Yeager: An Annotation-Based Framework for the Generation of Automated Long Sequence Regression Tests in Python

by

Casey Doran

Bachelor of Science
Software Engineering
Florida Institute of Technology
2015

A thesis
submitted to the School of Computing at
Florida Institute of Technology
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Software Engineering

Melbourne, Florida
September 2017

We the undersigned committee hereby recommend
that the attached document be accepted as fulfilling in
part the requirements for the degree of
Master of Science in Software Engineering.

"Yeager: An Annotation-Based Framework for the Generation of Automated
Long Sequence Regression Tests in Python",
a thesis by Casey Doran

Keith Brian Gallagher, Ph.D.
Associate Professor, Computer Sciences and Cybersecurity
Dissertation Advisor

William H. Allen, Ph.D.
Associate Professor, Harris Institude for Assured Information
Dissertation Co-Advisor

X. X. Xxxx, M.S.
Assistant Professor, Physics

X. X. Xxxxxxx, Ph.D.
Professor and Chair, Computer Science

X. X. Xxxx; Ph.D., P.E.
Visiting Professor, Space Sciences

Abstract

TITLE:	Yeager: An Annotation-Based Framework for the Generation of Automated Long Sequence Regression Tests in Python
AUTHOR:	Casey Doran
MAJOR ADVISOR:	Keith Brian Gallagher, Ph.D.

This work presents a Python software package, Yeager, designed to enable the generation and execution of high-volume automated long-sequence regression tests. Users apply the package to existing suites of automated regression tests by annotating individual test methods as state changes for the Software Under Test. Given a sufficiently connected state model (as inferred from these annotations), it becomes possible to generate and execute configurable random walks through the SUT's various states instead of simple regression suites as originally written.

Divided into three sections, this thesis provides a concise overview of an exemplar regression test suite in Python for a web application, a guide to the usage of Yeager itself within the context of the aforementioned regression test suite, and an extensive discussion of the benefits and drawbacks of High Volume Automated Testing in general, and Long Sequence Regression Testing in particular, within the scope of a typical software development organization.

Contents

Acknowledgements

This thesis would not exist if not for the assistance of:

- Dr. Cem Kaner and the Center For Software Testing Education and Research, for taking a chance on an enthusiastic freshman and introducing me to the world of software testing, as well as years of world-class training beyond valuation.
- The wider context-driven testing community, particularly the participants in the Workshops on Teaching Software Testing 11 and 12, my first exposure to the considerable ups and downs of academia.
- Ana Marafuga, Mike DeCabia, Jeff Farr, and Curtis Chambers, the team at Dycom Industries, the most formative internship I've ever had. They let an intern design their entire corporate test automation strategy, most of which informed this thesis, brave souls all.
- Dr. Richard Ford, whose infectious enthusiasm for others' learning and discovery knows no bounds.
- The Samuels family, particularly Bill Jr. and Rob, and Dave Pickerell, for their helpful contributions through most of my academic tribulations.
- The members of the Ruckus and the Harbor City Hooligans. Soccer clubs typically lack rigor, but on the space coast everybody's a rocket scientist.
- My immediate and adopted family, for loving me to where I am today, including
- kbg, for reminding me to thank them for it.

Dedication

TBD, TBH.

Chapter 1

A Concise Overview of A Python Regression Test Suite For a Web Application

This chapter is more of a how-to guide for writing a selenium web test, but it establishes the scenario used in chapter 2 so it's critical to paint a broad enough picture.

1.1 Technologies

this is where we discuss the tools we're using

1.1.1 Selenium

an open source project, meant for testing but used elsewhere, to expose scriptable interactions with web browsers. [Holmes and Kellogg, 2006; Bruns et al.,

2009; Razak and Fahrurazi, 2011; Wang and Xu, 2009; Kaur and Gupta, 2013; Kongsli, 2007; Artzi et al., 2011]

1.1.2 Python Test Runners

pytest vs unittest vs nose etc [Nielsen, 2014; Pajankar, 2017]

1.1.3 Developer Tools and Resources

Web Inspector, probably others[Odell, 2014]

1.2 Architecture

How web application tests are built

1.2.1 Page Objects

object oriented way of encapsulating all the things you can do on one page in a class [Liu et al., 2000; Kung et al., 2000; Leotta et al., 2013a; Marchetto et al., 2008]

1.2.2 DOM Identifiers And Other Constants

discussion of how we tie python abstractions to web page elements [Gupta et al., 2003; Web Hypertext Application Technologies Working Group, 2017; Nicholus, 2016]

1.2.3 Test Sequences

how individual tests are built as a sequence of page object actions [Leotta et al., 2013b]

1.3 Building The Test Suite

how to build the base test suite for the particular SUT we're discussing. may cite the boilerplate, [Sandström, 2015].

1.3.1 Planning A Set Of Tests

walkthrough of how to identify and abstract the list of program features to test, including building a list of actions to write [Nguyen, 2001]

1.3.2 Determining DOM Object Identifiers

how to use a web inspector while manually walking through the program

1.3.3 Scripting Actions

how to build the action sequences

1.3.4 Asserting Validity

assert() 101

1.3.5 Assembling The Final Test Scripts

building (and running) a suite

Chapter 2

Using Yeager To Generate Long Sequence Regression Tests

The test suite assembled in the previous chapter is a great way for a software development team to verify that the core functionality of the system under test is fundamentally operational. When executed, it will test the few well-understood scenarios we have outlined consistently and, assuming enough assertions are present, thoroughly. In fact, the suite requires the entire process from the previous chapter in order to accomodate the additon of new scenarios.

It's a boring, tedious, and repetitious task that can be the entire career of a test engineer. However, as any test automator will know, tasks which are boring, tedious, and repetitious are ripe targets for computer automation, and the task of scenario authorship is no different.

2.1 Software As A State Machine

Consider the system under test, Monica. As a relationship management web site, it has a few obvious states it can be in: logged out and on the landing page, logged in and on the dashboard, viewing a list of contacts, viewing a list of journal entries, or viewing the settings page. This maps nicely to the page objects we defined in the previous chapter. Actions on those page objects assume a current state (eg, we're logged in and on the dashboard) and after executions theoretically are in a new state which may or may not be the same state (eg, the `Dashboard.click_contacts_button()` method transitions from the dashboard to the contacts list). In fact, most modern programs can be looked at as systems composed of a finite set of states (pages, in this case) with some state transitions (links) and a data context (the stuff you've already typed into the system in those states).

2.1.1 States in Our Example System

Let's consider Monica's states which are already built into our test suite.

We have: the login page (`Login`) and logging in takes us to the `Dashboard` which has tabs for the `Contacts` list and the `Journal` log. There's also a `Settings` page which has subpages for `Import`, `Export`, `Users`, and `Tags`.

2.1.2 State Transitions As Actions In Our Example System

identifies actions developed in chapter 1 as lines connecting dots in this graph we're looking at

2.1.3 Our Example System, Illustrated

overview of the system as a whole, fully rendered and illustrated

2.1.4 Graph Connectedness

"Is it possible to get from here to here" and other questions, probably will introduce Dijkstra.

2.1.5 Capturing Contextual State

Yeah, getting past the login screen is cool, but there's other outside influences on the output of the program than just which page we're on

2.1.6 Taking A Walk On The Graph: Long Sequence Testing

introduce the concept of long sequence testing

2.2 Yeager State Transition Annotations

how to use yeager: mark up your existing code

2.2.1 State Identifiers

how to declare a state, formally

2.2.2 Basic State Transition Annotations (The 0-arg Case)

how to declare that a particular function is a transition from one declared state to another

2.2.3 Using The Yeager Connectedness Tester

how to check that yeager can see your states and transitions

2.3 Yeager Test Harnesses

in more advanced scenarios, we need to assist Yeager's execution

2.3.1 Application Configuration

where to put data yeager always needs

2.3.2 Test Setup and Entry Point

pulling a LSRT run up by its bootstraps, and what point on the graph the test starts at.

2.3.3 Exit Point

many scenarios won't deal with this, but how to note ways tests can end successfully.

2.3.4 Application Context Storage

sometimes a test needs more information than just what state we're in. this overviews how to store things relevant to tests (who's expected to be logged in, how many emails they have, how many contacts, etc for an email client app)

2.3.5 Test Method Helpers

special args to an annotation that specify a caller which pulls data from App Context Storage

2.3.6 Yeager-Only Assertions

hooks provided for each state transition which can make additonal assertions not in the original test

2.3.7 The Yeager Logger

how to know what happened

2.3.8 Advanced State Transition Annotations (With Context From Harness)

using the stuff from above to enable more rich/complex state transitions

2.4 Yeager Test Plans

the bread and butter, informing the test generator what you're wanting to do

2.4.1 Run-To-Crash vs. Run-Finitely

discussion of a couple scenarios the tester may wish to choose between

2.4.2 Controlling The Path: Blacklists

how to inform a test to NOT go to certain states

2.4.3 Controlling The Path: Weights

how to inform a test to prefer (or shun) certain states

2.4.4 Controlling The Path: Visitation Limits

how to limit the number of times a particular state should be visited (for instance, dont go to the logout state in this run, stay logged in)

2.4.5 Additional Configuration

tbd during Yeager development

2.4.6 Executing Test Plans

python -m yeager run yplan.py

2.4.7 Interpreting Results And Logs

what do logs look like anyways?

Chapter 3

High Volume Automated Testing And Long Sequence Regression Testing In Context

This is probably an article unto itself. This lends a "why" to the development of Yeager.

3.1 A Note On The Recorded History Of High Volume Automated Testing

what we know about HiVAT

3.1.1 High Volume Automated Testing Has Been Invented Six Times

and here's where we list all the inventors we can find. [Miller et al., 1990]

3.1.2 Every Industrial Inventor Thinks It's A Trade Secret

which is why I'm apologizing that this is sourced from a bunch of talks and interviews and less-than-academic sourcing.

3.1.3 A Call For HiVAT Documentation and Academic Consideration

so that the next poor sap who writes about it isn't going to have to do so much archaeology.

3.2 Anatomy Of A High Volume Automated Test

Let's look at the different legos we can play with

3.2.1 Driver: What Actions Are Taken

how to generate things. random entirely? random from list? build and run?

3.2.2 Interface: Black Box vs. White Box (And Shades Of Grey)

are you acting on the disassembled source or are you acting on the running end-user program, or something in between (like sending http requests to a ui-based app)

3.2.3 Oracle: Determining Correct Behaviour

how do you know things are going ok?

3.2.4 Logger: Figuring Out What Happened

how does the test report the results?

3.2.5 Testing Context: Cornering vs. Surveying vs. Abusing

what are you trying to do with this HiVAT anyways

3.2.6 Scalability: Parallelized vs. Sequential

how are you breaking down the work (and why should you care)

3.3 The High Volume Test Automation Family Tree

let's walk through some well-documented techniques

3.3.1 Long Sequence Regression Testing

uh, this is the one we're talking about [Lee and Yannakakis, 1996]

3.3.2 API Testing

i'm not sure if this belongs but i've seen it on some lists

3.3.3 Exhaustive Testing

ditto

3.3.4 "Fuzzing" And Other Monkey-Based Testing

"throw a fuzzer at it and see what happens"

3.3.5 Load-Based Testing

put one of the above techniques in a thread pool of a million or so

3.3.6 Testing In Production (Safely!)

Microsoft does this, siphons some user input from Bing to the live search engine and the next version of the search engine, comparing output from both versions. Sometimes users get output from the test version, even.

3.3.7 A/B Testing

An aggressive version of TIP invented by marketers to compare multiple versions of the same ad campaign.

3.3.8 Synthetic HiVAT Techniques

This is where I will wildly speculate about techniques not listed in above subsections (and therefore not discovered in literature review), but would make sense to implement in a context, as built from combinations of the building blocks listed in the Anatomy section.

3.4 High Volume Automated Testing Benefits and Drawbacks

this section might be merged into the above section simply due to the uniqueness of benefits and drawbacks among all the various HiVAT techniques. If, however, trends are apparent, they'll be discussed here.

3.5 The Case For Long Sequence Regression Testing

if there's something you could call a "conclusion", it's probably here. LSRT is a powerful, easy-to-adopt form of HiVAT in some scenarios, with otherwise-elusive bug discovery an eminently attainable outcome.

3.6 Scenarios For Yeager Adoption

A shameless ad for different ways Yeager can be adopted by different groups (a subsection per scenario)

Bibliography

Shay Artzi, Julian Dolby, Simon Holm Jensen, Anders Moller, and Frank Tip. A framework for automated testing of javascript web applications. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 571–580. IEEE, 2011.

Andreas Bruns, Andreas Kornstadt, and Dennis Wichmann. Web application tests with selenium. *IEEE software*, 26(5), 2009.

Suhit Gupta, Gail Kaiser, David Neistadt, and Peter Grimm. Dom-based content extraction of html documents. In *Proceedings of the 12th international conference on World Wide Web*, pages 207–214. ACM, 2003.

Antawan Holmes and Marc Kellogg. Automating functional tests using selenium. In *Agile Conference, 2006*, pages 6–pp. IEEE, 2006.

Harpreet Kaur and Gagan Gupta. Comparative study of automated testing tools: Selenium, quick test professional and testcomplete. *International Journal of Engineering Research and Applications*, 3(5):1739–43, 2013.

Vidar Kongsli. Security testing with selenium. In *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*, pages 862–863. ACM, 2007.

David Chenho Kung, Chien-Hung Liu, and Pei Hsia. An object-oriented web test model for testing web applications. In *Quality Software, 2000. Proceedings. First Asia-Pacific Conference on*, pages 111–120. IEEE, 2000.

David Lee and Mihalis Yannakakis. Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

Maurizio Leotta, Diego Clerissi, Filippo Ricca, and Cristiano Spadaro. Improving test suites maintainability with the page object pattern: An industrial case study. In *Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on*, pages 108–113. IEEE, 2013a.

Maurizio Leotta, Diego Clerissi, Filippo Ricca, and Paolo Tonella. Capture-replay vs. programmable web testing: An empirical assessment during test case evolution. In *Reverse Engineering (WCRE), 2013 20th Working Conference on*, pages 272–281. IEEE, 2013b.

Chien-Hung Liu, David Chenho Kung, and Pei Hsia. Object-based data flow testing of web applications. In *Quality Software, 2000. Proceedings. First Asia-Pacific Conference on*, pages 7–16. IEEE, 2000.

Alessandro Marchetto, Paolo Tonella, and Filippo Ricca. State-based testing of ajax web applications. In *Software Testing, Verification, and Validation, 2008 1st International Conference on*, pages 121–130. IEEE, 2008.

Barton P Miller, Louis Fredriksen, and Bryan So. An empirical study of the reliability of unix utilities. *Communications of the ACM*, 33(12):32–44, 1990.

Hung Q Nguyen. *Testing applications on the Web: Test planning for Internet-based systems*. John Wiley & Sons, 2001.

Ray Nicholus. Understanding the web api and vanilla javascript. In *Beyond jQuery*, pages 19–29. Springer, 2016.

Finn Årup Nielsen. Python programmingtesting. 2014.

Den Odell. Browser developer tools. In *Pro JavaScript Development*, pages 423–437. Springer, 2014.

Ashwin Pajankar. *Python Unit Test Automation: Practical Techniques for Python Developers and Testers*. Apress, 2017.

Rosnisa Abdull Razak and Fairul Rizal Fahrurazi. Agile testing with selenium. In *Software Engineering (MySEC), 2011 5th Malaysian Conference in*, pages 217–219. IEEE, 2011.

Martin Sandström. marteinn/selenium-python-boilerplate: A boilerplate for running selenium tests with python. `https://github.com/marteinn/Selenium-Python-Boilerplate`, September 2015. (Accessed on 07/05/2017).

Xinchun Wang and Peijie Xu. Build an auto testing framework based on selenium and fitnesse. In *Information Technology and Computer Science, 2009. ITCS 2009. International Conference on*, volume 2, pages 436–439. IEEE, 2009.

Web Hypertext Application Technologies Working Group. Document object

model standard. https://dom.spec.whatwg.org/, June 2017. (Accessed on 07/04/2017).