GUI Testing

Conrad Hughes
School of Informatics

Slides thanks to Stuart Anderson

GUI Testing: Overview

- GUI testing as an example of the use of regression testing.
- Automating GUI testing.
- The pitfalls of automating GUI Testing
- Approaches to the architecture of GUI testing
- Problems in GUI Test automation (Kaner)
- Summary

GUI Testing

- Most systems have a large GUI component. The GUI code is dense, often sensitive to small environment changes, difficult to test. But many bugs are present in GUI code.
- Common approach is to attempt to automate GUI testing:
  - One-off activity (Human):
    - Analyse product and design and write test code.
    - Run the test
    - May require debugging (both product and test code) and iteration to establish correct behaviour.
    - Capture and store GUI output
    - Package test code and results along with documentation.
  - Many-off activity (Machine):
    - Run the test code and capture the output.
    - Compare the output with the canned output.
    - If it matches OK, otherwise get someone to look at the output.
  - Many-off activity (Human): inspect erroneous output, maintain test.

Semi-automated GUI Testing Issues

- Creating code for automated tests is more expensive than normal testing.
- Maintenance costs are high (because code needs modifying).
- Often interfaces can be quite dynamic – but this approach tends to encourage no change to the interface.
- Depending on costly automated tests can make test regime too rigid.
- Regression test benefits come late (when the system is stable) but costs can come early and maintenance through early versions is costly.
- Need to take an engineering approach to the design and maintenance of GUI regression tests.

What's the Problem with Capture/Replay?

- Localisation:
  - Language
  - Local formats (e.g. UK/US Date format)
- Personalisation
  - Colour schemes, pictures, fonts, size and position of pop-ups, …
  - Preferred formats e.g. page a day diary vs week on a page, what is the first day of the week…
  - Short cuts, optional dialogue steps, …
- Environment
  - What Browser/Window Manager
  - Availability of helper applications
  - Different environments can lead to different GUI behaviour for the same sequence of interface gestures.

Sources of Variability in GUIs

- Variability in input devices, libraries etc.
- Variability in output devices.
- Variations in the desired results (because of personalisation).
- UI is subject to evolution (e.g. aspects of look and feel, presentation issues).
- Finally, over a long period test tools change, scripts become obsolete, …

Data Driven GUI Test Architectures

- GUIs have to do with the user supplying inputs in order that the system can create its response.
- For some systems we can exhaustively list all the possible input attributes (e.g. a diary: date, time slot, entry, + lots of configuration attributes).
- Build a data driven tester where we envisage a test suite as a table where each row represents a test and each column is a particular input variable.
- By placing a value v in row i and column j this means that for test i we want to set attribute j to the specified value.
- For each column, j, of the table there is a script that controls the GUI so that attribute j gets set to the value in that table entry.
- The test executor, executes test i by successively setting attribute 1, 2, 3, …

GUI Test Architectures

- Once the GUI has been driven to set the input attributes we need to test whether the result is correct. This could range from:
  - Comparing the screen image with a canned version.
  - Defining a similar set of observations that should be made of the system:
    - We could define a set of output attributes, specifying values appropriately.
    - The test interpreter would then drive the system to recover those attributes and compare them with the specified values.

Benefits of Simple Data Driven Architecture

- Data driven approach provides a quick, natural way to define and review tests.
- Resilient to change in all elements of the system.
- Testers can focus on tests **not** coding
- Provides a good platform for regression testing.
- Is applicable to a wide range of products.
- Requires simple, reusable coding to automate a large number of tests.

Deficiencies of Simple Data Driven Architecture

- Doesn't really provide a model of test (or does it?)
- Is there an explicit notion of coverage? (n – but it might be definable).
- No provision for working with state (e.g. databases, sequentiality in the interface, modes poorly supported).
- It presumes we can enumerate all attributes in the interface.
- Handles dependency between attributes poorly.

GUI Test Automation Readings

- Chris Agruss, Automating Software Installation Testing
- Tom Arnold, Visual Test 6 Bible
- James Bach, Test Automation Snake Oil
- Hans Buwalda, Testing Using Action Words
- Hans Buwalda, Automated testing with Action Words: Abandoning Record & Playback
- Elisabeth Hendrickson, The Difference between Test Automation Failure and Success
- Mark Fewster & Dorothy Graham, Software Test Automation
- Linda Hayes, The Automated Testing Handbook
- Doug Hoffman, Test Automation course notes
- Cem Kaner, Avoiding Shelfware: A Manager's View of Automated GUI Testing
- Cem Kaner, Architectures of Test Automation
- John Kent, Advanced Automated Testing Architectures
- Bret Pettichord, Success with Test Automation
- Bret Pettichord, Seven Steps to Test Automation Success
- Keith Zambelich, Totally Data-Driven Automated Testing

Common mistakes in GUI test automation (Kaner)

1. Don't write simplistic test cases.
2. Don't make the code machine-specific.
3. Don't automate bad tests.
4. Don't create test scripts that won't be easy to maintain over the long term.
5. Avoid complex logic in your test scripts.
6. Don't mix test generation and test execution.
7. Don't deal unthinkingly with ancestral code.
8. Don't forget to retire outdated or redundant regression tests.

Common mistakes in GUI test automation

9. Don't spend so much time and effort on regression testing.
10. Don't stop asking what bugs you aren't finding while you automate tests.
11. Don't use capture/replay to create tests.
12. Don't write isolated scripts in your spare time.
13. Don't assume your test tool's code is reliable or unlikely to change.
14. Don't put up with bugs and bad support for the test tool.
15. Don't "forget" to document your work.
16. Don't fail to treat this as a genuine programming project.

Common mistakes in GUI test automation

17. Don't insist that all your testers (or all the testers you consider skilled) be programmers.
18. Don't give the high-skill work to outsiders.
19. Don't underestimate the need for staff training.
20. Don't use automators who don't understand testing (or use them cautiously).
21. Don't use automators who don't respect testing.
22. Don't mandate "100% automation."

Common mistakes in GUI test automation

23. Don't underestimate the cost of automation.
24. Don't estimate the value of a test in terms of how often you run it.
25. Don't equate manual and automated testing.
26. Don't expect to be more productive over the short term.
27. Don't put off finding bugs in order to write test cases.
28. Don't expect to find most of your bugs with regression tests.
29. Don't forget to clear up the fantasies that have been spoon-fed to your management.

Summary

- Regression automation an be expensive: there are serious costs associated with the evolution of all the software elements.
- We are doing computer-assisted testing, not full automation so there are possibly significant human costs associated with false error reporting.
- Regression is just one target of (partial) automation. You can create and run new tests instead of reusing old tests but with an established product you probably want to know you haven't lost anything you care about
- Developing programmed tests is software development and many scripting environments encourage tight linkage to the UI framework.
- Maintainability is difficult to attain because GUIs are quite specific to their environment.
- The balance between writing new tests and rerunning old tests is very dependent on the type of product, time in the product lifecycle, type of product.