Advanced Software Testing

Understanding Code Coverage

RBCS

TIME TESTED.
TESTING IMPROVED.

www.RBCS-US.com

Advanced Software Testing

- A series of webinars, this one excerpted from *Advanced Software Testing: V3*, a book for technical test analysts, programmers, and test engineers
- Black box techniques are useful for checking behavior
- Static analysis allows us to scrutinize the code to look directly for defects
- White box techniques are useful for checking code coverage

Control Flow Coverage

- Many different levels of control flow coverage have been identified
  - Statement (instruction, code) coverage: every statement in the module is executed at least once
  - Decision (branch) coverage: every decision in the module has been tested for both outcomes
  - Condition coverage: each condition making up a decision is evaluated at least once (with two additional variants common)
  - Multiple condition coverage: all possible combinations of outcomes for individual conditions within a decision are tested
- Let's look at these types of coverage…

Statement Coverage Testing

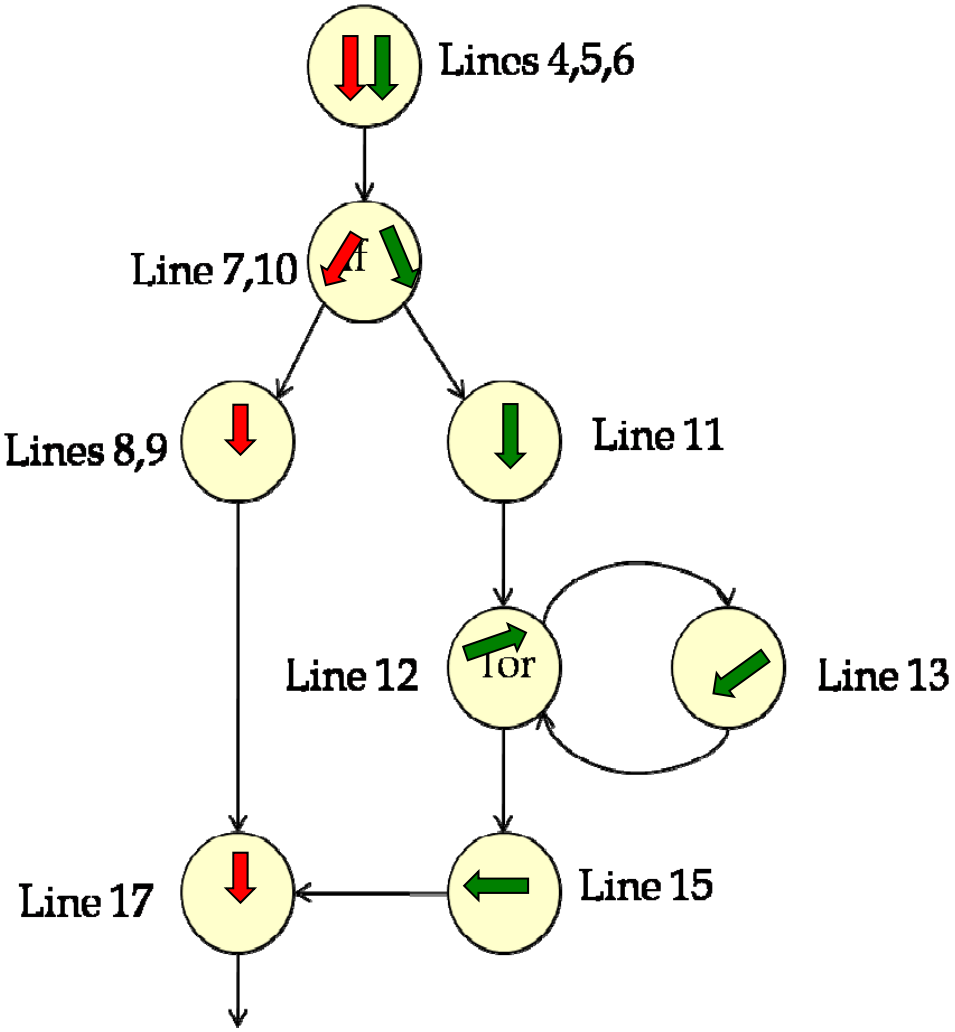
- Concept: executable statements are basis for test design selection
- Test derivation: identify test data to force execution of all statements
- Coverage criteria: number of statements executed / total number of statements
- Bug hypothesis: defects may exist in executable statements even though control flow is correct

Example: Statement Coverage

```
1 #include <stdio.h>
 2 main()
 3 {
 4   int i, n, f;
 5   printf("n = ");
 6   scanf("%d", &n);
 7   if (n < 0) {
 8     printf("Invalid: %d\n", n);
 9     n = -1;
10   } else {
11     f = 1;
12     for (i = 1; i <= n; i++) {
13       f *= i;
14     }
15     printf("%d! = %d\n", n, f);
16   }
17   return n;
18 }
```

Test value: **n < 0**

n > 0



Decision Coverage

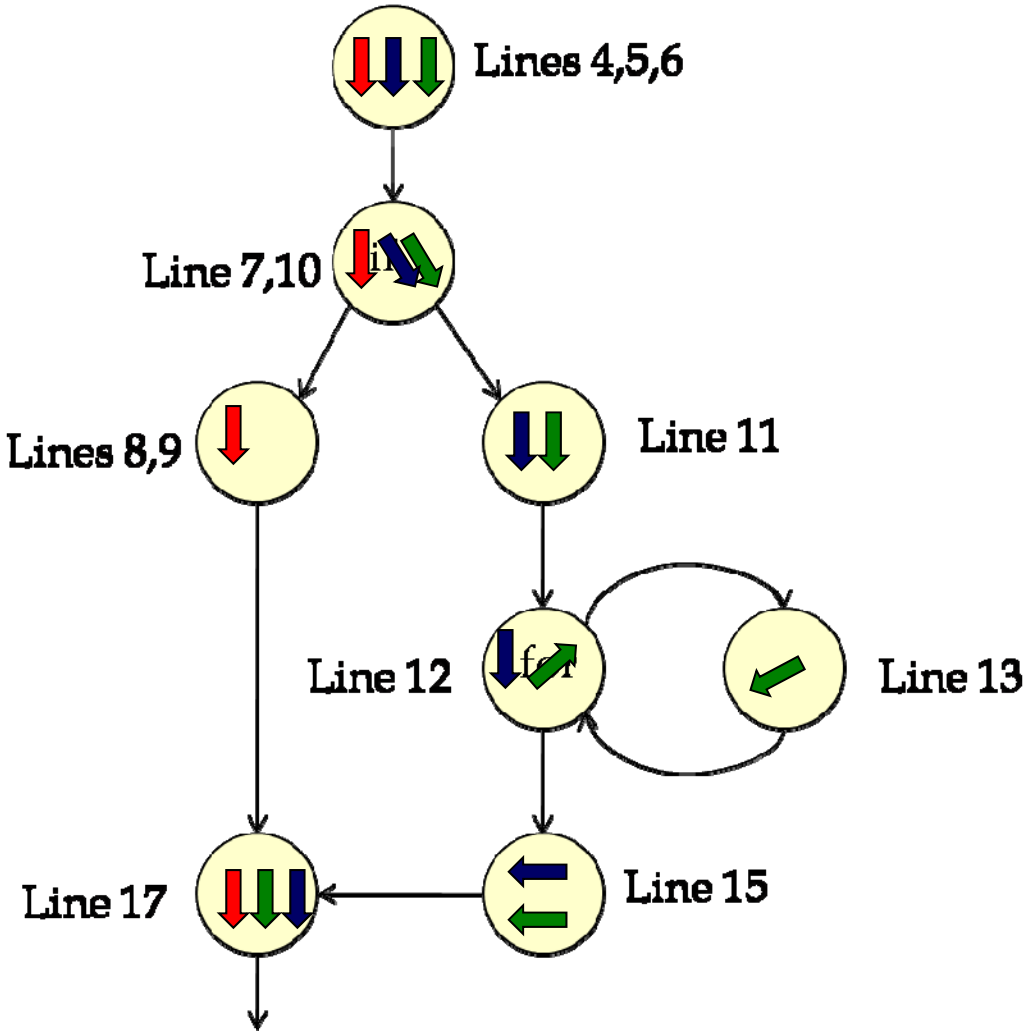
- Concept: control decisions are basis for test design selection
- Test derivation: identify data to force execution of each decision both ways (TRUE/FALSE)
- Coverage criteria: number of decision outcomes executed divided by total number of decision outcomes
- Bug hypothesis: defects may exist in untested decisions or branches
- 100% decision coverage gives 100% statement coverage

Example: Decision Coverage

```
1 #include <stdio.h>
 2 main()
 3 {
 4   int i, n, f;
 5   printf("n = ");
 6   scanf("%d", &n);
 7   if (n < 0) {
 8     printf("Invalid: %d\n", n);
 9     n = -1;
10   } else {
11     f = 1;
12     for (i = 1; i <= n; i++) {
13       f *= i;
14     }
15     printf("%d! = %d\n", n, f);
16   }
17   return n;
18  }
```

Statement coverage: We tested (**n < 0**) and (**n > 0**)

To get decision coverage, we still need (**n == 0**)



Loop Coverage

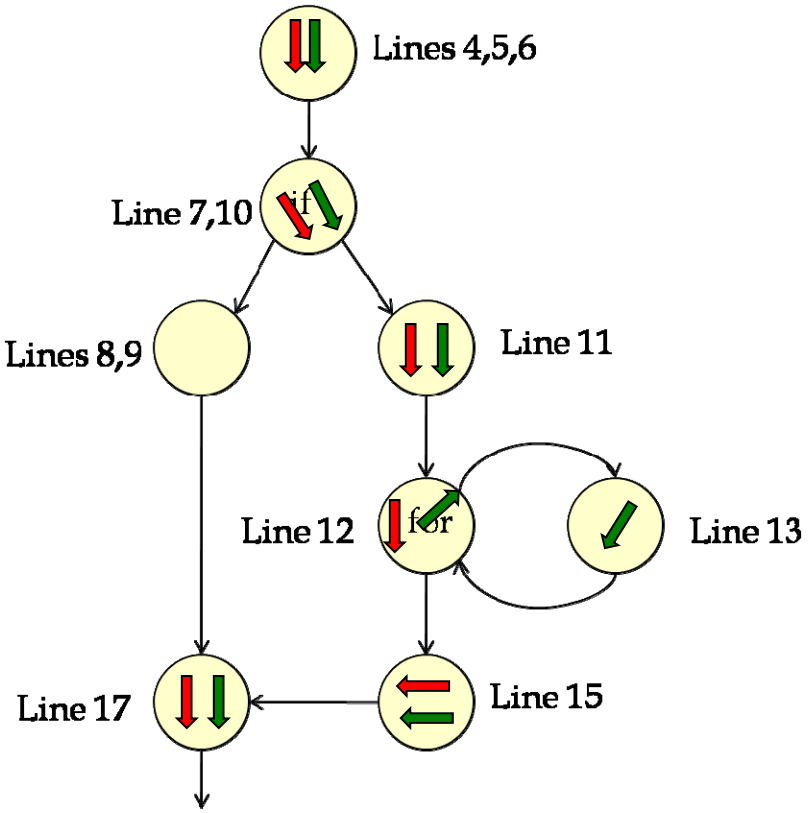
- Concept: because the number of paths when there is a loop may be close to infinite, we want to test a representative number of times through the loop
- Test Derivation: identify data that causes system to loop a finite number of different times
- Coverage criteria: tests such that the system loops 0 times, 1 time, n times where n equals the expected max number of loops (if possible)
- Bug hypothesis: failing to loop (or looping too often) may cause failures

Example: Loop Coverage (0, 1 Loops)

```
1 #include <stdio.h>
 2 main()
 3 {
 4   int i, n, f;
 5   printf("n = ");
 6   scanf("%d", &n);
 7   if (n < 0) {
 8     printf("Invalid: %d\n", n);
 9     n = -1;
10   } else {
11     f = 1;
12     for (i = 1; i <= n; i++) {
13       f *= i;
14     }
15     printf("%d! = %d\n", n, f);
16   }
17   return n;
18 }
```

Loop 0 times by testing **(n=0)**

Loop 1 time by testing **(n = 1)**



Condition Coverage

- Concept: each atomic condition must be evaluated in tests both ways (T/F)
- Test Derivation: identify data to force execution of each atomic condition both ways
- Coverage criteria: number of Boolean operand values executed divided by the total number of Boolean operand values
- Bug hypothesis: defects may exist in untested atomic conditions
- Atomic condition: the simplest form of code that can result in a TRUE or FALSE outcome *--Bath & McKay*

Atomic Conditions

- A decision is made up of one or more atomic conditions
- Examples:

x → TRUE

-- There is one atomic condition

D && F → FALSE

-- There are 2 atomic conditions

(A || B) && (C == D) → TRUE

-- There are 3 atomic conditions [**A, B, (C==D)**]

(a>b)||(x+y==-1)&&((d)!=TRUE) → TRUE

-- There are 3 atomic conditions

- In each case, there is still only one decision for the whole expression

Condition Testing

- The rule of condition testing is to ensure that each atomic condition has been evaluated to both TRUE and FALSE in testing
- In the previous slide:

x, D, F, A and B must all be tested TRUE and FALSE

(C == D) must be tested both TRUE and FALSE

(a > b) must be tested both TRUE and FALSE

(x+y==-1) must be tested both TRUE and FALSE

((d)!=TRUE) must be tested both TRUE and FALSE

Decision/Condition Coverage

- Concept: each atomic condition must be evaluated in a test both ways (T/F) **AND** decision coverage must also be satisfied
- Test derivation: identify data to force execution of each atomic condition both ways and add tests to ensure decision coverage
- Coverage criteria: must include both condition and decision coverage
- Bug hypothesis: testing all conditions alone may not be sufficient

Example: Decision/Condition Coverage

- Consider the following code snippet:

  if (a AND b) then…

- Condition coverage is satisfied with

  a==FALSE, b==TRUE → FALSE
  a==TRUE, b==FALSE→ FALSE

- Decision/Condition coverage is satisfied by adding

  a==TRUE, b==TRUE → TRUE

- Is that sufficient testing?

Modified Condition/Decision Coverage

- Concept: each atomic condition must be evaluated in a test both ways (T/F) **AND** decision coverage must also be satisfied **AND** each condition must affect the outcome decision independently
- Test derivation: identify data to force execution of each atomic condition both ways **AND** add tests to ensure decision coverage **AND** ensure each condition affects the decision outcome independently
- Coverage criteria: see next slide
- Bug hypothesis: testing all decisions and conditions alone may not be sufficient

MC/DC Coverage

- MC/DC is achieved when:
  1. Each decision tries every possible outcome
  2. Each condition in a decision takes on every possible outcome
  3. Each entry and exit point is invoked
  4. Each condition in a decision is shown to independently affect the outcome of the decision while holding fixed all other possible conditions
- When the programming language uses short circuits decision making, testing MC/DC is problematic

  if (a | | b) {} → if **a** is TRUE, **b** is never evaluated

Achieving MC/DC Coverage

- Given that definition, how do we achieve MC/DC coverage?
- Consider the following code snippet:

  if ((a OR b) AND c) then…

- Decision/Condition coverage can be achieved with

  a==TRUE, b==TRUE, c==TRUE → TRUE

  a==FALSE, b==FALSE, c==FALSE → FALSE

- Notice that b never independently affects the outcome of the decision

Achieving MC/DC Coverage

- The following test set will achieve the MC/DC coverage
  1. **a==TRUE, b==FALSE, c==TRUE → TRUE**
  2. **a==FALSE, b==TRUE, c==TRUE → TRUE**
  3. **a==TRUE, b==TRUE, c==FALSE → FALSE**
- Note that the decision would be reversed:
  - In test 1, if **a** changed with b and c held constant
  - In test 2, if **b** changed with a and c held constant
  - In test 3, if **c** changed with a and b held constant

Multiple Condition Coverage

- Concept: exhaustively test every single combination of conditions that can affect an outcome
- Test derivation: use Boolean truth table to generate all possible combinations of values that must be tested
- Coverage criteria: number of Boolean operand value combinations executed / total number of Boolean operand value combinations
- Bug hypothesis: bugs could be anywhere!

Conclusion

- In this webinar, we've seen how to apply code coverage to understand what's been tested…and what hasn't
- In our previous webinars, we looked at various test techniques, including behavior-based
- Code coverage can allow us to measure the degree to which these tests exercise the code
- Thus, code coverage can build confidence in your tests

…Contact RBCS

For over a dozen years, RBCS has delivered services in consulting, outsourcing and training for software and hardware testing. Employing the industry's most experienced and recognized consultants, RBCS conducts product testing, builds and improves testing groups and hires testing staff for hundreds of clients worldwide. Ranging from Fortune 20 companies to start-ups, RBCS clients save time and money through improved product development, decreased tech support calls, improved corporate reputation and more. To learn more about RBCS, visit www.rbcs-us.com.

Address:	RBCS, Inc. 31520 Beck Road Bulverde, TX 78163-3911 USA
Phone:	+1 (830) 438-4830
Fax:	+1 (830) 438-4831
E-mail:	info@rbcs-us.com
Web:	www.rbcs-us.com