UT ECE | THE UNIVERSITY OF TEXAS AT AUSTIN
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

EE360T/382V Software Testing

khurshid@ece.utexas.edu

April 2, 2018

Overview

Today

- Exam 2 is back
- Chapter 6: Practical considerations

Next time

- Building software testing tools

EE360T/382V Software Testing

khurshid@ece.utexas.edu

Practical considerations (Chapter 6)*

*Introduction to Software Testing by Ammann and Offutt

6.5 Identifying correct outputs

Oracle problem – "Is the output correct?"

- Often a complex problem to solve

This section discusses four techniques to check outputs:

- Direct verification of outputs
  - Behavioral specifications
- Redundant computations

  Alternative implementations
- Consistency checks
  - Class invariants or other partial properties, e.g., tree has no cycle, output is one of the inputs, …
- Data redundancy -- compare outputs for different inputs

Direct verification of outputs

Automated checking using an executable specification

- Implement code that checks expected properties
  - `boolean oracle(Object input, Object output);`

Same check can run against many inputs

Cost is often high

- Need to implement complex checking logic
  - E.g., relation between pre-state and post-state
    - add correctly adds the given element to a set

Example: specifying sort

Consider a method to sort an integer array

- void sort(int[] arr);

Post-condition: arr is in sorted order

- Pre-state: [8, 92, 8, 14]
- Post-state: [1, 2, 3, 4] – allowed by (weak) spec

ΔPost-condition: no new elements introduced

- Post-state: [92, 14, 8, 8] – still allowed by spec

ΔPost-condition: ascending order

- Post-state: [8, 14, 14, 92] – still allowed by spec

ΔPost-condition: pre-state is a permutation of post-state

Example: executable check

```
static boolean checkSort(int[] arr) {
    int[] copy = Arrays.copyOf(arr, arr.length);
    sort(arr);
    for(int x: arr) {
        if (count(x, arr) != count(x, copy)) return false;
    }
    for (int i = 0; i < arr.length - 1; i++) {
        if (arr[i] > arr[i + 1]) return false;
    }
    return true;
}
```

Similar in complexity to the method under test!

Redundant computation

Aka differential testing

- Use another implementation, e.g., one that has slower performance
  - Check if the outputs match

Can check many test executions

Can be expensive

May have coincidental failures

- Both programs may fail on the same input

Applies naturally in regression testing

- Check outputs of current version against outputs of previous version

Data redundancy

Compare outputs for different inputs

- Partial solution
  - Checks for specific faults

E.g., use knowledge about *identities*

- Pushing an element on a stack followed by popping it leaves the stack in its original state
- `min(1, 2, 3) == min(2, 1, 3)`
- $sin(a + b) == sin(a).cos(b) + cos(a).sin(b)$

Outline

Regression testing

Integration and testing

- Stubs and drivers
- Class integration and test order

Test process

Test plans

Identifying correct outputs

- Direct verification
- Redundant computations
- Consistency checks
- Data redundancy

?/!