Understanding White-Box Coverage

Better Testing Through Inside Knowledge

RBCS

TIME TESTED.
TESTING IMPROVED.

www.RBCS-US.com

Introduction

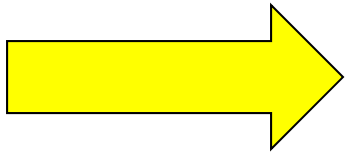
- With the prevalence of continuous integration frameworks, white-box coverage metrics are back in style
- Black-box techniques are useful for checking behavior
- Static analysis allows us to scrutinize the code to look directly for defects
- White-box techniques are useful for checking code coverage

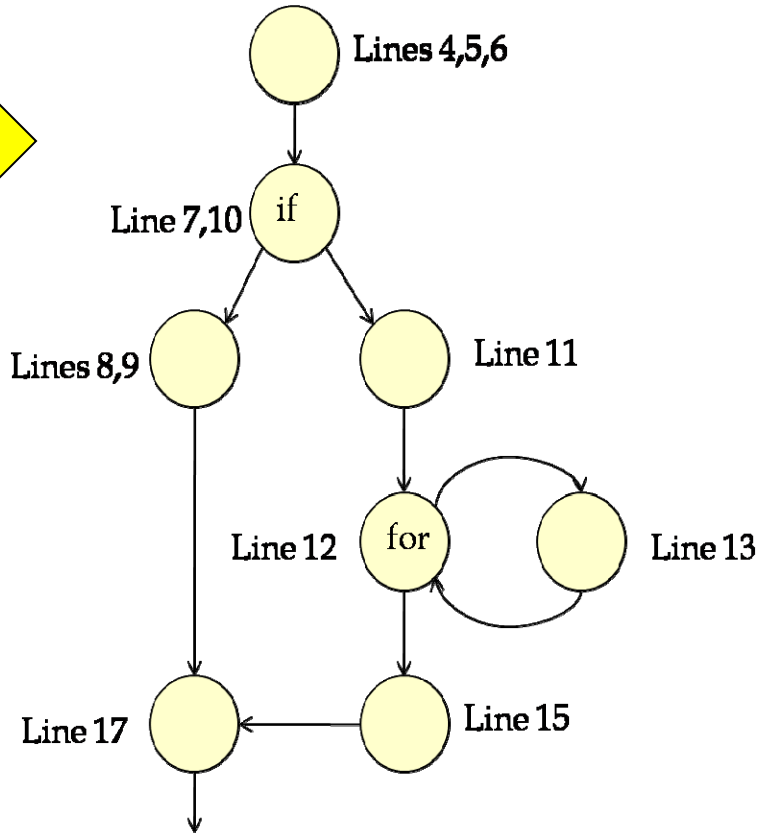
Control Flow Coverage

- Statement (instruction, code): every statement executed at least once
- Decision (branch): every possible outcome (T, F)
- Condition: each atomic condition once T, once F
- Decision/condition : condition and decision coverage
- Modified condition/decision: each condition alone determines the outcome twice (once T, once F)
- Multiple condition: all possible combinations of condition in a decision

Code to Control Flow Example

```
 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 }
```



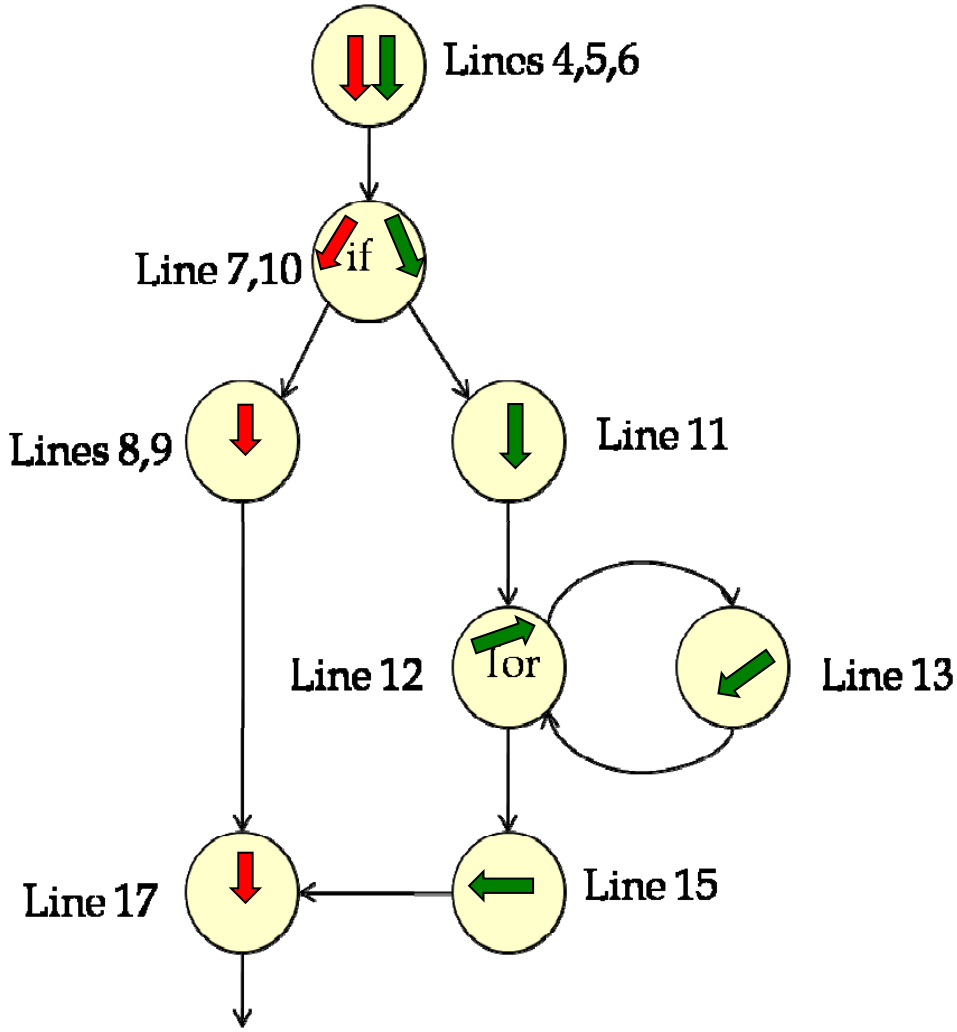


Example: Statement Coverage

```
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



What is a Decision?

- The ability to decide to take one path rather than another is the real power of a computer
- The decision may be a simple Boolean expression

  a > b

- Or, it might be arbitrarily complex

  (a>b) || (x+y==-1) && ((d) != TRUE)

- Different programming languages have different constructs for making decisions
  - if then (else)
  - while
  - until do
  - for
  - switch/case
  - and lots more…

Example: Decision Coverage

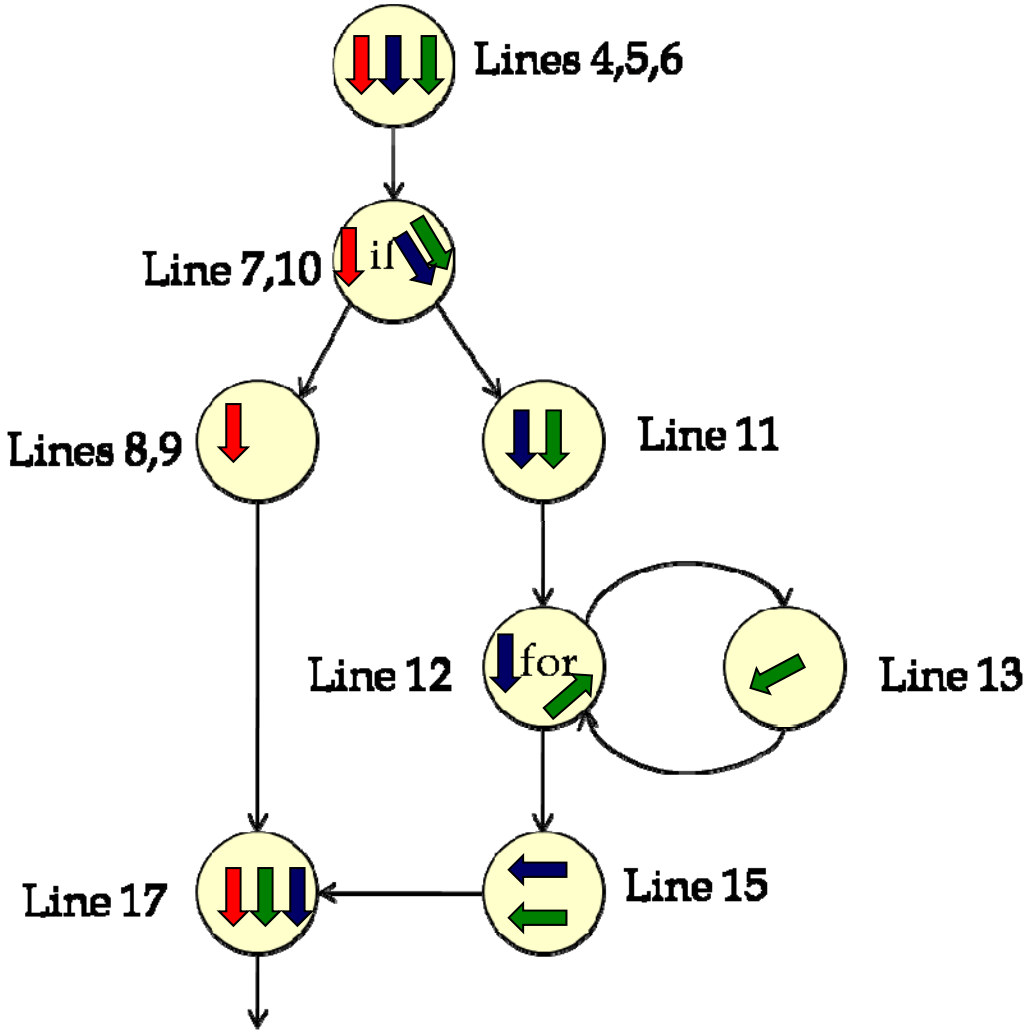
```
1 #include <stdio.h>
 2 main()
 3 {
 4   int i, n, f;
 5   printf("n = ");
 6   scanf("%d", &n);
 7   if (n < 0) {
 8     printf("Invalid: %d\n", n);
 9     n = -1;
10   } else {
11     f = 1;
12     for (i = 1; i <= n; i++) {
13       f *= i;
14     }
15     printf("%d! = %d\n", n, f);
16   }
17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

Statement coverage: We tested

(**n < 0**) and (**n > 0**)

To get decision coverage,

we still need (**n == 0**)



Condition Testing

- Decision testing looks at the overall outcome
- Condition testing looks at atomic conditions
- Each atomic condition must be at least once true and once false
- Applicability: mostly as part of understanding higher levels of coverage
- Limitations/difficulties: condition coverage doesn't always lead to decision coverage
- If a decision is based on a single atomic condition, condition testing is identical to decision testing

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 two atomic conditions

(A || B) && (C == D) → TRUE

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

(a>b)||(x+y==-1)&&((d)!=TRUE) → TRUE

-- There are three 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 Testing

- Testing condition coverage and decision coverage
- May not require additional tests beyond condition coverage
- Applicability: Important but not critical code
- Limitations/difficulties: Sometimes requires more tests than the decision level, which can exacerbate a time-crunch

Stronger Than Condition or Decision Alone

- Let's look at the decision process itself
- A decision may be a simple Boolean expression

a > b

- Or, it might be arbitrarily complex

(a > b) || (x+y == -1) && (d) != TRUE

- To achieve decision coverage, we only ever need two test cases; a test where the decision evaluates to TRUE and one FALSE

Example: Decision/Condition Coverage

- Consider the following code snippet again:

  if (a AND b) then…

- Condition coverage is satisfied with

  a == FALSE, b == TRUE → FALSE

  a == TRUE, b == FALSE → FALSE

- Decision/Condition coverage achieved by adding

  a == TRUE, b == TRUE → TRUE

- Is that sufficient testing?

Modified Condition/Decision Coverage

- MC/DC is decision/condition coverage plus at least one test where each atomic condition controls the decision outcome
- As a general rule, given N atomic conditions, requires N+1 tests
- Applicability: used in aerospace and many other safety-critical systems
- Limitations/difficulties: not always achievable when conditions are coupled or when the compiler short-circuits (more later)

Systematic MC/DC Process

1. Draw a table with three columns, with the logical expression and the two possible outcomes as shown
2. Add a blank row for each condition, with the number of blanks equal to the total number of conditions
3. Fill the diagonal blanks in the second column with "T" and in the third column with "F"
4. Replace the remaining blanks with neutral values (F for OR, T for AND)
5. Delete any duplicate tests from the cells

a OR b	**T**	**F**
a	_ _	_ _
b	_ _	_ _

a OR b	**T**	**F**
a	T _	F _
b	_ T	_ F

a OR b	**T**	**F**
a	T F	F F
b	F T	F F

a OR b	**T**	**F**
a	T F	F F
b	F T	*Del*

Example: 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, so we don't have MC/DC coverage here
- Let's try the process…

Example: Achieving MC/DC

1. Draw a table with three columns, with the logical expression and the two possible outcomes as shown
2. Add a blank row for each condition, with the number of blanks equal to the total number of conditions
3. Fill the diagonal blanks in the second column with "T" and in the third column with "F"
4. Replace the remaining blanks with neutral values (F for OR, T for AND); careful with compound expressions like this one!
5. Delete any duplicate tests from the cells

(a OR b) AND c	T	F
a	_ _ _	_ _ _
b	_ _ _	_ _ _
c	_ _ _	_ _ _

(a OR b) AND c	T	F
a	T _ _	F _ _
b	_ T _	_ F _
c	_ _ T	_ _ F

(a OR b) AND c	T	F
a	T F T	F F T
b	F T T	F F T
c	T F T	T F F

(a OR b) AND c	T	F
a	T F T	F F T
b	F T T	*Del*
c	*Del*	T F F

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
- 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
- Be careful when using tools, because some of these report these metrics differently, and some have bugs

…Contact RBCS

For over twenty 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
E-mail:	info@rbcs-us.com
Web:	www.rbcs-us.com
Twitter:	@RBCS, @LaikaTestDog
Facebook:	RBCS-Inc

