Building an Automation Ecosystem

©Clare McLennan 2014
The biotic components of a grassland ecosystem are the living organisms that exist in the ecosystem.
Software

Build and deployment system

Code control

Automated tests

Test data

Manual tests
The big picture

Ecosystem

Environment

Action
The big picture

Ecosystem

Environment

Why? When? Who?

Where? Test data?

How? Action
Otherwise...

Tests unreliable

Tests aren’t maintained

Not enough resource
Ecosystem

Environment

Action
What’s the goal?

To create an automated test suite

Why?
What’s the REAL goal?

- Reduce repetitive testing
- Smoke test

- Reduce testing cost
- Deliver quicker
- Deliver more frequently
- Reduce regression defects
- Develop features faster
Is it achievable?
Is it achievable?
Is it achievable?
Justifying the build cost

• Reduce manual regression testing
• Respond to customer needs quicker (more releases)
• Develop new features quicker
  – Find bugs faster = cheaper to fix
  – Refactor code to remove technical debt
• Reduce staff turnover by making work more enjoyable
Right people?

• Tests automation systems are complicated software systems

• They require
  – Design
  – Some programming
  – Best built by a team
  – Commitment
Right people?

• Support from
  – Developers
  – Management

• Support to
  – Change existing systems

• Bigger job – $$ to do it right

Don’t try this by yourself!
Get help from your team
Environment

• Where will tests run?
• Where does the test data come from?
Data setup
Data setup is critical

Reason 1: Tests need to run multiple times

• Develop the test
• Debug failing test

Once is enough – I know what I’ve done
Data setup is critical

Reason 2: Small changes can break automation

• Uniqueness
  – *Eg. Adding the same user*

• Paging
  – *Eg. The eleventh user may appear on a new page*
Just because you can doesn’t mean you should
Data setup is critical

Reason 3: Time

Timely test data is needed
  – Graph last 7 days data

Time stamps

Large time delays
  – Cookies expire after 30 days
Data setup is critical

Reason 4: Summary reports

• Any change in the system changes the values
  – Eg. Total $ received from all clients

Looks about right!
Options for data

• Recent copy of production
• Canned data aka Golden Database
• Create test data on-the-fly
Production Copy

✔ Implemented

❌ What is the data?

❌ Lots and lots and lots of maintenance

❌ Overcomplicated tests
Canned data

Test script has:
Reload testing database
Log on as “Clare McLennan”

Manually create new organisation
Upload data files
Store database

1

2
Canned data

- Known state
- Tests and database must be kept in sync
- Loading databases slow
- Writing new tests is slow
Canned data

- Upgrade path for database schema (including dev versions)
- Storage
- Prevent tests interfering
- Tools available
  - Transactions
  - Snapshots
Create test data

Test script includes:
Create new organisation
Load “doc-huts.kml”
Load “nz-fire-stations.kml”
Create test data

- Quick to write new tests
- Options include
  - Database insertions
  - Existing processes (eg CSV)
- GUI tests run faster when data *not* created through UI
Small is beautiful

- Run faster
- Easier to Debug/Maintain
Test Time
Change
application so test can specify time/date
No need to change data now!
An extensive, but low risk change to application
Environment

• Individual machines or shared environment?
Environment

• Correct set up
• Tests independent
• In control
  – Change time
  – Summed reports
Shared environment

Correct set up
Tests independent
In control
– Change time
– Summed reports

Separate organisations?
Per customer?

FREE

$$$$$

$$$$$
Own machine

Correct set up
Tests independent
In control
  – Change time
  – Summed reports
Team effort

- Database upgrades
- Software + test environment installer
- Source control
- Data extraction (small subset)
- Implementing test time
- Create test data
- Copy data (for customer/organisation...)
- Remove/Discard (customer/organisation...)
Action

Tools

First tests

Write to be maintained
Which to choose?
Design

Before GUI starts set up data
Create “TestOrg1”
Load “doc-huts.kml”
Load “nz-fire-stations.kml”
Design

Select layer “doc-huts”
Select layer “nz-fire-stations”

Click X
Visual recognition

New paradigm

Great for testing *parts* of application

• Visuals
• Difficult controls
Choosing a tool

- **Flexibility**
  - Setting up data & environment
  - Driven by another tool

- **Maintainability**

- **Can everyone have a copy?**

- **Manual testing**
GUI testing

GUI (Automated)

Integration

Component

Unit Testing

Accessible
Slow
Fragile
Business logic testing

- Unit Testing
- Component Testing
- Integration Testing
- GUI Testing (Automated)

- Fast
- Reliable
- May need devs help
Design

Test Specification

Glue

Api for data set up
Mock
Business logic
Gui
People aren’t computers

<table>
<thead>
<tr>
<th>Doing things differently</th>
<th>Doing things exactly the same</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adapting tests</td>
<td>Repeating tasks</td>
</tr>
<tr>
<td>Thinking</td>
<td>Checking numbers</td>
</tr>
<tr>
<td>Usability testing</td>
<td></td>
</tr>
</tbody>
</table>
Target start place

Difficult to do manually

Easy to automate

Valuable (reduces risk)

Sweet spot
Computers

• Hard to train to do a testing task

• Very good at doing it over and over ...and over and over...and over and over...
**Good targets**

- Reports
- Tax calculations
- Multiple ad impressions

End to end UAT test
Maintainability

Garden timer example
Example

Go [http://testsystem/admin](http://testsystem/admin)
Type “password” in input[id='password']
Click button[text()='OK']
Wait for div[class='timerstart']
Click div[text='Add interval']
Type “09:00” in input[class='time'][0]
Type “00:15” in input[class='time'][1]
Click button[text()='Save']
Wait for div[class='time'] has ‘09:00’
Wait for div[class='time'] has ‘09:00’

Example

Go to [http://testsystem/admin](http://testsystem/admin)

Type “password” in input[id = ‘password’])

Click button[text()=‘OK’]

Wait for div[class=‘timerstart’]

Click div[text= ‘Add interval’]

Type “09:00” in input[class=“time”][0]

Type “00:15” in input[class=“time”][1]

Click button[text()=‘Save’]

Wait for div[class=‘time’][0] has ‘09:00’

Wait for div[class=‘time’][1] has ‘00:15’

So what’s this test doing?

And what if we change the log in?
Example

Log on as admin
Add time interval (start = 9:00, duration = 0:15)
Wait until 08:55
Check timer is off
Wait until 09:00
Check timer is on
Example

If time interval is start = 9:00, duration = 0:15
then
Timer should be off at 08:55
Timer should be on at 09:00
**SUMMARY**
The timer should turn on at the correct time and off after the duration has elapsed.

**DETAILS**
Start time is the 24 hour clock time.
Duration is how long the guy wants the timer to be on.

**EXAMPLES**

<table>
<thead>
<tr>
<th>Should timer be on</th>
<th>Start Time</th>
<th>Duration</th>
<th>End Time</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before Start</td>
<td>09:00</td>
<td>0:15</td>
<td>08:59</td>
<td>Off</td>
</tr>
<tr>
<td>Start</td>
<td>09:00</td>
<td>0:15</td>
<td>09:00</td>
<td>On</td>
</tr>
<tr>
<td>During</td>
<td>09:00</td>
<td>0:15</td>
<td>09:04</td>
<td>On</td>
</tr>
<tr>
<td>End</td>
<td>09:00</td>
<td>0:15</td>
<td>09:15</td>
<td>Off</td>
</tr>
<tr>
<td>After End</td>
<td>09:00</td>
<td>0:15</td>
<td>09:20</td>
<td>Off</td>
</tr>
</tbody>
</table>
**TurnOnPeriod**

**Test Results** [history]

**Assertions:** 5 right, 0 wrong, 0 ignored, 0 exceptions (0.084 seconds)

**Summary**
The timer should turn on at the correct time and off after the duration has elapsed.

**Details**

```
Start time duration
```

Start time is the 24 hour clock time to start watering.
Duration is how long the garden should be watered for, specified in the `hours:minutes`

**Examples**

<table>
<thead>
<tr>
<th>Description</th>
<th>Start Time</th>
<th>Duration</th>
<th>Check Time</th>
<th>Timer Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before Start</td>
<td>09:00</td>
<td>0:15</td>
<td>08:59</td>
<td>Off</td>
</tr>
<tr>
<td>Start</td>
<td>09:00</td>
<td>0:15</td>
<td>09:00</td>
<td>On</td>
</tr>
<tr>
<td>During</td>
<td>09:00</td>
<td>0:15</td>
<td>09:04</td>
<td>On</td>
</tr>
<tr>
<td>End</td>
<td>09:00</td>
<td>0:15</td>
<td>09:15</td>
<td>Off</td>
</tr>
</tbody>
</table>
public class ShouldTimerBeOn {

    // Other methods...

    public void setStartTime(String onTime) {
        this.startTime = onTime;
    }

    // Other methods...

    public String timerStatus() {
        return Timer.isWaterOn(checkTime) ? "On" : "Off";
    }

    // Other methods...
}
Specification by Example
aka ATDD

- Tests expressed in a human readable and maintainable form
- Testing at GUI layer or calling code directly
- Tests relevant even when technology changed
Lots of options!
Ecosystem

Environment

Action
Nature

The biotic components of a grassland ecosystem are the living organisms that exist in the ecosystem.
As a separate activity

Coders → Testers → Automators
Specification by Example

Discuss feature
**TurnOnPeriod** [add child]

**SUMMARY**
The timer should turn on at the correct time and off after the duration has elapsed.

**DETAILS**

Start time is the 24 hour clock time to start watering.
Duration is how long the garden should be watered for, specified in the `hours:minutes`.

**EXAMPLES**

<table>
<thead>
<tr>
<th>Description</th>
<th>Start Time</th>
<th>Duration</th>
<th>Check Time</th>
<th>Timer Status?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before Start</td>
<td>09:00</td>
<td>0:15</td>
<td>08:59</td>
<td>Off</td>
</tr>
<tr>
<td>Start</td>
<td>09:00</td>
<td>0:15</td>
<td>09:00</td>
<td>On</td>
</tr>
<tr>
<td>During</td>
<td>09:00</td>
<td>0:15</td>
<td>09:04</td>
<td>On</td>
</tr>
<tr>
<td>End</td>
<td>09:00</td>
<td>0:15</td>
<td>09:15</td>
<td>Off</td>
</tr>
<tr>
<td>After End</td>
<td>09:00</td>
<td>0:15</td>
<td>09:20</td>
<td>Off</td>
</tr>
</tbody>
</table>
Specification by Example

Discuss feature

Write specs

Write Code

Automate

specs = tests

All test pass

Feature really DONE 😊
Team approach

Testers specify the tests

Tests reduce manual testing

Coders write (testing) code

Team members doing what they do best
As part of story

• Coverage
  – Rate of application development = rate of test development

• Team ownership
  – Coders consider how changes will affect tests

• Knowledge spread

• Easier hiring
All on board
The big picture

- Ecosystem
- Environment

Action
My Guiding Rules

• Tests run on debug environment
• Testers can run tests
• Different users can run tests simultaneously
• Tests run as part of DOD
• Test run < 1 hour
Tools

Selenium
http://seleniumhq.org/

Sikuli
http://sikuli.org/

FitNesse
http://fitnesse.org/
Gojko Adzic
http://specificationbyexample.com/
http://gojko.net/

http://concordion.org/Technique.html

Clare McLennan
clare.mclennnan@gmail.com
http://crazy-it-adventures.blogspot.com