

EECS 481 — Software Engineering — Exam #1

- **Write your name and UM username on the exam.**
- There are ten (10) pages in this exam (including this one) and seven (7) questions, each with multiple parts. Some questions span multiple pages. If you get stuck on a question, move on and come back to it later.
- You have 1 hour and 20 minutes to work on the exam.
- The exam is closed book, but you may refer to your two page-sides of notes.
- Even vaguely looking at a cellphone or similar device (e.g., tablet computer) during this exam **is cheating**.
- Please write your answers in the space provided on the exam. Clearly mark your solutions. You may use the backs of the exam pages as scratch paper. Do not use any additional scratch paper.
- Solutions will be graded on correctness and clarity. Each problem has a relatively simple and straightforward solution. We may deduct points if your solution is far more complicated than necessary.
 - *Good Writing Example:* Testing is an expensive activity associated with software maintenance.
 - *Bad Writing Example:* Im in ur class, @cing ur t3stz!1!
- If you leave a non-extra-credit portion of the exam blank, **you will receive one-third of the points for that small portion (rounded down) for not wasting time.**

UM username: _____

NAME (print): _____

UM username: (yes, again!) _____

Problem	Max points	Points
1 — Software Process Narrative	13	
2 — Test Inputs and Coverage	18	
3 — Short Answer	15	
4 — Mutation Testing	15	
5 — Dataflow Analysis	20	
6 — Quality Assurance Analyses	19	
Extra Credit	0	
TOTAL	100	

How do you think you did? _____

1 Software Process Narrative (13 points)

(1 pt. each) Read the following narrative. If a text segment corresponds to, or demonstrates, a concept below, fill in its ____ blank with the letter of the *most appropriate or specific* concept. **Or**, if a text segment is false or very unlikely in the real world (cf. the readings), fill in its blank with an X. Otherwise, leave it untouched. An option may be used more than once.

A. a/b testing	B. beta testing	C. effort estimation	D. integration testing
E. priority	F. regression testing	G. resolution	H. severity
I. streetlight effect	J. threat to validity	K. worksforme	X. false

Unrealistic Software is maintaining a hot new click-fest mobile game, *Clash of the Philosophers*.

- For part of a tie-in campaign with another company, Unrealistic Software wants to determine how much their players like various philosophers, so they measure how often their players purchase them.
- By slightly varying the icon border color, they find that in-app purchases of the Simone de Beauvoir character increase by 10%; they decide to implement that color change.
- Developers and program managers are most interested in learning how to generate 100% test coverage for this new color-change code.
- While development is focused on code coverage, new defects are reported by end users.
- The majority of the defect reports include stack traces.
- The stack traces seem to implicate the in-app purchase code.
- Management asks the developers how long it will take to fix this defect.
- It is decided that this defect must be fixed right now.
- However, developers need help pinning down the defect.
- Since they already have GPS location information about their players, they decide to look for a geographic correlation, but do not find one.
- They then hypothesize that the problem could be a database issue, so they decide to use Microsoft's CHESS tool to gain more information.
- Eventually, they test the new color-changing code in conjunction with the old in-app purchase code.
- Developers check in a change that they believe fixes the bug, but later the bug resurfaces and the defect report is reopened.

2 Test Inputs and Coverage (18 points)

(4 pts. each) Consider the following program with blanks. We are concerned with statement coverage, but only for the statements labeled S_1 through S_5.

```
1 void liskov(int a, int b, int c) {
2     S_1;
3     if (a == _____)
4         S_2;
5     if (b < _____)
6         S_3;
7     else if (b == a)
8         S_4;
9     if (a == b _____ c)
10        S_5;
11 }
```

Now consider the following three test inputs, T_1 through T_3:

T_1 = liskov(1,2,3)	T_2 = liskov(8,8,1)	T_3 = liskov(3,1,2)
---------------------	---------------------	---------------------

Using all three test inputs results in 100% coverage of the labeled statements. Using either T_1 or T_2 alone results in 40%. Using T_1 with T_3 results in 80%. Using T_2 with T_3 also results in 80%. Fill in each blank in the program with a **single** letter, integer or symbol so that it matches this coverage.

(6 pts.) Consider the following program:

```
1 void lovelace(int a, int b, int c) {
2     int local = 0;
3     if (a < b)                local += 1;           else local += 0;
4     if (c == 5)               local += 2;           else local += 0;
5     if (local == 3 && c < 4)  local = 6;           else local = 7;
6 }
```

Give a smallest set of test inputs (in terms of the number of test inputs) resulting in maximal branch coverage for this program:

3 Short Answer (15 points)

- (a) (3 pts.) Write a method accepting one input parameter for which test input generation via constraint solving will work better than test input generation at random.
- (b) (2 pts.) In at most three sentences, support or refute the claim that Microsoft's Driver Verifier is an instance of Mocking.
- (c) (3 pts.) In at most three sentences, support or refute the claim that the Cyclomatic Complexity metric helps to identify difficult-to-understand code.

(d) (*3 pts.*) You believe slow code is more likely to be buggy, so you design a dynamic analysis similar to Tarantula (coverage-based fault localization) that multiplies the default suspiciousness rating of each statement by the time spent in its enclosing method. In at most three sentences, describe the instrumentation for your analysis.

(e) (*2 pts.*) In at most three sentences, support or refute the claim that watchpoints help to debug race conditions.

(f) (*2 pts.*) Describe a defect for which Delta Debugging would struggle to find a minimal failure-inducing input.

4 Mutation Testing (15 points)

Consider this method to convert a number of days (since January 1, 1980) into a year. Note that 1980 *is* a leap year. The original program is shown on the left; three first-order mutants are each indicated by a comment on the right.

```
1 def zune(days):
2     year = 1980
3     while (days > 365):
4         if isLeapYear(year):
5             if (days >= 366): # Mutant 1 has if (days > 366):
6                 days -= 366
7                 year += 1 # Mutant 2 has year += 0
8         else:
9             days -= 365 # Mutant 3 has days = 0
10            year += 1
11    return year
```

(10 pts.) Complete the table below by indicating whether or not each test kills Mutant 2 and/or Mutant 3. (Mutant 1 is only killed by Test 2.)

	Input (days)	Oracle	Mutant 1	Mutant 2	Mutant 3
Test 1	365	1980	—		
Test 2	366	1981	killed		
Test 3	366 + 1	1981	—		
Test 4	366 + 365 + 1	1982	—		
Test 5	366 + 365 + 365 + 1	1983	—		

(1 pt.) What is the mutation score for Tests 1–5 using Mutants 1–3?

(1 pt.) What is the mutation score for Tests 1–5 using Mutants 1–2?

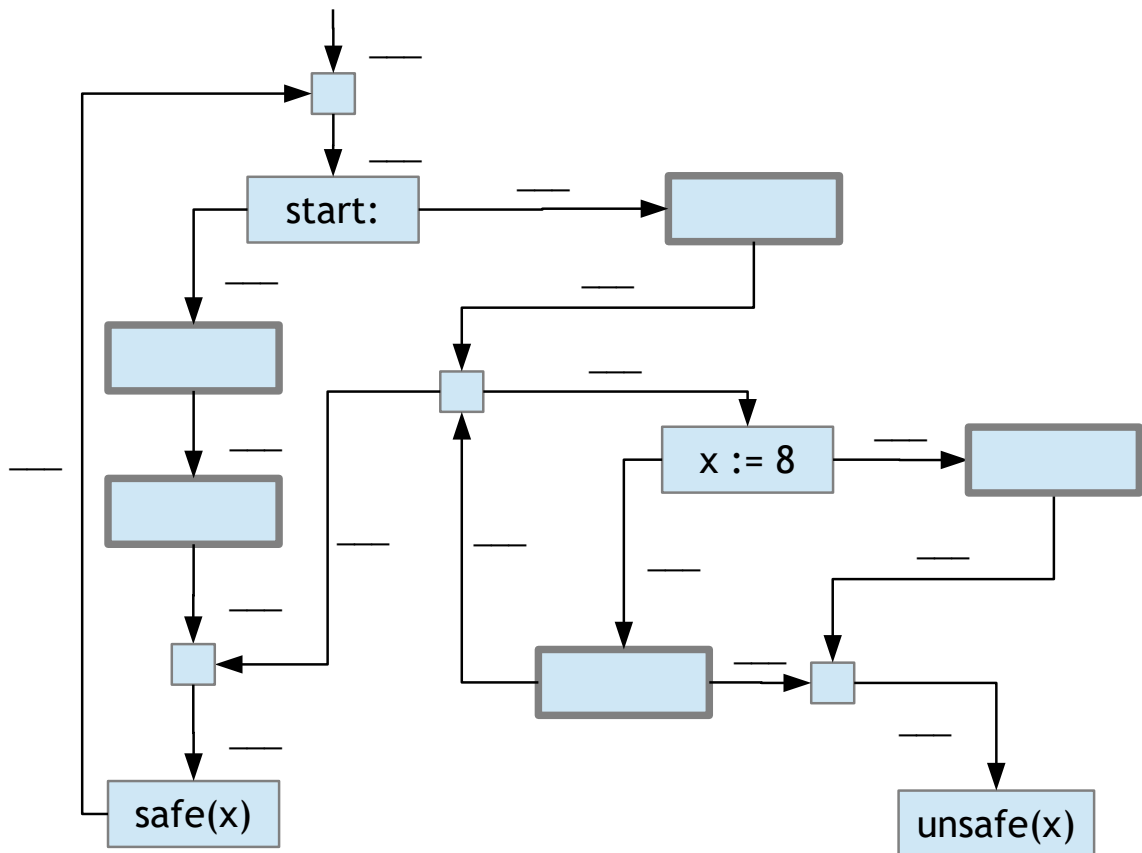
(1 pt.) What is the mutation score for Tests 1–3 using Mutants 1–3?

(2 pts.) In at most three sentences, support or refute the claim that mutation analysis agrees with your intuitive notion of test suite adequacy in this example.

5 Dataflow Analysis (20 points)

Consider the *constant propagation* dataflow analysis used in class to determine if a pointer variable is definitely null when used. We associate with each variable a dataflow analysis fact: either $*$ (“the variable holds a value, but our analysis cannot be certain which value”), $\#$ (“this point in the program has not yet been reached by our analysis”) or a number c (“at this point in the program, we are certain the value of this variable is exactly c ”).

For this problem, we also **extend** our notion of dataflow analysis to include simple arithmetic (called *constant folding*). For example, if we know that $x = 7$ before the statement $x = x+2$, we immediately conclude that $x = 9$ after it.



(10 pts.) Use each of the statements from the box below exactly once to fill in each of the five large bolded nodes so that the dataflow analysis correctly indicates that x is not null when used at `safe(x)` but conservatively indicates that x may be null when used at `unsafe(x)`.

$x := x + 1$	$x := 2$	$x := 3$	$x := x - 5$	$x := \text{foo}()$
--------------	----------	----------	--------------	---------------------

(10 pts.) Fill in each blank (---) with the final dataflow analysis fact associated with that edge.

6 Quality Assurance Analyses (19 points)

Consider the Eraser dynamic lockset analysis, which tracks and intersects the set of locks held by threads as they access variables.

```
1 def thread1(a):
2     global lock1
3     global shared
4     if (a == 5):
5         acquire(lock1)
6         shared = shared + 1
7     if (a == 5):
8         release(lock1)

def thread2(b):
    global lock1
    global shared
    if (b == 7):
        acquire(lock1)
        shared = shared + 1
    if (b == 7):
        release(lock1)
```

(3 pts.) Either give an input (a, b) that would cause Eraser to mistakenly conclude that there is no race condition or indicate that it is impossible to do so.

(3 pts.) Either give an input (a, b) that would cause Eraser to correctly conclude that there is a race condition or indicate that it is impossible to do so.

(5 pts.) Briefly summarize the sub-activities that make up modern (“passaround”) code review. Describe multiple outcomes.

(8 pts.) Pick **two** of manual code inspection, automatic static analysis, testing, and automatic dynamic analysis. For each chosen approach, describe a defect or quality concern and explain, in one sentence, why that defect would be better handled with that approach than with the other three approaches.

7 Extra Credit (1 point each)

What is one thing you would change about this class for next year?

What is one thing you would retain about this class for next year?

The “Producing Wrong Data Without Doing Anything Obviously Wrong!” paper argues that *which kind* of bias is significant and commonplace?

Name *one benefit* and *one cost* of a test suite augmented with MC/DC coverage (compared to standard functional testing) described in “An Empirical Evaluation of the MC/DC Coverage Criterion on the HETE-2 Satellite Software”.

List *one approach* to oracle automation from “The Oracle Problem in Software Testing: A Survey”.

What was the “surprising” *result* in “Gender differences and bias in open source: pull request acceptance of women versus men”?

List *one issue* discussed in “A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World”.

What *personality difference* was found between managers and testers in “Beliefs, Practices, and Personalities of Software Engineers: A Survey in a Large Software Company”?