About Doug Hoffman

I am a management consultant in testing/QA strategy and tactics. I help plan quality strategies and tactical approaches for organizations, especially esoteric test automation. I gravitated into quality assurance from engineering. I've been a production engineer, developer, support engineer, tester, writer, instructor, and I've managed manufacturing quality assurance, software quality assurance, technical support, software development, and documentation. My work has cut across the industry from start-ups to multi-trillion dollar organizations. Along the way I have learned a great deal about software testing and automation. I enjoy sharing what I've learned with interested people.

Current employment

- President of Software Quality Methods, LLC. (SQM)
- Management consultant in strategic and tactical planning for software quality

Education

- B.A. in Computer Science
- MS in Electrical Engineering, (Digital Design and Information Science)
- MBA

Professional

- President, Association for Software Testing
- Past Chair, Silicon Valley Section, American Society for Quality (ASQ)
- Founding Member and Past Chair, Santa Clara Valley Software Quality Association (SSQA)
- Certified in Software Quality Engineering (ASQ-CSQE, 1995)
- Certified Quality Manager (ASQ-CMQ/OE, 2003)
- Participant in the Los Altos Workshop on Software Testing and dozens of other offshoots

Douglas Hoffman 2

Today's Topics

- This Session (Part 1)
  - About Automated Testing
  - Non-Regression Automation
- Next Session (Part 2)
  - Test oracles and automation

Automation Background

Opportunities For Automation

- Program analysis
- Test design
- Test case management/selection
- Input selection/generation
- Automated test case
- Test execution control
- Actual results capture
- Expected results generation
- Results comparison
- Report generation

Automation Defined For This Class

Get the computer to do one or more of:

- Test design
- Input selection/generation
- Automated test case
- Test execution control
- Actual results capture
- Expected results generation
- Results comparison

Automated vs. Manual Tests

An automated test is not equivalent to the most similar manual test:

- Automated comparison is typically more precise (and may be tripped by irrelevant discrepancies)
- Skilled human comparison samples a wider range of dimensions, noting oddities that one wouldn't program the computer to detect
- Automated input is consistent, while humans cannot closely replicate their test activities

Automation Narrows Our Scope

An automated test is more limited than a manual test:

- The test exercise must be automated in advance
- People can integrate outside experience
- People may gain insights
- We can only check machine available results
- We only check result values that we pre-specify
- Cannot easily back up when something unexpected happens

Typical Automated Tests

- Are based on the functions of a test tool
- Automate a manual tester's actions
- Are a list of test activities (script)
- Work at the UI level
- Does program checking at specified points in the script
- Are used to repeat and speed up manual testing

Regression Test or Demo?

- Think of an interesting test
- Map out steps to perform the test
- Run the steps manually
  - if there is a bug; report and wait until fixed
  - if no bugs, capture the steps
- Rerun the steps to repeat the test

The goal is to find bugs

- Think of an interesting demo
- Map out steps to perform the demo
- Run the steps manually
  - if there is a bug; report and wait until fixed
  - if no bugs, capture the steps
- Rerun the steps to repeat the demo

The goal is to AVOID bugs

Valuable Regression Automation

- Build/smoke tests
- ***Required*** repeatability
- Rerun many times
- Demo script
- Comfort managers

Questions We Should Ask About Testing and Automation

- Should we limit our thinking to what a tool does?
- Should we focus automation on things we can do manually and script?
- Should we limit ourselves to UIs or APIs?
- Are we checking everything that's important?
- Do speedy manual tests find more or different bugs than manually running tests?
- Can inefficient or approximate tests be valuable?
- Must tests do the same things every time?

Exploratory Automated Tests

Exploratory Automation?

- Enables and amplifies tester's abilities
- Does something new every time
- May use massive numbers of iterations
- May use multiple parallel oracles
- Can find bugs we never imagined

Enable the Exploratory Tester

- Enables manual tester to do more things
- Enables manual tester to work faster
- Amplifies manual tester's abilities
- Can look under the covers
- Real time monitoring of the system

Randomness and Tests

- Random number generators
  - Pseudo-Random numbers
  - Generating random seeds
  - Repeatable by entering seed value
- Randomized input values
- Randomized data generation

Repeatable Random Series

```
# RUBY code
MAX_SEED = 1_000_000_000
 def initial_RNG_seed(myseed)
    if (myseed == nil) # Check if seed is provided
       # Create a random number to seed RNG
       puts "(no seed passed in, so generate one)"
       myseed = srand()
       myseed = rand(MAX_SEED)
      end
    # print the seed so that we know the seed used
    puts "myseed is #{myseed.to_s}\n"
    foo2 = srand (myseed) # initialize the RNG
    foo = rand()    # generate the [first] random number
    return foo
end
```


Random Series Output Example

```
puts ("First run: #{initial_RNG_seed(nil)} \n \n")
puts ("Second run: #{initial_RNG_seed(400)} \n \n")
puts ("Third run: #{initial_RNG_seed(nil)} \n")
```

→

(no seed passed in, so generate one)
myseed is 144288918
First run: 0.3705579466087263

myseed is 400
Second run: 0.6687289088341747

(no seed passed in, so generate one)
myseed is 108495905
Third run: 0.09838898989988143

RandSeed

Advantages of Exploratory Automation

- Does things a manual tester cannot do
- Does something new every time
- May use massive numbers of iterations
- May feed inputs directly to SUT
- Oracles may check internal information
- May have multiple parallel oracles
- Supplements baseline tests
- Can uncover obscure bugs
- Can uncover bugs impossible to find manually

Examples of Exploratory Automation

- Cem Kaner's "Telenova" example using random events
- Statistical packet profiles for data link testing
- Using "Dumb monkeys" and MS Word
- "Sandboxed" random regression tests
- 1/3 or 3x heuristic in test harness
- Periodic database unload/check
- Database links
- Database locking (single and multi-threaded)
- Device front panel state machine long walks
- Database load/unload dropouts
- Database unbalanced splitting
- Random machine instruction generation

Running Randomly Selected Regression Tests

```
# Ruby
threads = []
list = []
list = Dir.entries ("./executables")
list3 = list.reject   { |s| s =~ /^\./ }
TestCount = list3.length
for y in (1..10)
  name =  list3[rand(TestCount)]
  threads << Thread.new(name) do |x|
    print "running \n"
    print x.to_s
# use ruby for running all types of scripts and programs
    system ('ruby ./executables/' + x)
  end
  threads.each {|thr| thr.join}
end
puts ""
puts "Exiting"
#puts rand(TestCount)
#puts list3[rand(TestCount)]
```

Douglas Hoffman Copyright © 2012, SQM, LLC. 21

Well-Formed Input

```
#RUBY program that generates simple arithmetic phrases
def eq_gen
    jubilee = 1 + rand(8) # 8 choices for the 4 defined cases
    case jubilee
      when 1
         "(" << eq_gen() << ") + (" << eq_gen() << ")"
      when 2
         "(" << eq_gen() << ") - (" << eq_gen() << ")"
      when 3
         "(" << eq_gen() << ") * (" << eq_gen() << ")"
      when 4
         "(" << eq_gen() << ") / (" << eq_gen() << ")"
      else  # generate number half the time
         rand(100).to_s
    end
end
```

Douglas Hoffman Copyright © 2012, SQM, LLC. 22

Example of Well-Formed Input

```
puts eq_gen.to_s
puts eq_gen.to_s
puts eq_gen.to_s
puts eq_gen.to_s

→

(77) - ((62) / (6))

((10) - (40)) + (67)

53

(62) - ((96) * ((((77) - (72)) - ((7) * ((47) - (91)))) /
((34) + (((70) - (18)) + (4)))))
```

Douglas Hoffman Copyright © 2012, SQM, LLC. 23

Disadvantages of Exploratory Automation

- May not be repeatable (even with seeds)
- Difficulty of capturing program and system information for diagnosis
- May use multiple real-time oracles
- Coordination of autonomous oracles with the test
- Does not provide rigorous coverage
- Can uncover bugs that can't be fixed

Douglas Hoffman Copyright © 2004-12, SQM, LLC. 24

A Model of Test Execution

Oracles and Test Automation

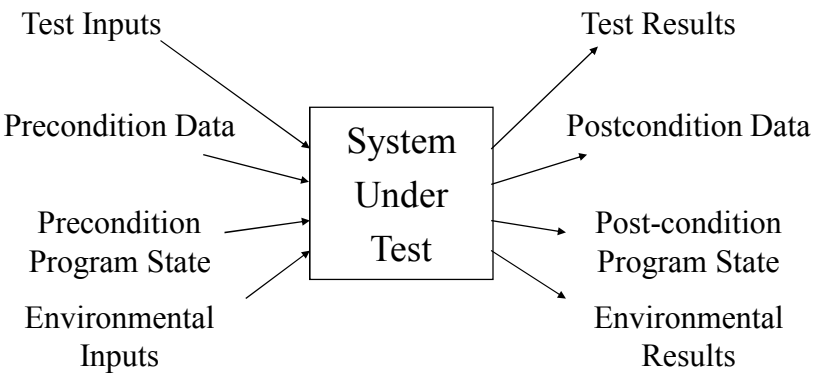
Good automated testing depends on our ability to programmatically detect whether the SUT behaves in expected or unexpected ways

Our ability to automate testing is fundamentally constrained by our ability to create and use oracles.

The Oracle

- The principle or mechanism for telling whether the SUT behavior appears OK or if further investigation is required
- Answers the question "is this expected behavior?"
- A fundamental part of every test execution
- Expected result is NOT required in advance
- Multiple oracles can be used for a test

Expanded Software Testing Model



Implications of the Expanded Model

- The test exercise is the easy part
- We don't control all inputs
- We can't check everything
- Multiple domains are involved
- We don't know all the factors

Important Factors in Outcome Comparison

- Which outcomes do we check?
- How do we know what to expect?
- Do we know how to capture it?
- Can we store the results?
- What differences matter?
- Are "fuzzy" comparisons needed?
- When do we compare outcomes?

A Preview of Part 2 (Oracles)

The Test Oracle

- Two slightly different views on the application of the word
  - ***Reference Function:*** **You ask the oracle what the "correct" answer is**
  - ***Reference and Evaluation Function:*** **You ask it whether the program behavior is abnormal**
- Using an oracle, you can compare the program's behavior to a reference (predicted behavior) and decide whether the program passed the test.
  - ***Deterministic oracle*** **(mismatch means abnormal)**
  - ***Probabilistic oracle*** **(means probably abnormal)**

Types of Test Oracles

- None
- Independent implementation
- Consistency
  - Saved master
  - Function equivalence
- Self-Verifying
- Model based
- Constraint based
- Probabilistic
- Statistical
- Property based
- Computational
- Diagnostic
- Hand-crafted
- Human

Today's Topics

- Last Session (Part 1)
  - About Automated Testing
  - Non-Regression Automation
- This Session (Part 2)
  - Test oracles and automation

Recap of Part 1

- Automated regression tests serve a purpose, but exploratory automation may have a higher ROI
- Automated tests can explore (not just repeat)
- Extend your reach into the system by doing things a human cannot do
- Oracles are key to good automated testing

The Test Oracle

- Two slightly different views on the application of the word
  - ***Reference Function:*** **You ask the oracle what the "correct" answer is**
  - ***Reference and Evaluation Function:*** **You ask it whether the program behavior is abnormal**
- Using an oracle, you can compare the program's behavior to a reference (predicted behavior) and decide whether the program passed the test.
  - ***Deterministic oracle*** **(mismatch means abnormal)**
  - ***Probabilistic oracle*** **(means probably abnormal)**

A Caveat About Test Results

- *It is important to recognize that evaluations are heuristic – reported "passes" and "fails" are only guesses*
  - **We can have false alarms:** A mismatch between actual and expected behavior might not matter. We must investigate when a test "fails" to know if there is something to report.
  - **We can miss defects:** A match between actual and expected behavior might result from the same error in both, a limitation or error in the test, not checking the right things, or incomplete coverage. These are "silent misses" and we don't ever know to investigate.

Reference Function

- Under this view, you compare your results to those obtained from the oracle using the same inputs
- The oracle is the method of generating the expected results
- Comparison and evaluation may be handled separately
- Check for equality between actual and expected results
  - If the results match, there is nothing interesting ("Pass")
  - If they don't match, further investigation is required ("Fail")

Reference And Evaluation Function

- Under this view, the oracle accepts both the inputs and results
- The oracle checks whether the results could be consistent with the inputs
- Comparison and evaluation are handled at once
- It checks the expected results for the possibility or likelihood they are wrong given the inputs
  - "Pass" if results are possible or likely
  - Further investigation is required if the results are not possible or unlikely ("Fail")

Evaluation Function

- Under this view, the oracle only looks at results
- The oracle checks whether the results are consistent with some conditions or rules
- May be real-time or post-execution
- Some examples:
  - Memory leak tests
  - Database consistency check
  - Statistical profiling
  - Self-Verifying data

Where To Get Expected Results

- Previous version
- Competitive product
- Model
  - System or subsystem
  - Heuristics (assertions, fuzzing, limits, etc.)
- Invariants
- Behavioral characteristics
- Secondary characteristics
- Embedded data

Which Results To Check

- Expected results
- Anticipated likely errors
- Major environmental factors
- Invariants
- Available easy oracles

Checking Results

- Exact comparison
  - Values
  - Ranges
  - Behaviors
  - Key words/values
- Heuristic comparison
  - Similarity
  - Algorithm based
  - Secondary characteristics
- Check during or after the test run

Software Test Automation:

Types of Test Oracles

Notes On Test Oracles

- At least one type is used in every test
- Implemented oracles may have some characteristics of multiple types of oracles
- Multiple oracles may be used for one test
- Oracles may be independent or integrated into a test
- The oracles are key to good automation

Types of Test Oracles

- None
- Independent implementation
- Consistency
  - Saved master
  - Function equivalence
- Self-Verifying
- Model based
- Constraint based
- Probabilistic
- Property based
- Computational
- Diagnostic
- Hand-crafted
- Human

Douglas Hoffman 47

Oracle Taxonomy (1 of 4)

	Definition	Advantages	Disadvantages
No Oracle	- Doesn't check correctness of results, (only that some results were produced)	- Can run any amount of data (limited only by the time the SUT takes)	- Only spectacular failures are noticed
Independent Implementation	- Independent generation of all expected results	- No encountered errors go undetected - Can be used for many different tests -Fast way to automate using an oracle (if available)	- Expensive to implement and maintain - Complex and often time-consuming when run
Consistency (Saved Master)	- Verifies current run results with a previous run	- Verification is straightforward - Can generate and verify large amounts of data	-Original run may include undetected errors -Can take large data storage space
Consistency (Function Equivalence)	- Verifies current run with another implementation	- Verification is straightforward in real time - Can generate and verify large amounts of data without storage	- Possibility of undetected common-mode errors

Douglas Hoffman 48

Oracle Taxonomy (2 of 4)

	Definition	Advantages	Disadvantages
Self-Verifying	- Embeds key to the answer within data	-Can check large amounts of data - Allows extensive run-time and post-test analysis - Straightforward to check - Verification can be based solely on message contents	- Test must require input data set with compliant structure - Must define answers and generate messages to contain them
Model Based	- Uses digital data model of SUT behavior	- May use digital model for multiple tests - Digital form of model easier to maintain than automated test - Tests may work for multiple SUTs by using different models	- Maintenance of complex SUT models is expensive - Model must match expected behavior
Constraint Based	- Verifies characteristics in compliance with constraints	- Faster and easier than most other oracles - Less expensive to create and use - May be reusable across SUTs and tests	- Can miss systematic errors - Can miss obvious errors

Douglas Hoffman Copyright © 2004-12, SQM, LLC. 49

Oracle Taxonomy (3 of 4)

	Definition	Advantages	Disadvantages
Probabilistic	- Checks a highly likely characteristic	- Faster and easier than most other oracles - Less expensive to create and use	- Can cause false alarms - Can miss systematic errors - Can miss obvious errors
Statistical	- Uses statistical properties of outcomes	- Allows checking of very large data sets - Allows checking of live systems' data - Sometimes allows post-test checking	- Can cause false alarms - May miss systematic errors - Can miss obvious errors
Property Based	- Checks an indirectly correlated characteristic	- Usually simple - Good for straightforward transformations	- Correlated variable may not exist - May be difficult to exploit the correlation

Douglas Hoffman Copyright © 2004-12, SQM, LLC. 50

Oracle Taxonomy (4 of 4)

	Definition	Advantages	Disadvantages
Computational	- Reverses the behavior of the SUT to revert results to inputs	- Good for mathematical functions - Good for straightforward transformations	- Limited applicability - May require complex programming
Diagnostic	- Programmatically checks assertions or traces activities and values	- May be built into code as defensive programming - Provides good diagnostic data when exception detected	-Can miss significant errors - Changes program characteristics - Can miss obvious errors
Hand-Crafted	- Result is carefully selected by test designer	- Useful for some very complex SUTs - Expected result can be well understood	- Does the same thing every time - Limited number of cases can be generated
Human	- Applies a person's brain power to decide correctness	- Available - Flexible (can always be applied) - Applies a broad spectrum of filters	- Slow - Error prone - Easily distracted

Douglas Hoffman 51

Software Test Automation:

Oracle Characteristics

Douglas Hoffman 52

To 127

Oracles: Challenges

- Completeness of information
- Accuracy of information
- Usability of the oracle or of its results
- Maintainability of the oracle
- Complexity when compared to the SUT
- Temporal relationships
- Costs

Oracle Completeness

- Input Coverage
- Result Coverage
- Function Coverage
- Sufficiency
- Types of errors possible
- SUT environments

There may be more than one oracle for the SUT

Inputs may affect more than one oracle

Oracle Accuracy

- How similar to SUT
  - Arithmetic accuracy
  - Statistically similar
- How independent from SUT
  - Algorithms
  - Sub-programs & libraries
  - System platform
  - Operating environment

  Close correspondence makes common mode faults more likely and reduces maintainability
- How extensive
  - The more ways in which the oracle matches the SUT, i.e. the more complex the oracle, the more errors
- Types of possible errors

Oracle Usability

- Form of information
  - Bits and bytes
  - Electronic signals
  - Hardcopy and display
- Location of information
- Data set size
- Fitness for intended use
- Availability of comparators
- Support in SUT environments

Oracle Maintainability

- COTS or custom
  - Custom oracle can become more complex than the SUT
  - More complex oracles make more errors
- Keep correspondence through SUT changes
  - Test exercises
  - Test data
  - Tools
- Ancillary support activities required

Oracle Complexity

- Correspondence with SUT
- Coverage of SUT domains and functions
- Accuracy of generated results
- Maintenance effort to keep correspondence through SUT changes
  - Test exercises
  - Test data
  - Tools
- Ancillary support activities required

Oracle Temporal Relationships

- How fast to generate results
- How fast to compare
- When is the oracle run
- When are results generated
- When are results compared

Oracle Costs

- Creation or acquisition costs
- Maintenance of oracle and comparators
- Execution cost
- Cost of comparisons
- Additional analysis of errors
- Cost of misses
- Cost of false alarms

Recapping

- Use automation to explore (not just repeat)
- Extend your reach into the system
- Oracles are key to good automated testing
- Automate based on available oracles
- Use different types of oracles