cs304
Software Engineering

TAN, Shin Hwei
陈馨慧
Southern University of Science and Technology

Slides adapted from cs427 (UIUC) and cs409 (SUSTech)

Administrative Info

- **The deadline for MP0 and Project Proposal has passed.** Late submission get 0 score!
- **MP1 part 1 and part 2 has been posted with two different deadlines**
  - Part 1 due on March 24
  - Part 2 due on April 14
- **Progress Report has been posted due on** April 23
- **All assignments should be written in English**
  - One exception: The selected issues could be written by the developers in Chinese
- **All lab exercise should be submitted before next lab to avoid accumulating too much assignments**
- **Attend lab today for tools-supported fixing session! Important for MP1!**
  - **What if you can run some tools to help you fix the bugs that you have selected?**

What you will be learning in the Lab this week

- What is code coverage tool?
  - Measure how well does your tests cover the code that you write.
- What is maven?
- What is Evosuite?

What is the build system that your selected project used?

- Ant
- Maven
- Gradle
- Others?

Note that this may affect how you run Astor and Prapr

What is maven?

- A a project management and comprehension tool
- Provide ways to help with managing:
  - Builds
  - Documentation
  - Reporting
  - Dependencies
  - SCMs
  - Releases
  - Distribution

POM

- Single configuration file that contains the majority of information required to build a project

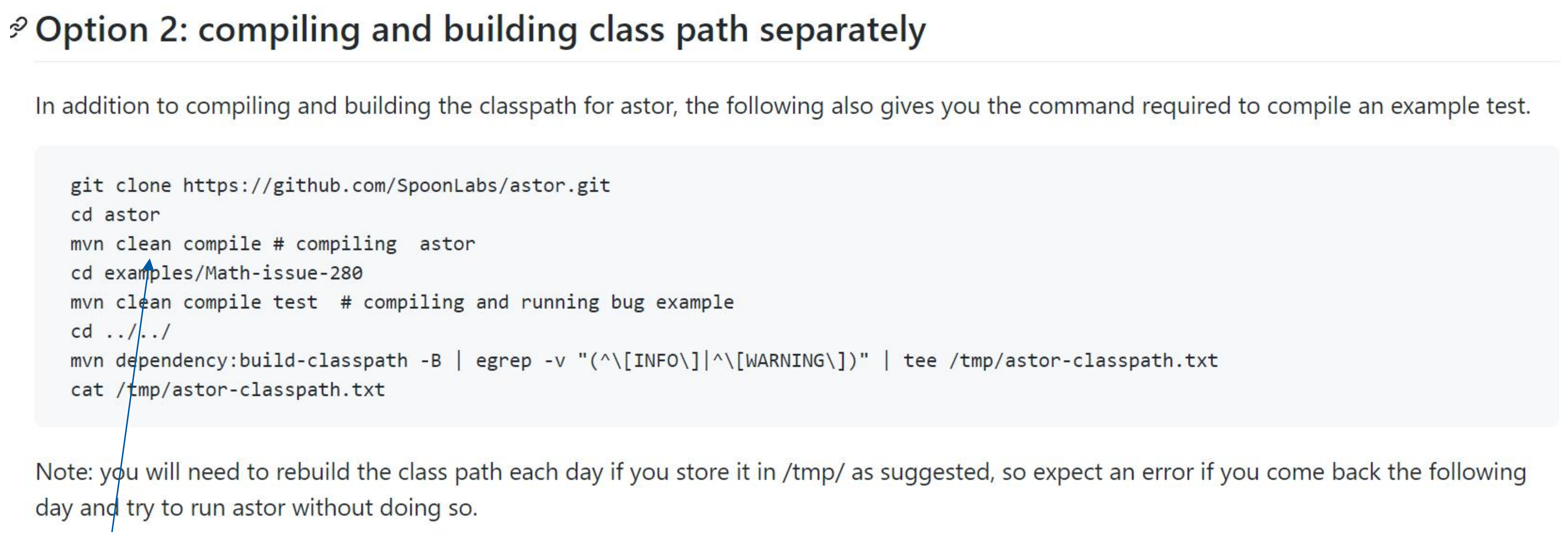
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
 <properties>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>
  <deendencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

declare dependencies

What is the maven command for removing all the files and directories generated by Maven during its build?

What is the maven command for running tests in your project?

Last Week lab on Astor



Option 2: compiling and building class path separately

In addition to compiling and building the classpath for astor, the following also gives you the command required to compile an example test.

```
git clone https://github.com/SpoonLabs/astor.git
cd astor
mvn clean compile # compiling  astor
cd examples/Math-issue-280
mvn clean compile test  # compiling and running bug example
cd ../../
mvn dependency:build-classpath -B | egrep -v "(^\[INFO\]|^\[WARNING\])" | tee /tmp/astor-classpath.txt
cat /tmp/astor-classpath.txt
```

Note: you will need to rebuild the class path each day if you store it in /tmp/ as suggested, so expect an error if you come back the following day and try to run astor without doing so.

Many students said that they don’t know maven

Phrases in Maven

Maven Phases

Common default lifecycle phases:

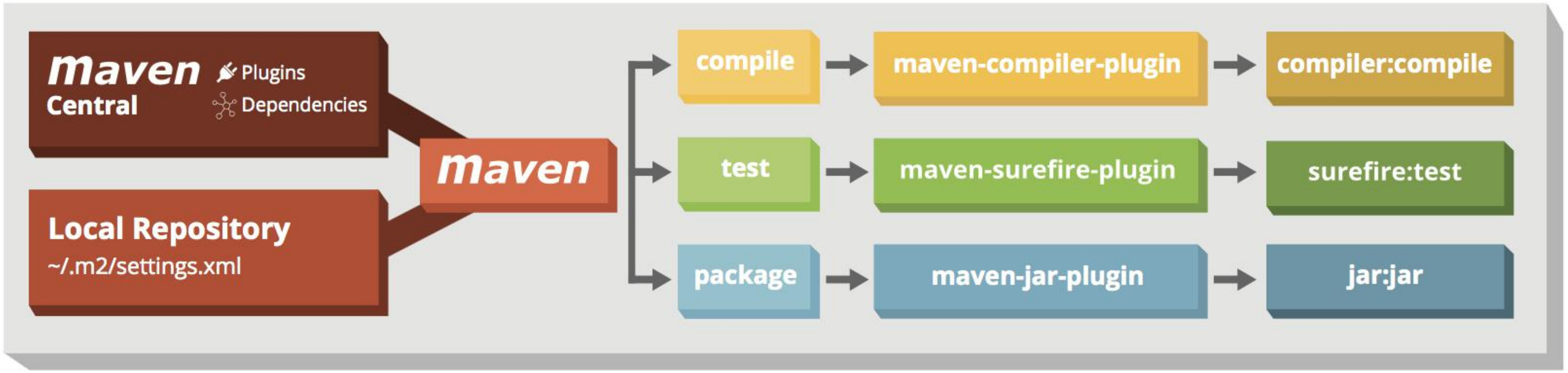
- **validate**: validate the project is correct and all necessary information is available
- **compile编译**: compile the source code of the project
- **Test测试**: test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
- **Package打包**: take the compiled code and package it in its distributable format, such as a JAR.
- **integration-test**: process and deploy the package if necessary into an environment where integration tests can be run
- **verify**: run any checks to verify the package is valid and meets quality criteria
- **Install安装**: install the package into the local repository, for use as a dependency in other projects locally
- **deploy**: done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

There are two other Maven lifecycles of note beyond the *default* list above. They are

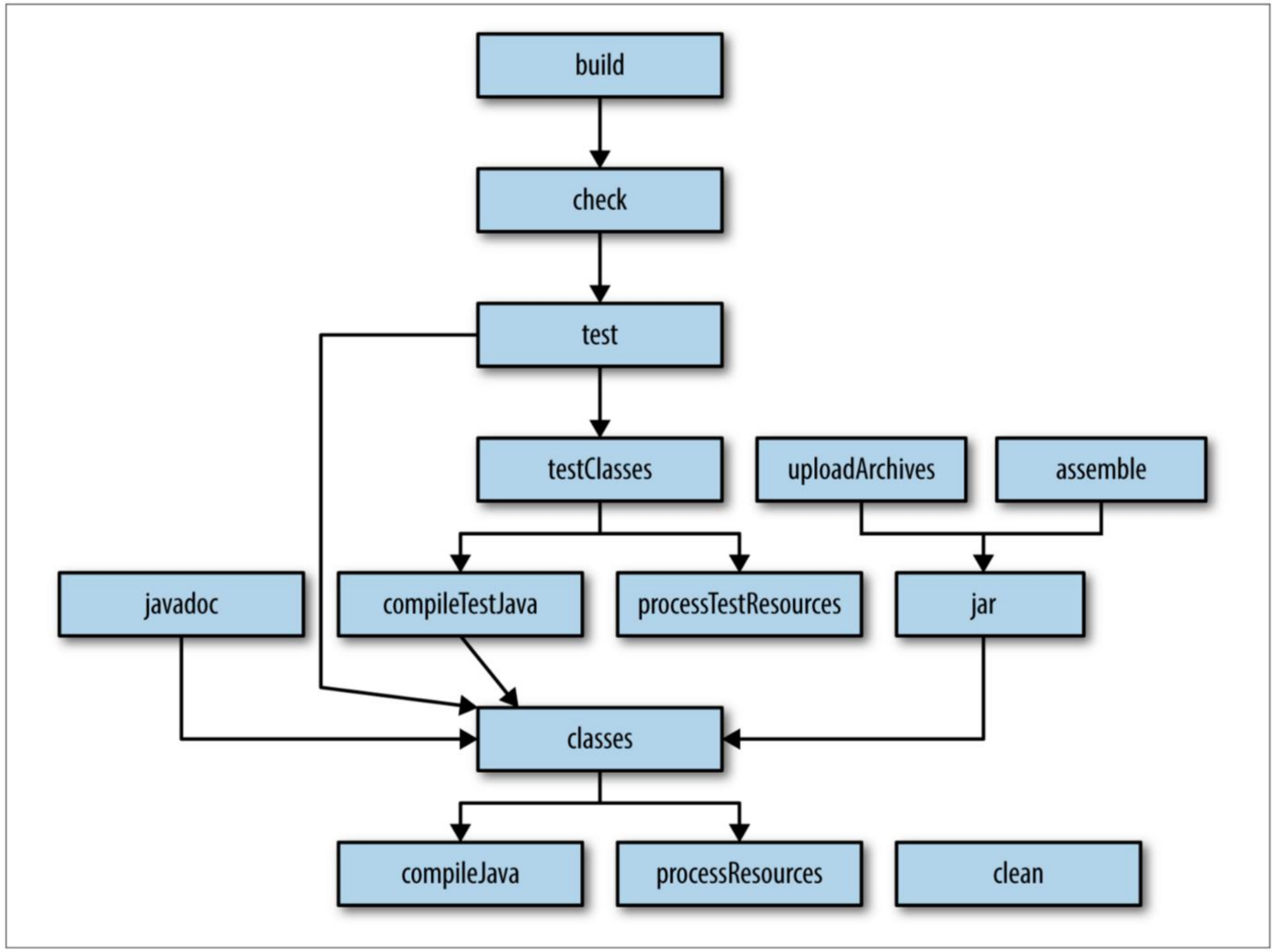
- **clean**: cleans up artifacts created by prior builds
- **site**: generates site documentation for this project

Example command: mvn test

Main commands in maven



Gradle: The build lifecycle



Refer to: https://proandroiddev.com/understanding-gradle-the-build-lifecycle-5118c1da613f

Can you find a test that show difference results for these two examples?

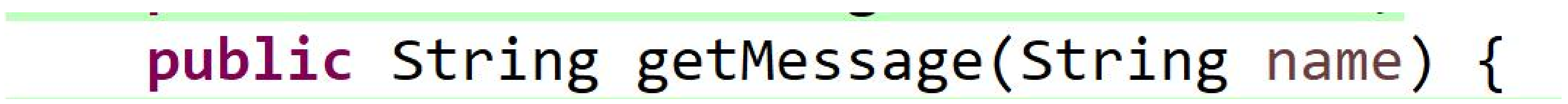
```
public boolean isPositive(int number) {

    boolean result = false;
    if (number >= 0) {
        result = true;
    }
    return result;

}
```

```
public boolean isPositive(int number) {

    boolean result = false;
    // mutator - changed conditional boundary
    if (number > 0) {
        result = true;
    }
    return result;

}
```

What is Coverage?

Coverage = Reachability (可达性)

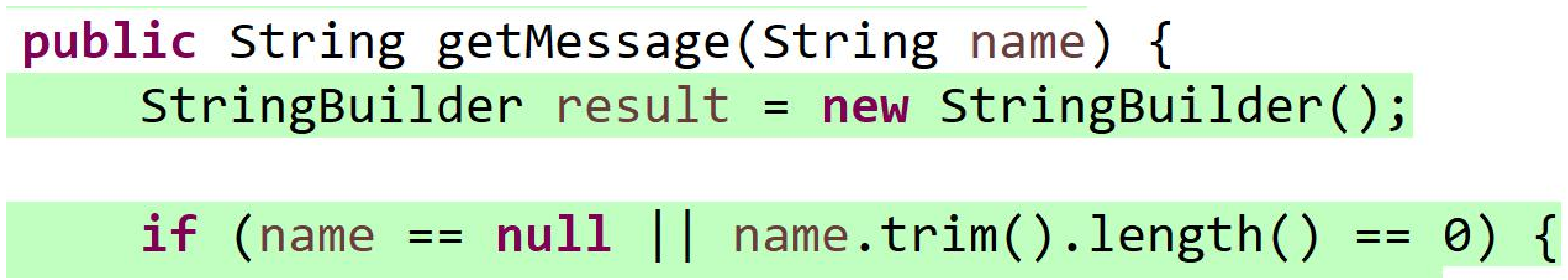
What is the input?



```
    public String getMessage(String name) {
```

What is Coverage?

Coverage = Reachability (可达性)

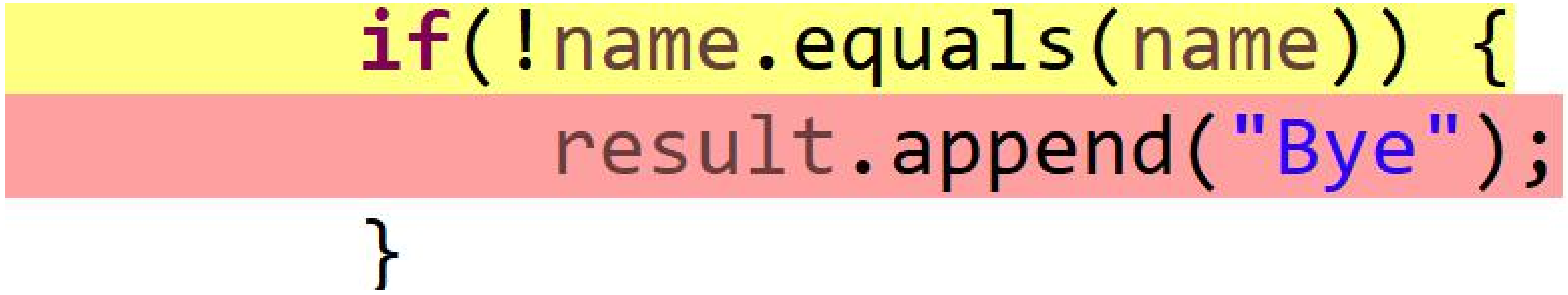


```
public String getMessage(String name) {
    StringBuilder result = new StringBuilder();

    if (name == null || name.trim().length() == 0) {
```

How to reach this line?

Recap: Coverage Lab

Why no test could cover this line?



```
        if(!name.equals(name)) {
            result.append("Bye");
        }
```

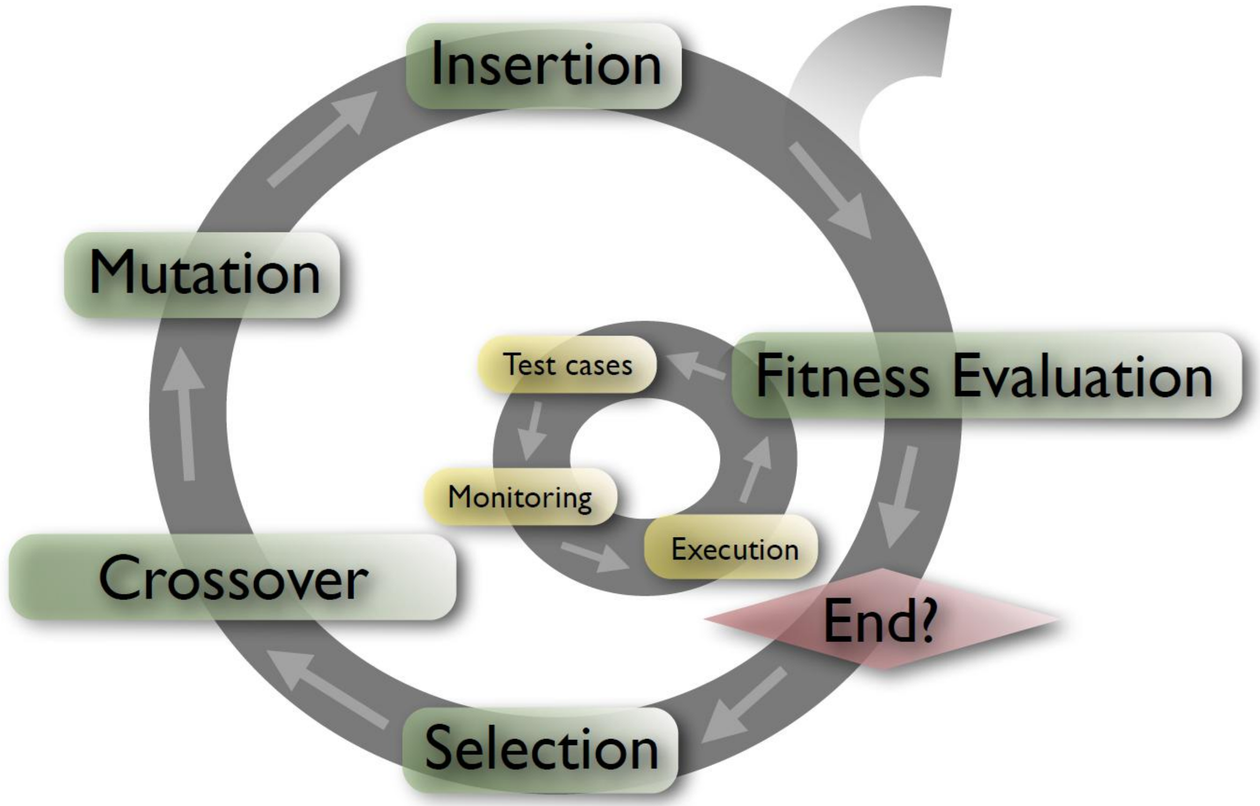
- !name.equals(name) = false
- Dead code: code that cannot be reached

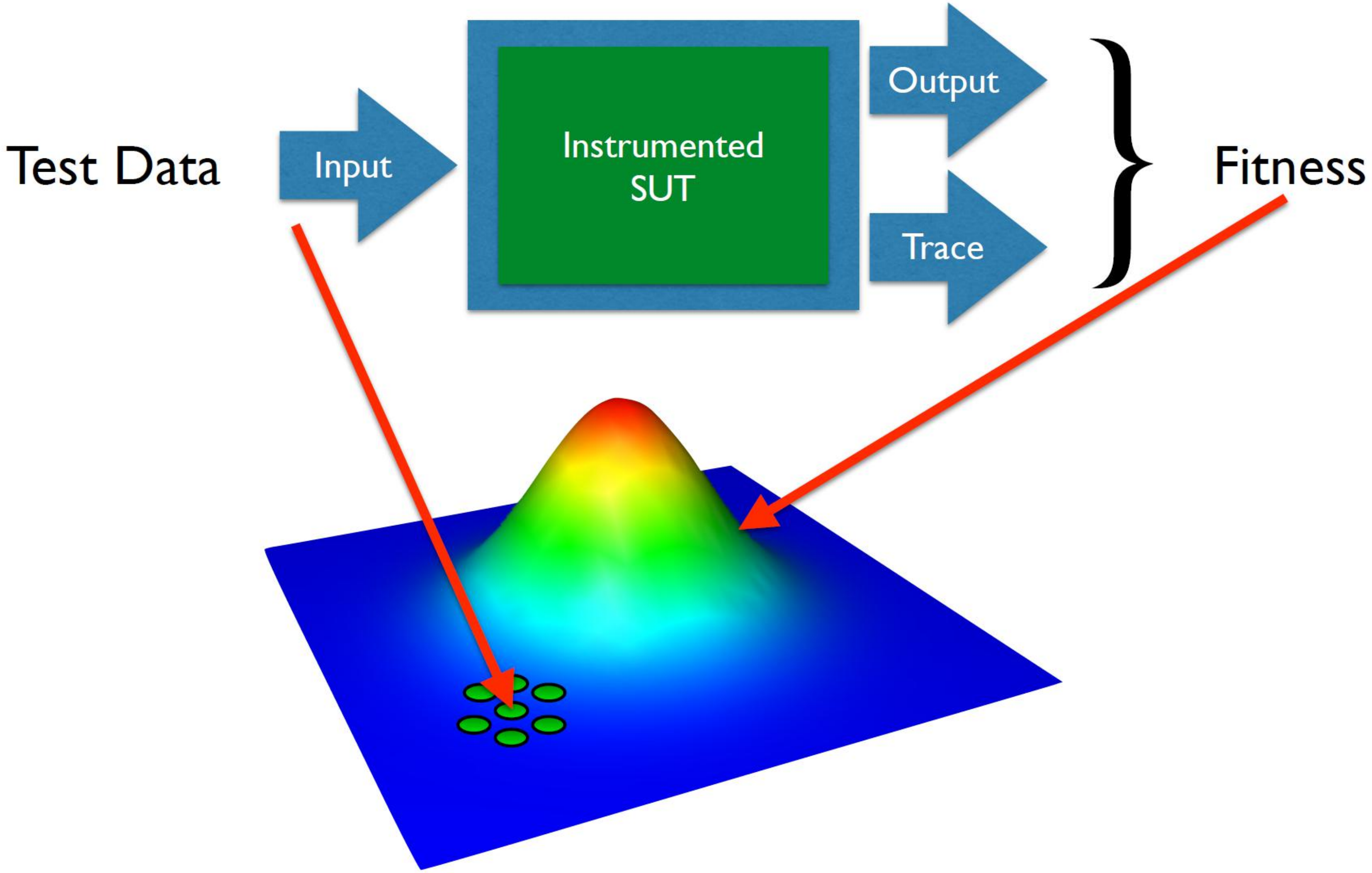
From:http://www.evosuite.org/evosuite-tutorials/

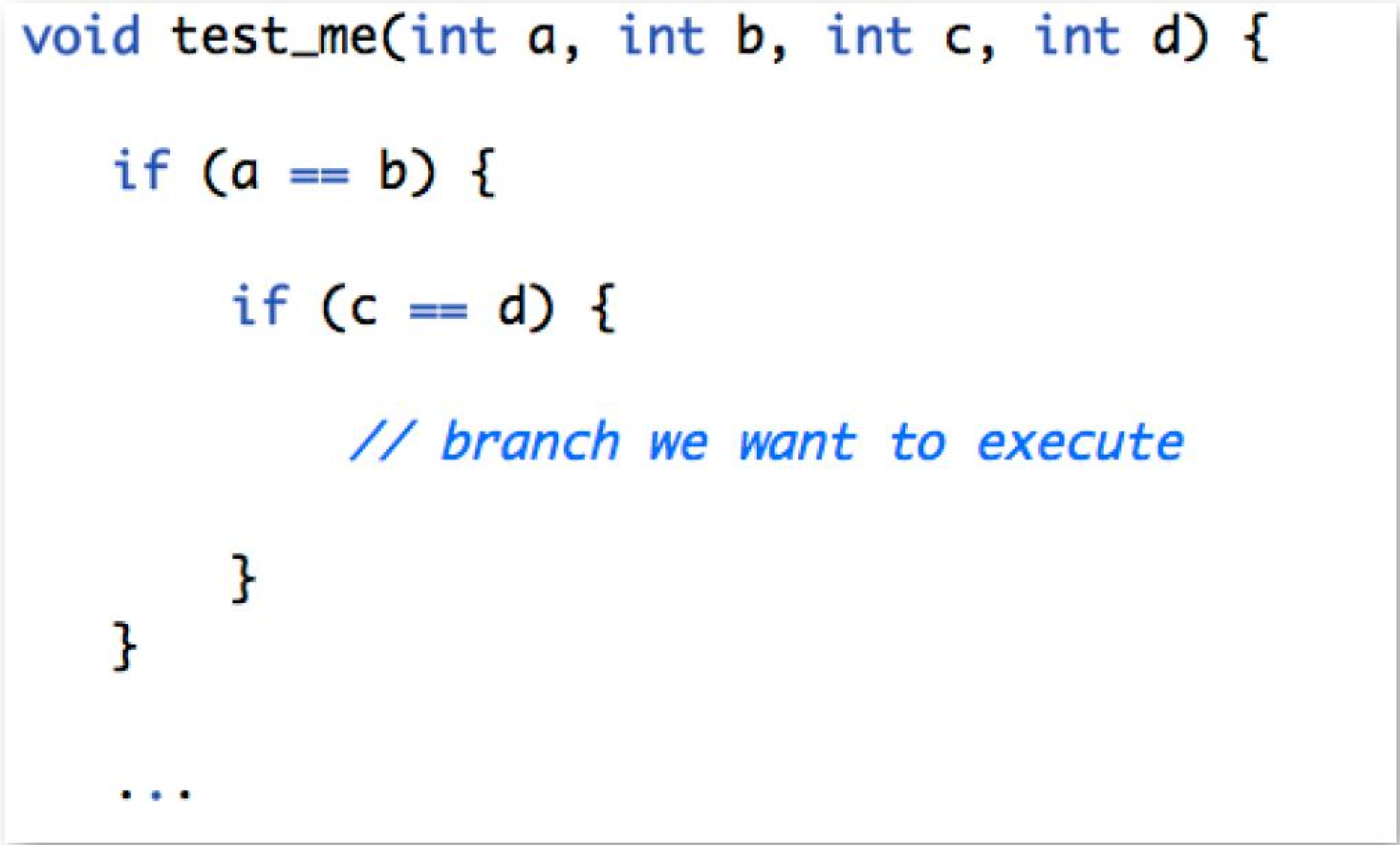
How does it works?

EvoSuite uses evolutionary algorithm to generate and optimize whole test suites towards satisfying a coverage criterion.

Evolutionary Testing



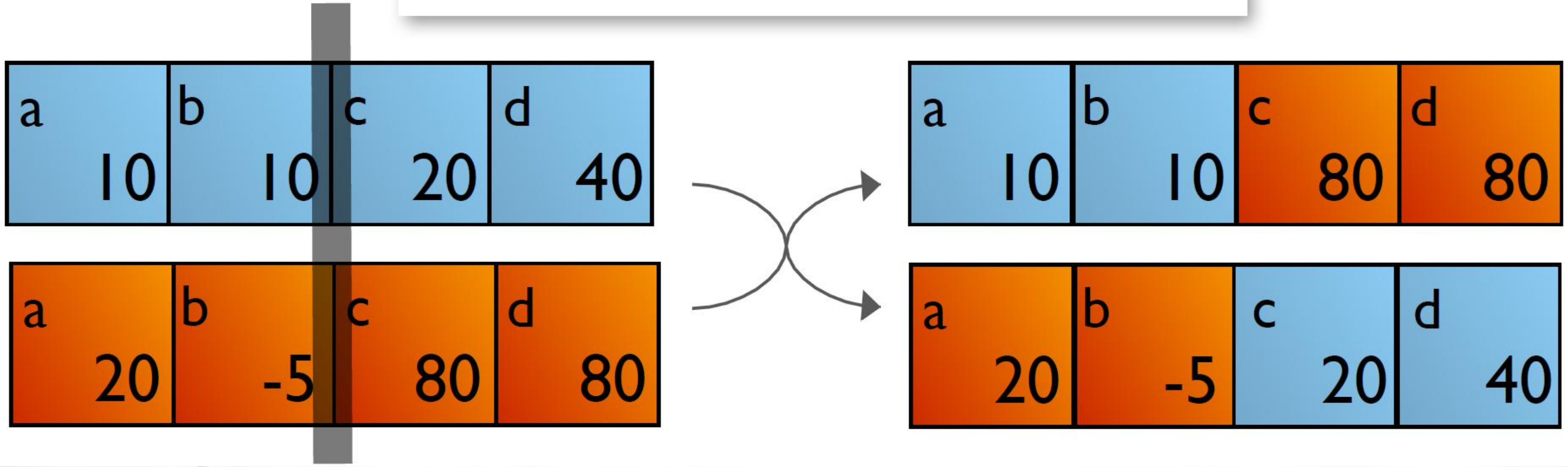

Test Data
Input
Instrumented SUT
Output
Trace
Fitness



```
void test_me(int a, int b, int c, int d) {

    if (a == b) {

        if (c == d) {

            // branch we want to execute

        }
    }

    ...
```

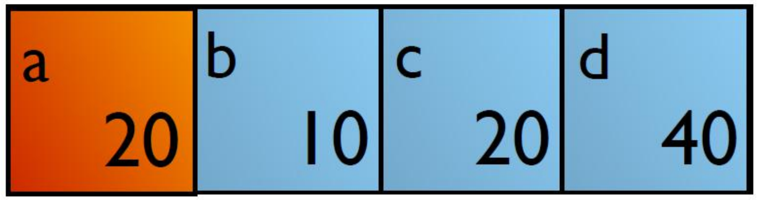
Crossover

```
void test_me(int a, int b, int c, int d) {

   if (a == b) {

      if (c == d) {

         // branch we want to execute

      }
   }

   ...
```



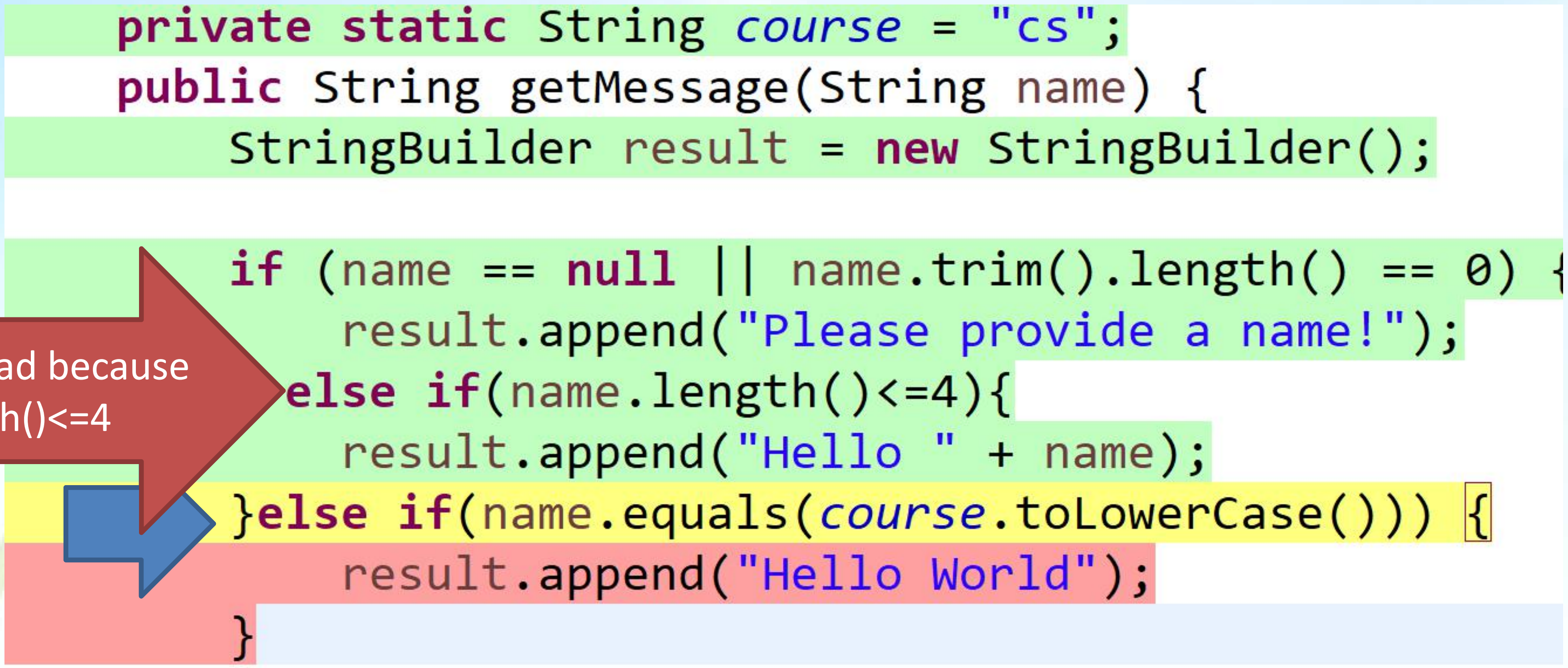
Mutation

```
void test_me(int a, int b, int c, int d) {

   if (a == b) {

      if (c == d) {

         // branch we want to execute

      }
   }

   ...
```



Coverage Lab

Why Evosuite couldn't cover this line?



```
    private static String course = "cs";
    public String getMessage(String name) {
        StringBuilder result = new StringBuilder();

        if (name == null || name.trim().length() == 0) {
            result.append("Please provide a name!");
         else if(name.length()<=4){
            result.append("Hello " + name);
        }else if(name.equals(course.toLowerCase())) {
            result.append("Hello World");
        }
```

Reach here instead because course.length()<=4

- course is a private field & the "course variable is <=4 so it will go to the line before
  - use reflection to change it to public

EVOSUITE

Research they are working on...

- Increasing coverage…
- Readability optimisation
- Better environment handling
- Mocking and private reflection
- Finding out how developers benefit most from using test generation
- User studies, replications

Method Names

```
@Test(timeout = 4000)
public voi  test3()  hrows Throwable
                                 {
      StringExample stringExample0 = new
      StringExample();  boolean boolean0 =
      stringExample0.foo("");  assertFalse(boolean0);
}
```

```
@Test(timeout = 4000)
public voi  testFooReturningFalse() hrows Throwable  {
      StringExample stringExample0 = new
      StringExample();  boolean boolean0 =
      stringExample0.foo("");  assertFalse(boolean0);
}
```

Variable Names

```
@Test(timeout = 4000)
public void testFooReturningFalse()throws Throwable

      {  StringExample stringExample0 = new
      StringExample();  boolean boolean0 =
      stringExample0.foo("");  assertFalse(boolean0);
}
```

```
@Test(timeout = 4000)
public void testFooReturningFalse()throws Throwable

      {  StringExample invokesFoo = new StringExample();
      boolean resultFromFoo = invokesFoo.foo("");
      assertFalse(resultFromFoo);
}
```

Online Tutorials

- Using EvoSuite on the command line:
  http://www.evosuite.org/documentation/tutorial-part-1/
- Using EvoSuite with Maven:
  http://www.evosuite.org/documentation/tutorial-part-2/
- Running experiments with EvoSuite:
  http://www.evosuite.org/documentation/tutorial-part-3/
- Extending EvoSuite:
  http://www.evosuite.org/documentation/tutorial-part-4/

Code Coverage is a metrics for test suite quality

Is there other metrics?

METRICS
ADAPTED FROM CS427 UIUC

TODAY’S GOALS

- How do we measure product and project progress?

METRICS

- What can we measure?
  - Process
    - Man hours
    - Bugs reported
    - Stories implemented
  - Product
- Measures of the product are called “technical metrics”

NON-TECHNICAL METRICS

- Number of people on project
- Time taken, money spent
- Bugs found/reported
  - By testers/developers
  - By users
- Bugs fixed, features added

TECHNICAL METRICS

- Size of code
  - Number of files
  - Number of classes
  - Number of processes
- Complexity of code
  - Dependencies / Coupling / Cohesion
  - Depth of nesting
  - Cyclomatic complexity

TECHNICAL OR NON-TECHNICAL?

- Number of tests
- Number of failing tests
- Number of classes in design model
- Number of relations per class
- Size of user manual
- Time taken by average transaction

MEASURING SIZE OF SYSTEM

- Lines of code (Source Lines of Code - SLOC)
- Number of classes, functions, files, etc.
- Function Points

- Are they repeatable?
- Do they work on analysis model, design model, or code?

LINES OF CODE (1)

- Easy to measure
- All projects produce it
- Correlates to time to build
- Not a very good standard

- Bill Gates:

  "Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weighs."

LINES OF CODE (2)

Blanks? Comments?

```
copy(char *p,*q) {while(*p) *q++ = *p++;}
```

```
copy(char *p,*q) {
   while(p) {
       *q++ = *p++;
   }
}
```

Use Source Lines of Code (SLOC): only counts source code (ignore blanks and comments)

LINES OF CODE: LANGUAGE MATTERS

- Assembly code may be 2-3X longer than C code
- C code may be 2-3X longer than Java code
- Java code may be 2-3X longer than …

LINES OF CODE (3)

- Lines of code is valid metric when
  - Same language
  - Standard formatting
  - Code has been reviewed

COMPLEXITY

- Complex systems are
  - Hard to understand
  - Hard to change
  - Hard to reuse
- Some metrics for complexity:
  - Cyclomatic complexity (Thomas J. McCabe, Sr. in 1976)
  - Function points
  - Coupling and cohesion
  - Many OO-specific metrics (may cover later)

CYCLOMATIC COMPLEXITY (1)

- A measure of logical complexity
- Tells how many tests are needed to execute every statement of program

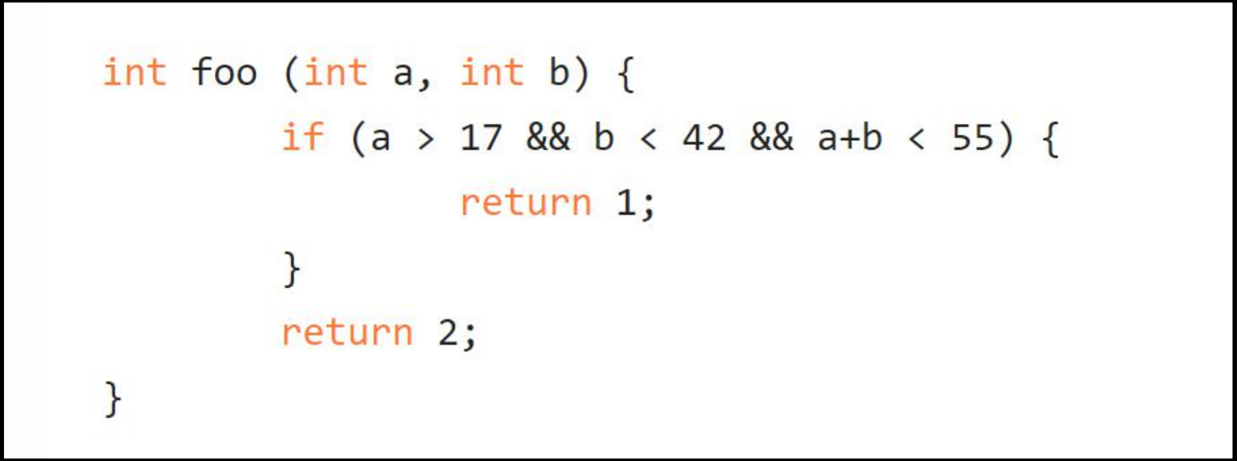
=Number of branches (`if`, `while`, `for`) + 1

WHY IS CYCLOMATIC COMPLEXITY USEFUL?

https://www.youtube.com/watch?v=UdjEb6t9-e4

DIFFERENT TOOLS TELLS YOU DIFFERENT VALUES OF CYCLOMATIC COMPLEXITY

- Original paper is not clear about how to derive the control flow graph
  - different implementations gives different values for the same code.
  - For example, the following code is reported with complexity 2 by the Eclipse Metrics Plugin, with 4 by GMetrics, and with complexity 5 by SonarQube:



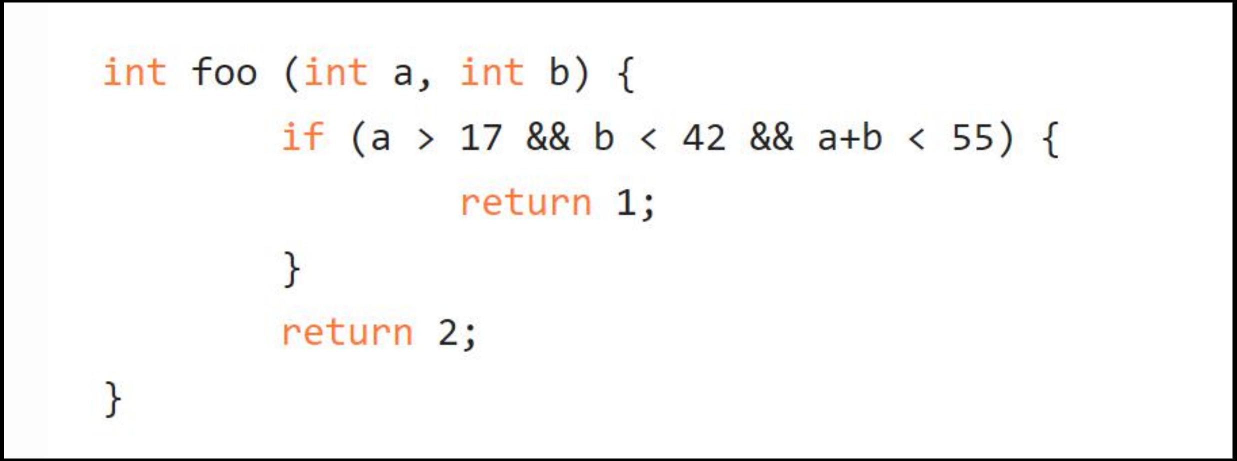
```
int foo (int a, int b) {
        if (a > 17 && b < 42 && a+b < 55) {
                return 1;
        }
        return 2;
}
```

CYCLOMATIC COMPLEXITY (4)

- Number of predicates + 1
- Number of edges - number of nodes + 2
- Number of regions of the flow graph

DIFFERENT TOOLS TELLS YOU DIFFERENT VALUES OF CYCLOMATIC COMPLEXITY

- Original paper is not clear about how to derive the control flow graph
  - different implementations gives different values for the same code.
  - For example, the following code is reported with complexity 2 by the Eclipse Metrics Plugin, with 4 by GMetrics, and with complexity 5 by SonarQube:



```
int foo (int a, int b) {
        if (a > 17 && b < 42 && a+b < 55) {
                return 1;
        }
        return 2;
}
```

CYCLOMATIC COMPLEXITY (4)

- Number of predicates + 1
- Number of edges - number of nodes + 2
- Number of regions of the flow graph

CYCLOMATIC COMPLEXITY (5)
TESTING VIEW

- Cyclomatic complexity is the number of independent paths through the procedure
- Gives an upper bound on the number of tests necessary to execute every edge of control graph

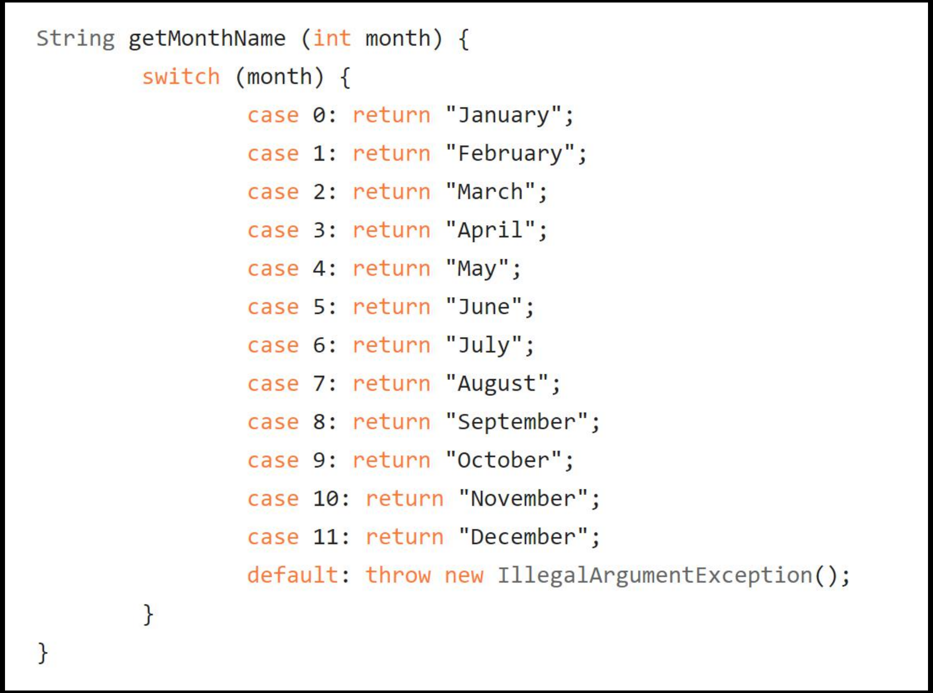
CYCLOMATIC COMPLEXITY (6)
METRICS VIEW

- McCabe found that modules with cyclomatic complexity greater than 10 were hard to test and error prone
- Look for procedures with high cyclomatic complexity and rewrite them, focus testing on them, or focus reviewing on them
  - Good target for refactoring, reverse engineering, reengineering

HIGH COMPLEXITY

cyclometric complexity
=Number of branches (`if`, `while`, `for`) + 1

- What is the cyclometric complexity of the code below?
  - 1
  - 12
  - 13
  - 14



```
String getMonthName (int month) {
        switch (month) {
                case 0: return "January";
                case 1: return "February";
                case 2: return "March";
                case 3: return "April";
                case 4: return "May";
                case 5: return "June";
                case 6: return "July";
                case 7: return "August";
                case 8: return "September";
                case 9: return "October";
                case 10: return "November";
                case 11: return "December";
                default: throw new IllegalArgumentException();
        }
}
```

CALCULATE CYCLOMATIC COMPLEXITY

```
int detect(int seconds, boolean isTimeSensitive) {
1       boolean isSlow = false;
2       int isPerIssue = 0;
3       if (seconds > 10 && seconds < 100) {
4           isSlow = true;
        }
5       if (isTimeSensitive) {
6           if (isSlow) {
7               isPerIssue = 1;
            }
        }
8       return isPerIssue;
  }
```

What is the cyclomatic complexity?

- 3
- 4
- 5
- 6

FUNCTION POINTS (1)

- Way of measuring “functionality” of system
- Measure of how big a system ought to be
- Used to predict size
- Several methods of computing function points, all complicated
- Most are proprietary

FUNCTION POINTS (2)

- Count number of inputs, number of outputs, number of algorithms, number of tables in database
- “Function points” is function of above, plus fudge factor for complexity and developer expertise
- You need training to measure function points

COUPLING AND COHESION (1)

- Coupling - dependences among modules
- Cohesion - dependences within modules
- Dependences
  - Call methods, refer to class, share variable
- Coupling - bad
- Cohesion - good

COUPLING AND COHESION (2)

- Number and complexity of shared variables
  - Functions in a module should share variables
  - Functions in different modules should not
- Number and complexity of parameters
- Number of functions/modules that are called
- Number of functions/modules that call me

COUPLING AND COHESION IN OO LANGUAGES

DHAMA'S COUPLING METRIC

Module coupling = 1 / (
  number of input parameters +
  number of output parameters +
  number of global variables used +
  number of modules called +
  number of modules calling
)

0.5 is low coupling, 0.001 is high coupling

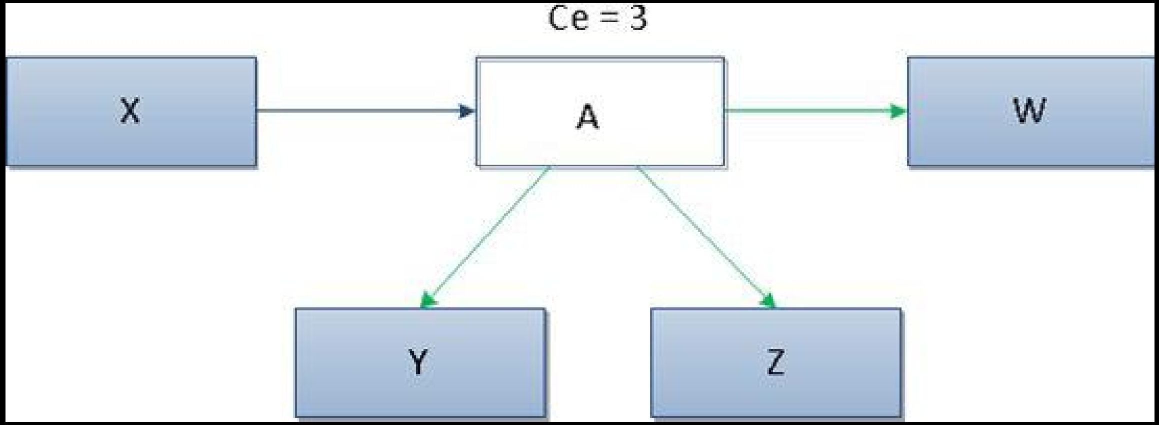
MARTIN'S COUPLING METRIC

- Ca : Afferent coupling: the number of classes outside this module that depend on classes inside this module
- Ce : Efferent coupling: the number of classes inside this module that depend on classes outside this module

Instability = Ce / (Ca + Ce)

EFFERENT COUPLING (CE)

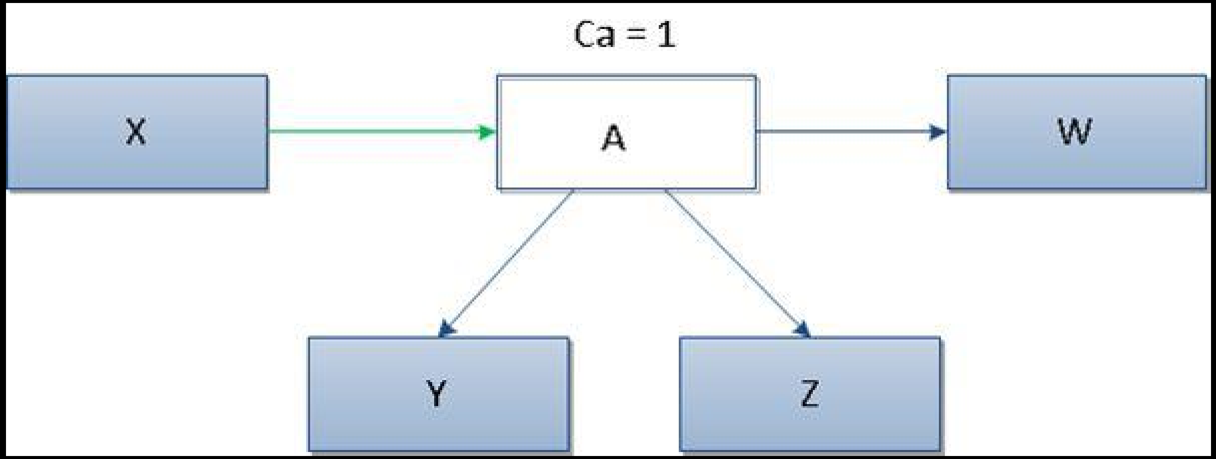
- Definition: A number of classes in a given package, which depends on the classes in other packages
- Measure interrelationships between classes.
- Enable us to measure the vulnerability of the package to changes in packages on which it depends.
- High value of the metric Ce> 20 indicates instability of a package,



Outgoing dependencies

AFFERENT COUPLING (CA)

- Measure incoming dependencies.
- Enables us to measure the sensitivity of remaining packages to changes in the analysed package.
- High values of metric Ca usually suggest high component stability.



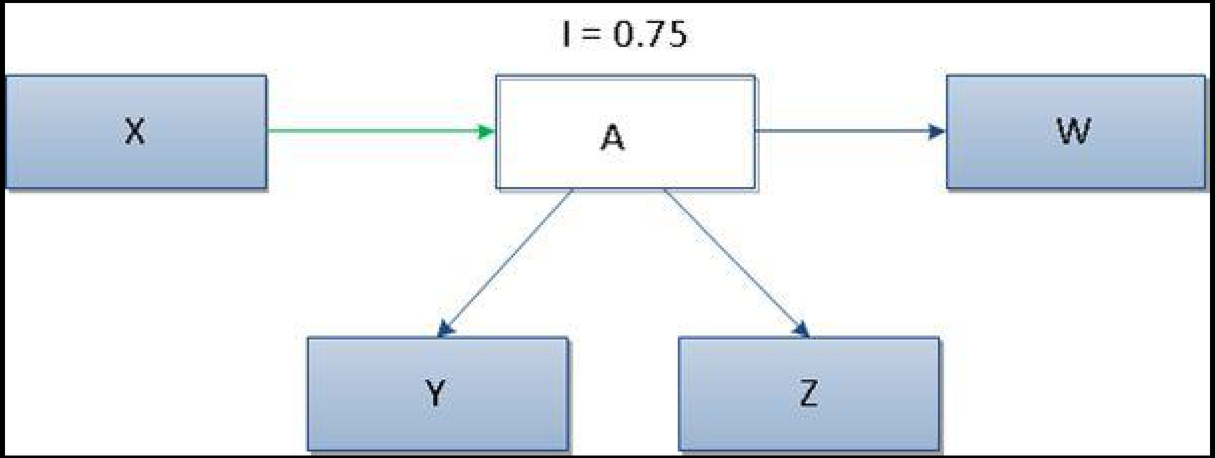
Incoming dependencies

INSTABILITY

$$I = \frac{Ce}{Ce+Ca}$$

There are two types of components:

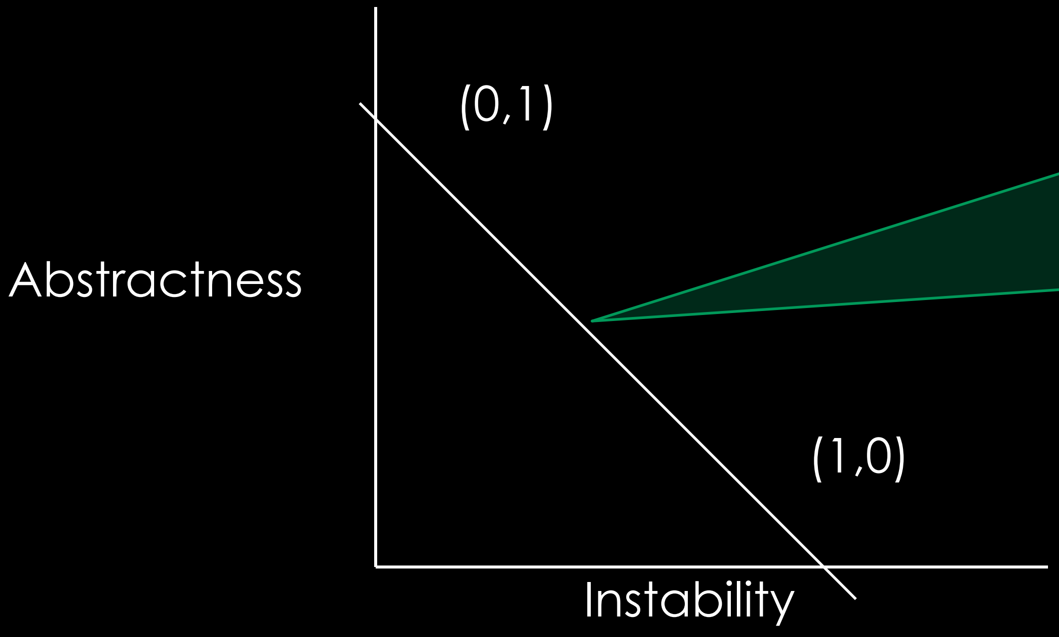
- Many outgoing dependencies and not many of incoming ones (value I is close to 1), are unstable due to the possibility of easy changes to these packages;
- Many incoming dependencies and not many of outgoing ones (value I is close to 0), therefore they are more difficult in modifying due to their greater responsibility.



ABSTRACTNESS

Abstractness = $A = \frac{T_{abstract}}{T_{abstract} + T_{concrete}}$

(number of abstract classes in module / number of classes in module)



Main sequence : "right" number of concrete and abstract classes in proportion to its dependencies

TECHNICAL OO METRICS

- Paper (not required for the exam, but you can read it if you are interested in this topic and the slides are not sufficient)
- A Metrics Suite for Object Oriented Design, Shyam R. Chidamber and Chris F. Kemerer IEEE Transactions on Software Engineering, June 1994, pp 476-493