EE360T/382C-16 Software Testing

khurshid@ece.utexas.edu

Lecture 1

January 22, 2020

Today

Introductions

Overview

Java and JUnit basics

Next time

Graph theory, logic, and discrete math basics

Introductions: this course

Introduction to software testing

- Systematic, organized approaches to testing
  - Based on models and coverage criteria
- Improve your testing (and development) skills
  - Not focused on research (EE382C-3)

Prerequisites: EE422C (or 322C) with a grade of at least C-

- Knowledge of data structures and object-oriented languages
- Programming experience

Lectures: MW 9am to 10:30am, CAL 100

Introductions: teaching team

Instructor

- Sarfraz Khurshid <khurshid@ece.utexas.edu>
  - Office: EER 7.880
  - Office hours: TuW 10:30-11:30am

TAs

- Grace Lee <gracewlee@utexas.edu>
- Wenxi Wang <wangwenxi0407@outlook.com>
- Jiayi Yang <jiayiyang1997@utexas.edu>
  - Office hours: TBD

Introductions: you

Undergrad/grad?

Programming experience?

Testing/verification experience?

Research experience?

Calendar—tentative

Week 1	1/22	Introduction, course overview, Java/JUnit basics
Week 2	1/27	Graph theory, logic, and discrete math basics
	1/29	Graph theory, logic, and discrete math basics
Week 3	2/ 3	Chapter 2 (1): Basic software testing principles and concepts
	2/ 5	Chapter 7 (2): Graph coverage Criteria
Week 4	2/10	Chapter 7 (2): Graph coverage Source code

…

Exam 1 – **February 26**

Exam 2 – **March 30**

Exam 3 – **May 6**

See more details on Canvas [syllabus]

Evaluation

Undergrads

- Homeworks (25%)
- Mid-term exams (75%)

Collaboration

No communication/texts during exams

You must individually write solutions for problem sets

You can discuss problem sets

Testing benefits from good communication skills

Textbook – recommended

Introduction to Software Testing by Paul Ammann and Jeff Offutt. ISBN: 9781107172012

Canvas

courses.utexas.edu

Course web-page

Slides

Handouts

Discussions

Problem sets

…

Software testing (1)

Testing plays a vital role in validating software quality

Not testing well can be very costly

- Bugs in code can lead to costly failures
    - E.g., Ariane 5, Mars PolarLander, USS Yorktown

Finding/fixing bugs cost >300B USD annually [Cambridge'13]

Conceptually, testing is simple:

- Create tests, run them, and check for failures

In practice, testing is ad hoc and expensive

Creating tests is a crucial and typically labor intensive part of testing

- Automation can help!

Software testing (2)

Testing is a dynamic approach for finding bugs

- Checks correctness for some – typically a small number compared to total number of – executions
- No test failures does not imply no bugs in code
- In contrast, static analysis can prove certain properties for all inputs

Testing is **not** the same as debugging, i.e., locating and removing specific faults

Topics in testing

Basic questions

- How to create test inputs?
- How many tests to create?
- How to check outputs?

There are many additional topics

- Selection, minimization, prioritization, augmentation, evaluation, …

Testing is not just about finding faults!

Terminology

Anomaly

Bug

Crash

Defect

Error, exception

Failure, fault, flaw, freeze

Glitch

Hole

Issue

…

“Bugs” in IEEE 610.12-1990

Fault – incorrect lines of code

Error – faults cause incorrect (unobserved) state

Failure – errors cause incorrect (observed) behavior

These terms are not always used consistently in literature

- But we’ll (try to) follow these standard definitions

Correctness

Expected correctness properties have two basic forms

- Common properties that we expect of a large class of programs
  - No segfaults, core dumps, deadlocks, memory leaks, etc.
- Specific properties that we expect of the specific program under test
  - May be derived from requirements or specs
  - E.g., method under test sorts correctly

JUnit

> Never in the field of software development was so much owed by so many to so few lines of code
>
> –Martin Fowler

Written by kent beck and erich gamma

Testing framework for writing and executing test cases

Automates testing of java programs

Website: junit.org

Has inspired a family of related tools

- E.g., cppUnit (C++), nUnit (C#), pyUnit (python)

JUnit example: addition

```
import static org.junit.Assert.*;
import org.junit.Test;

public class JUnitIntegerAddDemo {
    static int add(int x, int y) {
        return x + y;
    }

    @Test public void commutativity() {
        assertTrue(add(1, 2) == add(2, 1));
    }
}
```

org.junit.Assert

static void assertEquals(java.lang.Object expected, java.lang.Object actual)

static void assertNotNull(java.lang.Object object)

static void assertNull(java.lang.Object object)

static void assertTrue(boolean condition)

...

JUnit example: exceptions

```java
import org.junit.Test;

public class JUnitExceptionDemo {
    static int div(int x, int y) {
        return x/y;
    }

    @Test(expected=ArithmeticException.class)
    public void exceptionalDivide() {
        div(1, 0);
    }
}
```

JUnit example: timeout

```
import org.junit.Test;

public class JUnitPerformanceDemo {
    @Test(timeout=10) public void loop() {
        for (int i = 0; i < 1000; i++) System.out.print(i);
    }
}
```

?/!