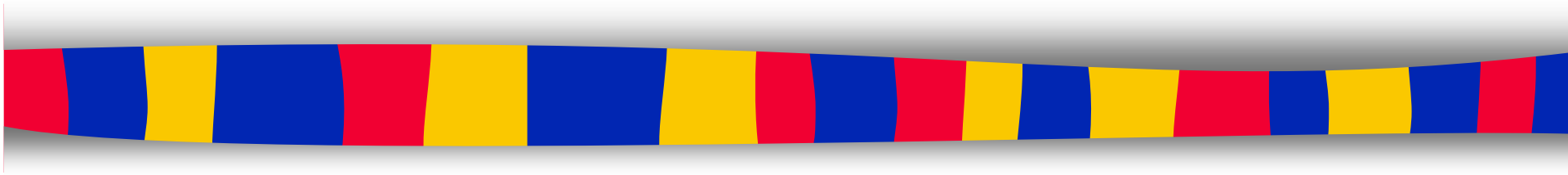
# EECS 4313

## Software Engineering Testing



## Topic 09:

## Slice-based Testing

Zhen Ming (Jack) Jiang

# Relevant Readings

- [Jorgensen] chapter 9
- Harman and Hierons. An overview of program slicing. Software Focus. 2011. Web version: http://www0.cs.ucl.ac.uk/staff/mharman/sf.html

# The point of program slice

- Analyze a program by focusing on parts of interest, disregarding uninteresting parts
  - The point of slices is to separate a program into components that have a useful functional meaning
  - Ignore those parts that do not contribute to the functional meaning of interest
  - Cannot do this with du-paths, as slices are not simply sequences of statements or statement fragments

# (Static) Program slice

- **What is a program slice?**
  - A program slice is a set of program statements that contributes to or affects the value of a variable at some point in a program
- **Backward slices S(v, n)**: refer to statement fragments that *contribute to* the value of v at statement n
  - Statement n is a Use node of variable v, Use (v, n)
- **Forward slices S(v, n):** refer to all the program statements that are *affected* by the value of v and statement n
  - Refers to the predicate uses and computation uses of the variable v

# Backward slicing example

```
1 void foo() {
2   x = 1;
3   y = 2;
4   z = y - 2;
5   r = x;
6   z = x + y;
7 }

// backward slicing on z
// at line 7
```

```
2   x = 1;
3   y = 2;


6   z = x + y;


// resulting backward slicing
// on z at line 7
```

# Backward slicing example (2)

```
1 n = readInt();
2 s=0;
3 p=0;
4 while (n>0) {
5   s=s+n;
6   p=p*n;
7   n=n-1;
8 }
9 System.out.println("p:"+p+"s:"+s);
// backward slicing on p at line 9
```

```
1 n = readInt();

3 p=0;
4 while (n>0) {

6   p=p*n;
7   n=n-1;
8 }

// resulting backward slicing
// on p at line 9
```

# Dynamic Slicing

- Static slicing can be used to assist debugging by simplying the program. However, the slices constructed by slicing can be very large.
- A **dynamic slice** is constructed with respect to the traditional static slicing criterion together with dynamic information (the input sequence supplied to the program, during some specific execution). Three piece of information (dynamic slicing criterion)
  1. Variables to be sliced *(same as static slicing)*
  2. The point of interest within the program *(same as static slicing)*
  3. Sequence of input values for which the program was executed
  - "A dynamic slice for a variable at statement n, on an input I

# Dynamic slicing example

```
1 n = readInt();
2 s=0;
3 p=0;
4 while (n>0) {
5   s=s+n;
6   p=p*n;
7   n=n-1;
8 }
9 System.out.println("p:"+p+"s:"+s);
// dynamic slicing on p at line 9
// for input <0>
```

```
3 p=0;
```

```
// resulting dynamic slicing on p
// at line 9 for input <0>
```

# Static vs. Dynamic Slicing

- Static slices will typically be larger, but will cater for every possible execution of the original program
- Dynamic slices will typically be much smaller, but they will only cater for a single input
- Code reuse:
  - Reusing part of a program which implements a well-tested approach to some problem. Often in such situations (especially for legacy code), the code we want to reuse will be intermingled with all sort of other unrelated code.
  - Static slicing is ideal as a technique for extracting the part of the program we require, while leaving behind the part of the program that we are not interested in

# Applications of program slicing

- Program comprehension
- Debugging
- Measuring the function cohesion
- Software maintenance and evolution

# Program Slicing Tools

- Java
  - Static slicer
    - http://indus.projects.cis.ksu.edu/
  - Dynamic slicer
    - https://www.st.cs.uni-saarland.de/javaslicer/
- C/C++
  - http://research.cs.wisc.edu/wpis/slicing_tool/