ee360t/ee382v
software testing

sarfraz khurshid

jan 18, 2017

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
- testing is not only about finding "bugs"
- improve your testing (and development) skills
- not focused on research (ee382c-3)

undergrads
- 6 problem sets, 2 mid-term exams, and a final exam

grads
- 6 problem sets, 2 mid-term exams, a final exam, and a group project

introductions: teaching team

Instructor
- sarfraz khurshid <khurshid@ece.utexas.edu>
  - office: ACES 5.120
  - office hours: MF 10:30-11:30am

TAs
- zijiang yang <yangzijiangjosh@gmail.com>
  - office hours: TBD
- jiaolong yu <joceyu@utexas.edu>
  - office hours: TBD

introductions: you

undergrad/grad?
programming experience?
testing/verification experience?
research experience?

administrative info

lectures: MW 9am to 10:30am, SZB 104

prerequisites: ee422c (or 322c) with a grade of at least C-

- knowledge of data structures and object-oriented languages
- programming experience

7

evaluation

undergrads

- homeworks (40%)
- mid-term exams (35%)
- final exam (25%)

grads

- homeworks (20%)
- mid-term exams (35%)
- final exam (25%)
- project (20%) – team of 2 or 3 students
  - proposal (due: march 12)
  - final report (due: may 1)
  - in-class presentation (last class week)

8

collaboration

no communication/texts during exams

you must individually write solutions for problem sets

you can discuss problem sets

testing is a social activity

- communication matters

9

textbook—required

Introduction to Software Testing
by Paul Ammann and Jeff Offutt. ISBN: 0521880386

10

canvas

courses.utexas.edu

course web-page

slides

handouts

discussions

problem sets

...

11

calendar—tentative

Week 1	1/18	Introduction, course overview, Java/JUnit basics
Week 2	1/23	Graph theory, logic, and discrete math basics
	1/25	Chapter 1: Basic software testing principles and concepts
Week 3	1/30	Chapter 2: Graph coverage Criteria
	2/ 1	Chapter 2: Graph coverage Source code
Week 4	2/ 6	Chapter 2: Graph coverage Designs/Specifications/use-cases

…

mid-term exam 1 – **february 15**

mid-term exam 2 – **march 22**

see more details on canvas [syllabus]

12

software testing

testing is a **dynamic** approach for bug finding
- run code for **some** inputs, check outputs
- checks correctness for **some** executions

testing is not the same as debugging (locating and removing specific faults)

main questions
- test-input generation
- test-suite adequacy (coverage criteria)
- test oracles

other testing questions

selection
minimization
prioritization
augmentation
evaluation
fault characterization
…
testing is not (only) 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

not used consistently in literature!

correctness

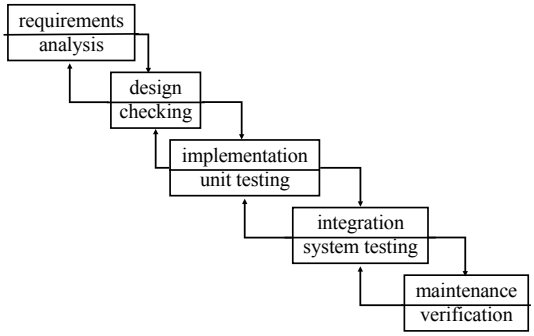
common (partial) properties
- segfaults, uncaught exceptions
- resource leaks
- data races, deadlocks

specific properties
- requirements
- specification

a look at textedit (or notepad)

what happens when you type the string “hello world” in notepad and select the “undo” operation?
1. it deletes the string “hello world”
2. it deletes the character ‘d’
3. it deletes the string “world”
4. it deletes part of the string “hello world”
5. it highlights part of the string “hello world”
6. it doesn’t do anything
7. none of the above

traditional waterfall model



19

phases (1)

requirements
- specify what the software should do
- analysis: eliminate/reduce ambiguities, inconsistencies, and incompleteness

design
- specify how the software should work
- split software into modules, write specifications
- checking: check conformance to requirements

20

phases (2)

implementation
- specify how the modules work
- unit testing: test each module in isolation

integration
- specify how the modules interact
- integration testing: test module interactions
- system testing: test entire system

maintenance
- evolve software as requirements change
- regression testing: test changes

21

topics related to finding bugs

how to eliminate bugs?
- debugging

how to prevent bugs?
- programming language design
- software development processes

how to show absence of bugs?
- theorem proving
- model checking, program analysis

22

testing topics to cover

test coverage and adequacy criteria
- graph, logic, input domains, syntax-based

test-input generation
test oracles
model-based testing
testing software with structural inputs
test automation
testing in your domain of interest?

23

java

java

object-oriented
architecture neutral
robust
exception handling
multi-threaded
garbage collection (no malloc, free)

example

```
public class HelloWorld {
    static public void main(String args[]) {
        System.out.println("Hello world!");
    }
}
```

25

java: data types

primitive
- numeric: 8-bit byte, 16-bit short, 32-bit int, and 64-bit long 32-bit IEEE 754 float, 64-bit IEEE 754 double
- character: 16-bit unicode char
- boolean: true or false

reference
array
- first-class objects
- have length, e.g., myArray.length
- have out-of-bounds access checks

string (java.lang.String and java.lang.StringBuffer)
- objects
- can use quotes to initialize, e.g., myString = "hello!";

26

java: constructs

packages
classes and objects
instance variables
constructors
methods
class variables and class methods
abtract classes and abstract methods
finalizers
access control
subclasses
interfaces

27

java: base system and libraries

basic java classes (java.lang)
- Object, Math, SecurityManager, Thread, System, Boolean, ...

input/output (java.io)
- File, InputStream, OutputStream, PrintStream, ...

utilities (java.util)
- Set, Map, Vector, StringTokenizer, Date, ...

abstract window toolkit (java.awt)
- Component, Color, Font, Image, Window, Applet, ...

28

java: type safety

strongly typed language
- there cannot be any type errors when the program runs

automatic storage management
- no dangling pointers

array index checks

29

java: type hierarchy

types are organized in a hierarchy
subtype's objects have all the methods defined by supertype
all reference types are subtypes of Object
Object defines a number of methods, e.g., toString, equals
apparent type of a variable is the type understood by compiler
actual type of object is its real type—type it receives at creation
java guarantees: apparent type is a supertype of actual type

30

java: differences from C/C++

no typedefs, defines, preprocessors
no structures, unions
no functions
no multiple inheritence
no goto
no operator overloading
no automatic coercions
no pointer data type, pointer arithmetic

31

eclipse

eclipse

extensible development platform and application framework for building software
built-in support for useful tools, such as, ant, cvs, and junit

to download on your home machine

- get it from eclipse.org
- unzip the zip file
- double-click on "eclipse.org"

to start

- create a new java project
- create a package
- …

33

eclipse: some useful features

auto-complete (ctrl-space)
organize import statements (ctrl-shift-o)
look up java API doc (shift-f2, cursor on method/class name)
(un)comment block of code (ctrl-/, ctrl-\, highlighted code)
mark TODO comments
generate get/set methods
refactoring code, e.g., renaming or moving packages, classes, methods, variables ("Refactor" >> "Rename")
emacs key binding

34

eclipse: demo

35

junit

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)

mini quiz

1. write a method that adds two integers; write a test case to check that addition commutes, i.e., a + b = b + a
2. write a method that divides two numbers; write a test case to check that division by 0 raises an ArithmeticException

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(double expected, double actual, double delta)
static void assertEquals(float expected, float actual, float delta)
static void assertEquals(java.lang.Object[] expecteds, java.lang.Object[] actuals)
static void assertEquals(java.lang.Object expected, java.lang.Object actual)
static void assertEquals(java.lang.String message, double expected, double actual, double delta)
static void assertEquals(java.lang.String message, float expected, float actual, float delta)
static void assertEquals(java.lang.String message, java.lang.Object[] expecteds, java.lang.Object[] actuals)
static void assertEquals(java.lang.String message, java.lang.Object expected, java.lang.Object actual)
static void assertEquals(boolean condition)
static void assertFalse(java.lang.String message, boolean condition)
static void assertNotNull(java.lang.Object object)
static void assertNotNull(java.lang.String message, java.lang.Object object)
static void assertNotSame(java.lang.Object unexpected, java.lang.Object actual)
static void assertNotSame(java.lang.String message, java.lang.Object unexpected, java.lang.Object actual)
static void assertNull(java.lang.Object object)
static void assertNull(java.lang.String message, java.lang.Object object)
static void assertSame(java.lang.Object expected, java.lang.Object actual)
static void assertSame(java.lang.String message, java.lang.Object expected, java.lang.Object actual)
static void assertTrue(boolean condition)
static void assertTrue(java.lang.String message, boolean condition)
static void fail()
static void fail(java.lang.String message)
```


junit example: exceptions

```
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: performance tests

```
import org.junit.Test;

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

junit cookbook by beck and gamma

http://junit.sourceforge.net/doc/cookbook/cookbook.htm

```
@Test public void simpleAdd() {
    Money m12CHF= new Money(12, "CHF");
    Money m14CHF= new Money(14, "CHF");
    Money expected= new Money(26, "CHF");
    Money result= m12CHF.add(m14CHF);
    assertTrue(expected.equals(result));
}
```

43

junit cookbook: fixtures

fixture—set of background objects against which a test runs

fixtures allow you to share code among different tests

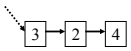
```
public class MoneyTest {
    private Money f12CHF;
    private Money f14CHF;
    private Money f28USD;

    @Before public void setUp() {
        f12CHF= new Money(12, "CHF");
        f14CHF= new Money(14, "CHF");
        f28USD= new Money(28, "USD");
    }
}
```

44

implementing and testing a linked list

consider a singly-linked acyclic list




```
public class SLList {
    Node header;
    int size;

    static class Node {
        int elem;
        Node next;
    }
}
```

45

?/!

46