Mason Dinardi & Conner Cross

ESOF 322 - Homework 5

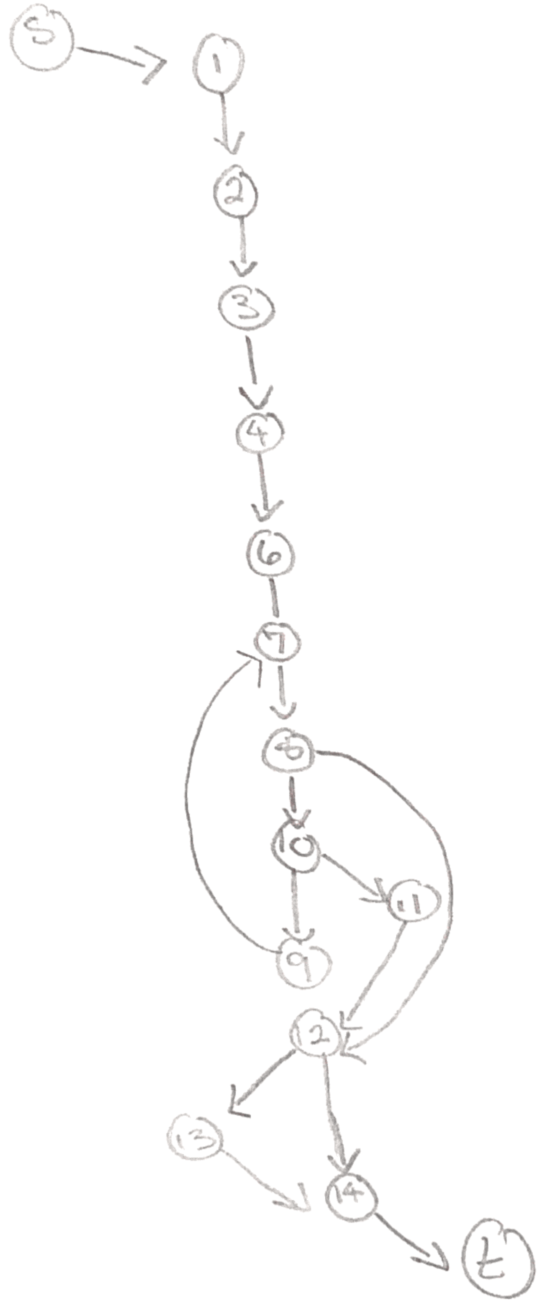
11/5/2019

ESOF 322 Software Engineering I
Homework #5 (32 points total)
Due: Tuesday November 5th
Hand in your printed copy in class, and submit a PDF file into D2L.

Question 1 (10 pts)

a) Create a control flowgraph for the *sieve* algorithm. To the left of the line numbers in the source code clearly identify the nodes that will be used in your graph. Once you have identified the nodes, draw the control graph. **(4 pts)**

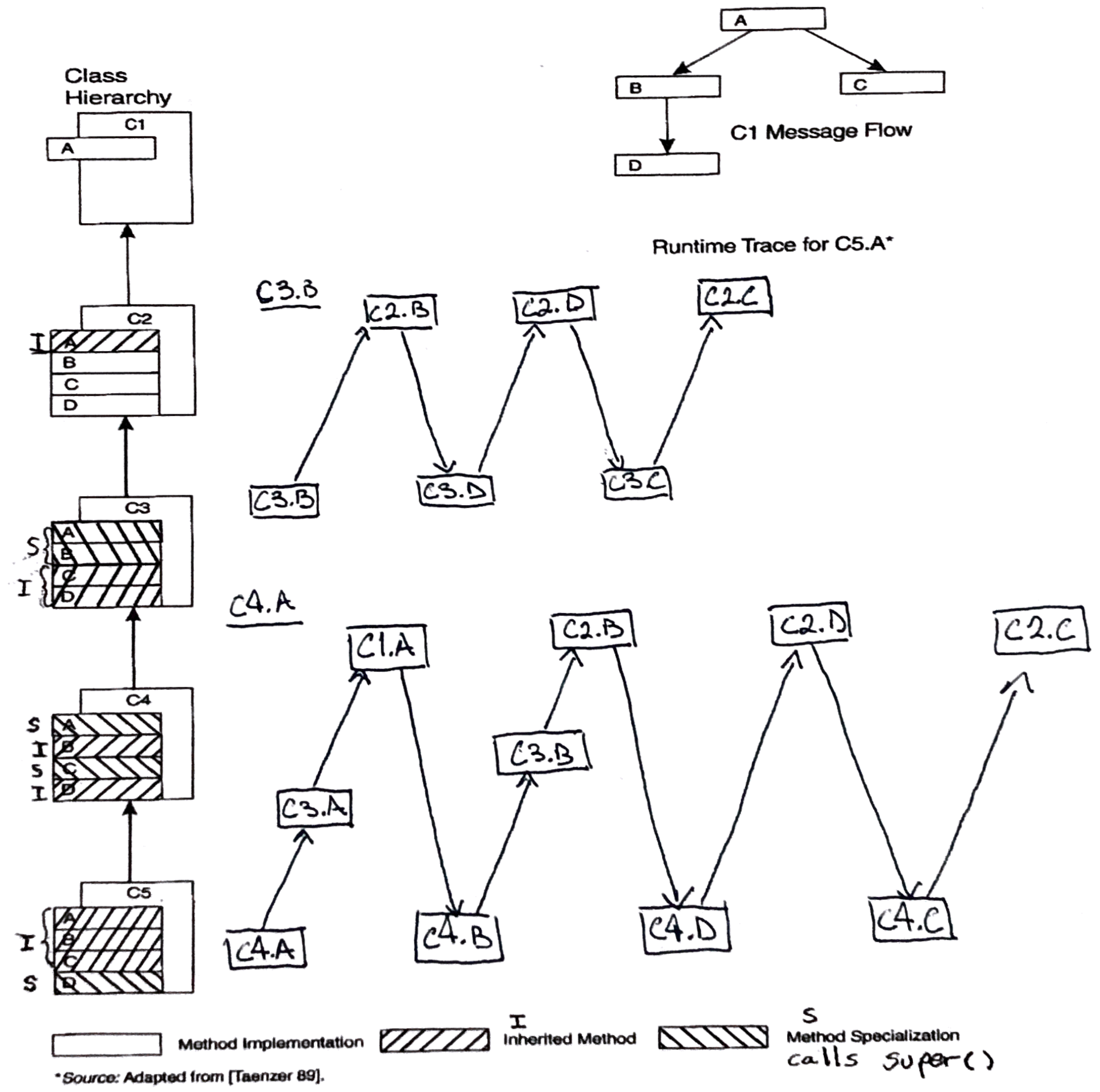
```
      1. /* Find all primes from 2-upper_bound using Sieve of Eratosthanes */
      2.
      3. #include
 1    4. typedef struct IntList {
      5.           int value;
      6.           struct IntList *next;
      7.           } *INTLIST, INTCELL;
      8. INTLIST sieve ( int upper_bound ) {
      9.
   [ 10.        INTLIST prime_list = NULL;    /* list of primes found */
 2 [ 11.        INTLIST cursor;               /* cursor into prime list */
   [ 12.        int candidate;                /* a candidate prime number */
   [ 13.        int is_prime;                 /* flag: 1=prime, 0=not prime */
     14.
     15.         /* try all numbers up to upper_bound */
 3   16.         for (candidate=2;
     17.
 4   18.              candidate <= upper_bound;
 5   19.              candidate++) {
     20.
 6   21.          is_prime = 1; /* assume candidate is prime */
 7   22.          for(cursor = prime_list;
     23.
 8   24.               cursor;
 9   25.               cursor = cursor->next) {
     26.
10   27.             if (candidate % cursor->value == 0) {
     28.
     29.               /* candidate divisible by prime */
     30.              /* in list, can't be prime */
11 [ 31.               is_prime = 0;
   [ 32.               break;   /* "for cursor" loop */
     33.             }
     34.          }
12   35.          if(is_prime) {
     36.
     37.            /* add candidate to front of list */
   [ 38.            cursor = (INTLIST) malloc(sizeof(INTCELL));
13 [ 39.            cursor->value = candidate;
   [ 40.            cursor->next  = prime_list;
   [ 41.            prime_list = cursor;
     42.          }
     43.         }
15   44.         return prime_list;
     45.        }
```


s
1
2
3
4
6
7
8
10
9
11
12
13
14
t

* b) Provide a set of test cases that would give 100% Node Coverage (NC). **(2 pts)**
* c) Provide a set of test cases that would give 100% Edge Coverage (EC). **(2 pts)**
* d) Is 100% NC or 100% EC possible in general? Why, or why not? **(2 pts)**

<u>Question 2 (10 pts)</u>

a) Draw the execution of the calls that exhibit the YoYo problem for a runtime trace of C3.B, and for C4.A (Draw both on the diagram below). **(8 pts)**
* b) Describe what happens when we call C1.D **(2 pts)**



*Source: Adapted from [Taenzer 89].

* SEE ATTACHED

Question 1

1b) T = {{1, 2, 3, 4, 6, 7, 8, 10, 9, 7, 8, 12, 13, 14, t}, {1, 2, 3, 4, 6, 7, 8, 10, 11, 12, 13, 14, t}

1c) T = {{1, 2, 3, 4, 6, 7, 8, 10, 11, 12, 13, 14, t}, {1, 2, 3, 4, 6, 7, 8, 10, 9, 7, 8, 12, 14, t}}

1d) While 100% Node coverage is possible in general, 100% Edge coverage is generally not. This is because to guarantee 100% Edge coverage you would need to have a solution to the Halting problem. Things like infinite loops make determining if every edge has been covered impossible.

Question 2

2b) When we call C1.D during the runtime trace of a call to a lower class, we will call the D method on the lower class (i.e. C5.D). However, if we just directly call D from C1 (C1.D) we will get an error as C1 does not have any visibility of a D method.

Question 3

3a) T = {1}
3b) T = {0}
3c) T = {4}

The test case that would kill all three mutants simultaneously would be T = {{1, 0, 4}}