High Volume Automated Testing with Yeager

Casey Doran

Florida Institute of Technology

cdoran2011@my.fit.edu

November 28, 2017

Overview

Acknowedgements

This work would not be possible without the support of:

- Cem Kaner, CSTER, and WTST participants
- Curtis Chambers, Jeff Farr, Mike DeCabia at Dycom Industries
- the Ruckus, the Harbor City Hooligans, the Samuels family
- Rob Atilho and Ryan Bomalaski, and many more on campus
- kbg, Richard Ford, actual and adopted family

Relevant URLs

- github.com/elementc/yeager
- github.com/elementc/monica-tests-traditional
- github.com/elementc/monica-tests-yeagerized
- github.com/elementc/thesis
- github.com/monicahq/monica
- monica-doran.herokuapp.com

Why Automate Testing?

- Save time
- Save money
- Test thoroughness
  - Humans miss details
  - Humans get bored or tired

How Do We Automate?

- Write functions that exercise the system under test
- Put these functions in a format that can be consumed by a test runner
- Call test runner
- Interpret test runner's output

Languages

- Test frameworks exist for many languages
- Testers prefer "easier" scripting languages like Perl, Ruby, Python
- This discussion will center around Python
  - Much can be implemented in Ruby

Frameworks

- Has a suite of assertion convenience methods
- Has logging/reporting facilities
- Has a runner
- Python: `unittest`, `nose`, `pytest`
- `unittest` is in the Python Standard Library

Glass Box Testing

- Test code interacts directly with the System Under Test's source
- Can probe very deeply into execution
- Use mock interfaces & shims to isolate tests

Black Box Testing

- Test code interacts with the user or service interface of the running program
- Use external toolkits like Selenium to drive user interfaces
- Often in a special test environment but otherwise the unmodified software

Selenium

- Programmatic control of web browsers for testing and other automation
- Driver class allows navigation and document queries
- Node class allows interaction, data retrieval, and limited Driver-like queries for children

HTML (summary)

- XML- based documents for the web
- Tree-structured
- Nodes have properties, including text, in addition to children

CSS (summary)

- Language for styling HTML documents
- Format- `selector: rule;`
- Selectors: strings that identify one, many, or none of the nodes in an HTML document
- Rules: specific styling attributes to apply to each node matched by attached rule

Monica: A Personal CRM

- Open-Source
- Life-tracker
- Friend-keeper
- Journal
- In the cloud

Contacts

Relationship Management

Journal

Page Object Modeling

- Each page on a site corresponds to a Python class.
- Fields or important strings on pages get getters and setters.
- Clickable buttons or links get click() functions.
  - If the click should transition to a new page, construct and return that new page's class.
- In class constructors, assert invariants about that page.

How Web Test Suites Come Together

- Build all the page objects and put them in /pages/.
- Write step-by-step test plan as comments in the body of a function in the runner's format.
- Translate english steps into Python code.

Running Tests

- Same as running any other Python script
- `python3 test_contacts.py`
- Some frameworks have a multi-script runner
- `python3 -m unittest`

'Bugs' That Traditional Testing Finds

- Known bugs, whether previously fixed or bugs that are defended against
- Unfinished features
  - Write the tests before you write the feature.
- Clear and obvious program faults
  - Obvious to the computer
  - Crashes, for instance
  - Nonzero return codes

What Traditional Testing Does Not Find

- Faults the tester did not think to test for
- Faults that are not obvious
- Faults the tester deems improbable

How To Find What Traditional Testing Does Not Find

- All the bugs missed are failures of imagination.
  - If a scenario can be imagined, a test can be written for it.
- Computers are really bad at imagining, too, but are passable at rolling dice.

Examples of The Bugs We Want To Find

- Digital phone system that crashes when the 22nd line is put on hold
- Flaky text editor that has been running for months on a grad student's laptop
- System that buckles when 200k users log on at the start of a workday
- Other "hard to reproduce" failures

Software Is A Finite State Machine

- Software can be represented as a machine with states, state transitions, inputs, outputs, and other tuples.
- FSM exactly describes the software's behavior
- Technique is popular in EE and for testing protocols

Testers Write Based On The System's States

- Page Object Model testing pattern emulates the system's underlying state model, and includes state transitions.
- Implied state model is significantly simplified compared to a formal FSM specification.
- POM provides a detailed look at how the system is built.

State Models Can Help Us Plan New Tests

- Given a printout of a state model, one can trace a pen along the model and plan a new test sequence.
- What parts of the SUT are tested and what parts are not yet tested becomes obvious.

Context: What Simplified State Models Don't Capture

- Input typed into the program
- Data the program read from some external source
- Overheating CPUs, cosmic rays, etc.

S

implified State Models Can Be Represented As Directed Multigraphs

- System states are vertexes, or nodes.
- Test functions are edges, connecting an in-node to an out-node.
- Each edge connects one in-node to one out-node, however
  - a given function might work as a transition to an out-node from multiple compatible in-nodes.
  - This behavior is a byproduct of convenicence features in the software under test, like having a logout button on every page.
  - For brevity's sake, treat a list of in-nodes on an edge's definition as a separate edge definition for each listed in-node.

Random Walks: Generating New Test Plans Automatically

Given one of these simplified state models represented as a graph, and a source of random numbers, automatically generating test plans is straightforward.

- For a given node, the current state, from the set of nodes
- Gather all of the edges, the transition functions, which have that state as their from-node
- Select one of these gathered functions at random and execute it
- The selected function's to-node becomes the new current state
- Repeat until some planned condition is met or execution of a selected function is not possible

What Bugs Look Like From A Modeling Perspective

- Bugs manifest as nodes which the model says should be reachable, but execution cannot successfully reach.
- Such occurrences might be bugs in the software.
- Such occurrences might be bugs in the tester's model.

Prior Art: Model Based Testing

- Jonathan Jacky, in Radiation Oncology, of the University of Washington, made an excellent Python model-based tester called PyModel.
- PyModel consumes a handcrafted model.
- PyModel can emit a test plan that covers the whole model.
- PyModel can emit a test plan that takes a random, should-be valid walk of the software under test.

Weaknesses in PyModel

- PyModel requires a handcrafted model in a finicky domain-specific language.
  - Not Plain Old Python.
- PyModel is difficult to connect to test execution.
- PyModel requires a lot of time to get running.

What Is Yeager?

- Python version 3 module
- Annotate funtions indicating that they cause a state transition.
- Infers a state model
- Can take a random walk on that model
  - Can terminate random walks under selectable conditions
- Has debug tools to understand the inferred model

Yeager's API Fits On A Notecard

- `import yeager`
- `@yeager.state_transition(from, to)`
- `yeager.walk()`
- Tweak: `yeager.add_state_to_blacklist()`, `yeager.add_transition_to_blacklist()`, `yeager.remove_state_from_blacklist()`, `yeager.remove_transition_from_blacklist()`, and `yeager.set_edge_weight()`
- Debug: `yeager.enumerate_transitions()`, `yeager.reachable_states()`, `yeager.orphaned_states()`

Write a Function

```
def login(driver):
    from pages.login import LoginPage
    lp = LoginPage(driver)
    lp.log_in_correctly(USERNAME, PASSWORD)
```

Annotate the State Transition

```
@yeager.state_transition("login", "dashboard")
def login(driver):
    from pages.login import LoginPage
    lp = LoginPage(driver)
    lp.log_in(USERNAME, PASSWORD)
```

Debug Yeager Models

- Using `enumerate_transitions` function
- Using `orphaned_states` & `reachable_states` functions

Plan A Test Run

- `yeager.walk()`
- `yeager.walk(50)`
- `yeager.walk(exit_state="state-to-exit-on")`
- In development: after some visitation goal

Run It

- `python3 yeager_test.py`

Test Monica With Yeager

- Have a robust suite of Page Object Models
- Intuitive and meaningful system
- Public service

Intuitive States of Monica

- login page
- dashboard
- contacts list
- looking at a contact
- editing a contact
- logging a phone call or meeting with a contact
- writing in the journal
- etc.

States Necessitate Transitions

- Filling in the login form transitions from the login page to the dashboard
- Clicking a contact in the contacts list transitions to the viewing-a-contact state

Use Existing Page Object Models As A Guide

- Emulates the Page Object Models' structure
- States are pages
- Methods are state transitions
  - Some transitions can be loopbacks

Write Some Glue and Go

For each method in the page object models:

- create a relatively stateless function that calls it.
- annotate any state transition that function triggers.

"Relative Statelessness"

- This will vary from tester to tester according to their gumption.
- It's reasonable for a test function to require a shared webdriver so page objects can be used.
- It might be reasonable for a test function to require a list of all the Contact names put into the system so far.
- It's unreasonable for a test function to require a memoizing key-value store with hundreds or thousands of entries.

Example Suite's Model

Give It A Run

- Execution begins with a call to `yeager.walk()`

What It Looks Like When Everything Is Good

- No crash
- No assertions being tripped
- Software appears to be being executed

What It Looks Like When The Model Is Wrong

- Crash on an illogical sequence
- Example:
  - Click "Create Contact"
  - Click "Add this Contact"
  - Expected: On Contact pages
  - Actual: On Add Contact Page with an error message about needing to input a name

What It Looks Like When The Software Is Wrong

- Crash on a perfectly logical sequence.
- Example:
  - Open a contact
  - Click "Add Reminder"
  - Fill in a date
  - Fill in a title
  - Check the "Remind me about this just once" box
  - Click the save button
  - Expected: On the contact's page, with a new reminder
  - Actual: On a 500 internal server error page
- `https://github.com/monicahq/monica/issues/326`

What Is High Volume Test Automation?

Tests that algorithmically generate, execute, and evaluate the results of arbitrarily many test actions on a system, in such volume as to:[Kaner, 2013]

1. Exceed the volume a reasonable testing staff could do manually.
2. Expose behaviors of the system not normally exposed during traditional testing techniques.
3. Simulate use and abuse of the system more realistically and dynamically than would be attainable through traditional techniques.
4. Generate test scenarios that are not outside the realm of possibility or even probability due to the high-availability nature of modern software systems.

Generators

- How you hook up the RNG to the SUT.
- In short, how the test drives the System.

Interface

- Black box?
- White box?
- In between? (hitting an HTTP api that's not public but also not unit-ey)

Oracle

- How do you know if you've exposed bad behavior?

Loggers and Diagnostics

- How you know what happened.

Context

- What are you trying to do? Corner a bug? Survey the system? Abuse the system?

Scalability

- Single-threaded? Many-threaded? Hammering the system from thousands of cloud servers?

Parveen and Tilley [2010]

Inventors One Through Three

- HP, "evil", 1966
- TI
- Bell

Inventors Four Through Six (And Beyond)

- Microsoft
- Telenova
- Fuzz Tester [Miller et al., 1989]

Everybody Thinks It's A Trade Secret

- Why wouldn't they?
- It's massively better at testing than what they read about.

A Call For Academic Consideration

- TBD

Family Tree: Exploring (And Abusing) The Anatomy

- TBD

Long Sequence Regression Testing

- TBD

State Model Testing

- TBD

Exhastive Testing

- TBD

Fuzz Testing

- TBD

Load Testing

- TBD

Testing In Production

- TBD

A/B Testing

- TBD

Kohavi and Thomke [2017]

Model-Based LSRT

- TBD

Quick To Implement

- TBD

Good Enough Detail

- TBD

Benefit From Existing Test Code

- TBD

References I

Cem Kaner. An overview of high volume automated testing, 2013. URL `http://kaner.com/?p=278`.

Ron Kohavi and Stefan Thomke. The surprising power of online experiments, 2017. URL `https://hbr.org/2017/09/the-surprising-power-of-online-experiments`.

Barton P Miller, Lars Fredriksen, and Bryan So. An empirical study of the reliability of operating system utilities. Technical Report 830, University of Wisconsin–Madison, 1989.

Tauhida Parveen and Scott Tilley. When to migrate software testing to the cloud? In *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on*, pages 424–427. IEEE, 2010.