

6.170 Lecture 9 Glass-box testing



John Chapin
MIT EECS
2-16-2000

Some material in this lecture is derived from a tutorial by Steve Cornett



Outline

Introduction to glass-box testing
5 coverage metrics
Testing as part of software development

© John Chapin/John Guttag

2



Black-box vs glass-box testing

Black-box

implementation hidden
includes:
specification coverage
random
boundary based on
possible implementations

may be designed before
writing code

Glass-box

implementation visible
includes:
test suite from BB
random

© John Chapin/John Guttag

3



Glass-box testing

Goal:

Ensure test suite covers (executes) all of the program
Measure quality of test suite with % coverage

Assumption:

high coverage =>
(no errors in test suite output
=> few mistakes in the program)

Focus: features not described by specification

Control-flow details
Performance optimizations
Alternate algorithms for different cases

© John Chapin/John Guttag

4



Glass-box challenges

Definition of all of the program

What needs to be covered?

Options:

Statement coverage
Decision coverage
Loop coverage
Condition/Decision coverage
Path-complete coverage

↑
increasing
number of
partitions
↓

Target % coverage

100% may be unattainable (dead code)
high cost to approach the limit

© John Chapin/John Guttag

5



Coverage metric depends on the application

RTCA DO-178B FAA standard for commercial aircraft

Level C: failure reduces safety margins
radio data link
statement coverage

Level B: failure reduces capability of the aircraft or crew
GPS, collision alert system
decision coverage

Level A: failure can cause loss of aircraft
engine controls, flight computer
modified condition/decision coverage

© John Chapin/John Guttag

6

6-170 Statement coverage

FAA level C

Measures:
whether each line of code has been executed

Advantage:
can be measured on object code

Limitation:

```

y = 0;
if (x > 0) {
  y = 5;
}
z = z+3/y;

```

SC reports 100% coverage even if x>0 in all tests
If statements without else are common

© John Chapin/John Guttag 7

6-170 Decision Coverage

FAA Level B

Measures:
whether all control-flow edges have been traversed

```

y = 0;
do {
  if (x > 0) {
    y = 5;
  }
  z = z+3/y;
} while (z < 10);

```

Watch out for caught exceptions!

© John Chapin/John Guttag 8

6-170 Decision coverage

Advantage:
relatively simple (compared to condition/decision)
reveals more control-flow errors than SC

Limitations:
Reports 100% even if while loops executed only once

Ignores branches within boolean expressions

```

if (b1 && (b2 || myfunction())) {
  // do something
}

```

reports 100% even if myfunction() never called here

© John Chapin/John Guttag 9

6-170 Loop coverage

Measures:
Whether all loop bodies executed 0, 1, and >1 times

Advantages:
Reveals the most common looping errors
e.g. failure to reinitialize variable
Can combine with SC, DC, or C/DC to strengthen
Can combine with PC to weaken

Limitations:
Assumes two or more iterations are equivalent
Loops expressed as recursive calls hard to measure automatically

© John Chapin/John Guttag 10

6-170 Condition/Decision coverage

FAA Level A

Measures:
Decision coverage + whether each boolean subexpression evaluates to both true and false

Advantage:
More complete control flow coverage

```

if (b1 && (b2 || myfunction())) {
  // do something
}

```

Limitations:
Very expensive
Unclear how much benefit beyond DC

© John Chapin/John Guttag 11

6-170 Path-complete coverage

Measures:
whether each path through the program has executed

Advantage:
thorough testing

Limitations:
of paths is exponential function of # of branches
Difficult to measure coverage automatically
e.g. Are there 2 or 4 paths in:

```

if (b1) statement1;
statement2;
if (b1) statement3;

```

Must prove effect of s1 and s2 on b1 to answer

© John Chapin/John Guttag 12

Summary of coverage metrics

Stronger metric subsumes the weaker metric
 All test suites that reach 100% on the stronger metric are at 100% for the weaker one

```

  graph TD
    Path-complete --> Decision
    Condition/Decision --> Decision
    Decision --> Statement_Basic_Block[Statement, Basic Block]
    Loop --> Decision
    Path-complete --> Loop
  
```

© John Chapin/John Guttag 13

Testing as part of software development

Unit and integration testing
Coverage goals
Testing strategy

© John Chapin/John Guttag 14

Unit and integration testing

Unit testing
 validate each procedure or ADT in isolation
 challenges:
 building stubs

Integration testing
 validate overall program
 challenges:
 achieving coverage
 specification errors
 debugging

© John Chapin/John Guttag 15

Coverage goals

100% is ideal, but:
 dead code
 arcane test cases required for last bits of coverage

Example tradeoff:
 write test cases to boost coverage from 90% to 95%
 OR
 do a formal technical review

Practitioner recommendation:
 90-95% coverage in unit test
 80-90% coverage in integration test
 (even FAA level A allows <100%, if explained)

© John Chapin/John Guttag 16

Testing strategy

Goals:
 detect errors with the least effort
 detect errors early enough to fix before release date

Strategy:
 breadth-first search (not depth-first)

Example integration test strategy:

1. Invoke at least one function in 90% of the classes
2. Invoke 90% of the functions
3. Achieve 90% decision coverage in 100% of functions

More later on large-scale programming and testing

© John Chapin/John Guttag 17

Basic block coverage (variant of SC)

Measures:
 whether each basic block has been executed

Advantage:
 avoids a reporting problem in statement coverage

```

  if (x != 0) {
    // 100 lines of straightline
    // code
  } else {
    x = -1;
  }
  
```

One test with $x = 0$ reports 1% statement coverage
 One test with $x \neq 0$ reports 99% SC

Basic block coverage reports 50% for each one

© John Chapin/John Guttag 18