CS 130 SOFTWARE ENGINEERING

TESTING

REGRESSION TESTING

Professor Miryung Kim

UCLA Computer Science

Based on Materials from Miryung Kim

PREVIEW

- We will discuss regression test selection, prioritization, and augmentation methods.

REGRESSION TESTING

AGENDA

- Regression testing selection
- Change impact analysis: which tests are affected by program changes?

REGRESSION TESTING (RETESTING)

- Suppose that you've tested a product thoroughly and found no errors.
- Suppose that the product is then changed in one area and you want to be sure that it still passes all the tests it did before the change.
- Testing designed to make sure that the software hasn't taken a step backward, or "regressed" is called "regression testing"

REGRESSION TEST SELECTION

- P: old version
- P’: new version
- T is a test suite for P
- Assume that all tests in T ran on P. => Generate coverage matrix C.
- Given the delta between P and P’ and the coverage matrix C, identify a subset of T that can identify all regression faults. (Safe RTS)

HARROLD & ROTHERMEL'S REGRESSION TEST SELECTION

- A safe, efficient regression test selection technique
- Regression test selection based on traversal of control flow graphs for the old and new version.
- The key idea is to select tests that will exercise **dangerous edges** in the new program version.

HARROLD & ROTHERMEL'S RTS

- Dangerous edges are the edges in the old CFG where target node is different in the new CFG
- If you find all test cases that exercised dangerous edges, those tests are as effective as running all test cases.

STEP 1. BUILD CFG

Control flow graph for the old version

```
     Procedure avg

S1.  count = 0
S2.  fread(fileptr,n)
P3.  while (not EOF) do
P4.    if (n<0)
S5.       return(error)
       else
S6.       numarray[count] = n
S7.       count++
       endif
S8.    fread(fileptr,n)
     endwhile
S9.  avg = calcavg(numarray,count)
S10. return(avg)
```

STEP 2. RUN T = {T1, T2, ...} ON P

Test Information			
Test	Type	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 p3->p4-s6->s7-s8-> s9-s10

STEP 3. BUILD EDGE COVERAGE MATRIX

Test History

Edge	TestsOnEdge(edge)
(entry, D)	111
(D, S1)	111
(S1, S2)	111
(S2, P3)	111
(P3, P4)	011
(P3, S9)	101
(P4, S5)	010
(P4, S6)	001
(S5, exit)	010
(S6, S7)	001
(S7, S8)	001
(S8, P3)	001
(S9, S10)	101
(S10, exit)	101

STEP 4. TRAVERSE TWO CFGS IN PARALLEL

Control flow graph for the old version

```
Procedure avg2

S1'. count = 0
S2'. fread(fileptr,n)
P3'. while (not EOF) do
P4'.    if (n<0)
S5a.       print("bad input")
S5'.       return(error)
        else
S6'.       numarray[count] = n
        endif
S8'.    fread(fileptr,n)
     endwhile
S9'. avg = calcavg(numarray,count)
S10'.return(avg)
```

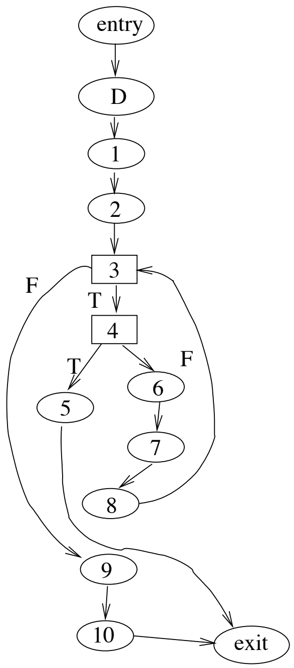
STEP 5. SELECT TESTS RELEVANT TO DANGEROUS EDGES

- Which tests are relevant to changes and thus must be rerun?

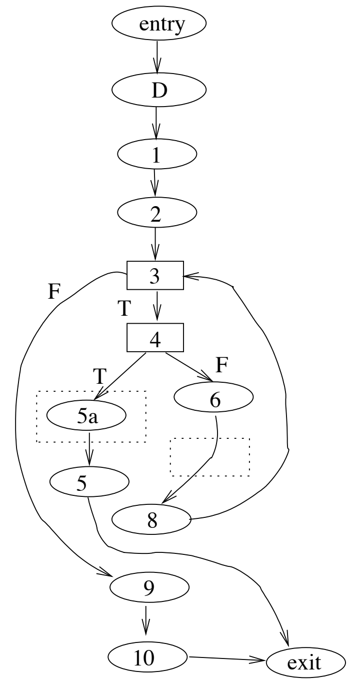
- Answers are shown in the next slides step by step.

OLD AND NEW CFG FOR AVG

```
procedure avg

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

        endif
8.      fread(fileptr,n)
   endwhile
9. avg = calcavg(numarray,count)
10. return(avg)
```



A TEST SUITE FOR PROGRAM AVG

Test Case	Input	Expected Output
1	empty file	0
2	-1	error
3	1 2 3	2

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

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

PRACTICE QUESTION: RTS #1

```
void fun(int N, int k, int j){
  int sum = 0;
  int product = 1;
  for(i = 1; i <= N; i=i+1){
    sum = sum + i;
    if (k%2==0) product = product*i;
  }
  write(sum);
  write(product);
}
```

```
void fun(int N, int k, int j){
  int sum = 0;
  int product = 1;
  for(i = 1; i <= N; i=i+1){
  sum = sum + i;
  if (k%2==0) product =
product* 2*i;
  }
  write(sum);
  write(product);
}
```

PRACTICE QUESTION: RTS #1

- Draw the control flow graphs for the new version of the program
- Mark the dangerous edges on your control flow graphs.
- Dangerous edges mean the control flow graph edges that have different target nodes in the new version.

PRACTICE QUESTION: RTS #1

- Identify a subset of the following tests that are relevant to the edits and thus must be re-run for the new version of the program.
- Mark if and only if the test must be selected for the new version.
- T1 (N=0, k=1, j=1)
- T2 (N=10, k=2, j=0)
- T3 (N=1, k=3, j=1)
- Answer: T2 should be rerun for the new version.

HARROLD ET AL. RTS FOR JAVA

- Regression Test Selection for Java Software
- OOPSLA 2001
- What are main challenges for making RTS work in Java?

MAIN CHALLENGES FOR MAKING RTS WORK IN JAVA

- Java language features: in particular, (1) polymorphism, (2) dynamic binding, and (3) exception handling
- Why is polymorphism & dynamic binding difficult to handle in RTS?
- The target of method calls depends on the dynamic type of a receiver object.

- Java language features: in particular, (1) polymorphism, (2) dynamic binding, and (3) exception handling
- Why is polymorphism & dynamic binding difficult to handle in RTS?
  - The target of method calls depends on the dynamic type of a receiver object.

```
class B extends A {
};
class C extends B {
 public void m(){...};
};
void bar(A p) {
  A.foo();
  p.m();
}
```

```
class B extends A {
 public void m(){...};
};
class C extends B {
 public void m(){...};
};
void bar(A p) {
  A.foo();
  p.m();
}
```

EXTERNAL LIBRARIES AND COMPONENTS

- Why is it important to model interaction between the main code and its libraries?
- External library code can invoke internal methods if the internal methods override external methods.

- External libraries and components
  - Why is it important to model interaction between the main code and its libraries?
  - External library code can invoke internal methods if the internal methods override external methods.

```
class B extends A {
  public void foo() {...};
}
class C extends B {
 public void bar() {...};
};
```

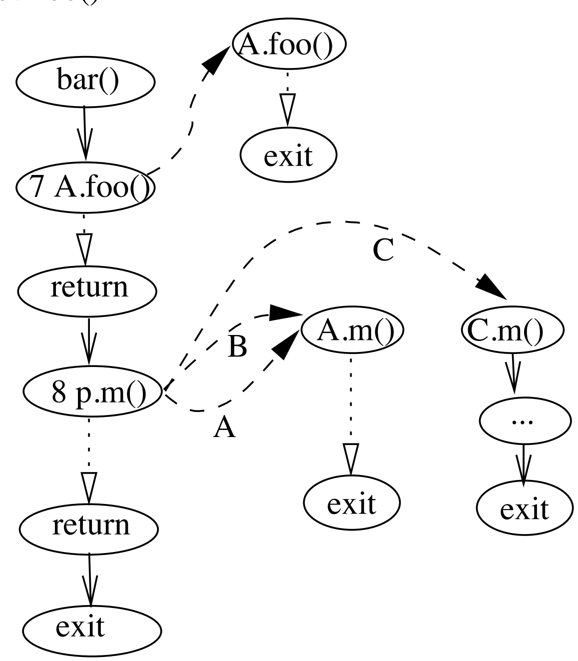
```
class B extends A {
  public void foo() {...};
  public void bar() {...};
}
class C extends B {

};
```

JIG (JAVA INTERCLASS GRAPH)

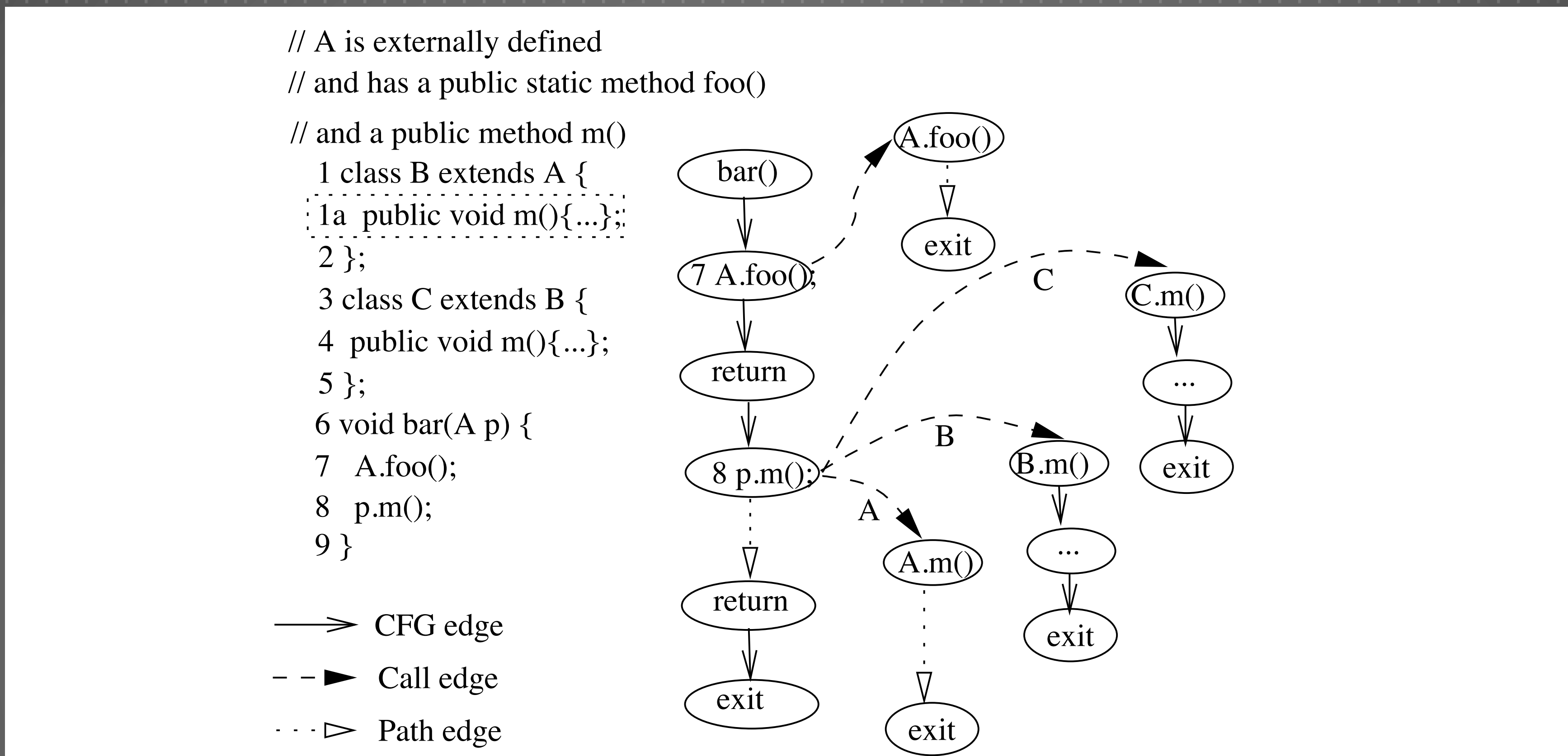
- JIG extends CFG to handle five kinds of Java features
- (1) variable and object type information
- (2) internal and external methods
- (3) interprocedural interactions through calls to internal methods or external methods from internal methods.
- (4) interprocedural interactions through calls to internal methods from external methods
- (5) exception handling

OLD CFG WITH DYNAMIC DISPATCHING

```
// A is externally defined
// and has a public static method foo()
// and a public method m()
1 class B extends A {
2 };
3 class C extends B {
4  public void m(){...};
5 };
6 void bar(A p) {
7   A.foo();
8   p.m();
9 }
```



NEW CFG FOR DYNAMIC DISPATCHING

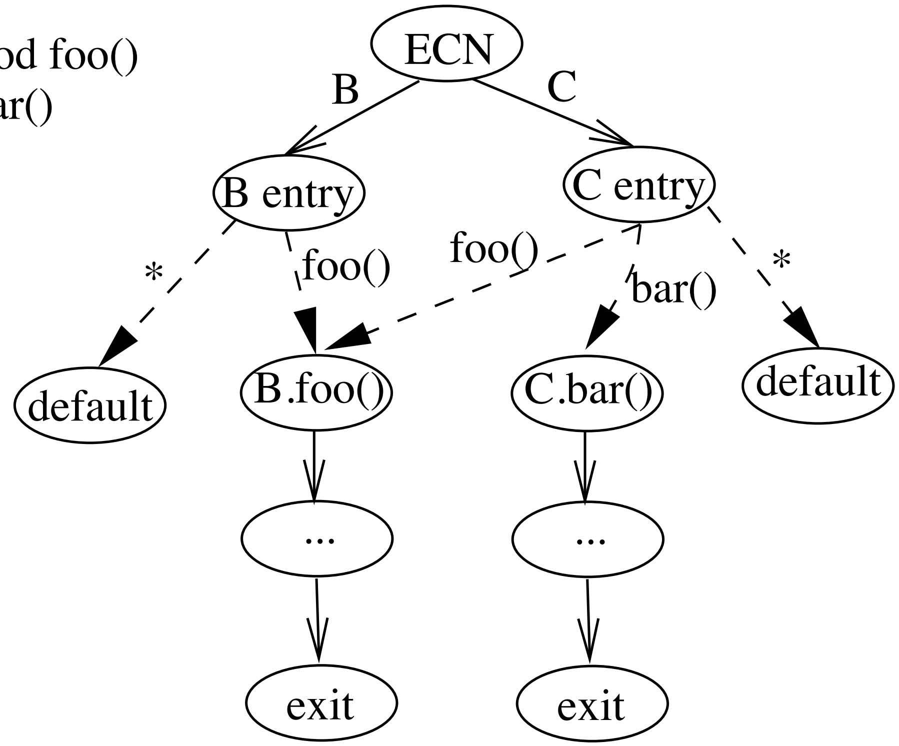


DANGEROUS EDGES

- Calling p.m() on an object with type B

OLD INTERACTION GRAPH

```
//A is externally defined
// and has a  public method foo()
// and a public method bar()
class B extends A {
  public void foo() {...};
}
class C extends B {
 public void bar() {...};
};
```

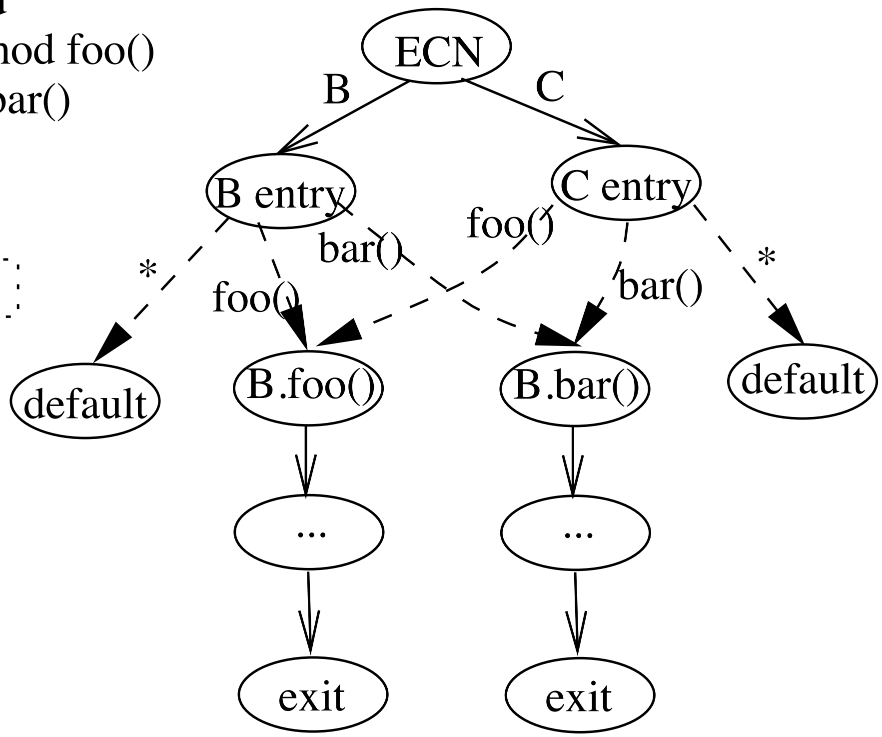


CFG edge

Call edge

NEW INTERACTION GRAPH

```
//A is externally defined
// and has a  public method foo()
// and a public method bar()

class B extends A {
  public void foo() {...};
  public void bar() {...};
}
class C extends B {

};
```



CFG edge

Call edge

DANGEROUS EDGES

- Calling bar() on object of type B
- Calling bar() on object of type C

JAVA RTS ALGORITHM

- start from either main () node, the ECN node, static method entries,
- the algorithm traverses the JIgs and add the dangerous edges that it finds to E.
- It marks N as "N visited" to avoid comparing N and N' again in a subsequent iteration
- If the target along the same edge is different between two graphs, then it becomes a dangerous edge.
- One way to determine the equivalence of two nodes is to examine the lexicographic equivalence of the text associated with the two nodes.

TAKE AWAY FOR OBJECT ORIENTED TESTING

- You changed A.m().
  - It is your job to look what classes A extends.
  - It is your job to investigate who extends A.
  - It is your job to investigate who uses A.
  - Whether it creates any changes in dynamic dispatching behavior of your code.

RECAP

- We have studied sub-problems within regression testing.
- We have studied an algorithm for regression test selection.
- Preview model based testing.
  - How do you test? Test well.

QUESTIONS?