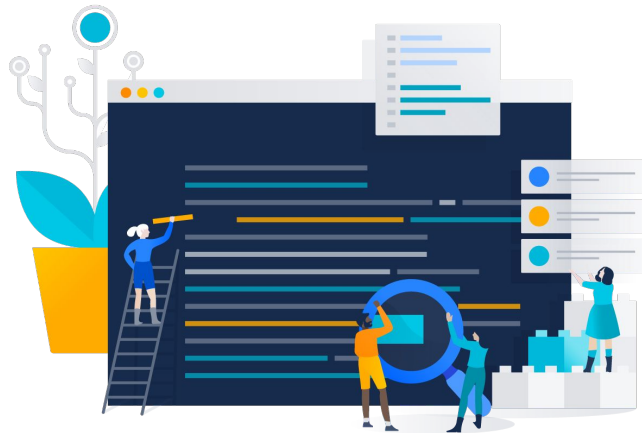


CS130: Software Engineering

Week 8



Agenda

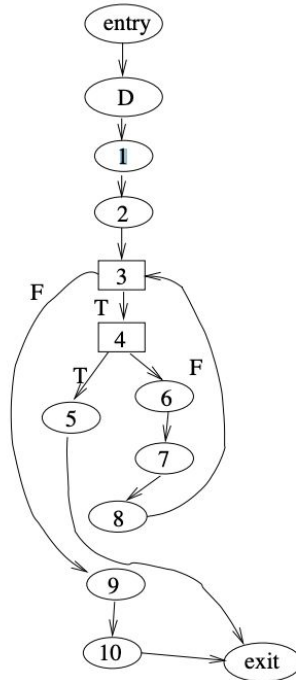
1. Regression Testing & Dangerous Edges
2. Mutation Testing
3. Hoare's Logic
4. Randoop
5. Team Time for Part C

Regression Testing & Dangerous Edges

What are the dangerous edges?

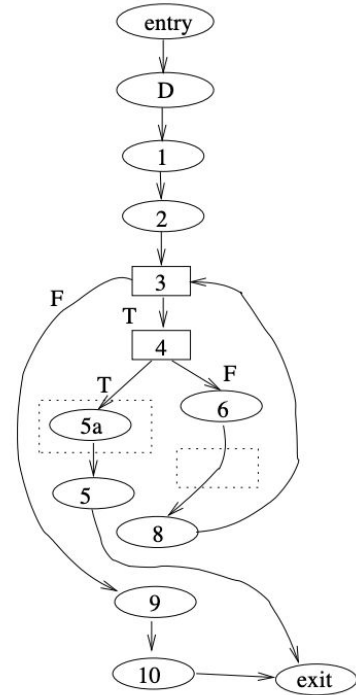
procedure avg

```
1. int count = 0
2. fread(fileptr,n)
3. while (not EOF) do
4.   if (n<0)
5.     return(error)
6.   else
7.     numarray[count] = n
8.     count++
9.   endif
10.  fread(fileptr,n)
11. endwhile
12. avg = calcavg(numarray,count)
13. return(avg)
```



procedure avg'

```
1. int count = 0
2. fread(fileptr,n)
3. while (not EOF) do
4.   if (n<0)
5a.    print("bad input")
5.    return(error)
6.    else
7.      numarray[count] = n
8.    endif
9.    fread(fileptr,n)
10. endwhile
11. avg = calcavg(numarray,count)
12. return(avg)
```



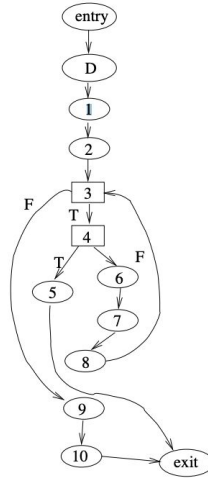
What are the dangerous edges?

procedure avg

```

1. int count = 0
2. fread(fileptr,n)
3. while (not EOF) do
4.   if (n<0)
5.     return(error)
6.   else
7.     numarray[count] = n
8.     count++
9.   fread(fileptr,n)
10. endwhile
11. avg = calcavg(numarray,count)
12. return(avg)

```

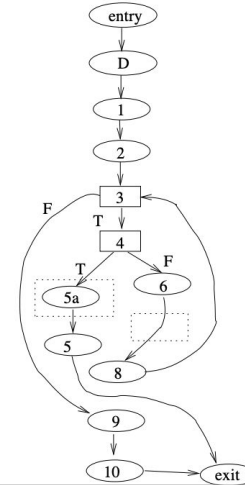
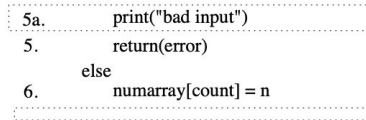


procedure avg'

```

1. int count = 0
2. fread(fileptr,n)
3. while (not EOF) do
4.   if (n<0)
5a.    print("bad input")
5.     return(error)
6.   else
7.     numarray[count] = n
8.   endif
9.   fread(fileptr,n)
10. endwhile
11. avg = calcavg(numarray,count)
12. return(avg)

```



Test #	Input	Output	Edges Traversed
T1	Empty File	0	(Entry -> S1-> S2 -> P3-> S9 -> S10)
T2	-1	Error	(Entry -> S1-> S2 -> P3-> P4 -> S5-> exit)
T3	1 2 3	2	(Entry -> S1-> S2 -> P3-> P4->s6->s7->s8>p3->p4-s6->s7 p3->p4-s6->s7-s8-> s9-s10)

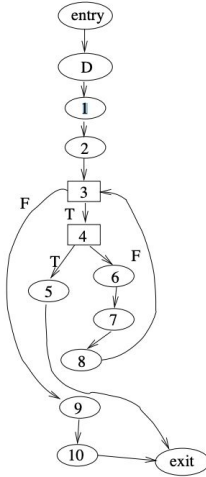
Which tests should be run again after change?

procedure avg

```

1. int count = 0
2. fread(fileptr,n)
3. while (not EOF) do
4.   if (n<0)
5.     return(error)
6.   else
7.     numarray[count] = n
8.     count++
9.   endif
10.  fread(fileptr,n)
11. endwhile
12. avg = calcavg(numarray,count)
13. return(avg)

```

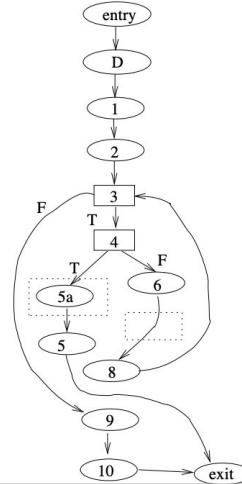
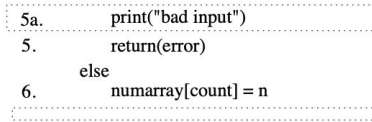


procedure avg'

```

1. int count = 0
2. fread(fileptr,n)
3. while (not EOF) do
4.   if (n<0)
5a.    print("bad input")
5.     return(error)
6.   else
7.     numarray[count] = n
8.   endif
9.   fread(fileptr,n)
10. endwhile
11. avg = calcavg(numarray,count)
12. return(avg)

```



Edge Coverage Matrix

Edge	Test Case
(entry, 1), (1, 2), (2, 3)	1, 2, 3
(3, 9), (9, 10), (10, exit)	1, 3
(3, 4)	2, 3
(4, 5), (5, exit)	2
(4, 6), (6, 7), (7, 8), (8, 3)	3

Test #	Input	Output	Edges Traversed
T1	Empty File	0	(Entry -> S1-> S2 -> P3-> S9 -> S10)
T2	-1	Error	(Entry -> S1-> S2 -> P3-> P4 -> S5-> exit)
T3	1 2 3	2	(Entry -> S1-> S2 -> P3-> P4->s6->s7->s8>p3->p4-s6->s7 p3->p4-s6->s7-s8-> s9-s10)

**Any tests that exercised edges (4,5) and (6,7) in the old version are selected.
=> T2 and T3 should be rerun for the new version.**

Mutation Testing

What is mutation testing?

Bugs, or *mutants*, are automatically inserted into your production code. Your tests are run for each mutant. **If your tests *fail* then the mutant is *killed*. If your tests passed, the mutant *survived*.** The higher the percentage of mutants killed, the more *effective* your tests are.

It's really that simple.

- <https://stryker-mutator.io/>

Stryker

Test your tests with mutation testing.



Types of Mutations

Language Feature	Operator	Description
Access Control	AMC	Access modifier change
Inheritance	IHD	Hiding variable deletion
	IHI	Hiding variable insertion
	IOD	Overriding method deletion
	IOP	overriding method calling position change
	IOR	Overriding method rename
	ISK	<i>super</i> keyword deletion
	IPC	Explicit call of a parent's constructor deletion
Polymorphism	PNC	<i>new</i> method call with child class type
	PMD	Instance variable declaration with parent class type
	PPD	Parameter variable declaration with child class type
	PRV	Reference assignment with other comparable type
Overloading	OMR	Overloading method contents change
	OMD	Overloading method deletion
	OAQ	Argument order change
	OAN	Argument number change
Java-Specific Features	JTD	<i>this</i> keyword deletion
	JSC	<i>static</i> modifier change
	JID	Member variable initialization deletion
	JDC	Java-supported default constructor creation
Common Programming Mistakes	EOA	Reference assignment and content assignment replacement
	EOC	Reference comparison and content comparison replacement
	EAM	Accessor method change
	EMM	Modifier method change

Hoare's Logic

State Predicates

A predicate is a Boolean function on the program state.

Eg: $x == 8$, $x < y$, true

Hoare's Triples

$\{P\} S \{Q\}$

$S \rightarrow$ program

$P \rightarrow$ pre-condition

$Q \rightarrow$ post-condition

$\{P\} S \{Q\}$ means that if S is started in a state satisfying P , then it terminates in Q .

Eg:

$\{\text{true}\} x:=12 \{x=12\}$

$\{x < 40\} x:=12 \{10 < x\}$

$\{m \leq n\} j := (m+n)/2 \{m \leq j \leq n\}$

(Note- $:=$ is the assignment operator)

Q: If $\{P\}S\{Q\}$ and $\{P\}S\{R\}$, then does $\{P\}S\{Q \wedge R\}$ hold?

Yes!

Q: If $\{P\}S\{R\}$ and $\{Q\}S\{R\}$, then does $\{P \vee Q\}S\{R\}$ hold?

Yes!

Strongest Postcondition & Weakest Precondition

Consider the following Hoare Triples:

- A. $\{x=5\} x := x*2 \{x==true\}$
- B. $\{x=5\} x := x*2 \{x>0\}$
- C. $\{x=5\} x := x*2 \{x==10 \vee x==5\}$
- D. $\{x=5\} x := x*2 \{x==10\}$

Which Hoare Triple is the most useful?

(D)

The most precise Q such that $\{P\} S \{Q\}$ is called the ***strongest postcondition*** of S with respect to P.

Consider the following Hoare Triples:

- A. $\{x = 5 \ \&\& \ y = 10\} z := x / y \{ z < 1 \}$
- B. $\{x < y \ \&\& \ y > 0\} z := x / y \{ z < 1 \}$
- C. $\{y \neq 0 \ \&\& \ x / y < 1\} z := x / y \{ z < 1 \}$

Which Hoare Triple is the most useful?

(C)

The most general P such that $\{P\} S \{R\}$ is called the ***weakest precondition of S*** with respect to R, written ***wp(S, R)***

Quick example

Consider code $x:=x+1$ and postcondition $(x > 0)$

One valid precondition is $(x > 0)$, so in Hoare Logic the following is true:

$\{x > 0\} x:=x+1 \{x > 0\}$

Another valid precondition is $(x > -1)$, so:

$\{x > -1\} x:=x+1 \{x > 0\}$

Answer:

$(x > -1)$ is *weaker* than $(x > 0)$ (...because $(x > -1) \Rightarrow (x > 0)$)

Now, your turn!

Find the weakest precondition such that the following code terminates without any exceptions.

```
if(x > 0) {  
    a = a - 1;  
} else {  
    b = b + 1;  
}  
  
arr = new int[3];  
c = arr[-1 * a * b];
```

Random Testing

Random testing is a black-box software testing technique where programs are tested by generating random, independent inputs. Results of the output are compared against software specifications to verify that the test output is pass or fail.

Random testing has the following strengths:

- It is cheap to use: it does not need to be smart about the program under test.
- It does not have any bias: unlike manual testing, it does not overlook bugs because there is misplaced trust in some code.
- It is quick to find bug candidates: it typically takes a couple of minutes to perform a testing session.
- If software is properly specified: it finds real bugs.

Randoop

What is Randoop?

- Randoop is a unit test generator for Java. It automatically creates unit tests for your classes, in JUnit format.

How does Randoop work?

- Randoop generates unit tests using feedback-directed random test generation.
- This technique pseudo-randomly, but smartly, generates sequences of method/constructor invocations for the classes under test.
- Randoop executes the sequences it creates, using the results of the execution to create assertions that capture the behavior of your program. Randoop creates tests from the code sequences and assertions.

Randoop can be used for two purposes:

- To find bugs in your program
- To create regression tests to warn you if you change your program's behavior in the future

Quick example

Consider code $x:=x+1$ and postcondition $(x > 0)$

One valid precondition is $(x > 0)$, so in Hoare Logic the following is true:

$\{x > 0\} x:=x+1 \{x > 0\}$

Another valid precondition is $(x > -1)$, so:

$\{x > -1\} x:=x+1 \{x > 0\}$

Q. So which of the above preconditions do you choose as the *weakest precondition*?

Team Time!