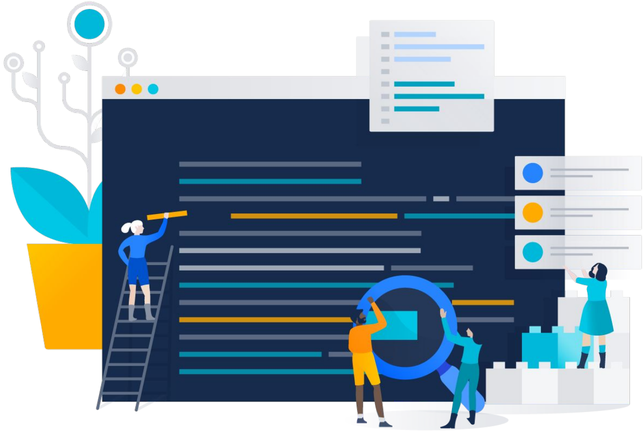
CS130: Software Engineering

Week 6

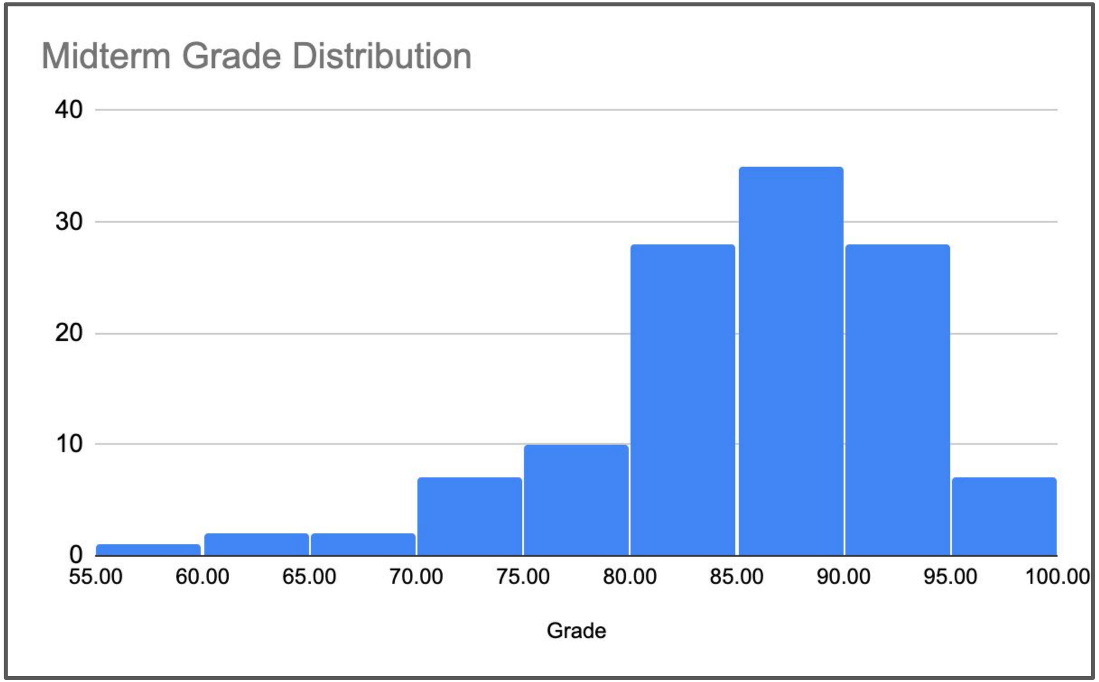


Agenda

1. Mid-point Feedback
2. Midterms
3. Path & Branch Coverage
4. Make Tools
5. Testing
    a. JUnit
    b. UI testing
6. Part B : Sample Presentation
7. Team Time for Part B

Midterms

Average	84.71
Median	86
Max	96
Std Dev	7.55



Path and Branch Coverage

A. Which of the following statements are true about Regression Testing?

a. Regression Testing is the execution of software in its final configuration, including integration with other software and hardware systems.

b. Regression Testing is the repetition of previously executed test cases for the purpose of finding defects.

c. Regression Testing is the process of testing changes to a software program to make sure that the older code still works with the new changes.

A. Which of the following statements are true about Regression Testing?

   a. Regression Testing is the execution of software in its final configuration, including integration with other software and hardware systems. **FALSE**

   b. Regression Testing is the repetition of previously executed test cases for the purpose of finding defects. **TRUE**

   c. Regression Testing is the process of testing changes to a software program to make sure that the older code still works with the new changes. **TRUE**

B. Which of the following statements are true about Branch Coverage?

a. The coverage criteria is that each control structure should evaluate one of the two conditions i.e. true or false.

b. Branch coverage is a testing method which aims to ensure that each control structure should evaluate each one of the possible branch at least once.

c. A 100% Statement Coverage always implies a 100% Branch Coverage.

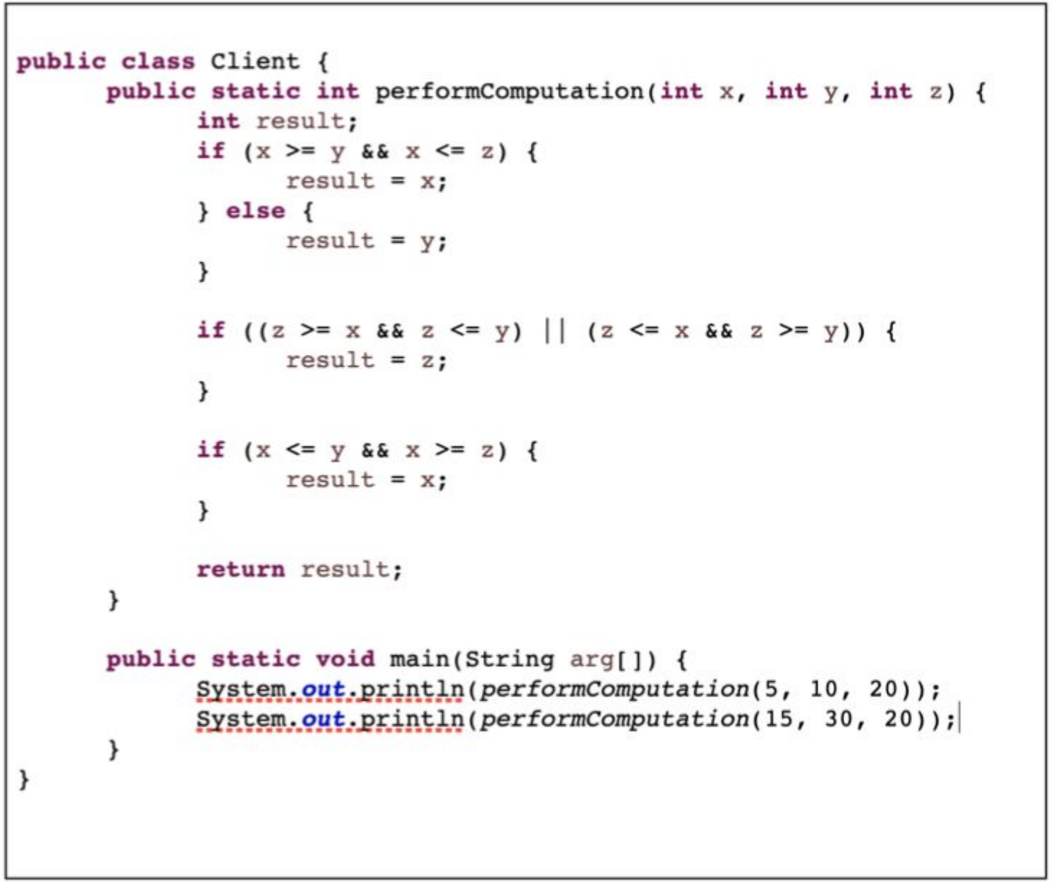
B. Which of the following statements are true about Branch Coverage?

a. The coverage criteria is that each control structure should evaluate one of the two conditions i.e. true or false. **FALSE**

b. Branch coverage is a testing method which aims to ensure that each control structure should evaluate each one of the possible branch at least once. **TRUE**

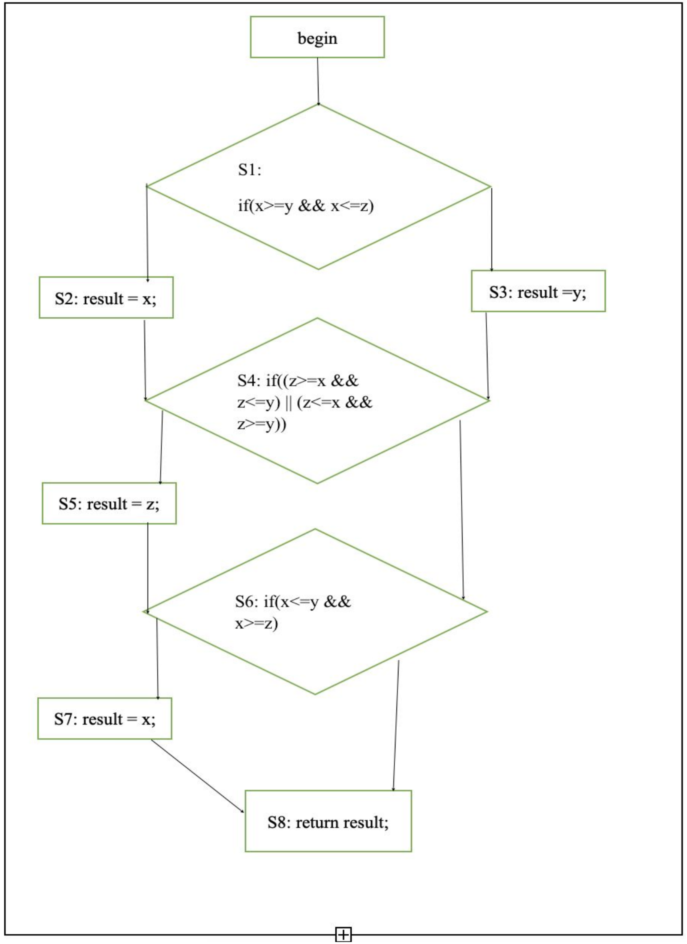
c. A 100% Statement Coverage always implies a 100% Branch Coverage. **FALSE**

2. Refer the code below and answer the following questions.

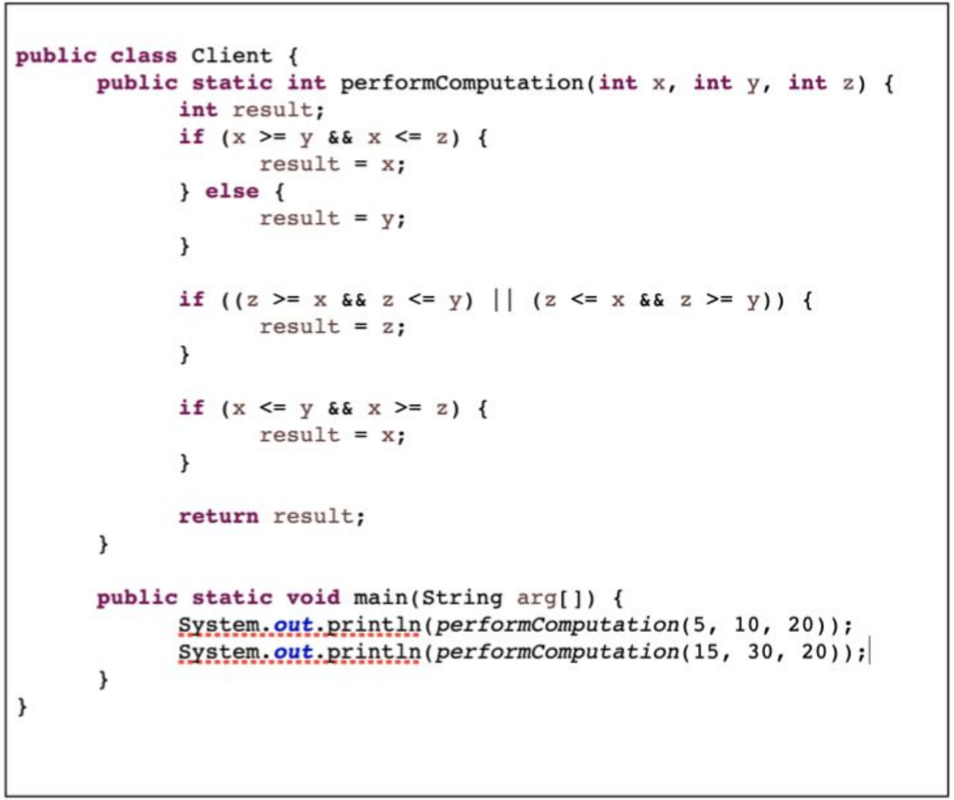


```
public class Client {
      public static int performComputation(int x, int y, int z) {
            int result;
            if (x >= y && x <= z) {
                  result = x;
            } else {
                  result = y;
            }

            if ((z >= x && z <= y) || (z <= x && z >= y)) {
                  result = z;
            }

            if (x <= y && x >= z) {
                  result = x;
            }

            return result;
      }

      public static void main(String arg[]) {
            System.out.println(performComputation(5, 10, 20));
            System.out.println(performComputation(15, 30, 20));
      }
}
```

a. Draw a control-flow graph for the above `performComputation` function

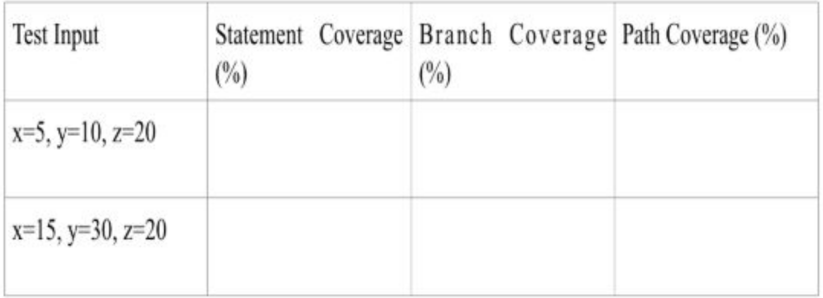

begin
S1:
if(x>=y && x<=z)
S2: result = x;
S3: result =y;
S4: if((z>=x &&
z<=y) || (z<=x &&
z>=y))
S5: result = z;
S6: if(x<=y &&
x>=z)
S7: result = x;
S8: return result;

2. Refer the code below and answer the following questions.

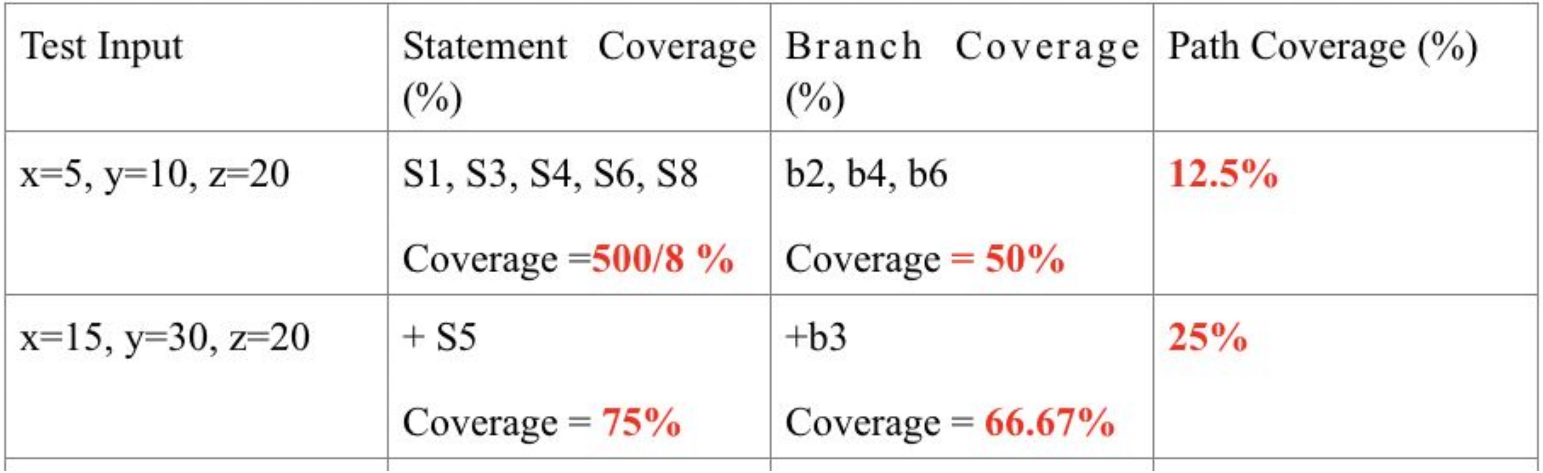


```java
public class Client {
    public static int performComputation(int x, int y, int z) {
        int result;
        if (x >= y && x <= z) {
            result = x;
        } else {
            result = y;
        }

        if ((z >= x && z <= y) || (z <= x && z >= y)) {
            result = z;
        }

        if (x <= y && x >= z) {
            result = x;
        }

        return result;
    }

    public static void main(String arg[]) {
        System.out.println(performComputation(5, 10, 20));
        System.out.println(performComputation(15, 30, 20));
    }
}
```

b. Fill in the table



Test Input	Statement Coverage (%)	Branch Coverage (%)	Path Coverage (%)
x=5, y=10, z=20			
x=15, y=30, z=20			

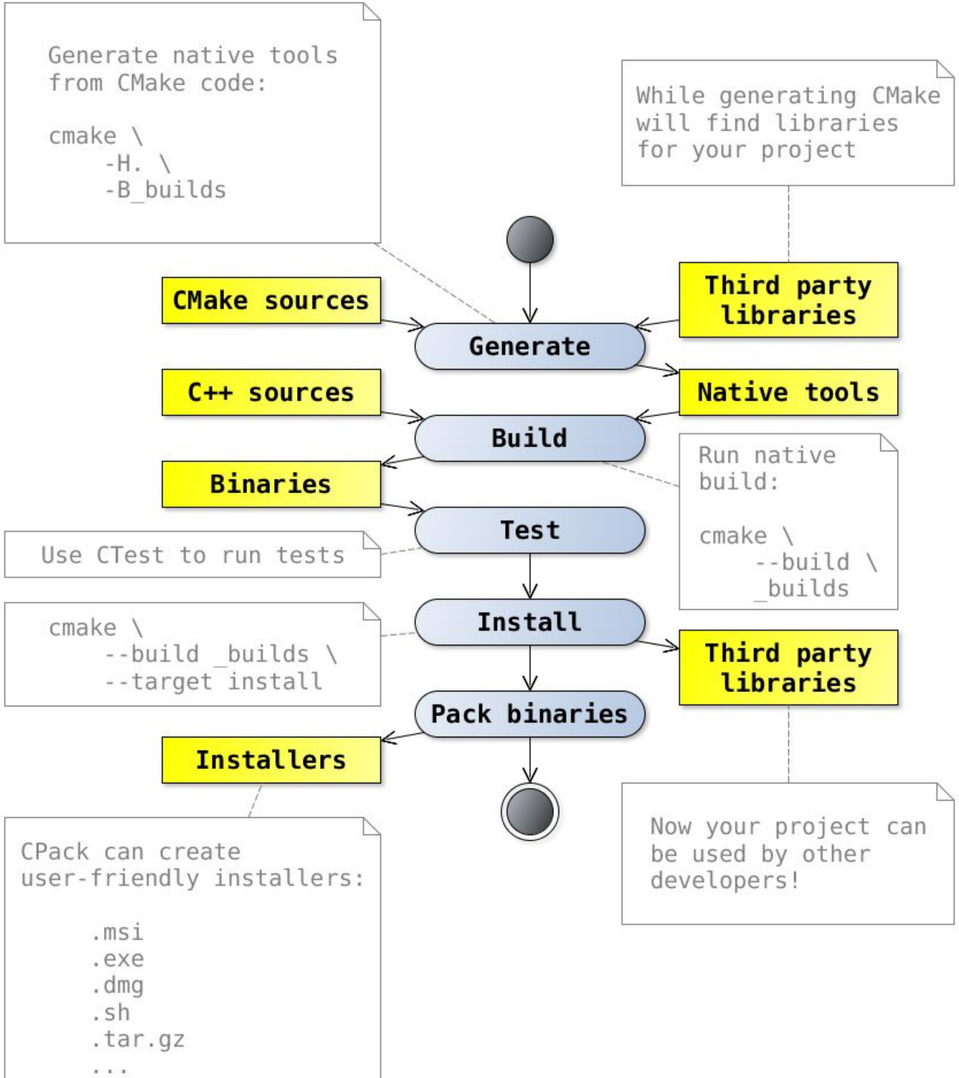


Test Input	Statement Coverage (%)	Branch Coverage (%)	Path Coverage (%)
x=5, y=10, z=20	S1, S3, S4, S6, S8 Coverage =**500/8 %**	b2, b4, b6 Coverage **= 50%**	**12.5%**
x=15, y=30, z=20	+ S5 Coverage = **75%**	+b3 Coverage = **66.67%**	**25%**

Make Tools

CMake

CMake is a cross-platform free and open-source software tool for managing the build process of software using a compiler-independent method.



Generate native tools from CMake code:

```
cmake \
    -H. \
    -B_builds
```

While generating CMake will find libraries for your project

Run native build:

```
cmake \
    --build \
    _builds
```

Use CTest to run tests

```
cmake \
    --build _builds \
    --target install
```

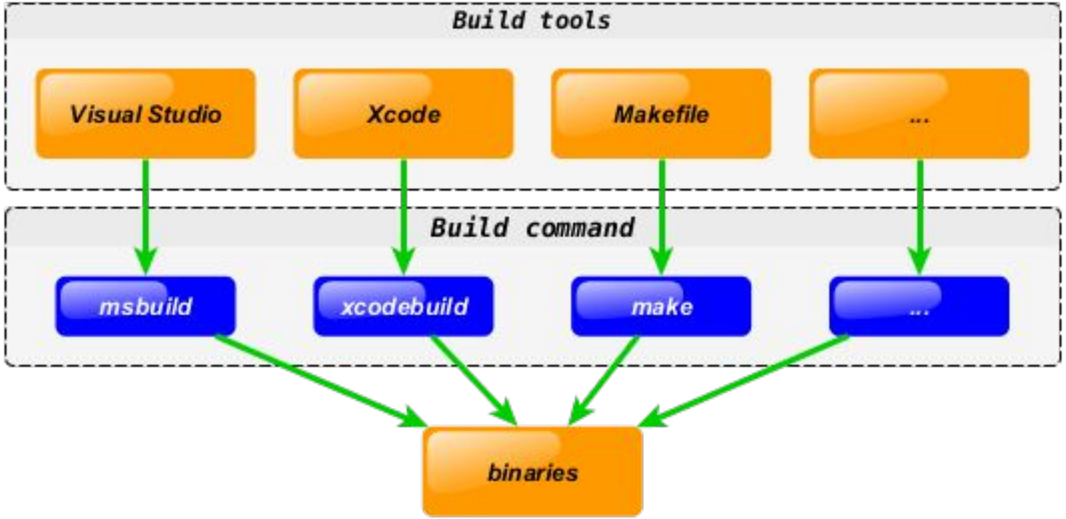
Now your project can be used by other developers!

CPack can create user-friendly installers:

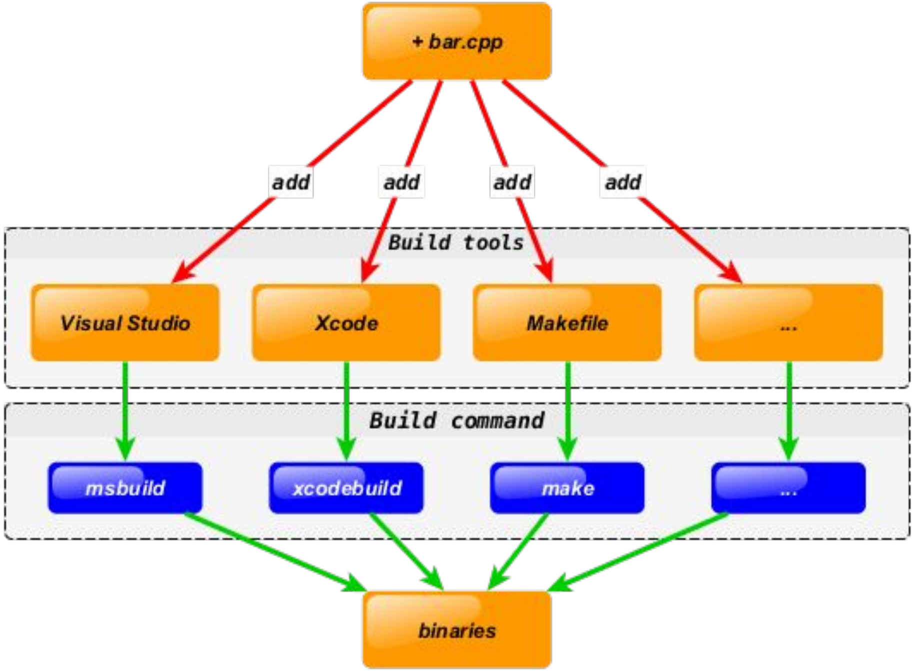
```
    .msi
    .exe
    .dmg
    .sh
    .tar.gz
    ...
```

Cross Platform Development

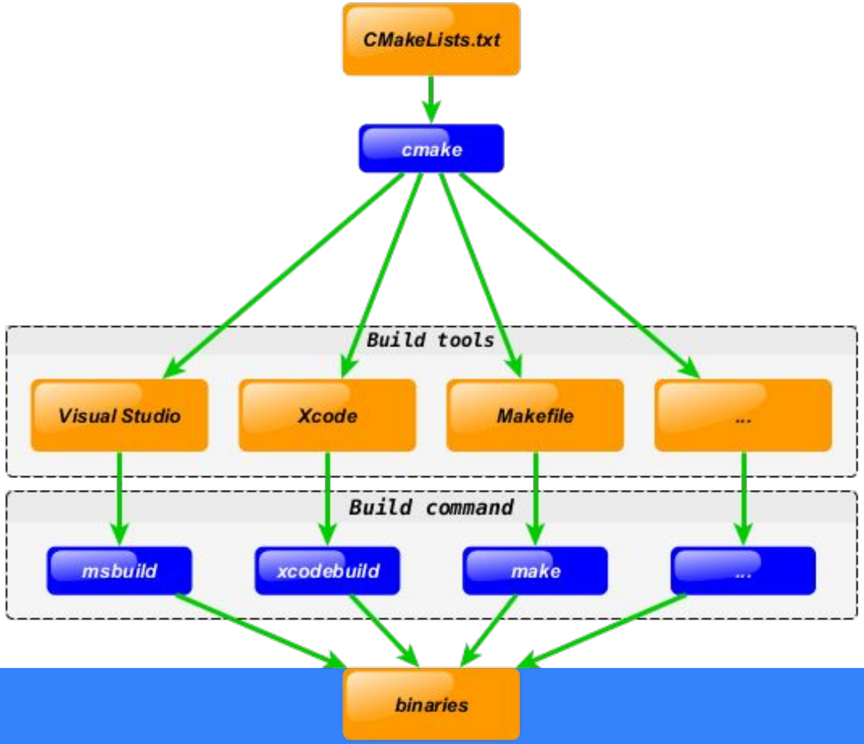
Let's assume you have some cross-platform project with C++ code shared along different platforms/IDEs. Say you use `Visual Studio` on Windows, `Xcode` on OSX and `Makefile` for Linux:



What you will do if you want to add new `bar.cpp` source file? You have to add it to every tool you use



CMake solve this design flaw by adding extra step to development process. You can describe your project in `CMakeLists.txt` file and use CMake to generate tools you currently interested in using cross-platform CMake code:



Testing

JUnit 5

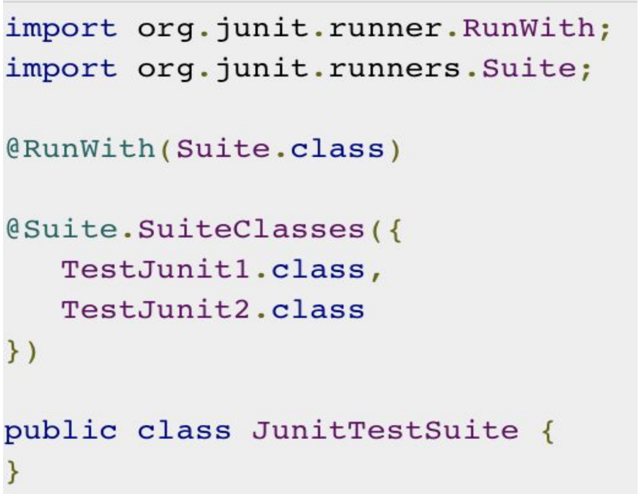
JUnit without an IDE:
https://medium.com/@pelensky/java-tdd-with-junit-without-using-an-ide-cd24d38adff

JUnit with Eclipse:
https://courses.cs.washington.edu/courses/cse143/11wi/eclipse-tutorial/junit.shtml#creating

JUnit 5

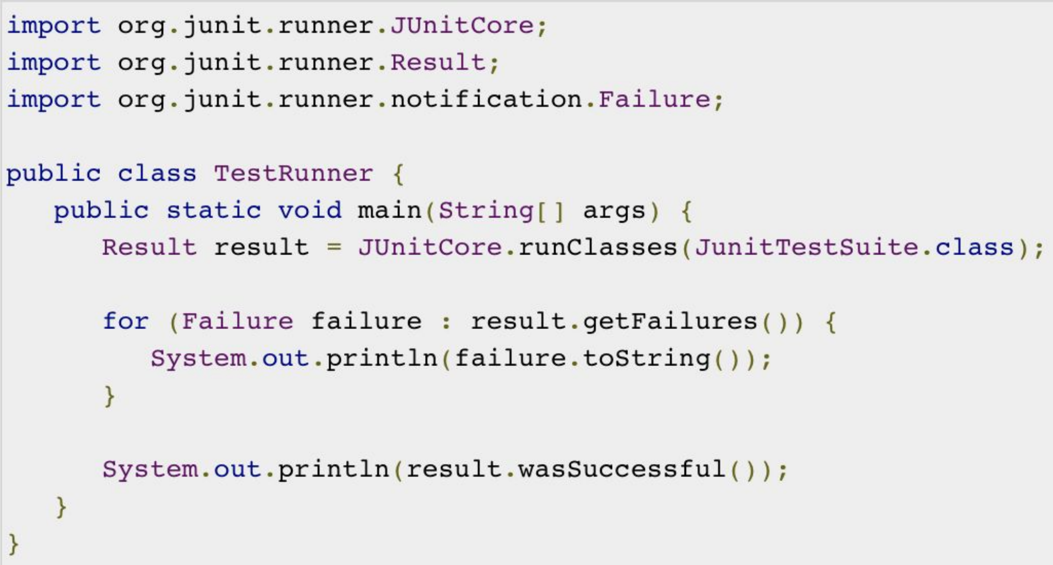
To handle multiple test cases, we use a Test Suite

TestSuite.java



```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)

@Suite.SuiteClasses({
   TestJunit1.class,
   TestJunit2.class
})

public class JunitTestSuite {
}
```

TestRunner..java



```
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class TestRunner {
   public static void main(String[] args) {
      Result result = JUnitCore.runClasses(JunitTestSuite.class);

      for (Failure failure : result.getFailures()) {
         System.out.println(failure.toString());
      }

      System.out.println(result.wasSuccessful());
   }
}
```

UI Testing

Selenium

https://selenium-python.readthedocs.io/

https://www.npmjs.com/package/selenium-webdriver

Jest + Enzyme

https://medium.com/codeclan/testing-react-with-jest-and-enzyme-20505fec4675

Team Time!