The following are modified versions of the publicly-available slides for Chapters 1 and 2 in the Ammann and Offutt Book, “Introduction to Software Testing” (http://www.cs.gmu.edu/~offutt/softwaretest)
Outline for today’s class

- Fundamental testing terminology
- The costs of insufficient, non-existent, or late testing
- The goals of a software tester
- Foundations of software testing
- Levels of software testing
- Types of testing activities
- Model-Driven Test Design
Coverage Criteria

- Exhaustive testing of even small programs requires **too many inputs**
  - private static double computeAverage (int A, int B, int C)
  - On 32-bit machines: A, B, C have over **4 billion** possible values
  - Over **80 octillion** possible tests!!
  - Input space might as well be infinite

- Testers **search** a huge input space
  - Trying to find the **fewest inputs** that will find the **most problems**

- **Coverage criteria** give structured, practical ways to search the input space
  - **Search** the input space thoroughly
  - Not much **overlap** in the tests
Advantages of Coverage Criteria

- Maximize the “bang for the buck”
- Provide traceability from software artifacts to tests
  - Source, requirements, design models, …
- Make regression testing easier
- Give testers a “stopping rule” … when testing is finished
- Can be well supported with powerful tools
There are many Coverage Criteria

- Testing researchers have defined dozens of criteria

- One view: they are all just a few criteria on four types of software structures:
  - Input Domains
  - Graphs
  - Logic Expressions
  - Syntax Descriptions

- This class: We will learn about test design using these four structures
Model-Driven Test Design

- Test Design is the process of designing input values that will effectively test software.

- Test design is one of several activities for testing software:
  - Most mathematical
  - Most technically challenging
Using MDTD in Practice

- First do the math or analysis to obtain test requirements

- Then
  - Find input values that satisfy the test requirements
  - Automate the tests
  - Run the tests
  - Evaluate the tests
Software Artifact: Java Method

```java
/**
 * Return index of node n at the
 * first position it appears,
 * -1 if it is not present
 */

public int indexOf (Node n)
{
    for (int i=0; i < path.size(); i++)
        if (path.get(i).equals(n))
            return i;
    return -1;
}
```

Control Flow Graph:

1. $i = 0$
2. $i < \text{path.size()}$
3. if
4. return -1
5. return i
Example (2)

Graph
Abstract version

Initial Node: 1
Final Nodes: 4, 5

Edges
1 2
2 3
3 2
3 4
2 5

6 requirements for Edge-Pair Coverage
1. [1, 2, 3]
2. [1, 2, 5]
3. [2, 3, 4]
4. [2, 3, 2]
5. [3, 2, 3]
6. [3, 2, 5]

Test Paths
[1, 2, 5]
[1, 2, 3, 2, 5]
[1, 2, 3, 2, 3, 4]
In-Class Exercise (8 minutes)

- [Link](https://docs.google.com/document/d/1e5YgQ5WSXselfg1lUaUpjCbULaGglAZm4y5Kfnj7v43c/edit)

Tasks:

- meet one another
- Complete the task
Outline for today’s class

- Fundamental testing terminology
- The costs of insufficient, non-existent, or late testing
- The goals of a software tester
- Foundations of software testing
- Levels of software testing
- Types of testing activities
- Model-Driven Test Design