Exploratory Automated Testing

PNSQC 2013
October 14, 2013

Douglas Hoffman, BACS, MBA, MSEE,
ASQ-CSQE, ASQ-CMQ/OE, ASQ Fellow
Software Quality Methods, LLC. (SQM)
www.SoftwareQualityMethods.com
doug.hoffman@acm.org

About Doug Hoffman

I am a management consultant in testing/QA strategy and tactics. I help transform organizations.

Effective software testing requires a special skillset and mindset. The problems and solutions are unique and specific to every set of relevant factors; the context. Over several decades in the business I have learned a great deal about software testing and automation in a large range of contexts from big government and commercial companies to Silicon Valley startups; from hardware and microcode to full blown applications. 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

- Board of Directors and Past President, Association for Software Testing
- Past Chair, Silicon Valley Section, American Society for Quality (ASQ)
- Founding Member and current Chair, Santa Clara Valley Software Quality Association (SSQA)
- Certified in Software Quality Engineering (ASQ-CSQE)
- Certified Quality Manager (ASQ-CMQ/OE)
- Participant in the Los Altos Workshops on Software Testing and many others

Automation For This Discussion

Get the computer to do one or more of:

- Test data generation
- Setting-up or capturing pre-conditions
- Controlling test execution
- Selecting/generating inputs
- Running the test exercise
- Outcome monitoring
- Verdict determination:
  - Expected outcome generation
  - Actual outcome monitoring and/or capture
  - Outcome evaluation

Most Automated Tests Today

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

The Regression Test Approach

- Identify potential problems and tests necessary to expose them
- Identify and prioritize important risks
- Standardize documentation and procedures
- Design and create tests that are repeatable
- Rerun the tests often

Regression testing targets specific potential errors and repeatedly (and only) checks for those errors

Questions We Should Ask About Testing and Automation

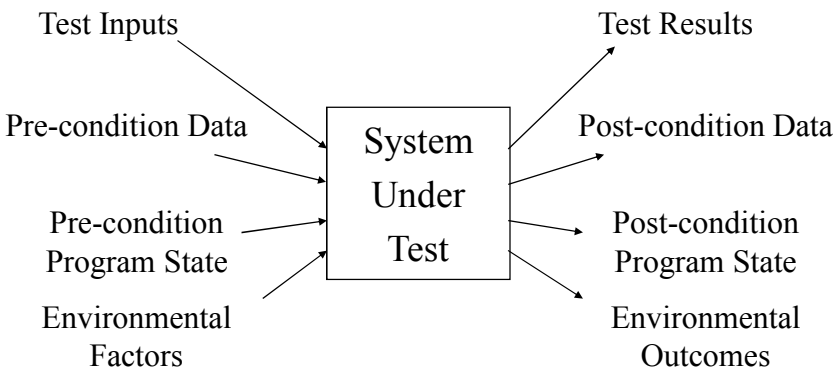
- Should we limit our thinking to what a tool does?
- Should we focus automation on things we can do manually and then script?
- Should we limit ourselves to UIs and APIs?
- Are we checking everything that's important?
- Do speedy manual tests find more or different bugs than manually running tests?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

Exploratory Test Automation

- Does something new every time (except…)
- Does things a manual tester cannot do
- Gets 'behind the scenes" or "under the covers"
- May use massive numbers of iterations
- May use multiple parallel oracles

Exploratory test automation can find bugs we never imagined and couldn't find any other way

A Test Execution Model



Some Implications of the Model

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

The Oracle Is Key For Test Automation:

- The principle or mechanism for telling whether the SUT behavior appears OK or if further investigation is warranted
- Addresses the question "is this unexpected behavior?"
- It's a basic part of every test execution

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

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

High Volume Random Tests

- Principle idea
  - High-volume testing using varied stimulus
  - Results checking based on individual results or population's statistical characteristics
- Fundamental goal is to have a huge number of iterations
  - The individual tests may not be not all that powerful or compelling
  - Stimulus is varied for each step
  - Individual results may not be checked for correctness (e.g., heuristics or population statistics)
  - The power of the approach lies in the large number of tests

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

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)]
```

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
```


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)))))
```

Low Volume Exploratory Tests

- Principle idea
  - One-at-a-time testing using varied inputs
  - Use automation to make exploration easier
- Fundamental goal is to enable exploration
  - Variations on a theme (modification of existing tests)
  - Quick-and-dirty generation of tests/data/comparisons
  - Checking in the background
    - Memory leak detection
    - File modification
    - Etc.

Examples of Exploratory Automation

- Random events (Cem Kaner's "Telenova" example)
- "Dumb monkeys" (Noel Nyman)
- "Sandboxed" random regression tests (Kaner/Hoffman)
- Single and multi-threaded database locking (Hoffman)
- Statistical packet profiles for data link testing (Hoffman)
- Random machine instruction generation (Hoffman)
- Database unbalanced splitting (Hoffman)
- Database load/unload dropouts (Hoffman)
- Database forward/backward link consistency (Hoffman)
- Device front panel state machine long walks (Hoffman)
- Periodic database unload/check (Hoffman)
- 1/3 or 3x timing difference heuristic in test harness (Oracle)

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

Disadvantages of Exploratory Automation

- May not be repeatable
- Difficulty capturing program and system information for diagnosis
- May have multiple parallel oracles
- Coordination of autonomous oracles with the test
- Does not provide rigorous coverage
- Can uncover bugs that can't be fixed

Summary

- Exploratory Automated Testing uses tests that get the computer to look for unexpected conditions
- Varies what the test does
- Opposite of regression tests
- Test oracles are fundamental in testing, especially in exploratory automated testing