

# Software Test

---

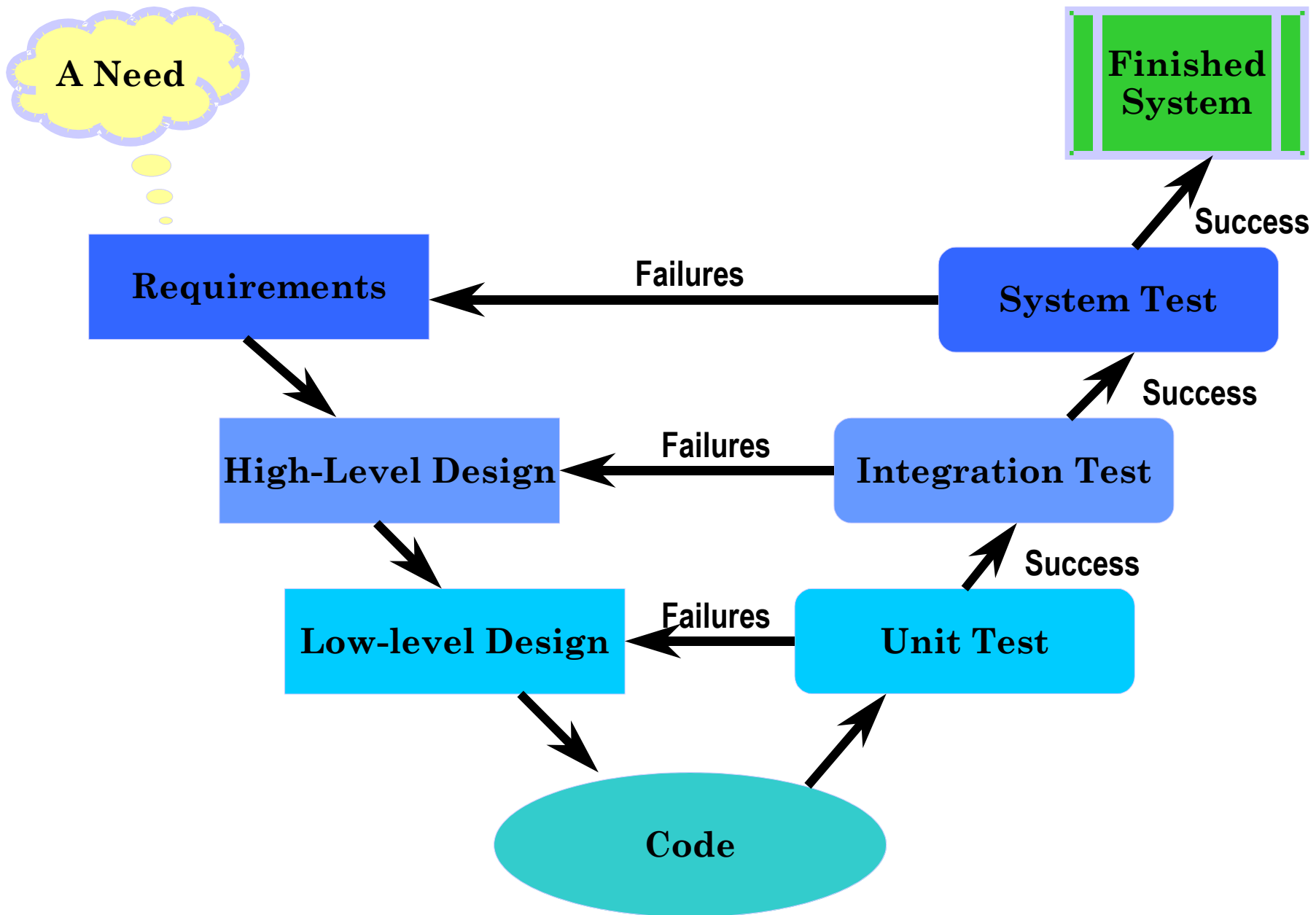
- Types of test
  - System test
  - Integration test
  - Unit test
- Types of test
  - Black box
  - White box
- Regression test
- The Junit tool for unit testing Java programs

# Introduction

---

- „Software test is the process of executing a program with the intent of find errors“ [Glen Myers].
  - „This should not be done by customers“ [K.Torp]
- Software test is an activity taking place after the program has been implemented and before the program is debugged.
  - Program implementation is an activity where design is converted to actual source code.
  - Debugging is an activity where the causes of program malfunctions are found.
- Software testing is a very resource and time consuming activity
- A test can show that there is an error in a program. However, it can never show that program is error free!

# Unit, Integration, and System Tests



# Problems Related to Testing

---

- Test is done late in a project
  - If it is done at all, „it takes too long time to write test cases!“
- Test cases gets outdated
  - Test cases are for version 1.0 of the system which is currently now release in version 5.2.
- Test is incomplete
  - Not all parts of the program are tested

# Advantages of Testing

---

- Fewer bugs
  - Less annoying product patching, more interesting development
- A better and more robust product
  - Users are more likely to buy more software from your company
- Better sleep at night
  - You know there are no obvious bugs in your program
- More proud of your company and its products
  - "See Darling, what I have built!"
- Faster to release new versions of the product
  - Building test cases is not a *detour* it is a *short-cut*!
  - "Test More, Spend Less!"
- More aggressive changes of system
  - Refactoring to build the best system possible

# Disadvantages of Testing

---

- None!
- Absolutely none!
- Absolutely, definitively none!
- Absolutely, definitively, surely none!
- Absolutely, definitively, surely, conclusively none!
- Absolutely, definitively, surely, conclusively, decisively none!
- Absolutely, definitively, surely, conclusively, decisively, determinatively none!
- Absolutely, definitively, surely, conclusively, decisively, determinatively, positively none!
- Ran out of synonyms!

# Testability

---

- Observability
  - The results are easy to see
  - Different outputs are generated for different inputs
  - Incorrect outputs are easy to identify
- Controllability
  - Processing can be controlled (e.g., wall clock)
  - Running test cases can be automated (and easily repeated)
- Decomposability
  - Modules can be tested individually
- Simplicity
  - No huge and complex modules in the system
- Understandability
  - The design of the system can be easily communicated from implementor to tester

# Testability, cont.

---

- Testability must be taken into account during the design fase
- High testability
  - `cos(x)`
  - `integer2string(int)` return string
  - `insert(Object)`
- Low testability
  - GUI, low controllability because hard to automate
  - Recovery system of database management system
    - ◆ Hard to thing of all possible error (low controllability, inputs hard to guess)
    - ◆ The system is highly complex (low simplicity, low understandability)
    - ◆ Extremely badly if it does not work correct



# Test Units

---

- Single expression or statement
- Method
- Class
- Package
- Entire program

# Black Box Test

---

- Look at the program from the outside, no knowledge of the internals
  - Are requirement fulfilled
  - Are interfaces available and working
  - Can also reveal performance problems with the program
    - ◆ Takes too long to compute
    - ◆ Consumes too much main memory or disk space
- Can be applied to all levels of testing (unit, integration, system)
- Can be done by independent testers or even customers
- It is impossible to test for all possible inputs!

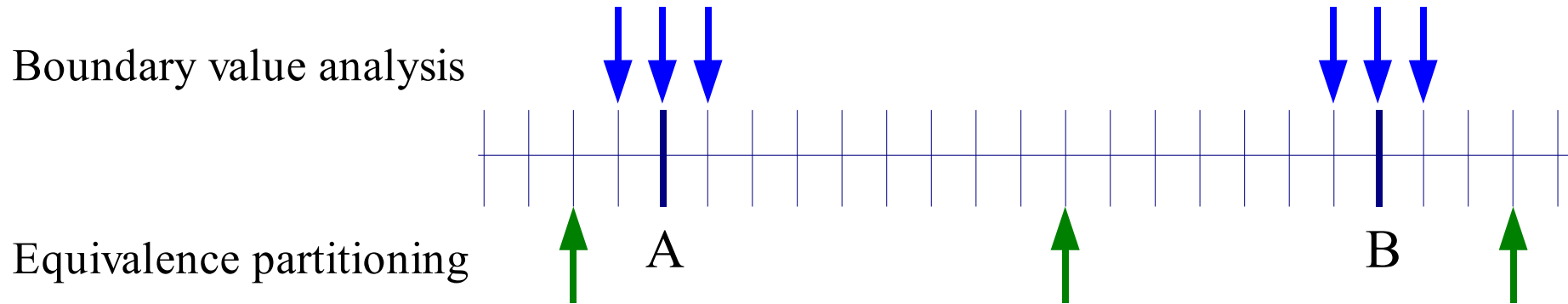
# Black Box Test, cont.

---

- The challenge in black box testing is picking the input values
- Equivalence partitioning (input domain partitioning)
  - Partition input space into a small number of *equivalent classes* all elements in one class should be handled identically by the program
- Boundary value analysis
  - A technique on identifying test cases to explore boundary conditions
  - At boundaries there are typically many errors

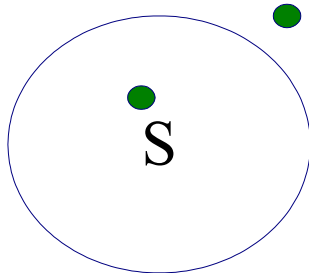
# Black Box Test, Examples

- Look at an interval  $[A..B]$

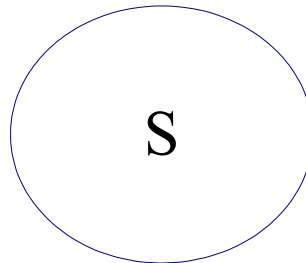


- Look at a set  $S$

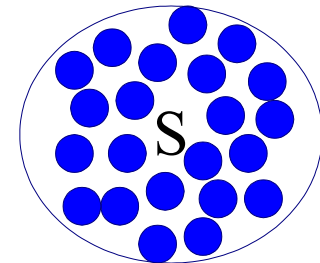
Equivalence partitioning



Boundary value analysis  
(empty)



Boundary value analysis  
(full)



# White-Box Test

---

- Look inside the program
  - Conditions and path taking within the program
  - Data flow
- Can be applied to level unit test
- Can be done by the developers or in house
- Block-box testing can be considered a sub set of white-box testing for unit testing
- It is impossible to check all paths a program can take internally.

# White-Box Test, cont.

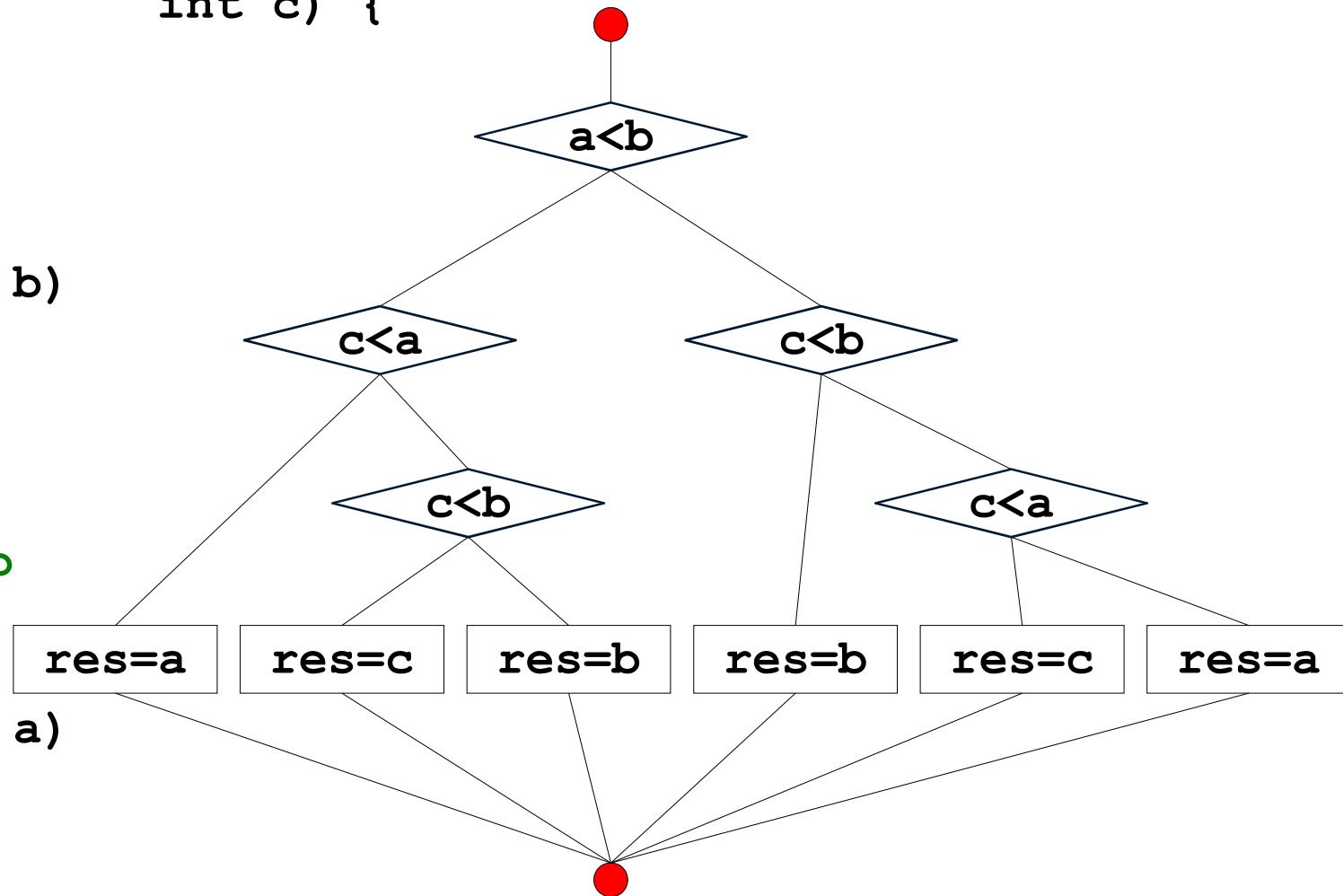
---

- The challenge in white box testing is to exercise each line of code in the unit being tested at least once.
- A number of techniques exists for this
  - Basis path coverage also called *Cyclomatic Complexity*
  - Logical coverage
  - Dataflow coverage
- Basis path coverage is based on finding the basis set of paths for a programs path space.
- Defines the number of independent paths in the basis set
- *Path coverage set* is the set of paths that will execute all statements and all conditions in a program at least once
  - Are not unique

# White-Box Test, Example

```
public static int mid(int a,  
                     int b,  
                     int c) {
```

```
    int res;  
    if (a < b){  
        if (c < a)  
            res = a;  
        else if (c < b)  
            res = c;  
        else  
            res = b;  
    }  
    else{ // a >= b  
        if (c < b)  
            res = b;  
        else if (c < a)  
            res = c;  
        else  
            res = a;  
    }  
    return res;  
}
```



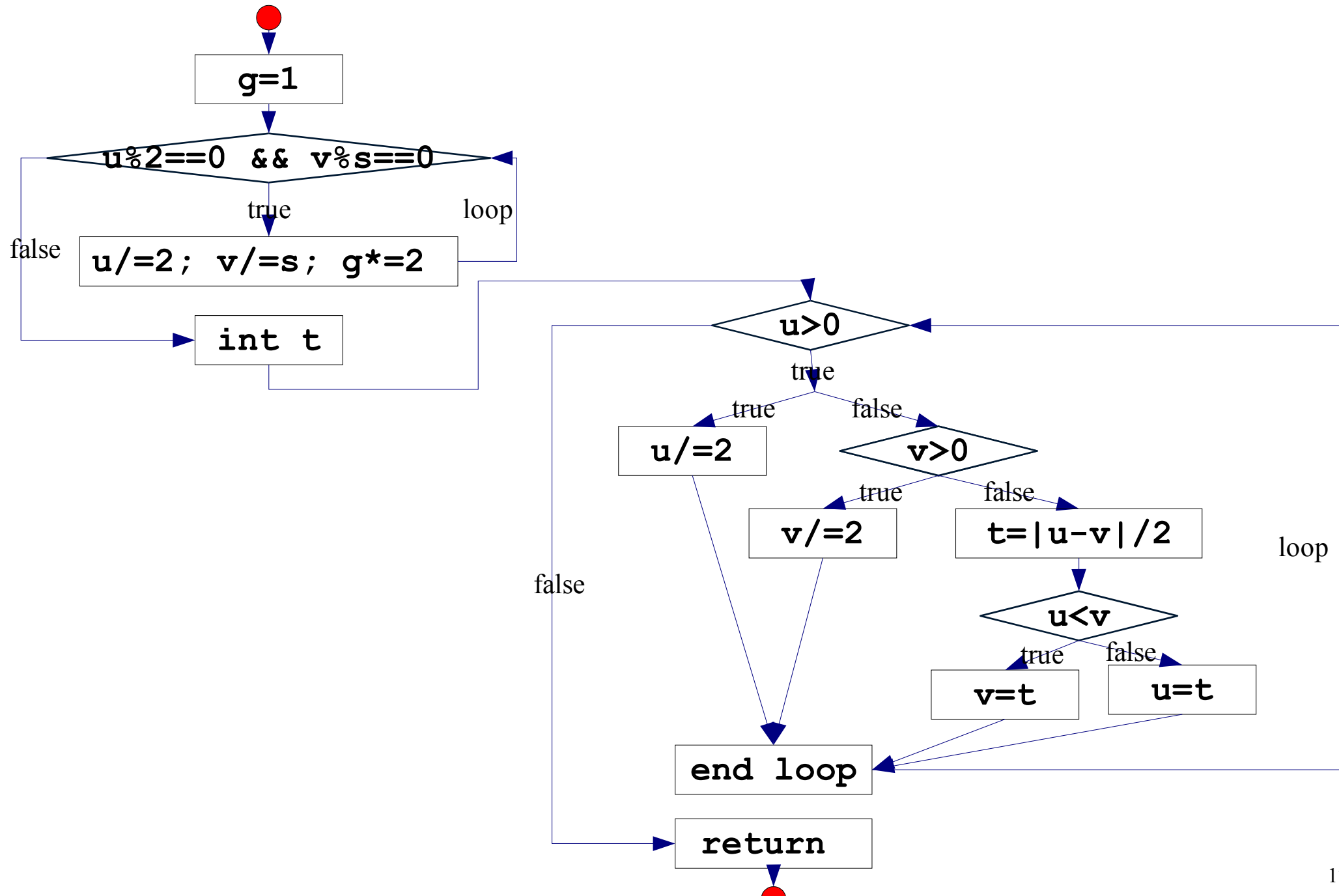
# White-Box Test, Another Example

---

```
// greatest common divisor
public static int gcd(int u, int v){
    int g = 1;
    while(u%2==0 && v%2==0){ // u is even and v is even
        u /= 2; // right shift
        v /=2;  // right shift
        g *= 2; // (left shift)
    }
    // now u or v (or both) are odd
    int t;
    while (u > 0){
        if (u%2==0){ u /= 2; } // u is even, u = u/2
        else if (v%2==0){ v /= 2;} // v is even, v = v/2
        else{
            t = java.lang.Math.abs(u-v)/2; //t = |u-v|/2
            if (u < v) {v = t;}
            else      {u = t;}
        }
    }
    return g*v;
}
```



# White-Box Test, Another Example, cont.



# Test Cases

---

- Describe how to test a unit (system/module/method/statement)
- Must include
  - System state before execution of the test
  - Part of system tested
  - Input to the run test
  - Expected outcome of the test (system state, output to screen, etc.)
- Good test cases will find bugs
- Good test cases are based on requirement specification

# Regression Test

---

- Regression testing is a technique not a tool
- After a change to a piece of code two things must be tested
  - That the changed piece of code works properly
  - That the functionality of the entire system other than the change piece of code is unaffected
- It is very time consuming to test the two things above
  - Automate
  - modularization of the system (with solid interfaces)
- The foundation for maintaining a good software product
  - To avoid introducing error in the maintenance phase

# JUnit and Unit Test

---

- For each unit ("program atoms") write at least one test
- All unit tests can be executed at any time to ensure entire program is working properly
- For each change finished (before commit to CVS) all unit tests must succeed, i.e., no errors found
- Unit test central to the extreme programming paradigm
  - Rapid feed back
  - Assumed simplicity
  - Incremental change
  - Quality work

# Unit Test Concepts

---

- Assertion
  - The smallest building blocks of a unit test
  - Expression that determines if a test succeeds or fails
- Test case
  - A collection of test methods e.g., the test for an entire class or method
- Test suite
  - One or more test cases
  - Entire test suite can be execute with a single command

# Assertions in JUnit

---

```
assertTrue (boolean)
assertFalse (boolean)
assertEquals (Object, Object)
assertNull (Object)
assertNotNull (Object)
```

```
// examples
```

```
assertTrue(true);
assertFalse(false);
assertFalse("true is not false", true)
assertNull(null);
assertNotNull("hey");
```

# Test Cases in JUnit

---

- A test case in JUnit must inherit from the class  **junit.framework.TestCase**
- Methods of special importance
  - setUp() set up the test fixture
    - ◆ Called before every test method
  - tearDown() tear down the test fixture
    - ◆ Called after every test method
  - test<method name>() a single test method
    - ◆ The set of these methods in a class is a test case

# Test Cases in JUnit, Example

---

```
public class TestMiddleValue extends junit.framework.TestCase{
    MiddleValue md; // to have something to set up
    /** Sets up the test fixture */
    protected void setUp() { md = new MiddleValue(); }
    public void testMidAEqualBEqualC() {
        int val = md.mid(1,1,1);
        assertTrue(val==1);
    }
    public void testMidASmallerThanB() {
        int val;
        val = md.mid(50, 100, 10);
        assertTrue("Did not return 50 but " + val, val==50);
        val = md.mid(50, 100, 70);
        assertTrue("Did not return 70 but " + val, val==70);
    }
}
```



# Test Suites in JUnit

---

- Collection of test cases in a single logical unit
  - BlueJ will do this automatically for you!
- Test coverage
  - No help from JUnit or BlueJ
    - ♦ Add test until you run out of ideas for good things to test for
    - ♦ Expensive products exists (not aware of any open source projects)
  - Every time you write a `System.out.println` for test move it to a JUnit test case instead
    - ♦ Reuse your work!!!!

# Limits of JUnit

---

- Works very well with
  - Function libraries
  - API in general
- Having problem to deal with
  - GUI
  - Network
  - Web
  - Database
- Most test framework have these limitations

# Automate the Testing

---

- You have a combination of very strong tools at your disposition
  - JUnit
  - Cron jobs
  - CVS
  - Ant (build-tool like make)
- You can now
  - Extract a specific version of your product
  - Compile entire program and test
    - ◆ Send an eail if there are errors in nightly tests
  - Run test of experimental version of your program
  - Generate documentation
- Combination used by several Aalborg companies.

# Advantages and Disadvantages of JUnit

---

- Advantages
  - Small and very-well documented framework
  - Can structure test better and make it much easier to run the tests
  - Available for most Java development tools including BlueJ
  - Testing becomes very systematic and partly automatic
  - Defactor standard for unit testing Java programs
- Disadvantages
  - No matic you have to write the test code your self
  - Java specific but being ported to other languages
- JUnit is highly relevant for your projects!!!

# Summary

---

- Testing is vital to provide a high-quality program.
- Up to 50% of time in real-world project may be testing.
- JUnit is a small Java based framework for unit testing
  - Very hand and well-integrated with BlueJ, Eclipse, and other Java IDEs

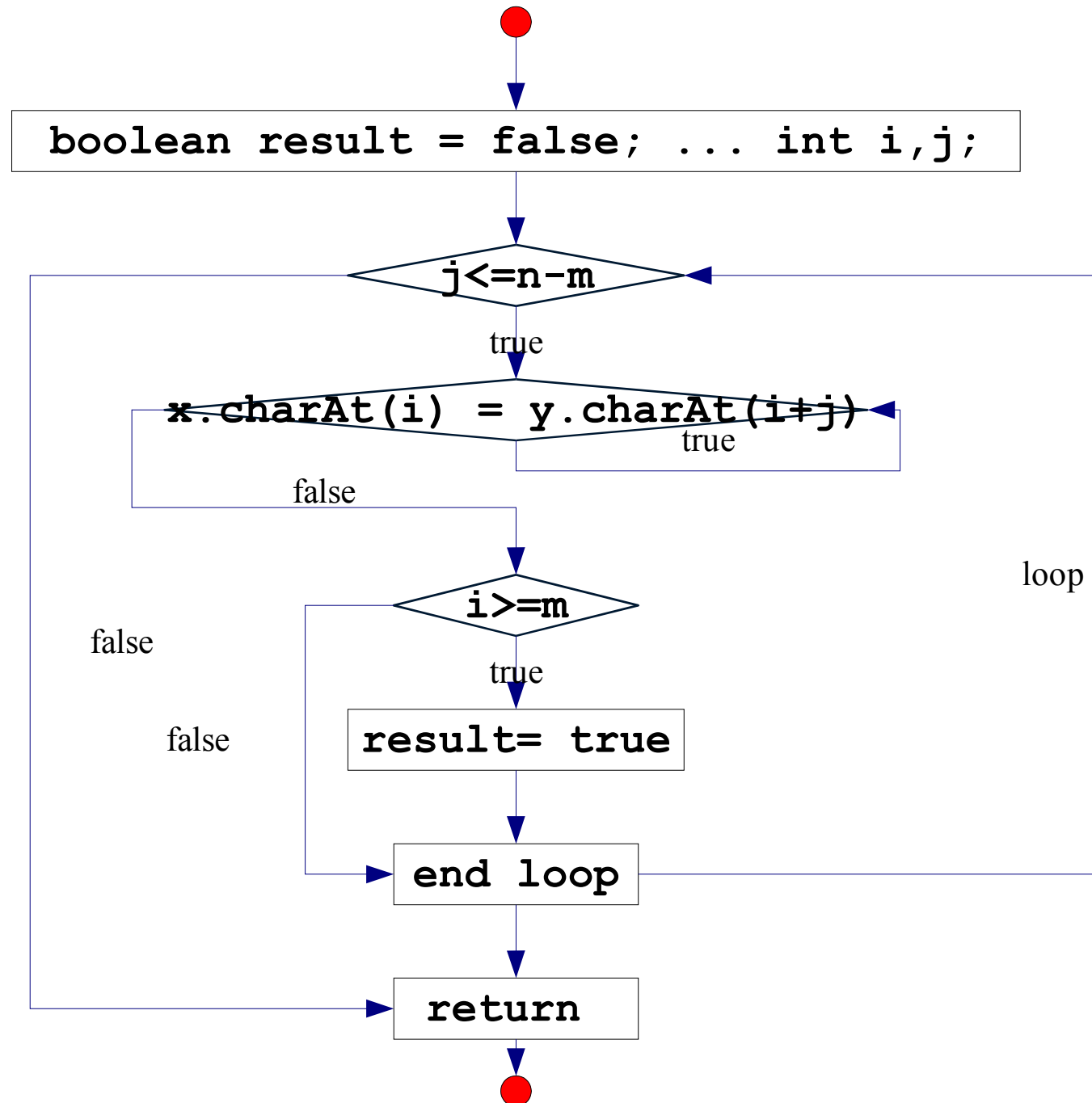
# White-Box Test, A Third Example

---

```
// greatest common divisor
public static boolean stringCompare(String x, String y){
    boolean result = false;
    int m = x.length();
    int n = y.length();
    int i, j;

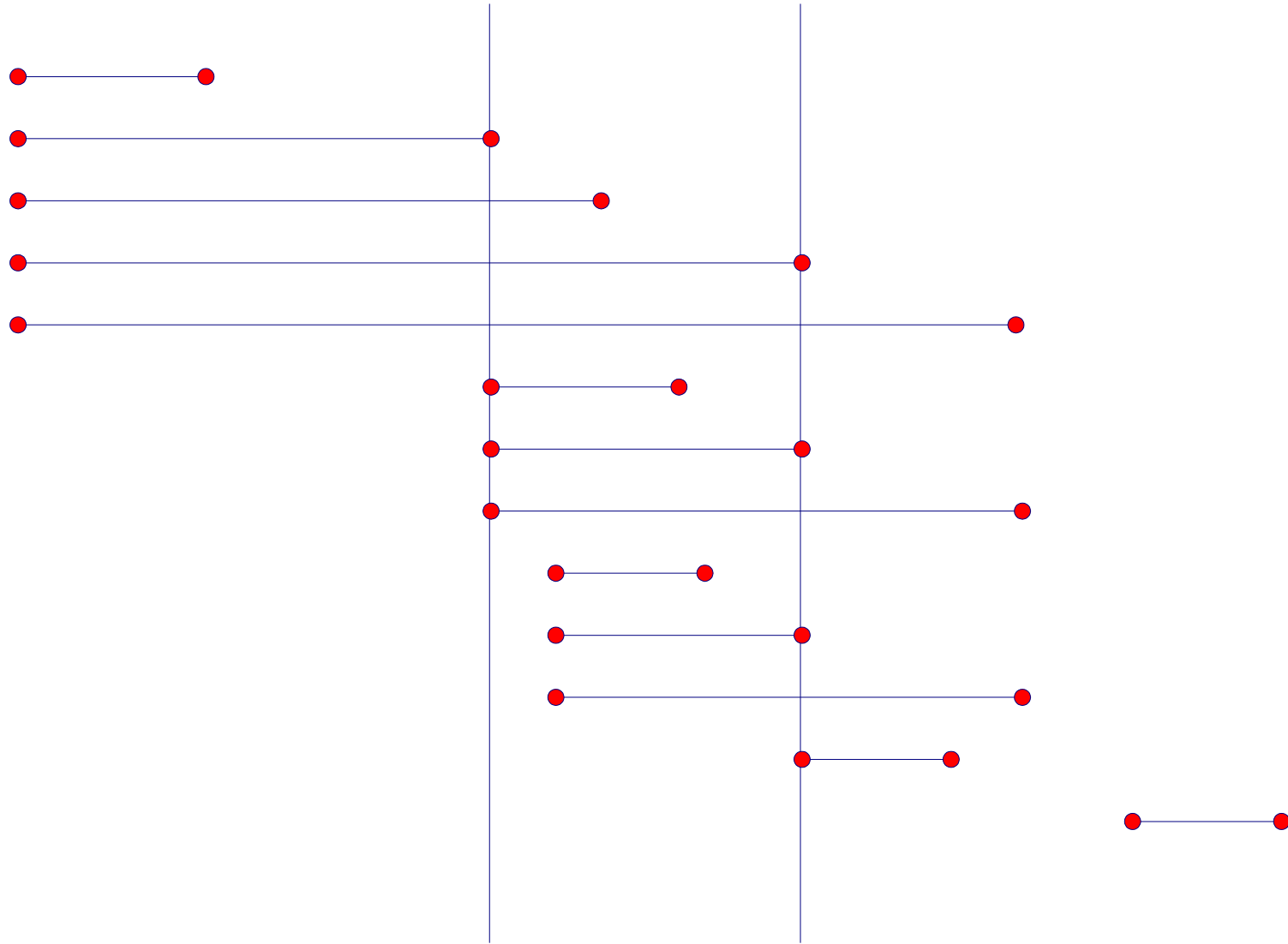
    /* Searching */
    for (j = 0; j <= n - m; ++j) {
        for (i = 0; i < m && x.charAt(i) == y.charAt(i + j); ++i);
        if (i >= m)
            result = true;
    }
    return result;
}
```

# White-Box Test, A Third Example, cont.



# Interval Test Cases

---





# Test Coverage

---

- *Goal:* To ensure that all statements and conditions have been executed at least once.
- *Why:*
  - Studies show that logic errors and incorrect assumptions are inversely proportional to a path's execution probability.
  - Typographical errors are distributed random across the program, it is therefore likely that untested paths will contain such errors.
  - Often assumed by programmer that a particular path is not likely to be executed. However, reality is often counter intuitive.

# Cyclomatic Complexity

---

- Industry studies have shown that the higher  $V(G)$ , the higher the probability of errors.
- Defines the number of independent paths in the basis set
- *Path coverage set* is the set of paths that will execute all statements and all conditions in a program at least once
  - Are not unique
- Define test cases for basis set