Use-case recording: Testing a rich client UI by recording in a domain-specific language

Geoff Bache, August 2008

JEPPESEN
Making Every Mission Possible

Overview

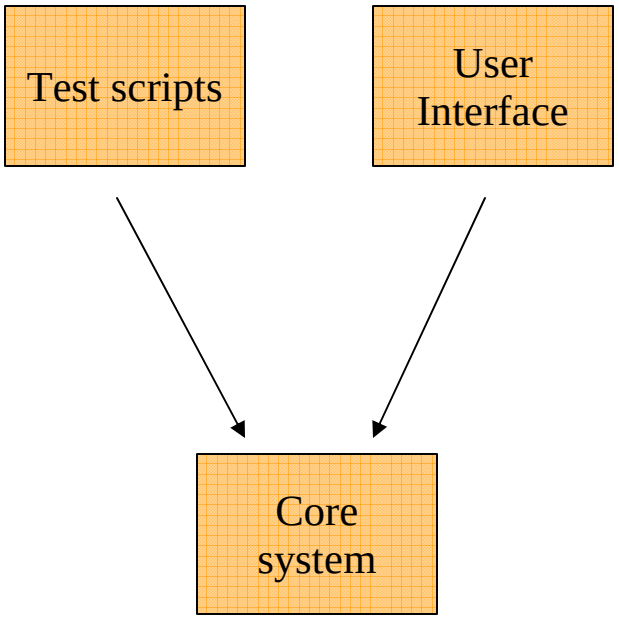
- Record/playback and data-driven testing
- Why record/playback is worth saving
- Maintainability issues
- Synchronising multi-threaded UIs
- Asserting application behaviour
- xUseCase tools in practice
- Demo

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
- Has a reputation for producing scripts that are very hard to maintain.

JEPPESEN
Making Every Mission Possible

Data-driven GUI testing



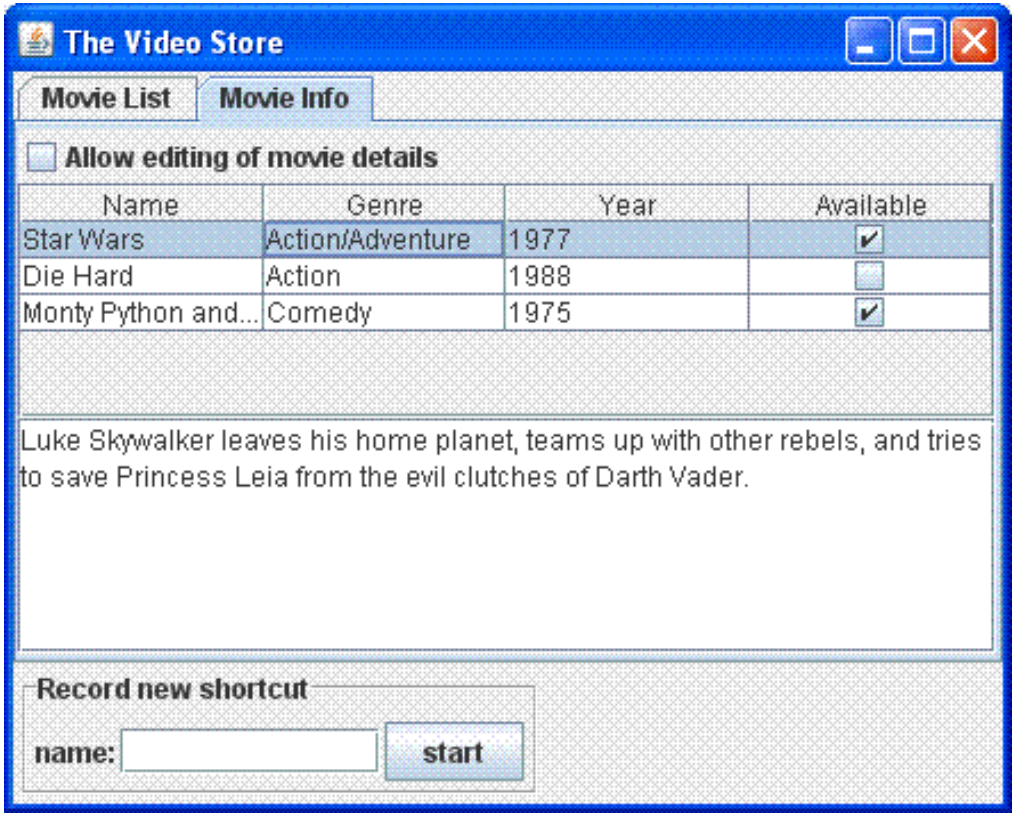
- The UI itself is a maintenance problem, so we bypass it
- Create a ”domain API” into the system.
- Write test scripts that call it and simulate the actions of a user.
- Assert that returned values are as expected.
- Test the UI itself manually

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 and learn how to interact with them. A long way towards programming.
- Observing it executing provides this confidence, and allows tests to double as system demos and/or tutorials.

Limitations of current record/playback implementations



- 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 user is trying to do?

JEPPESEN
Making Every Mission Possible

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,&amp;quot;Battle Royale&amp;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,&amp;quot;King
Pin&amp;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.tester.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.

JEPPESEN®
Making Every Mission Possible

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.

JEPPESEN
Making Every Mission Possible

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

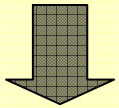
```
select popular movie Serpico
add movie
load movies
wait for movies to be loaded
sort movies
set movie selection to Battle Royale, King Pin
remove selected movies
close
```

- 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?

JEPPESEN
Making Every Mission Possible

Mapping the GUI to a domain language

```
JTextField originField = new JTextField(“ANY");
```



```
JTextField originField = new JTextField(“ANY");

ScriptedTextField.connect("choose origin", originField, ScriptedTextField.EDIT);
```

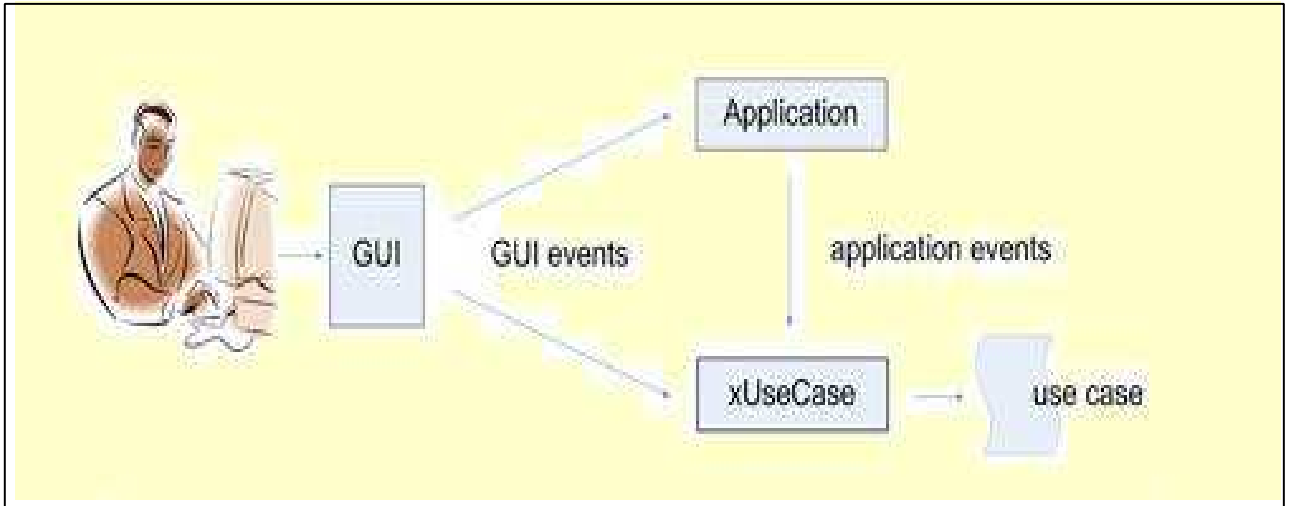
- Explicitly map GUI controls to domain level statements about what the user is doing.
- Simple code changes that have no effect when switched off
- Easiest and most flexible approach is to instrument the code to call a “use-case recording library” (“xUseCase”)
- Could also imagine trying to represent this mapping externally to the code and using widget naming.

JEPPESEN®
Making Every Mission Possible

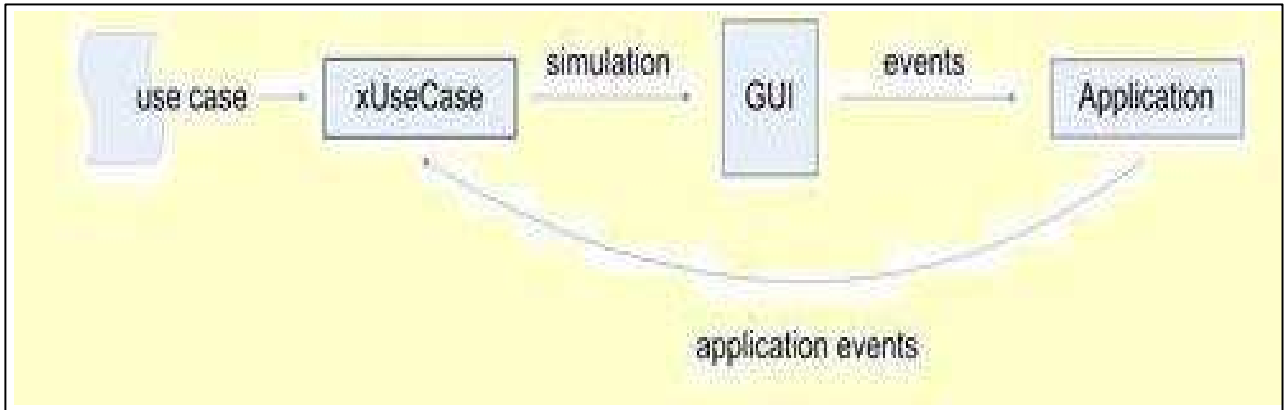
Synchronising GUI tests

- A large number GUIs are multithreaded, because they don’t want to lock the user out while processing is happening.
- Often have the situation where usecases need to wait for something to happen before proceeding.
- In the current tools, the best you can do is wait for the GUI to change in some way the tool understands, which might not be enough. At worst, you have to “sleep”.
- Lose a lot of the advantages of recording if you have to go in and program in the test script after recording it.

“Application events”: synchronising by instrumenting



xUseCase Recording, with Application Events



xUseCase Replay, with Application Events

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

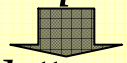
JEPPESEN
Making Every Mission Possible

Asserting application behaviour

- So far, we can only ensure that it is possible to walk through a sequence of actions. We can't see what happens along the way.
- E.g. Marathon provides a way to include assertions about widget state as part of the recording…
  - … but this ties us even more to the current GUI mechanics
  - … and is rather limited, we can't examine anything else
- Or we can go and get a programmer and insert generic assertions into the script…
  - … but then the test writer loses control of the script
  - … and we introduce dependency on internal structures
- What about using system logging as a measure of behaviour?

Using logged information to assert application behaviour

Test Writer: *assertEquals(numberOfMovies, 3)*



Developer: *logger.info("Loaded " + str(numberOfMovies) + " movies.")*

- ✓ One less thing for the test writer to worry about
- ✓ Keep the use-cases clean: don't turn them into scripts
- ✓ "Assertions" live with the code and don't depend on its structure
- ✓ Fairly easy to write reusable generic logs about GUI state
- ✓ Easy to introduce into a legacy system
- ✓ The same tests can function on many levels of detail
- ✓ Debugging builds up a knowledge base (unlike using debuggers)
- ✘ Probability of "false failure" somewhat higher (but decreases with time)
- ✘ Tests need updating when log schema changes ("press Save")

JEPPESEN
Making Every Mission Possible

Tool support is available…

- TextTest tool starts tested programs via the command line, and compares produced text files with those from previous runs. Also:
  - Filters output to avoid false failure
  - Manages test data and isolation from global effects
  - “Nightjob website” to get a view of test progress over time
  - Performance testing
  - Integrates with Sun Grid Engine for parallel testing (and LSF)
  - Various “data mining” tools for automatic log analysis and interpretation
- “log4x” tools are a good idea for the actual logging.

JEPPESEN
Making Every Mission Possible

xUseCase in practice?

- Two implementations of this approach exist on SourceForge, JUseCase and PyUseCase.
- JUseCase tests Java/Swing GUIs and PyUseCase tests Python/PyGTK GUIs. Both are free and open source and work in very similar ways.
- Requires some developer effort to create the GUI/domain language mapping.
- Pure simulators, i.e. they don't have any means to write assertions about the application behaviour.
- Not “complete”: they support the widgets that users of them have needed.
- Have in practice been used together with TextTest (but not operationally dependent on it).

JEPPESEN
Making Every Mission Possible

xUseCase in practice, Part 2

- I work daily with a testsuite that uses TextTest and PyUseCase.
- It has existed for about four years and contains about 900 tests.
- New tests created by recording unless they're very similar to other tests.
- Use-case scripts generally maintained via a normal text editor
- When the GUI changes, I re-record one test to get a feel for the new process, and update the rest via multi-file search-and-replace.
- Project runs "continuous release": i.e. checked in code goes directly into production. There are no unit tests.
- I know of one other project using PyUseCase, and three using JUseCase. All have less than 50 tests.
- TextTest alone is much more widely used.

JEPPESEN
Making Every Mission Possible

Conclusions

- Non-technical people cannot really have confidence in tests that they can't observe.
- Testing via the UI has been plagued with maintenance issues, but there are ways to mitigate them.
- Set up a mapping from GUI controls to a "domain language" that can describe the interaction with your system.
- Instrumenting the code simplifies this process and provides a neat way to handle synchronisation.
- Keeping track of behaviour by monitoring logged output separates concerns neatly and has many other advantages.
- xUseCase tools can help you if you use Swing or PyGTK. The same concept should be widely applicable and such tools aren't very hard to write.

JEPPESEN
Making Every Mission Possible