SE 2AA4, CS 2ME3 (Introduction to Software Development)

Winter 2018

32 White Box Testing (Ch. 6)

Dr. Spencer Smith

Faculty of Engineering, McMaster University

March 27, 2018

32 White Box Testing (Ch. 6)

- Administrative details
- Nonfunctional testing
- Functional testing
- Testing phases
- Theoretical foundations of testing
- Complete coverage principle
- Statement coverage
- Edge coverage
- Condition coverage
- Path coverage

Administrative Details

- A4: Due April 9 at 11:59 pm
- No classes on Thurs, Mar 29 or Fri, Mar 30
- Final tutorial (examination review)
  - Monday, Mar 26
  - Tuesday, Mar 27
  - Friday, Apr 6 (no tutorial on Fri, Mar 30)
- Course evaluations
  - https://evals.mcmaster.ca
  - Start: Tues, Mar 27, 10:00 am
  - Close: Tues, Apr 10, 11:59 pm
  - **Your participation is highly valued!**
  - *Grade bonus for class participation!*

Quality Testing: Installability

- How might you test installability?

Bugs Per Lines of Code

Some numbers

- Indust avg: 15–50 errors per 1000 lines of delivered code
- Microsoft apps: 10–20 errors per 1000 lines in-house testing, 0.5/1000 for released product
- Harlan Mills: 3/1000 in-house testing, 0.1/1000 released product
- Space shuttle: 0/500 000

How do we estimate these numbers?

Fault Testing

- Common analogy involves planting fish in a lake to estimate the fish population
- T = total number of fish in the lake (to be estimated)
- N = fish stocked (marked) in the lake
- M = total number of fish caught in lake
- M' = number of marked fish caught
- T = (M - M')*N/M'
- Artificially seed faults, discover both seeded and new faults, estimate the total number of faults

Fault Testing Continued

- Method assumes that the real and seeded faults have the same distribution
- Hard to seed faults
  - By hand (not a great idea)
  - Independent testing by two groups and obtain the faults from one group for use by the other
- Want most of the discovered faults to be seeded faults
- If many faults are found, this is a bad thing
- The probability of errors is proportional to the number of errors already found

Fault Testing Usage Homework

After completion of a complex software project, two independent groups test it. The first group finds 300 errors and the second group finds 200 errors. A comparison of the errors discovered by the two teams shows that they found the same error in 50 cases. What is the range that estimates the number of errors in the original software project?

A. 900–1000

B. -17–25

C. 600–1500

D. 0–150

E. 150–250

Nonfunctional System Testing

- Stress testing: Determines if the system can function when subject to large volumes
- Execution: Determines if the system achieves the desired level of proficiency in production status (performance)
- Recovery: Determines if the system has the ability to restart operations after integrity has been lost
- Operations: Determines if the operating procedures and staff can properly execute the system (documentation)
- Compliance (to process): Determines if the system has been developed in accordance with information technology standards, procedures and guidelines
- Security: Determines if the system can protect confidential information

Functional System Testing

- Requirements: Determines if the system can perform its function correctly and that the correctness can be sustained over a continuous period of time
- Error Handling: Determines the ability of the system to properly process incorrect transactions
- Manual Support: Determines that the manual support procedures are documented and complete, where manual support involves procedures, interfaces between people and the system, and training procedures
- Inter-systems: Determines the that interconnections between systems function correctly

Functional System Testing Continued

- Control: Determines if the processing is performed in accordance with the intents of management
  - Includes data validation, file integrity, audit trail, backup and recovery, documentation and other aspects related to integrity
  - Controls are designed to reduce risks
- Parallel: Determines the results of the new application are consistent with the processing of the previous application or version of the application

Testing Phases

1. Unit testing
2. Integration testing
3. System testing
4. Acceptance testing

Theoretical Foundations Of Testing: Definitions

- P (program), D (input domain), R (output domain)
  - P: D $\rightarrow$ R (may be partial)
- Correctness defined by OR $\subseteq$ D $\times$ R
  - P(d) correct if $\langle$ d, P(d) $\rangle \in$ OR
  - P correct if all P(d) are correct
- Failure
  - P(d) is not correct
  - May be undefined (error state) or may be the wrong result
- Error (Defect)
  - Anything that may cause a failure
    - Typing mistake
    - Programmer forgot to test "x=0"
- Fault
  - Incorrect intermediate state entered by program

Definitions Questions

- A test case t is an element of D or R?
- A test set T is a finite subset of D or R?
- How would we define whether a test is successful?
- How would we define whether a test set is successful?

Definitions Continued

- Test case t: An element of D
- Test set T: A finite subset of D
- Test is successful if P(t) is correct
- Test set successful if P correct for all t in T

Theoretical Foundations of Testing

- Desire a test set T that is a finite subset of D that will uncover all errors
- Determining and ideal T leads to several undecideable problems
- No algorithm exists:
  - To state if a test set will uncover all possible errors
  - To derive a test set that would prove program correctness
  - To determine whether suitable input exists to guarantee execution of a given statement in a given program
  - etc.

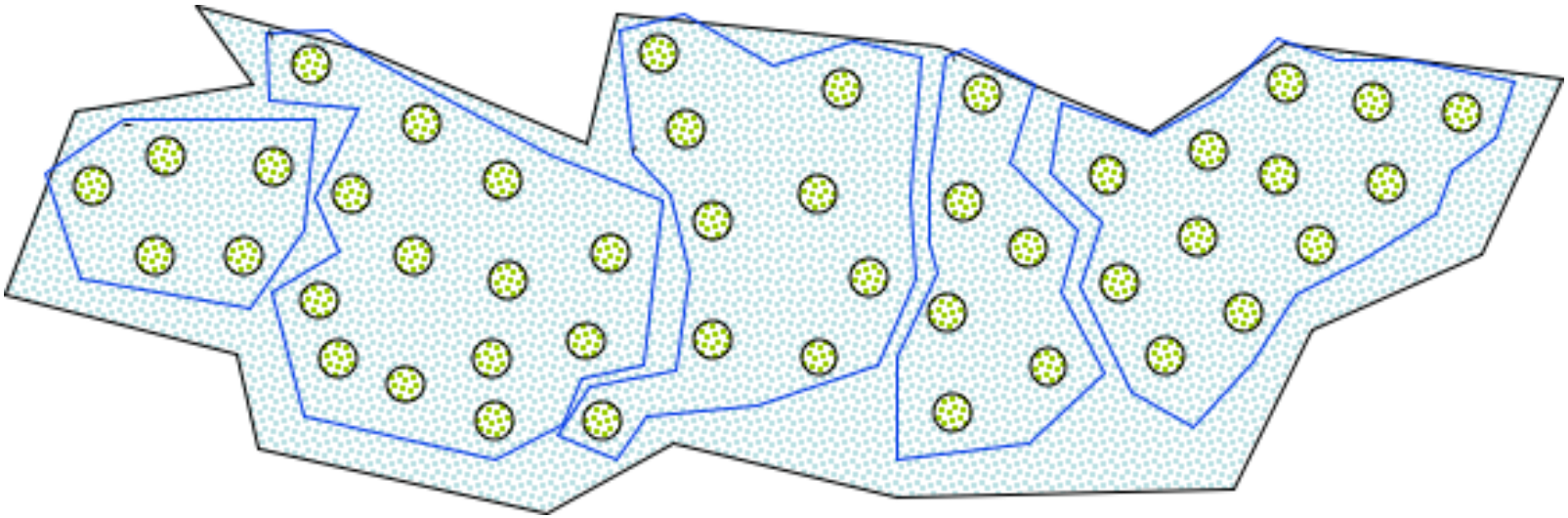
Empirical Testing

- Need to introduce empirical testing principles and heuristics as a compromise between the impossible and the inadequate
- Find a strategy to select significant test cases
- Significant means the test cases have a high potential of uncovering the presence of errors

Complete-Coverage Principle

- Try to group elements of D into subdomains D_1, D_2, ..., D_n where any element of each D_i is likely to have similar behaviour
- $D = D_1 \cup D_2 \cup ... \cup D_n$
- Select one test as a representative of the subdomain
- If $D_j \cap D_k = \emptyset$ for all $j \neq k$, (partition), any element can be chosen from each subdomain
- Otherwise choose representatives to minimize number of tests, yet fulfilling the principle

Complete-Coverage Principle



White-box Testing

- Intuitively, after running your test suites, what percentage of the lines of code in your program should be exercised?

White-box Coverage Testing

- (In)adequacy criteria - if significant parts of the program structure are not tested, testing is inadequate
- Control flow coverage criteria
  - Statement coverage
  - Edge coverage
  - Condition coverage
  - Path coverage

Statement-Coverage Criterion

- Select a test set T such that every elementary statement in P is executed at least once by some d in T
- An input datum executes many statements - try to minimize the number of test cases still preserving the desired coverage

Example

```
read (x); read (y);
if x > 0 then
        write ("1");
else
        write ("2");
end if;
if y > 0 then
        write ("3");
else
        write ("4");
end if;
```

How would you write a test case?

What is the minimum number of test cases?

Example

```
read (x); read (y);
if x > 0 then
        write ("1");
else
        write ("2");
end if;
if y > 0 then
        write ("3");
else
        write ("4");
end if;
```

{<x = 2, y = -3>, <x = - 13, y = 51>, <x = 97, y = 17>, <x = - 1, y = - 1>} covers all statements

{<x = - 13, y = 51>, <x = 2, y = - 3>} is minimal

Weakness of the Criterion

```
if x < 0 then
        x := -x;
end if;
z := x;
```

{<x=-3>} covers all statements. Why is this not enough?