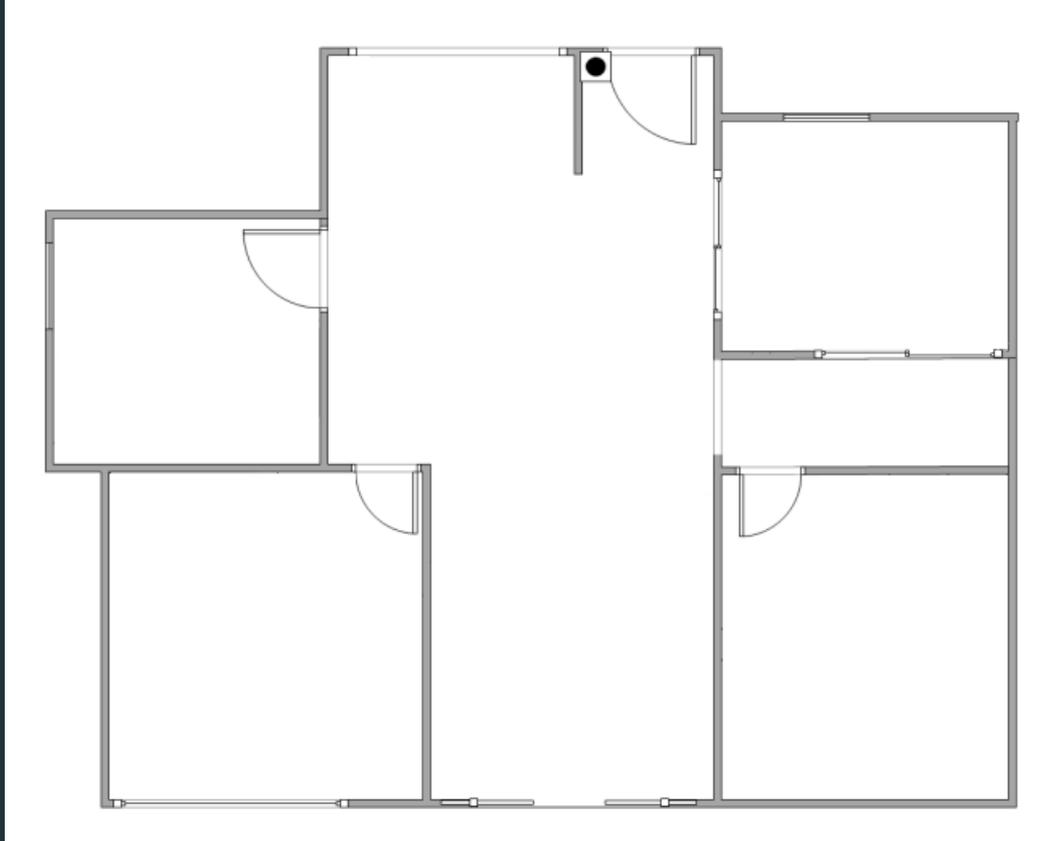
- 1) Modular – Test cases are reused to mitigate reinventing the wheel every time a test is written.
- 2) Maintainable – Traceability between requirements and tests – when the requirements changes there is a low cost to identify and update the associated tests
- 3) Notion of coverage – Tests are relevant to support the test requirements, minimising the number of tests, while ensuring the maximum amount of functionality is covered
- 4) The right data at the right time – The right test data is utilized and deployed for each test at the right location at the right time during execution.

Model based Testing for Test Automation

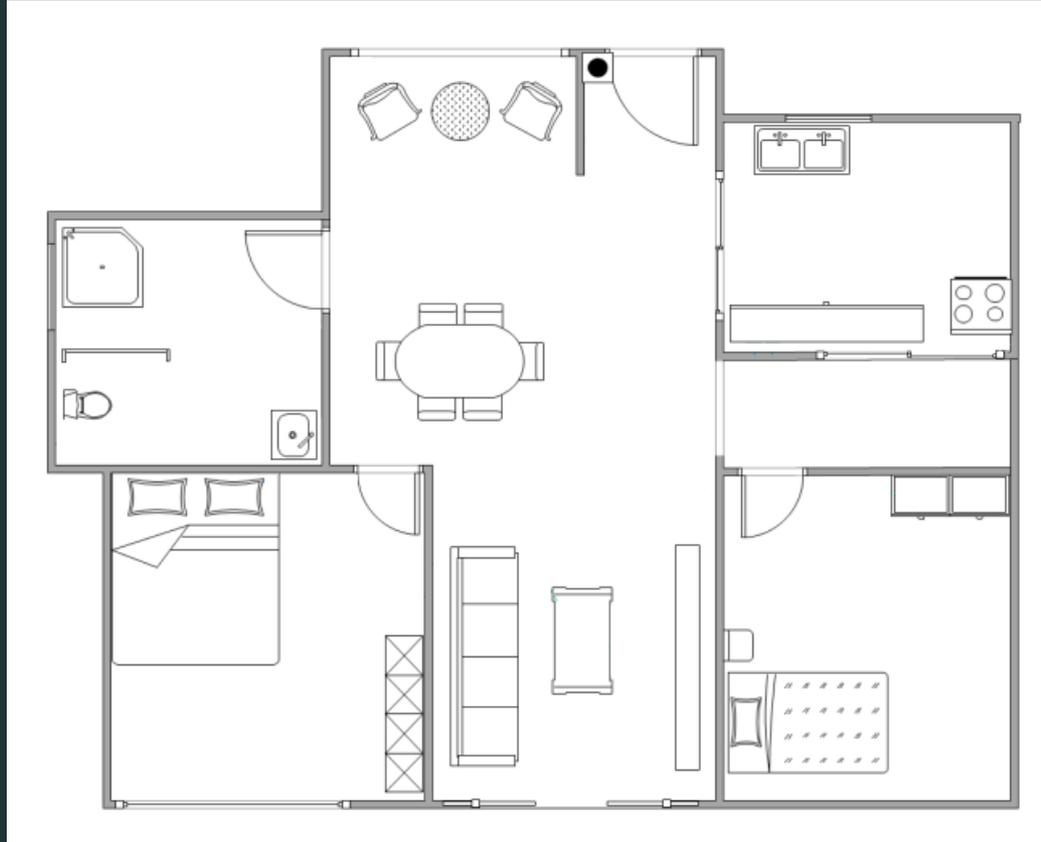
Persona 1 – The Home Owner



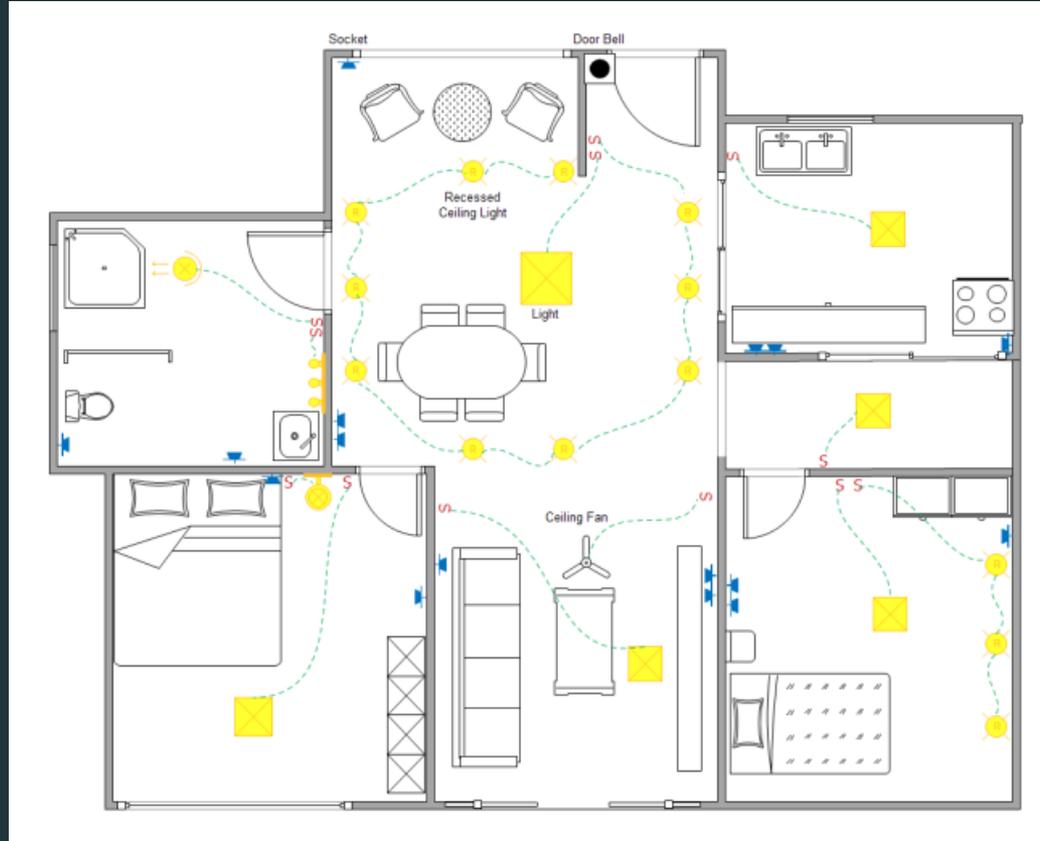
Persona 2 – The Architect



Persona 3 – The Interior Designer

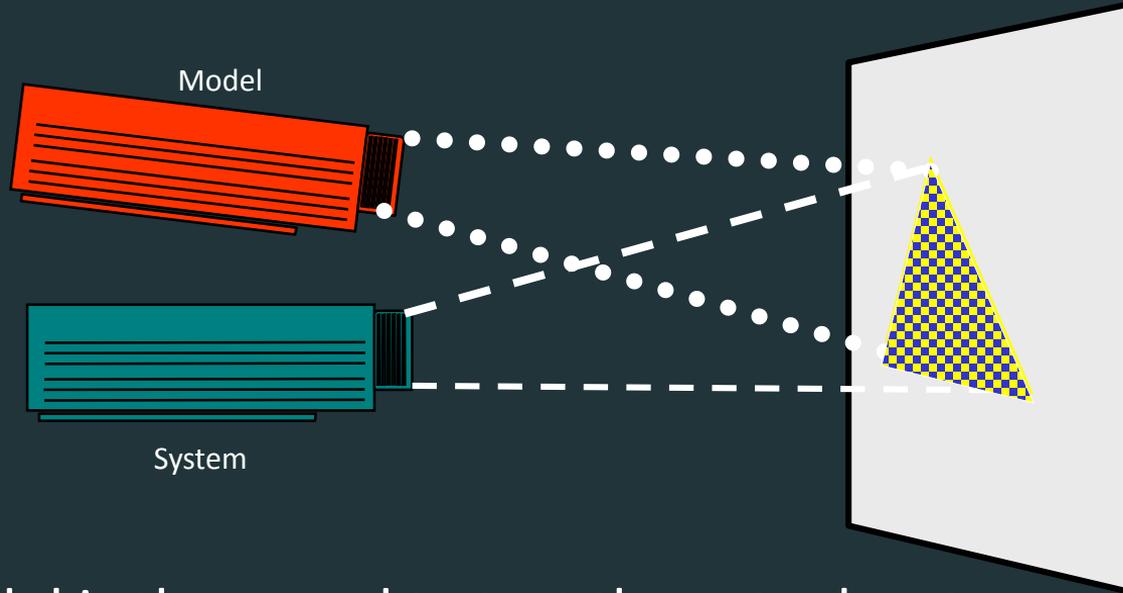


Persona 4 – The Electrical Engineer



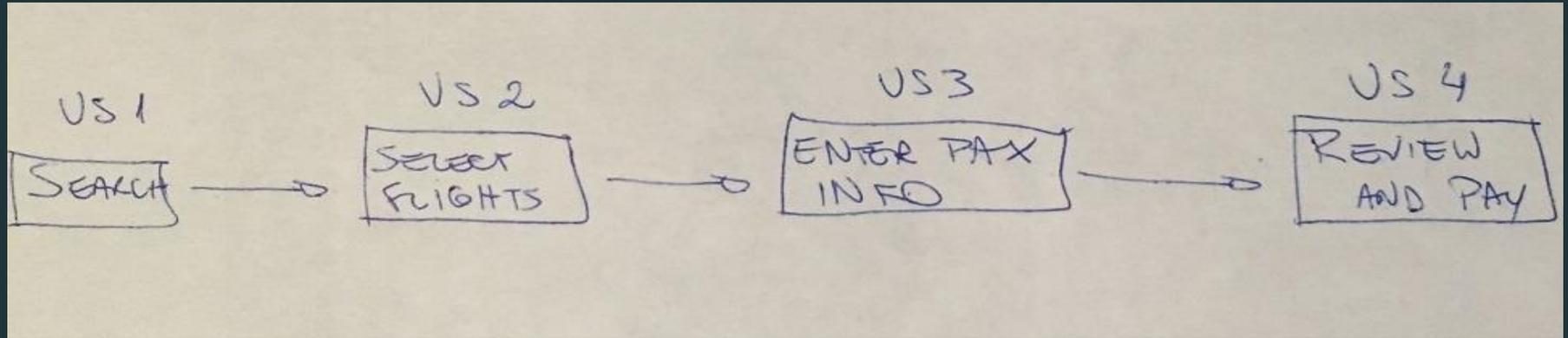
What is Model Based Testing?

Model based testing lets you define the behavior of a system under test – in other words what is supposed to happen.

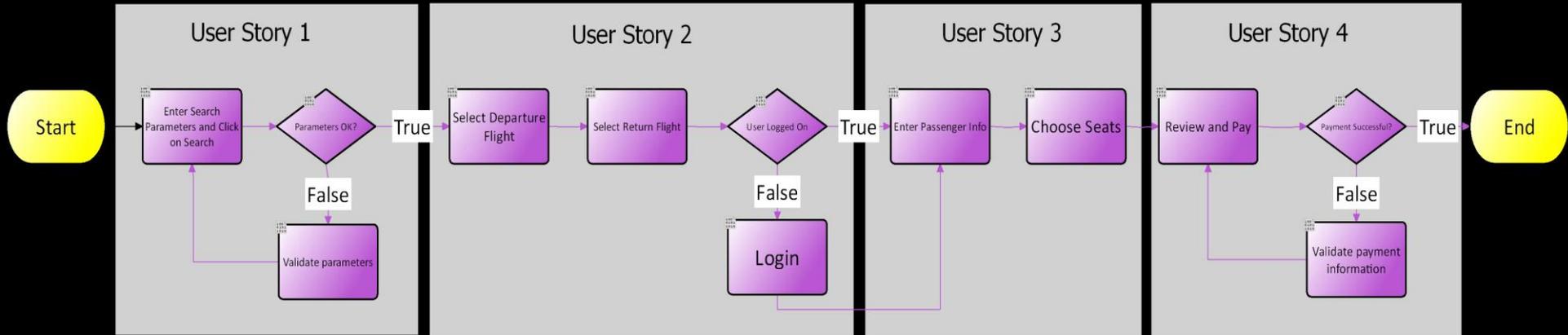


The model is then used to test the actual system to see if it does what it is meant to.

Persona 1 – The Product Owner



Persona 2 – The Modeler



- Streamlines understanding across all teams in the value stream by removing all ambiguity from the scope
- Foundational layer others will build on top of (single source of truth)

Persona 4 – The SOA Engineer

Enter Search Parameters and Click on Search

General

Process Details

Test Data

Make System Data

Find System Data

Images

Custom Fields

Automation

Properties

Details

Messaging

Tables

Stored Paths

Application Links

People & Roles

Requirements IDs

Virtualized

End point:

Operation:

Save Cancel

Enter Search Parameters and Click on Search

Persona 4 – The Test Automation Engineer

The screenshot shows a software application window titled "Valid Data". On the left is a sidebar with a list of menu items: General, Process Details, Test Data, Make System Data, Find System Data, Images, Custom Fields, Automation, Properties, Details, Messaging, Tables, Stored Paths, Application Links, People & Roles, and Requirements IDs. The "Test Data" item is highlighted in orange. The main area contains a table with the following data:

Name	Type	Action	Variable	Repeater	Code Snippet
1	Assertion	equal to URL	1: MainPage	1	<pre>browser.getCurrentUrl().then(function (url) { assert.equal(url, '~url~'); });</pre>

At the bottom of the table area are three buttons: "Add", "Edit", and "Remove". To the right of these buttons are three icons: an up arrow, a down arrow, and a green checkmark. At the bottom right of the window are "Save" and "Cancel" buttons. A help icon (question mark in a square) is located at the bottom left of the window.

Enter Search
Parameters and Click
on Search

Active Automation

Active Automation

- Test cases and scripts are created automatically from “Active” requirements
- Testing is Model Based for maximum coverage
- Use a modular component library of common and optimized tests to promote reusability
- Test Data and Virtual End Points are created or found as part of the Automation
- If the requirements change the tests are updated.

How CA Use Agile Requirements Designer

Drinking our own champagne

- We use CA Agile Requirements Designer in our own software development
- Model requirements in ARD
- Overlay the Ranorex automation framework
- Automatically generate automated functional tests

Conclusion

Conclusion

- Have ARD create a perfect test suite for my coverage requirements.
- For each test case automatically generate me an automation script.
- How we can edit the ARD model to reflect a change in the requirements.
- How that change instantly updates our automation scripts.
- Providing automatic automation with MBT.

Thank you.

Stay connected at communities.ca.com