

Making GUI testing productive and agile

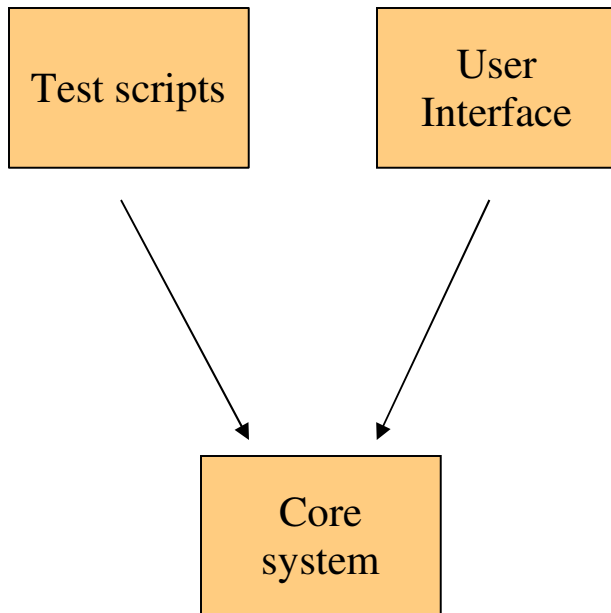
Geoff Bache, ScanDev 2010



Talk Overview

- Data-driven testing
- Record-playback and problems with current tools
- PyUseCase demo
- How the xUseCase tools work
- Why this is both productive and agile
- Experiences using this approach

Data-driven GUI testing



- Avoid volatile UI by bypassing it
- Create a more stable "domain API" into the system.
- Write test scripts that call it and simulate the actions of a user.
- Assert that returned values are as expected.
- Test the UI itself manually

Limitations of the data-driven approach

Fixtures.PowerPoint		
open	usecase_recording.ppt	
check	slidecount	18
check	view	Normal

- This "Fit-style" test is clear and probably very maintainable.
- But can it really give a non-coder confidence in PowerPoint?
- Deals in abstractions: what does "slidecount" mean in practice?
- Abstract tests require test writers who can think in abstractions. A long way towards programming.

GUI Testing by record/playback

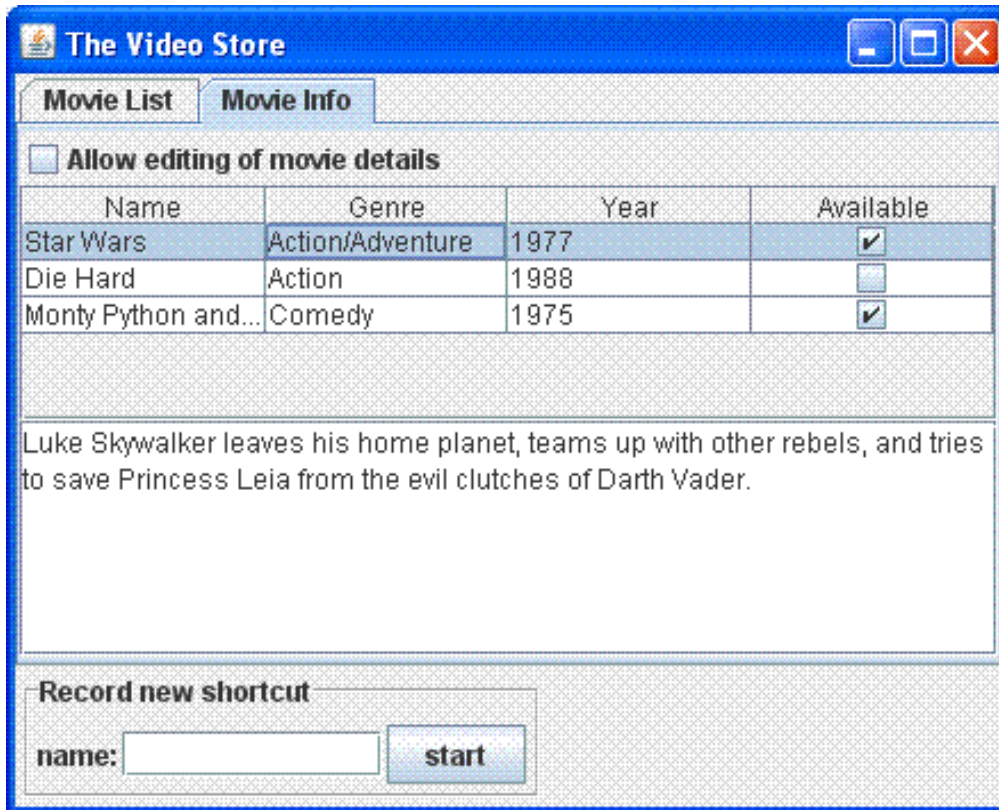


- Click through your application as a user would.
- Record a script of all the actions performed
- Replay the script and check the application behaves as expected
- Old and somewhat discredited idea

Limitations of current record/playback implementations

- The key is **what** they record.
 - How easy is it to understand?
 - How easy is it to maintain when the GUI changes?
- Too much re-recording makes the approach unworkable
 - Recorder output tightly coupled to the current UI layout
 - Usually need to go in and “re-program” after recording the test.
 - Difficult for non-programmers to create maintainable tests.

A GUI testing example



- Chosen a simple use case for a “video store” application.
- Recorded a replayable test script in a couple of modern open source tools.
- Can you guess what the tests are about?

Abbot

```
<action args="JComboBox Instance,Serpico"
class="javax.swing.JComboBox"
        method="actionSelectItem" />
<action args="add" class="javax.swing.AbstractButton"
        method="actionClick" />
<action args="load movie list"
class="javax.swing.AbstractButton"
        method="actionClick" />
<action args="5000" method="actionDelay" />
<action args="sort list" class="javax.swing.AbstractButton"
        method="actionClick" />
<action args="JList Instance,&quot;Battle Royale&quot;"
        class="javax.swing.JList"
method="actionSelectRow" />
<event component="JList Instance" keyCode="VK_CONTROL"
kind="KEY_PRESSED"
        modifiers="CTRL_MASK" type="KeyEvent" />
<action args="JList Instance,&quot;King
Pin&quot;,BUTTON1_MASK|CTRL_MASK"
        class="javax.swing.JList"
method="actionClick" />
<event component="JList Instance" keyCode="VK_CONTROL"
kind="KEY_RELEASED"
        type="KeyEvent" />
<action args="remove selected movies"
class="javax.swing.AbstractButton"
        method="actionClick" />
<action args="The Video Store"
class="abbot.testers.WindowTester"
        method="actionClose" />
```

- Lots of details, although XML format makes it look worse
- Compared to the tools of 10 years ago, this is very short...
- The age of pixel positions is over.
- But this script is tightly coupled to the current GUI

Marathon

```
window('The Video Store')
select('Movie ComboBox', 'Serpico')
click('add')
click('load movie list')
sleep(5)
click('sort list')
click('Movie List', '0')
click('Movie List', '1')
click('remove selected movies')
close()
```

- Python script – we can program our way out of trouble!
- Uses widget names to identify widgets, need to set these for every control
- Details reduced but still widget mechanics present (“click”, “window”, “0”, “1”)
- Have to insert a “sleep” for synchronisation after recording.

If the computer spoke English, how would we describe the test?

```
select popular movie Serpico
add movie
load movies
wait for movies to be loaded
sort movies
set movie selection to Battle Royale, King Pin
remove selected movies
close
```

- No fluff – 8 lines for 8 actions. No GUI mechanics mentioned. “Wait” says why it’s there.
- Easy to edit without re-recording. Easy to write without having a system yet. Is basically a description of a usecase.
- Is there really any reason why we can’t record and replay such a script?

Demo – PyUseCase in practice

Enter Usecase names for auto-recorded actions

Previously unseen actions: provide names for the interesting ones

<u>Widget Type</u>	<u>Identified By</u>	<u>Action Performed</u>	<u>Usecase Name</u>
Entry	Name=Movie Name	edited text	set movie name to
Button	Label=Add	clicked	add movie
Window	Title=The Video Store	closed	close

Current Usecase Preview

```
set movie name to Star Wars  
add movie  
close
```

OK

Recording in a “domain language”

<u>Widget Type</u>	<u>Identified By</u>	<u>Action Performed</u>	<u>Usecase Name</u>
Entry	Name=Movie Name	edited text	set movie name to
Button	Label=Add	clicked	add movie
Window	Title=The Video Store	closed	close

Current Usecase Preview

set movie name to Star Wars
add movie
close

OK

```
[Title=The Video Store]
delete-event = close

[Label=Add]
clicked = add movie

[Name=Movie Name]
changed = set movie name to
```

- When we record an action we haven't seen before, ask the user “what they intended” (which might be nothing)
- Store this info in a “UI map file”, which maps the current GUI mechanics to “usecase names”. Record only the name they provide.
- Widgets identified by Name, Title, Label and Type.
- When the GUI changes, update only this file.

How most tools check the application behaved corectly

```
select popular movie Serpico
add movie
movieList = getWidget("Movielist").getData()
assert("Serpico" in movieList)
load movies
movieList = getWidget("Movielist").getData()
assert(len(movieList) == 5)
close
```

- The usual “unit test” approach of adding assertions to the script
- Assertions mean variables. Variables mean programming.
- Our script above is not a high-level usecase any more. Now we have code to maintain.

Logging the GUI

```
----- Window 'The Video Store' -----  
Focus widget is 'Movie Name'  
  
Menu Bar :  
  'File' Menu :  
    'Add'  
    'Delete'  
    'Show buttons' (checked)  
    'Quit'  
'New Movie Name ' , Text entry , Button 'Add' ,  
Button 'Delete'  
  
Showing Notebook with tabs: text info , video view  
Viewing page 'video view'  
  
Showing Movie Tree with columns: Movie Name  
-----  
  
'set new movie name to' event created with  
arguments 'Star Wars'  
Edited 'Movie Name' Text entry (set to 'Star Wars')  
  
'add movie' event created with arguments ''  
  
Updated : Movie Tree with columns: Movie Name  
-> Star Wars  
...
```

- PyUseCase provides a log of all GUI changes.
- Supplement as needed/ desired with app-specific logs.
- Use a test tool, such as TextTest, that compares textual output
- Save the logs when correct behaviour changes
- Can leave the usecase in peace

A Test is made up of:



A use case
recording

+



A saved
GUI log

This part recorded by anyone who can
use the GUI

This part is generated
automatically, supplemented by
application-specific logs

Creating new tests requires little or no programming

Synchronising GUI tests

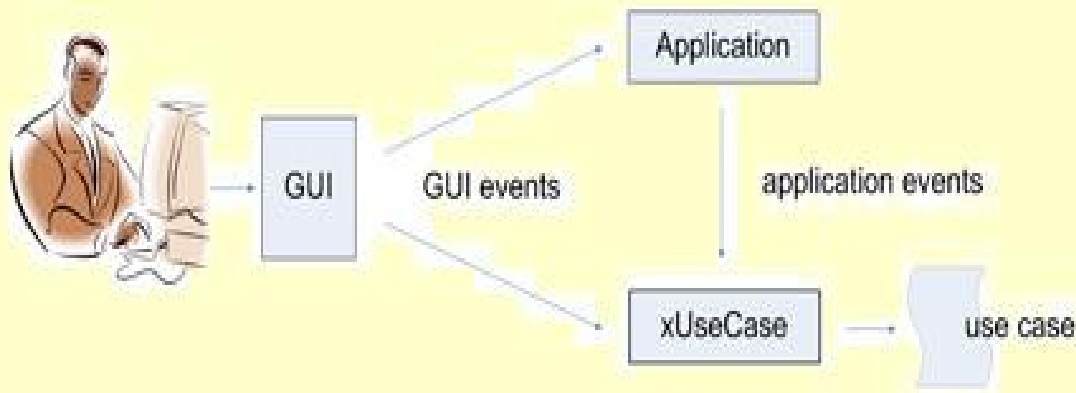
```
sleep(10)
```

```
movieList = getWidget("Movielist").getData()  
wait(len, movieList, 5)
```

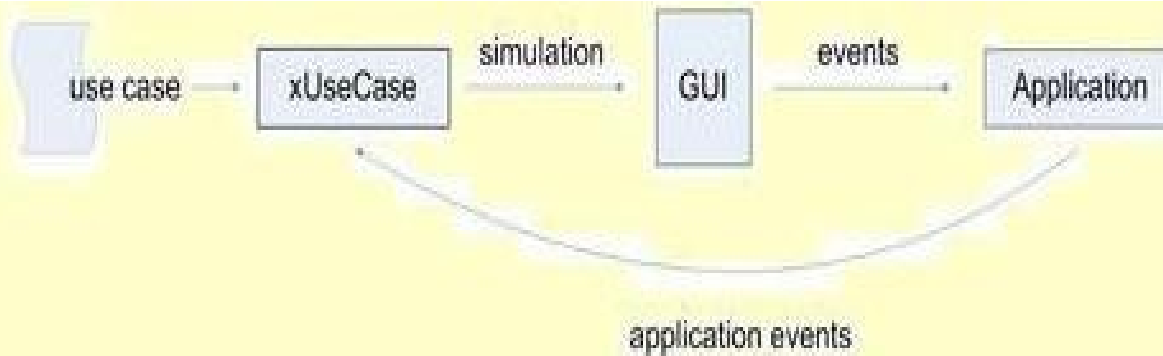
```
wait for movies to be loaded
```

- A large number of GUIs are multithreaded – tests need to wait.
- The test writer usually has to do this explicitly themselves
 - Means they need to understand when it's needed.
- At best, you specify a wait for the GUI to change in some way
 - More GUI mechanics get hard-wired into the test.
- At worst, you have to “sleep”.
 - Tests fail when system is loaded
 - Or tests are much slower than necessary.

“Application events”: synchronising by instrumenting



xUseCase Recording, with Application Events



xUseCase Replay, with Application Events

- Application is doing some processing.
- Calls a method “applicationEvent” when it is complete
- In record mode, this just turns into a “wait for” command
- When replaying, the tool suspends replay until the same call is made.

xUseCase in practice



- There are three open source tools with this model.
 - PyUseCase for Python GUIs
 - Fairly complete support for PyGTK
 - Fairly basic support for Tkinter
 - JUseCase for Java GUIs
 - Dormant Swing version
 - Revival effort ongoing with SWT support
 - nUseCase for .net
- Not “complete”: support the widgets their users have needed.

TextTest



- Companion tool to xUseCase (not operationally necessary)
- Handles filtering and comparison of the generated logs
- Organises log differences into groups
- Organises tests into suites, manages test data and UI map files
- User interfaces around recording tests
- A host of more advanced features. Tool isn't specific to GUI testing.



Not just a toy project...

- I work daily with a testsuite that uses TextTest and PyUseCase.
- It has existed for about five years and contains about 1400 acceptance tests, of which about 650 are GUI tests with PyUseCase. Coverage = 99%.
- New tests created by recording unless they're very similar to other tests.
- Use-case scripts and UI map generally maintained via a text editor.
- Has undergone many major GUI changes in that time.
- Releases happen once per week with around 60 daily users.
- There are few unit tests and no manual testing is performed.

Productive and Agile GUI testing

- “Productive” because no programming is involved
 - Can involve business users
 - Recording tests is fast
- “Agile” because tests are readable and robust
 - UI map file decouples from current GUI layout
 - Easy to update tests when GUI changes
 - UI log changes predictable & grouped by TextTest
 - Synchronisation handled seamlessly with some trivial developer effort

Conclusions



- Current approaches to GUI testing create brittle tests which are too tightly coupled to the current UI layout.
- Address this by introducing an extra level of indirection, the “UI map file”.
- Keeping track of behaviour by monitoring logged output separates concerns neatly and has many other advantages.
- Tool support is available in the form of TextTest and the xUseCase tools.

Little or no programming => Productive
Readable, Robust tests => Agile

Pros and cons of logging as an assertion mechanism

- ✓ One less thing for the test writer to worry about
- ✓ Keep the use-cases clean: don't turn them into scripts
- ✓ Get it for free where the look of the GUI is concerned
- ✓ “Assertions” live with the code and don't depend on its structure
- ✓ Easy to introduce into a legacy system

- ✗ Somewhat easier to be lazy (read+save instead of write)
- ✗ UI changes cause many tests to “fail” (e.g. TextTest groups them)