COMS W4156 Advanced Software Engineering (ASE)

December 6, 2022

Agenda

1. Questions about second assessment
2. advertisement for my spring class
3. if time permits: Test Oracles

Repository Cloning Demo: Satyam Sharma

https://docs.google.com/document/d/1wbF7M0vxuJLe4NJ-rcjM5Q1QA7BgzN2R/edit

Agenda

1. Questions about second assessment
2. advertisement for my spring class
3. if time permits: Test Oracles

COMS E6156 Topics in Software Engineering

6156 is not "more" 4156, and not "more advanced" 4156. 6156 is a “topics” course, like 6998, where each section has a different topic. Prof. Donald Ferguson teaches his very popular Cloud Computing course as a section of 6156, but my section has nothing to do with Cloud Computing

My section is a seminar where students read, present and discuss research papers; write a midterm literature review paper; and conduct a final research project.   There are no lectures after the first week or so, after that all class sessions consist of student presentations and discussions. “Live” attendance is required. The paper must be individually written but the project can be joint with any reasonably-sized team. The student or student team chooses their own subtopic within “software engineering and security”, broadly construed.

Some of Last Year’s Student-Chosen Project Topics

Single encoder / multiple decoder method generation architecture for CodeXGlue (fine-tunes CodeGPT model)

Formal methods-based validation of APR-generated bugfixes (Defects4J dataset)

Efficacy of Deepfake detection methods

Capture-the-flag based series of lab assignments for a hands-on Security I course

Code embeddings for vulnerability detection

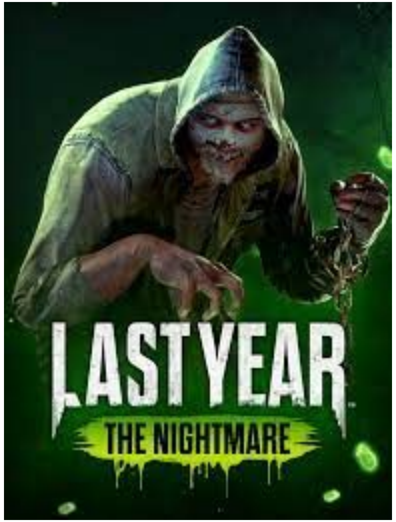
Combined scanner for several root causes of security vulnerabilities

Blockchain-based tamper-proofing for relational databases

NPC population control for video games

Static analysis for likely performance bugs in SQL database applications

(I intentionally removed projects from last spring’s 6156 students who are currently taking 4156)



Agenda

1. Questions about second assessment
2. advertisement for my spring class
3. if time permits: Test Oracles

What is a Test Oracle?

The entity that already knows the expected outputs and/or can check that the actual outputs are the same as or consistent with the expected outputs

Often the human tester is the test oracle - either knows the correct output in advance or, given an actual output, can determine whether or not it is correct

For manual testing, the human tester checks the results

For automated testing, the tester writes [assertions] to check the results

Test Assertions

Notation for checking the results of executing a test

Usually supports (at least) checking for true/false result, comparing an output to a known value, and checking whether an exception has been raised

[Google Test assertions]

[JUnit 5 Assertions]

Writing Better Assertions

Assume that we test a Java method whose signature is

```java
List<Employee> getEmployees(Predicate<Employee> constraint)
```

This method accepts a predicate on `Employee` and retrieves from some database the list of all employees that match the given predicate.

Let's say we want to check that the function retrieves all employees older than 25 when it is called with a predicate of the form "*the age of an employee is greater than 25*". We could do so in a JUnit test like this one:

```java
@Test
public void getEmployeesGreaterThan25() {
  // arrange
  Employee mark = new Employee ("Mark", 19);
  Employee nina = new Employee ("Nina", 26);
  Employee jules = new Employee ("Jules", 31);
 clearDatabase();
  insertEmployee(mark);
  insertEmployee(nina);
  insertEmployee(jules);
  Predicate<Employee> constraint = e -> e.getAge() > 25;
```

Example continued

```
// act
List<Employee> result = getEmployees(constraint);
// assert
assertEquals(result.size(), 2);
assertEquals(result.get(0).getName(), "Nina");
assertEquals(result.get(0).getAge(), 26);
assertEquals(result.get(1).getName(), "Jules");
assertEquals(result.get(1).getAge(), 31);
    }
```

We used five assertions to check that the function returns all employees older than 25.

But, looking closely, these assertions actually check something else: they check that the result list always contains Nina in the first entry and Jules in the second one.

This particular detail could change as we update the implementation of getEmployees, or could even be altered by external sources of non-determinism.

This is not the best set of assertions: should assert requirements, not implementation details

Assert the exact desired behavior; not more, not less

Alternatively, we could have written the following lines in place of those 5 assertions:

```
for (Employee e : result) {
  assertTrue(e.getAge() > 25);
}
```

This version has problems, too. We now check that every employee returned by the function is over 25.

But we are not checking that *all* such employees are returned. If `getEmployees` was modified to always return an empty list, this test would still pass.

Good assertions check what is precisely requested from the code under test, as opposed to checking an overly precise or an overly loose condition.

It is easy to write overly precise assertions, but the resulting test is often not maintainable or might be affected by slight non-deterministic variations in the environment.

It is also tempting to write overly loose assertions, but the resulting test might not catch regressions that it should catch.

One assertion, one condition

It may be tempting to aggregate multiple checks in one assertion. We could have written something like:

```
boolean first = result.get(0).getAge() > 25;
boolean second = result.get(1).getAge() > 25;
    assertTrue(first && second);
```

If that assertion fails, we will not immediately know if it is because the first or the second employee. Splitting into two assertions gives us a better clue about the cause:

```
assertTrue(result.get(0).getAge() > 25);

assertTrue(result.get(1).getAge() > 25);
```

Don’t write assertions that check conditions that could be split into multiple assert statements.

Always check the simplest condition possible.

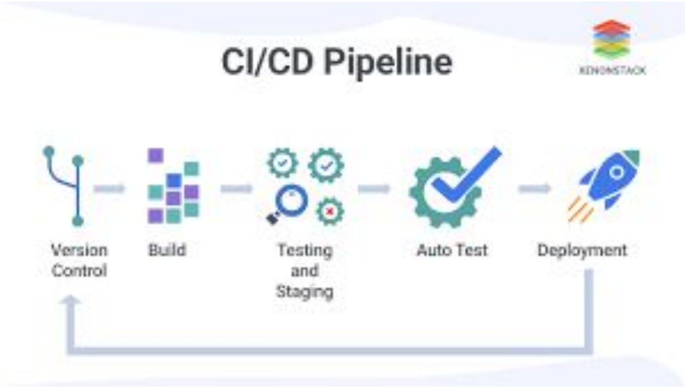
Any gain in speed when writing complex conditions is lost when someone needs to debug why such assertion fails.

Why Are Good Test Assertions Important?

Continuous integration (CI) tools hook a version control system with a build tool to automatically re-run the build after every commit to the shared repository, to detect errors quickly

Continuous Delivery/Deployment (CD) adds automatic delivery and/or deployment to continuous integration = produces a patch for customers and/or installs new build on the production server(s)

Continuous deployment (and possibly continuous delivery) has no human in the loop!



CD requires that testers be able to use automated testing tools, with automatically checked assertions, for all testing, including end-to-end system testing and acceptance testing - not just for unit and integration testing

How Does the Tester Know the Correct Answer to Assert?

If it doesn't crash, hang, or produce obviously nonsensical output, it works!

Stick to math and string processing libraries, with well-understood functions

Wait until users complain that the software doesn't do what they expected, then write test cases to check for what they say they expected

Specifications

Detailed prose specification - common for networking, database, file system and other standard protocols necessary for interoperability, rare for user-facing functionality of business/consumer applications

```
[Search] [txt|html|pdf|with errata|bibtex] [Tracker] [WG] [Email] [Diff1] [Diff2] [Nits]

From: draft-ietf-uri-url-07                                  Proposed Standard
Obsoleted by: 4248, 4266                                          Errata exist
Updated by: 1808, 2368, 2396, 3986, 6196, 6270,
            8089
Network Working Group                                          T. Berners-Lee
Request for Comments: 1738                                               CERN
Category: Standards Track                                         L. Masinter
                                                            Xerox Corporation
                                                                  M. McCahill
                                                      University of Minnesota
                                                                      Editors
                                                                December 1994


                        Uniform Resource Locators (URL)

Status of this Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements.  Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol.  Distribution of this memo is unlimited.

Abstract

   This document specifies a Uniform Resource Locator (URL), the syntax
   and semantics of formalized information for location and access of
   resources via the Internet.

1. Introduction

   This document describes the syntax and semantics for a compact string
   representation for a resource available via the Internet.  These
   strings are called "Uniform Resource Locators" (URLs).

   The specification is derived from concepts introduced by the World-
   Wide Web global information initiative, whose use of such objects
   dates from 1990 and is described in "Universal Resource Identifiers
   in WWW", RFC 1630. The specification of URLs is designed to meet the
   requirements laid out in "Functional Requirements for Internet
   Resource Locators" [12].

   This document was written by the URI working group of the Internet
   Engineering Task Force.  Comments may be addressed to the editors, or
   to the URI-WG <uri@bunyip.com>. Discussions of the group are archived
   at <URL:http://www.acl.lanl.gov/URI/archive/uri-archive.index.html>
```

Specifications

Formal Specification - expensive to write and verify, mostly used for safety-critical systems

$$\frac{P \to P' \quad \{P'\}\, C\, \{Q'\} \quad Q \to Q'}{\{P\}\, C\, \{Q\}}\text{WEAK}$$

$$\frac{}{\{A[E/x]\}\; x := E\; \{A\}}\text{ASG}$$

$$\frac{\{P\}\, C_1\, \{A\} \quad \{A\}\, C_2\, \{Q\}}{\{P\}\, C_1; C_2\, \{Q\}}\text{SEQ}$$

$$\frac{\{P \wedge B\}\, C_1\, \{Q\} \quad \{P \wedge \neg B\}\, C_2\, \{Q\}}{\{P\}\ \textbf{if}\ B\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \{Q\}}\text{IF}$$

$$\frac{\{I \wedge B\}\, C\, \{I\}}{\{I\}\ \textbf{while}\ B\ C\ \{I \wedge \neg B\}}\text{While}$$

Sometimes There Isn’t Any Test Oracle

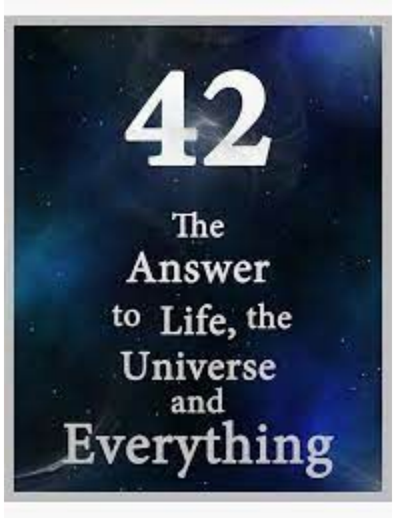
Consider this search: https://www.google.com/search?q=gail+kaiser

Is the answer correct? Are the answers on the first page the best ones?

Now consider this search:
https://www.google.com/search?q=gail+kaiser&tbm=isch&hl=en&chips=q:gail+kaiser,online_chips:software+engineering

Is the answer correct? Are the answers on the first page the best ones?

Non-testable program = *“Programs which were written in order to determine the answer in the first place. There would be no need to write such programs, if the correct answer were known.”* (Weyuker 1982)

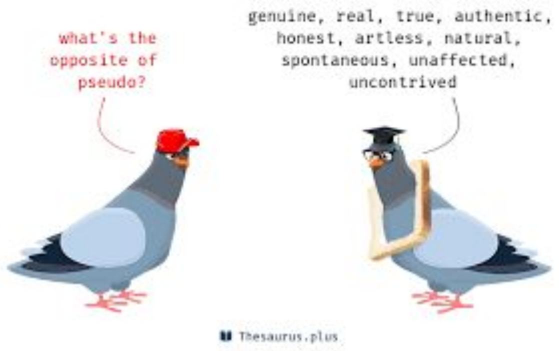


Pseudo-Oracles

Pseudo-oracle = one test case acts as an oracle for another test case

You may not know whether the result of either test case is right, but if the results are inconsistent you know (at least) one of them is wrong

They could both be wrong



Pseudo-Oracles: Regression Testing

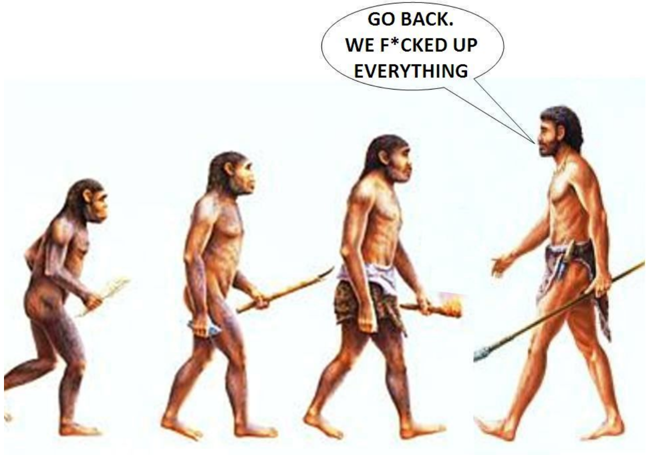
Regression testing is the most common form of pseudo-oracle

The test results for the old version are known and we re-run the tests on the new version to see if they have the same or different results

Some test cases *should* change results on the new version ⇒ when we try to fix a bug that caused a test case to fail on the old version, we want that test case to pass on the new version

But unrelated test cases should not be affected ⇒ if they are, that indicates a hidden dependency and (probably) a newly introduced bug

```
program_v1(input)->output_v1

program_v2(input)->output_v2

output_v1 =? output_v2
```



stopped here

Pseudo-Oracles: Differential Testing

Differential testing requires a second *independently developed* implementation of the same functionality.

Practical for standard protocols, e.g., there are many different implementations of SSL/TLS and HTTP

- ➢ If they disagree, one or both is wrong
- ➢ If they agree, that does not guarantee bot are right, but does provide some confidence that they are right

```
program_1(input)->output_1

program_2(input)->output_2

output_1 =? output_2
```

It's critical that the implementations are truly independent, with no common third-party libraries or other common factors

But "independent" developers can make the same mistakes

Pseudo-Oracles: Metamorphic Testing

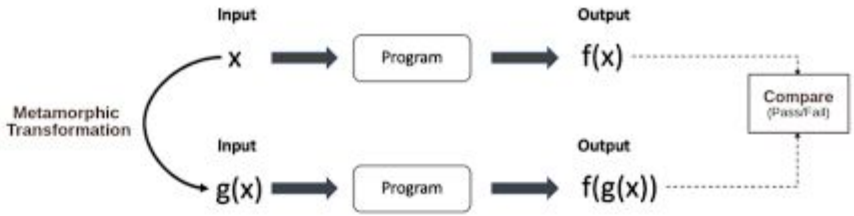
Metamorphic testing requires only one implementation

Starts with some original input (possibly chosen randomly) and its known original output. We do not need to know whether this output is correct

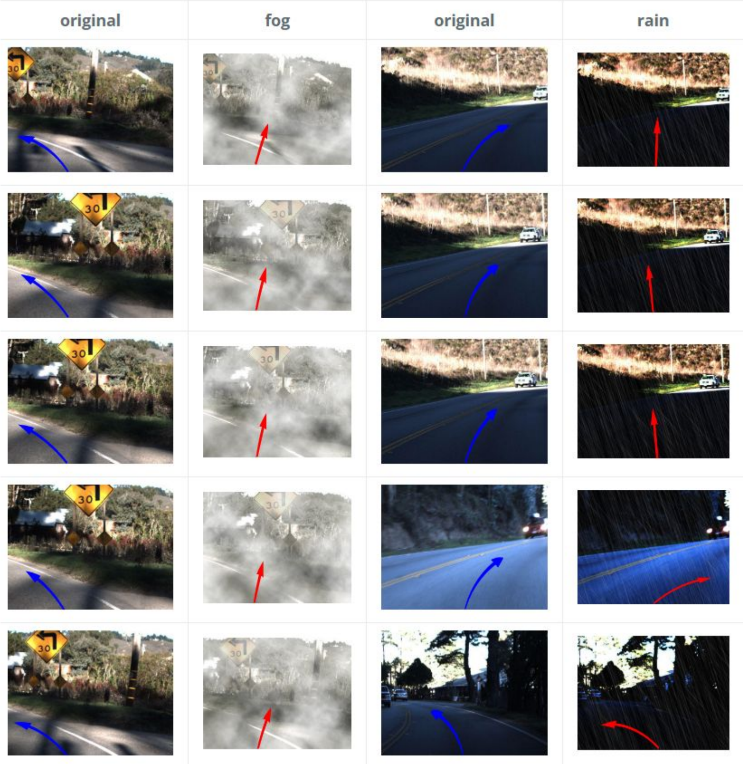
Create a new test case by *deriving* a new input from the original input, and *predicting* the expected new output from the original input, the derived input and the original output - the prediction is usually that the output is the same, nearly the same, or changed in a simple way

If the actual new output deviates too much from the predicted output, there is (probably) a bug

```
program(input_1)->output_1

deriveInput(input_1)->input_2

predictOutput(input_1, input_2,
output_1) -> predicted_output_2

program(input_2)->output_2

check(predicted_output_2, output_2)
-> do they match?
```



Example Metamorphic Testing



Say we have a machine learning application that controls the steering of a self-driving car, so it obeys traffic laws and doesn't cause accidents

How do we know what is "correct" steering for a given real-world driving scenario? How can we write test cases?

Different human drivers will not necessarily steer at exactly the same angle and the same human driver will not necessarily steer at the same angle every time

DeepTest: Synthetic but realistic changes to driving images such as adding snow, fog, rain, more sunlight, less sunlight, etc. should not (in most cases) change automobile steering

Predicted Output is not Necessarily the Same Output

Metamorphic relations for a sorting program (lowest first) applied to an array of numbers

sort(5,2,3,1,4) -> 1,2,3,4,5

Permutative: if we shuffle the order of the input array, the sorted output array should be the same

sort(3,4,1,2,5) -> 1,2,3,4,5 😀

Additive: if we add N to every element of the input array, the sorted output array should be in the same order

sort(25,22,23,21,24) -> 21,22,23,24,25 😀

Multiplicative: if we multiply every element of the input array by (positive) N, the sorted output array should be in the same order

sort(15,6,9,3,12) -> 3,6,9,12,15 😀

Invertive: if we multiply every element of the input array by (negative) N, the sorted output array should be in the opposite order

sort(-5,-2,-3,-1,-4) -> -5,-4,-3,-2,-1 😀

Inclusive: if we add one new element to the input array, the sorted output array should be the same except for the placement of that new element

sort(5,2,42,3,1,4) -> 1,2,3,4,5,42 😀

Exclusive: if we remove one element from the input array, the sorted output array should be the same except it is missing that one element

sort(5,3,1,4) -> 1,3,4,5 😀

Using Metamorphic Relations to Detect Bugs

Original

buggysort(5,2,3,1,4) -> 1,2,3,4,5

Permutative: if we shuffle the order of the input array, the sorted output array should be the same

buggysort(3,4,1,2,5) -> 2,4,1,5,3 😞

Additive: if we add N to every element of the input array, the sorted output array should be in the same order

buggysort(25,22,23,21,24) -> 25,25,25,25,25 😲

Multiplicative: if we multiply every element of the input array by (positive) N, the sorted output array should be in the same order

buggysort(15,6,9,3,12) -> 6,12,3,15,9 😟

Invertive: if we multiply every element of the input array by (negative) N, the sorted output array should be in the opposite order

buggysort(-5,-2,-3,-1,-4) -> -1,-4,-3,-2,-5 😕

Inclusive: if we add one new element to the input array, the sorted output array should be the same except for the placement of that new element

buggysort(5,2,42,3,1,4) -> ⌛ 😖

Exclusive: if we remove one element from the input array, the sorted output array should be the same except it is missing that one element

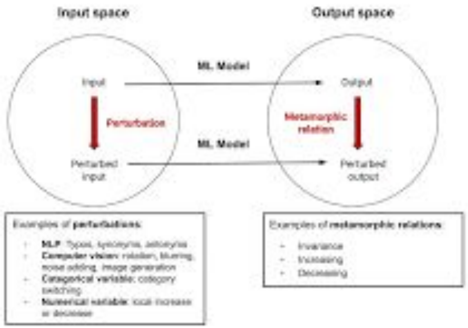
buggysort(5,3,1,4) -> 🌋 😳

Where Do Metamorphic Relations Come From?

Some are generic, e.g., permutation can be applied to any ordered input

Others are specific to domain or application

Metamorphic Testing automates generation of additional tests from existing test suite whether or not there is a test oracle - it’s a test generation technique as well as a pseudo-oracle technique



Upcoming Assignments

[Second Individual Assessment] from 12:01am Tuesday December 6 through 11:59pm Friday December 9 (this Friday)

[Demo Day] Monday December 19 10am to 4pm   signup sheet [here]

Thursday

more questions about second assessment

another advertisement for my spring class

finish Test Oracles

a little bit on Test Generation

Ask Me Anything