BIO
PRESENTATION

W3

5/16/2007 11:30:00 AM

"BEHAVIOR PATTERNS FOR DESIGNING AUTOMATED TESTS"

Jamie Mitchell
Test & Automation Consulting LLC

International Conference On
Software Test Analysis
And Review
May 14-18, 2007
Orlando, FL USA

Jamie Mitchell CTAL, CSTE

Jamie Mitchell brings over 25 years of testing experience, both hardware and software, to his consulting company.

Mr. Mitchell is a pioneer in the test automation field. He has been working with a variety of test automation tools since the first Windows tools were released with Windows 3.0. He has written test tools for several platforms, including Windows, AIX, and AS/400. His company recently released T-WorMS (Testing – Workflow Management Suite), a modestly priced set of tools that help organizations take control of their testing processes, integrating their test planning, design, execution, artifacts, results, and metrics into one easy to manage entity.

In his role as Principal Consultant at Jamie Mitchell Consulting Inc., Jamie Mitchell is responsible for developing and implementing test automation initiatives for a range of clients. In addition, Mr. Mitchell also provides training, mentoring, and expert technical support.

Jamie is the former Lead Automation Engineer for American Express Distributed Integration Test (lab) / Worldwide, and has successfully architected and implemented test automation projects for many top companies including American Express, Mayo Clinic, IBM AS/400 division, ShowCase Corporation and others.

Mr. Mitchell holds a Master of Computer Science degree from Lehigh University in Bethlehem, Pa. He holds the Certified Software Test Engineer certification from QAI and is an ISTQB Certified Tester (Advanced Level.) He is a charter member of the Austin Workshop on Test Automation, and a regular speaker at several international conferences, including SQE, QAI and PSQT.

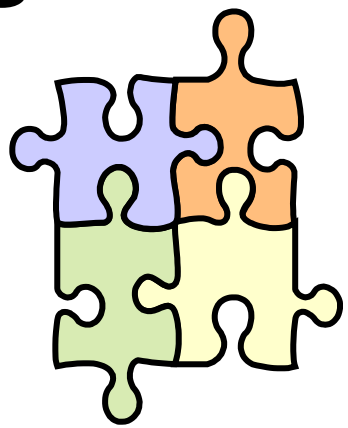
Principal Consultant @ Jamie Mitchell Consulting Inc.
jamie@go-tac.com
612-801-5285

Behavior Patterns for Designing Automated Tests

Jamie Mitchell

Jamie Mitchell Consulting Inc.

The Value of Patterns



- Humans tend to repeat things
  - Assumptions
  - Designs
  - Ssshhhhh: to put it softly:

<<<<<<<<<< Mistakes >>>>>>>>>>

From Wikipedia:

Design patterns provide general solutions, documented in a format that doesn't require specifics tied to a particular problem

On Patterns

- After automating for the last 15 years, I have seen many similar patterns
- **Failure patterns**: how to guarantee that your automation project will be a giant sucking black hole to resources
- **Success patterns**: how to give your project at least an even chance of generating positive ROI

Caveat

- You can do everything right in automation and still fail
- Lots of good projects fail due to outside influences
- Understanding the following patterns of behavior should give you the best chance of success…

First the Bad News

Failure Patterns

A Man's Got to Know His Limitations!

- Automation that is not focused will fail
  - Start out focused on a small test subset
  - Look for your critical testing problems
- Human testers are not going away
  - Most bugs will still be found by humans
  - Many (most?) bugs are found indirectly by running a test looking for something else

Do not try to automate everything

Automation Has Limitations

The standard lifecycle of a script in an immature organization

- The more we spend on automation, the more testing we want
- The more we want out of it, the more we cram into each test
- The more stuff the test evaluates, the more complex the test
- The more complex the test, the more likely it will fail
- The more it fails, the more the test will be “dumbed down”
- The dumber the test, the less value it has
- Repeat until test is worthless

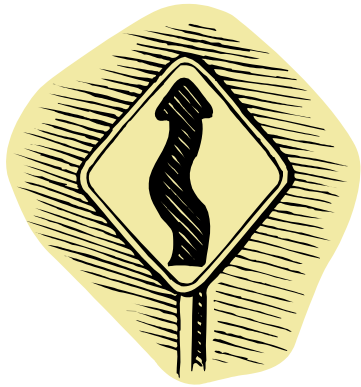
Test design must be focused on Need vs. ROI

Scripts Fail

- Even good automated tests fail
  - From extraneous events, system changes, bad synchronization
- Tests that cry wolf too often tend to get ignored when they fail
- Tests tend to get commented out until we “have some spare time to fix them”
- We never have spare time!

Entropy Exists

Dealing with Entropy



- Aggressively fix automation bugs – NOW!
- Include sanity checking in each test
- Constantly error check throughout test
  - A test with 20 steps has at least 21 failure points
- Recover after failure (go to next test)
- Leverage good engineering [programming] principles
- The most important test you will ever run:

The Next One!

Good Automation – Bad Testing

- A test case must have expected results.
- Many automators deem a test successful if it survives the run.
- Automated tests must survive the run.
- However, that is not enough!

Make sure you do GOOD TESTING!

Beware the Bitmaps!

- Determining results has always been the number 1 problem in automation
- Many tools recommend that you compare bitmaps
- Because of failure rates of these compares, some tools make them fuzzy…

This is often horrible testing!

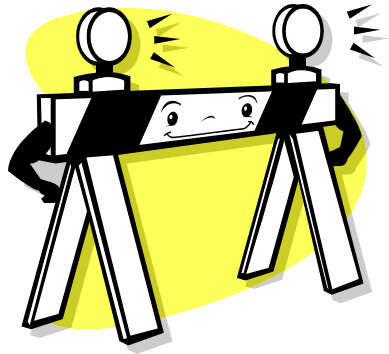
Imagine your baby...

<1% Difference: Same Picture?

Context **is** the key

Basic Warnings on Failure Patterns

- Good processes using fewer, less expensive tools get you much further than expensive tools and poor processes
- You are investing for the long haul – short term thinking will kill you
- Scalability is the holy grail
- The number 1 killer of automation:



Unrealistic Expectations!

It’s Not All Bad News

Success Patterns

Automation Might Not Be All Testing

- There are many repetitive tasks that automation tools can help with
  - Building data sets
  - Setting up environments
  - Comparing result sets
  - Singleton tests
    - EG. Troubleshooting a 3AM network problem

Leverage your tools everywhere!

Test at the Right Level

- Many applications are hierarchical
- Use GUI tools to test at the GUI level
- Build API automation to test at the business object level
  - Much of the system functionality may be completely divorced from the GUI
  - Business object interface tends to change much slower than GUI interface

Different Objectives, Different Tests,

Different Tools

Understand Manual Testing

- Manual testing has worked for the last 60 years.
- By understanding how manual testing works, we can design automation for success
- The following slides will discuss GUI type testing – although the principles also apply to many other types of tests

The Manual Test Pattern

- When reduced to bare minimum, a test case often can be documented in 3 columns:

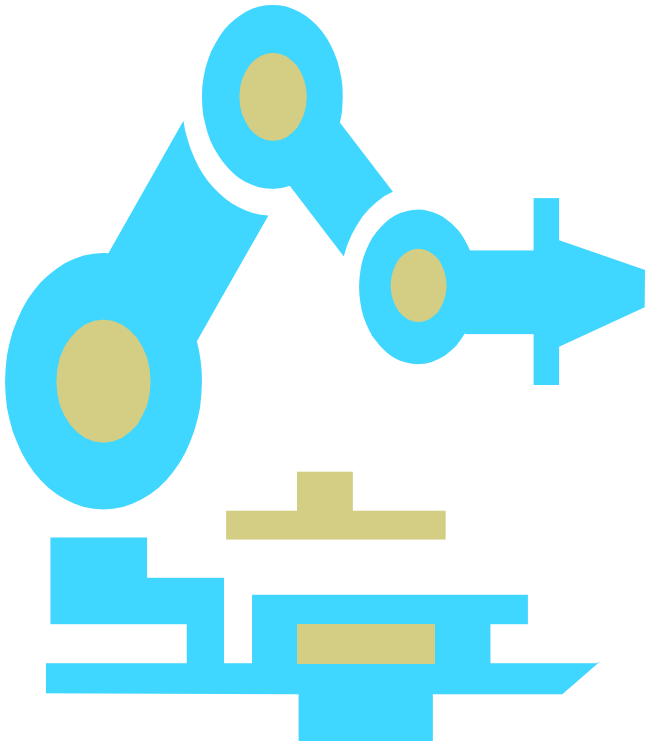
1. Abstract task to be performed
2. The data to do it with
3. The expected result from the step

A [partial] Reasonable Test

Start Application	MyWordProc	App starts and inits to main screen empty
Paste in long text with no line breaks	Get text from file g:\Test Data\LongStringTest D453.Txt	Expect format to occur correctly
Add text into middle of file	Random text including HTML tags	Allowed
Save file as HTML	C:\temp\Test D453.htm	Expect warning message that no HTML header was found
Etc.		

Points to Note

- Abstract Tasks
  - Slow evolution of abstract functions
- Scalability and reuse
- This test is incomplete
  - Needs the human tester to add
    - Reasonableness
    - Context



In automation, we add context and reasonableness though programming!

4 Step Dance Pattern

- When dealing with any GUI, humans tend to perform the same 4 steps repeatedly
- Understanding this common dance can frame your automation tasks
- Use to add reasonableness and context through programming
- Allows us to more closely simulate the human interaction with the system

4-Step Fandango

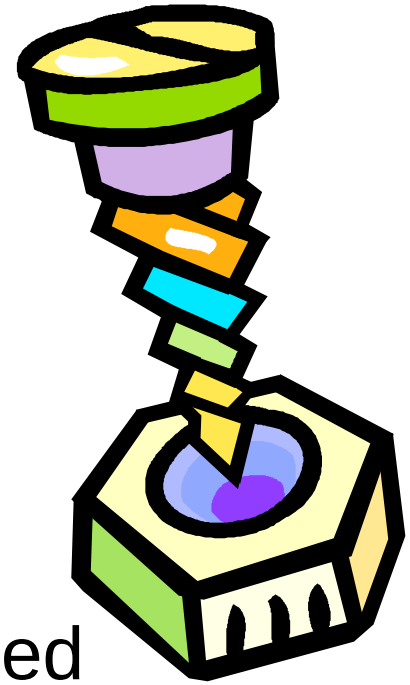
1. Make sure you are in the right place to perform the abstract task
2. Fill in the interface (instruct the system on what you want it to do)
3. Trigger the action
4. Wait for completion – ensure completion was as expected

Step 1 – Right Place?

- Are we in the right place?
- Are we getting to the right place?
- Is the right place ready for us?
  - In focus?
  - No pop-ups?
  - Enabled?

Humans do this without thinking!

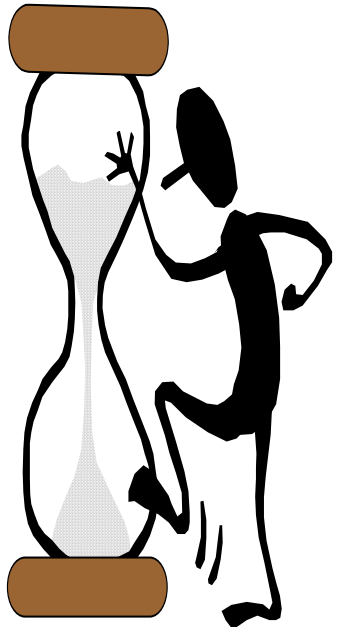
Location Sanity Check



- We must be able to determine
  - When a window is created
  - When a window becomes enabled
  - When a window becomes re-enabled
  - When a window goes away
- Must be able to wait for a finite time
- Must be able to check for an arbitrary number of different windows

Bart Simpson Has the Answer

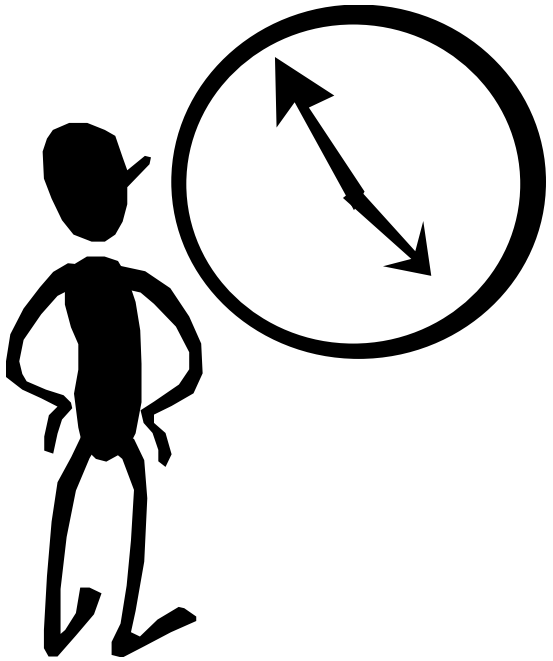
Are we there yet? Are we there yet? Are we there yet? Are we there yet?
Are we there yet? Are we there yet? Are we there yet? Are we there yet?
Are we there yet? Are we there yet? Are we there yet? Are we there yet?
Are we there yet? Are we there yet? Are we there yet? Are we there yet?
Are we there yet? Are we there yet? Are we there yet? Are we there yet?
Are we there yet? Are we there yet? Are we there yet? Are we there yet?
Are we there yet? Are we there yet? Are we there yet? Are we there yet?
Are we there yet? Are we there yet? Are we there yet? Are we there yet?
Are we there yet? Are we there yet? Are we there yet? Are we there yet?
Are we there yet? Are we there yet? Are we there yet? Are we there yet?



WaitForWin(Win, Seconds, isEnabled);

WaitFor…()

- You can wait for almost anything:
  - WaitForWinToClose()
  - WaitForObjEnabled()
  - WaitForStatusBarMsg()
  - WaitForAppletToStabilize()
  - WaitForStableBitmap()
  - WaitForTextToChange()



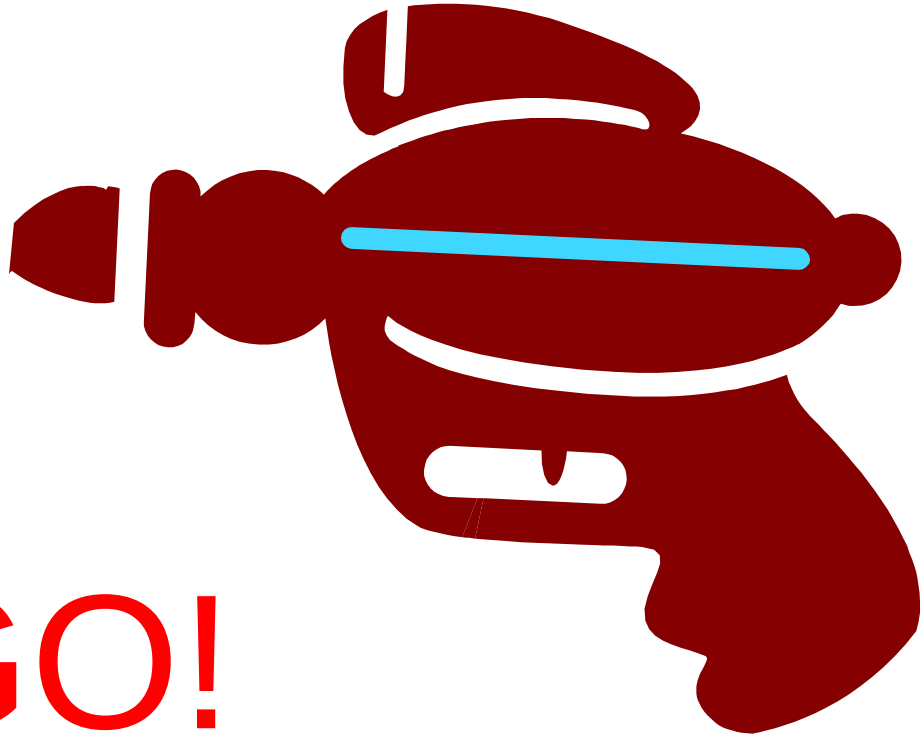
Step 2: Fill in the Interface

- Piece of cake
  - Every tool, no matter how complex or simple, can interact with controls
  - Set values based on test data
  - Even low level actions are fairly simple:
    - Click on object, send type message

Get’er Done

Step 3: Trigger It

- More cake
  - Press Next button
  - Press OK
  - Click on menu item



GO!

Step 4: Wait For Completion

- Did it finish correctly?
- Did it complete in a timely manner?
- Did we get an error message?
- Did we get a warning message?
- Did it go to never-never land?
- This is the most complex problem in automation

What happened?

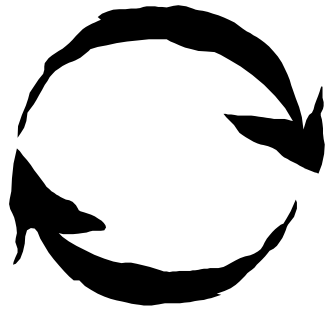
Waiting For Multiple Windows

- Problem: most tools are “single-threaded”
  - Reality: several possible outcomes to any action
- Need to cycle through all known windows quickly; repeat as necessary

```
While
    If (WaitForWin(WinA, 0, TRUE) {  }
    If (WaitForWin(WinB, 0, TRUE) {  }
    If (WaitForWin(WinC, 0, FALSE) {  }
End While
```

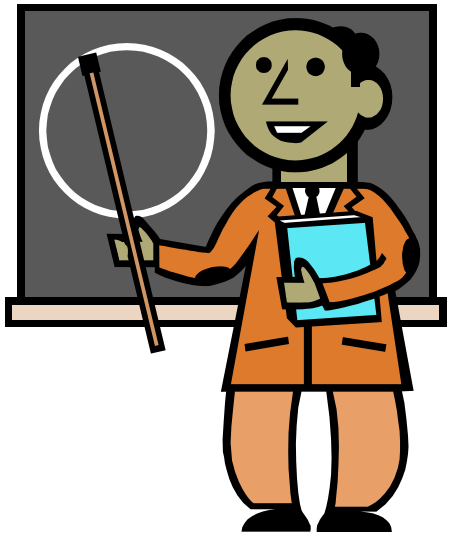
Single Action While Loop

- Perform action to leave current state (step 3)
- Loop until get to expected state
- Three ways to leave loop…
  - Get to final state
  - Get known error which cannot be handled
  - Time out (unknown state)
- Each time through the loop
  - Only 1(or 0) action is handled
  - No order assumption is made
  - Counter prevents infinite loop



Learning Architecture

- Iterative approach to programming
- Build abstract functions first, refine later
- SAWL gives us a way to
  - Add new code without breaking old
  - Revise portions of the code
  - Quick fix when broken
  - Time out if unexpected happens
  - Log exactly what happened



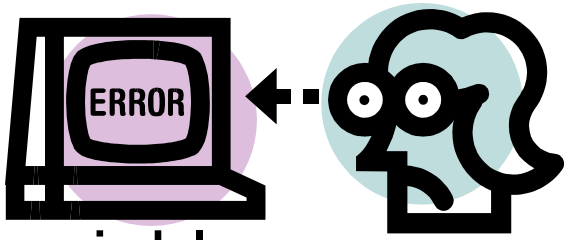
Expected Behavior In SAWL

- Negative testing can be added to loop
- 5 conditions Must be considered
  - Expecting no error, get no error
  - Expecting no error, get error
  - Expecting error, get expected error
  - Expecting error, get different error
  - Expecting error, get no error

- Assumption: Test case over immediately when error box pops

Expected Behavior

- Add single global variable to script (ExpErr)
- Place expected error string in it
  - If no error expected, pass in NULL string
- On error, match expected with actual
  - If match, negative test passes
  - If no match, test fails
- At end of test, if no error, check variable



Conclusion

- Automation is…
  - More complex than you think
  - Really really expensive
  - A long term investment
  - May return great benefit when done right
- Watch out for the patterns – they can lead you to success

Contact me:

Jamie Mitchell

Jamie Mitchell Consulting Inc

jamie@go-tac.com