Code Coverage Metrics

And How to Use Them

```
int main(int argc, char* argv[])
{
        long int i, n=0;
        ubcd pp, p, c;

        if (argc > 1) {
                n = atol(argv[1]);
        } else {
                cout << "Enter element count: ";
                cin >> n;
        }

        if (n < 0) {
                cout << "No " << n << " series!\n";
                n = -1;
        } else {
                cout << "{ ";
                pp = 0;
                p = 1;
                for (i=0; i < n; i++) {
                        c = pp + p;
                        cout << c.csrep() << ' ';
                        pp = p;
                        p = c;
                }
                cout << "} \n";
        }

        return n;
}
```


A Hands-on Demonstration

Introduction

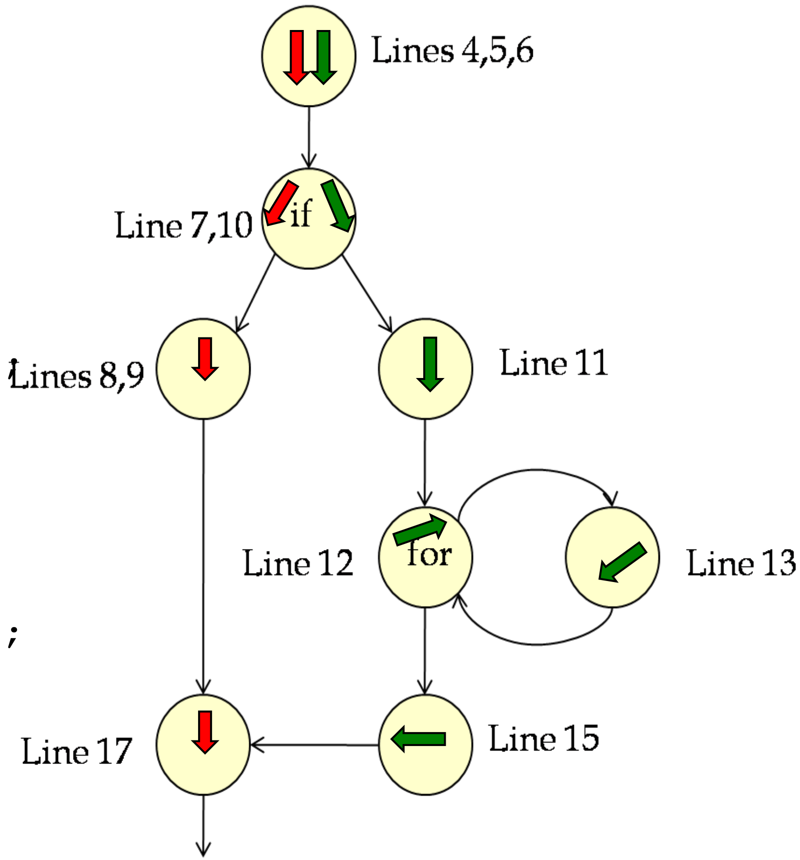
- More and more testers and programmers use tools that provide code coverage metrics
  - How do these tools work?
  - What do these metrics tell us?
- Some testers and programmers use tools that measure code maintainability
  - How do these tools work?
  - What do these metrics tell us?
- Let's look at three examples to examine these questions
  - Statement and branch coverage metrics
  - McCabe Cyclomatic complexity and basis path coverage
- I'll illustrate with a real program, along with open source tools, using a Virtual Box environment

Statement Coverage Testing

- Concept: executable statements are basis for test design and selection
- Test derivation: identify test data to force execution of all statements
- Coverage criterion: (number of statements executed / total number of statements) = 100%
- 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 values: **n < 0, n > 0**

Decision Coverage

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

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

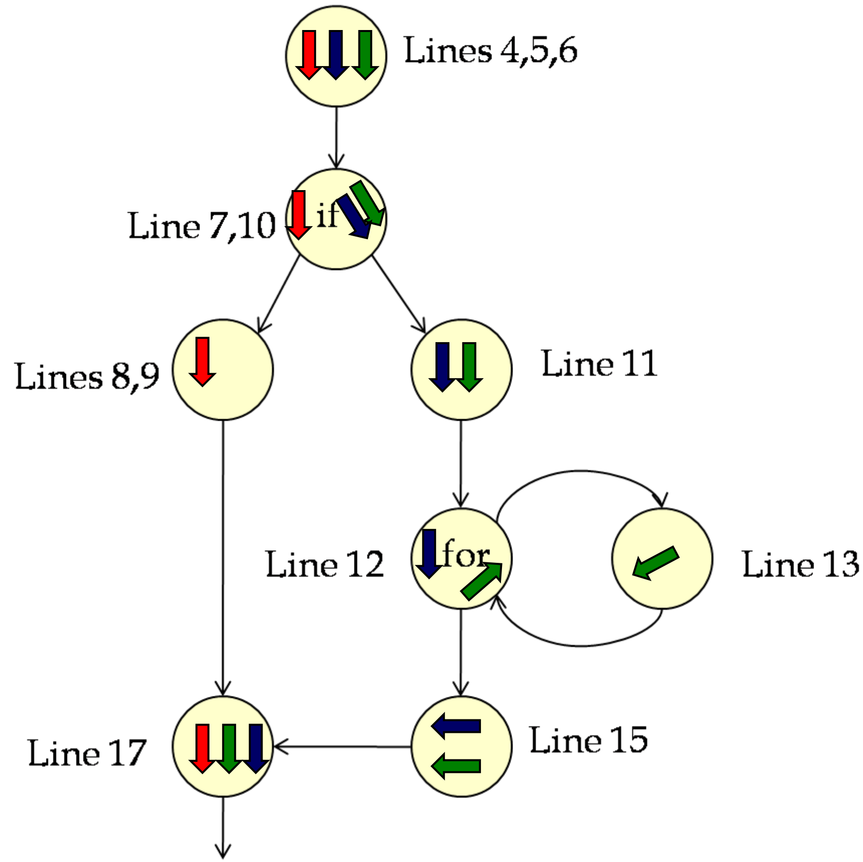
  (a>b) || (x+y==-1) && ((d) != TRUE)

- 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 }
```

Test values: **n < 0**, **n > 0**, **n == 0**



RBCS
TIME TESTED. TESTING IMPROVED.
www.RBCS-US.com

Demo: Control Flow Coverage

- Compile WordTree with code coverage enabled

  g++ -Wall -fprofile-arcs -ftest-coverage -o wordtree wordtree.cpp driver.cpp

- Run tests designed to achieve 100% equivalence partition and boundary value coverage

- Let's see if we obtain complete statement and decision coverage

  gcov wordtree.cpp | less

  less wordtree.cpp.gcov

McCabe Cyclomatic Complexity

RBCS
TIME TESTED. TESTING IMPROVED.
www.RBCS-US.com

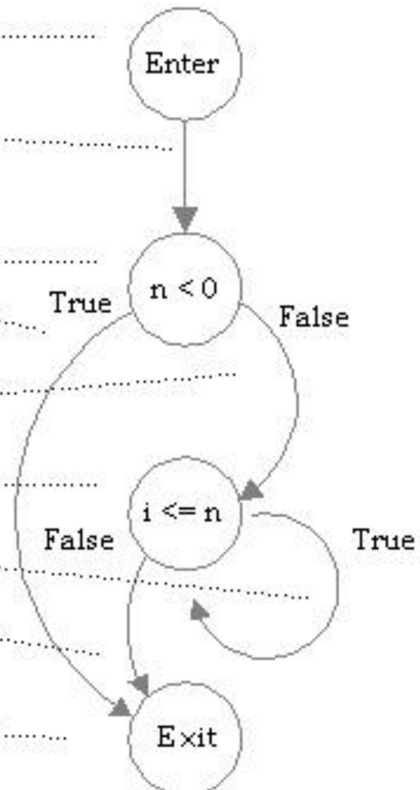
- McCabe's Cyclomatic Complexity measures control flow complexity
  - Measured by drawing a directed graph
  - Nodes represent entries, exits, decisions
  - Edges represent non-branching statements
- It has some useful testing implications
  - High-complexity modules are inherently buggy and regression-prone
  - The complexity is approximately equal to the number of tests you'll need to run to get decision coverage
- Let's see how…

Cyclomatic Complexity for Factorial

Program

```
main()
{
 int i, n, f;
 printf("n = ");
 scanf("%d", &n);
 if (n < 0) {
  printf("Invalid: %d\n", n);
  n = -1;
 } else {
  f = 1;
  for (i = 1; i <= n; i++) {
   f *= i;
  }
  printf("%d! = %d.\n", n, f);
 }
 return n;
}
```

Flow Diagram



Cyclomatic Complexity

C = #R + 1 =
2 + 1 = 3

or

C = #E - #N + 2 =
5 - 4 + 2 = 3

Complexity Rule of Thumb

- Calculating the Cyclomatic Complexity graphically can be tedious and error prone
- A useful rule of thumb: Count the branching and looping constructs and add 1
  - For if/else, use 1 for each if (ignore else)
  - For switch/case, count the case blocks (ignore default)
- This rule of thumb has proven accurate for me, but there might be a situation where it doesn't give the exact answer (but it'll be close)

RBCS
TIME TESTED. TESTING IMPROVED.
www.RBCS-US.com

Demo: Cyclomatic Complexity

- Use the pmccabe program to calculate the Cyclomatic Complexity for each of the member functions in the Tnode and WordTree classes

  pmccabe -v wordtree.cpp

- Use the pmccabe program to calculate the Cyclomatic Complexity for the triangle program

  pmccabe -v triangle.c

- Check to see if the calculations are correct using the rule of thumb

Conclusion

- Code coverage can tell us what we've tested, and what we haven't tested
- We won't find all the bugs in the code we have tested, but we can't find bugs in any of the code we haven't tested
- Complexity can give us an idea of how hard it will be to maintain our code
- As with code coverage, not perfect, but useful
- I hope you find these techniques useful tools in your work as a tester or programmer

...Contact us

For over 25 years, RBCS has delivered consulting, training, and expert services to clients, helping them with software and hardware testing. Employing the industry's most experienced and recognized consultants, RBCS advises its clients, trains their employees, 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, @MisterSDET, @LaikaTestDog
Facebook:	@TestingImprovedbyRBCS
LinkedIn:	https://www.linkedin.com/in/rex-black
YouTube:	https://www.youtube.com/user/RBCSINC