# EECS 481 — Software Engineering
# Winter 2019 — Exam #1

- **Write your UM uniqname and UMID and your name 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 uniqname: _____

UM ID: _____

Name (print): _____

UM uniqname: (yes, again!) _____

UM ID: (yes, again!) _____

| Problem | Max points | Points |
|---|---|---|
| 1 — Software Process Narrative | 13 | |
| 2 — Test Inputs and Coverage | 18 | |
| 3 — Short Answer | 18 | |
| 4 — Mutation Testing | 16 | |
| 5 — Dataflow Analysis | 20 | |
| 6 — Quality Assurance Analyses | 15 | |
| Extra Credit | 0 | |
| TOTAL | 100 | |

How do you think you did? _____

# 1 Software Process Narrative (13 points)

*(1 pt. each)* Read the following narrative. Fill in each ____ blank with the single *most specific or appropriate* corresponding concept from the answer bank. (Each ____ blank does have a corresponding answer.) Each option from the answer bank will be used *at most once*.

| | | | |
|---|---|---|---|
| A. Alpha Testing | B. Beta Testing | C. Call-Graph Profile | D. Comparator |
| E. Conditional Breakpoint | F. Dataflow Analysis | G. Development Process | H. Effort Estimation |
| I. Formal Code Inspection | J. Instrumentation | K. Integration Testing | L. Mocking |
| M. Oracle | N. Passaround Code Review | O. Perverse Incentive | P. Priority |
| Q. Resolution | R. Severity | S. Software Metric | T. Spiral Development |
| U. Threat to Validity | V. Triage | W. Watchpoint | X. Waterfall Model |

Unrealistic Software is developing a hot new online multiplayer game, *ApexUnknown's Defense League of OverGrounds Legendary Pals 76*.

____ Managers decide to plan the organization of various software development activities into distinct phases.

____ Managers decide to focus on the construction of an increasingly-complex series of working prototypes.

____ Managers decide to use the Maintainability Index to assess the code because it is built in to Visual Studio.

____ The Maintainability Index misleads the managers by pointing out code that is large, rather than code that is truly difficult to maintain.

____ Developers are instructed to lower the Maintainability Index at any cost, resulting in methods that are very short but almost impossible to comprehend or maintain.

____ Developers are worried about the quality of the software prototype. Internally, they use a tool to randomly generate and run inputs.

____ Developers decide, for simplicity, that at the very least their software should not crash or throw exceptions on those random inputs.

____ A new defect report comes in. QA finds it to be a valid report and not a duplicate. Developers are instructed to address it.

____ Management decides that the new defect report must be addressed immediately: the business cost of not fixing it is deemed to be too high.

____ Management asks developers to predict how long it will take to localize and fix the defect.

____ Developers modify the code to print out which lines are visited during execution.

____ The bug report mentions the corruption of a particular memory address, so the developer uses a debugger to halt execution whenever the value at that address changes.

____ However, the code works fine when the developers run it. They cannot reproduce the defect, so they move on to other tasks.

## 2 Test Inputs and Coverage (18 points)

*(4 pts.)* Give a smallest test suite for `lamarr()` below that maximizes *statement coverage* for statements S_1 through S_8. (A test for `lamarr()` is a value for `x` and a value for `y`. A test suite is a set of tests.)

```
1  void lamarr(bool x, bool y) {
2    S_1;
3    if (x == y)    S_2;
4    else           S_3;
5    if (x || y)    S_4;
6    else           S_5;
7    if (x && y)    S_6;
8    else           S_7;
9    S_8;
10 }
```

*(2 pts.)* What is the maximum *path coverage* possible for `lamarr()` in practice? (Express your answer as a fraction.)

*(4 pts.)* Give a smallest test suite for `lamarr()` that maximizes *path coverage*.

*(4 pts.)* Suppose we define full *boolean coverage* to require that every boolean variable or expression evaluate to true (e.g., on one input) and also to false (e.g., on another input). Give a smallest test suite for `hedy()` below that maximizes *boolean coverage*.

```
1  void hedy(int a, int b) {
2    bool p, q;
3    p = a < b;
4    q = a > b;
5    if (p || q)    S_1;
6    else           S_2;
7  }
```

*(4 pts.)* Give a smallest test suite for `hedy()` that maximizes *branch coverage* but *not* boolean coverage.

# 3 Short Answer (18 points)

(a) *(4 pts.)* Support or refute the claim that spectrum-based fault localization (such as the Tarantula tool), which is based on differences in coverage between passing and failing runs, would be a useful technique for localizing memory corruption bugs.

(b) *(3 pts.)* For each of *mocking*, *unit testing* and *integration testing*, describe a defect or development scenario for which that technique is likely to work well.

(c) *(3 pts.)* Explain the relationship between *reopened* bug reports and *regression testing*. What process would you suggest to minimize reopened bug reports?

(d) *(4 pts.)* Support or refute the claim that modern passaround code review should be performed *after* the new code is subjected to unit and regression testing if goals of code review are taken to be "code improvement" and "reduce costs".

(e) *(4 pts.)* Choose a modern passaround code review requirement from a large company (such as Facebook's "all changes have at least two reviewers" or Google's "you must have a readability badge in the programming language") and support or refute the claim that that requirement aids the goals of "defect finding" and "knowledge transfer".

# 4 Mutation Testing (16 points)

Consider the following implementation of ascending bubble sort and some associated test inputs, oracles and suites:

```
1  def BubbleSort(arr):
2    n = len(arr)                    # reminder: len([5,6]) = 2
3    for i in range(n):              # reminder: range(3) = [0,1,2]
4      for j in range(i):
5        if (arr[j] >= arr[j+1]):
6          arr[j], arr[j+1] = arr[j+1], arr[j] # cool python swap
7
8  testInput1 = [11, 22]             # oracle1 = [11, 22]
9  testInput2 = [88, 66, 77]         # oracle2 = [66, 77, 88]
10 testInput3 = [55, 33, 0, 99]      # oracle3 = [0, 33, 55, 99]
11
12 testSuiteA = [ testInput1 ]
13 testSuiteB = [ testinput2, testInput3 ]
```

*(6 pts.)* Suppose the only mutation operator available to you is *single variable renaming*: on line $A$ change the $B$th instance of variable $C$ to variable $D$. For example, one mutation might be "on line 6 change the 3rd instance of `j` to `i`". You may only mutate the method body (lines 2–6), not the tests. Give an example of a single mutation such that both `testSuiteA` and `testSuiteB` kill that mutant (i.e., that mutant fails at least one test in `testSuiteA` and also fails at least one test in `testSuiteB`) *without* any Python runtime errors (i.e., no mutants that reference undefined variables and no index out of bounds accesses, etc.).

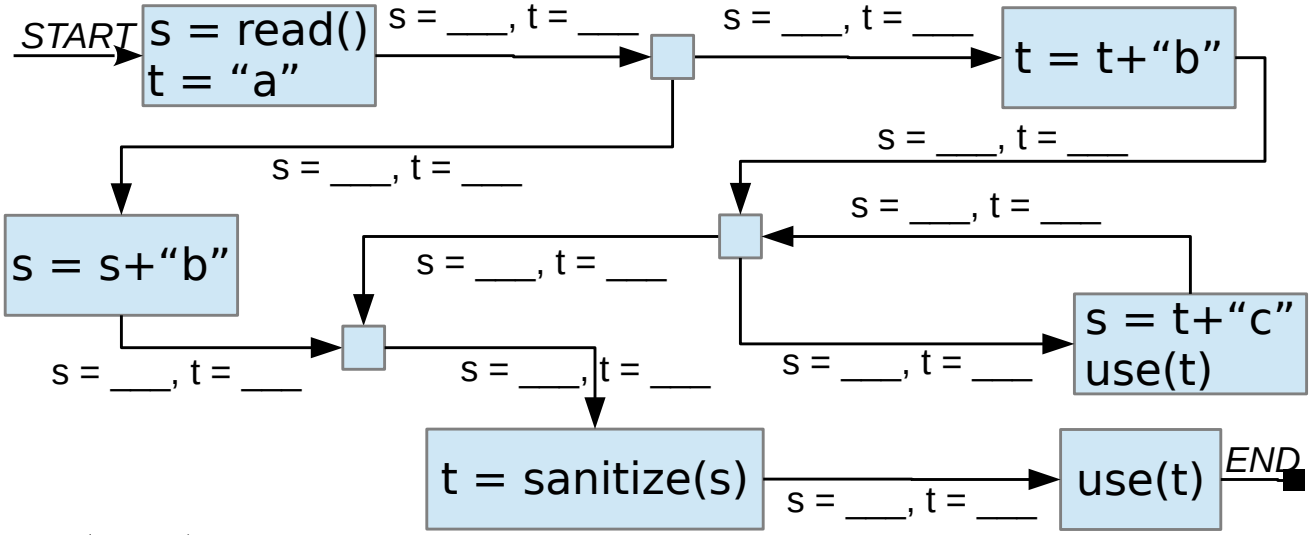
*(6 pts.)* You are still limited to single variable renaming mutations. Give an example of a single mutation such that *one of* `testSuiteA` or `testSuiteB` kills the mutant *but not both* (with no Python runtime errors) — and say which one kills the mutant.

*(4 pts.)* You may now consider any and all mutation operators. Give two advantages of higher-order mutation (i.e., multiple mutations per mutant) and two disadvantages of higher-order mutation. Use at most four sentences total.

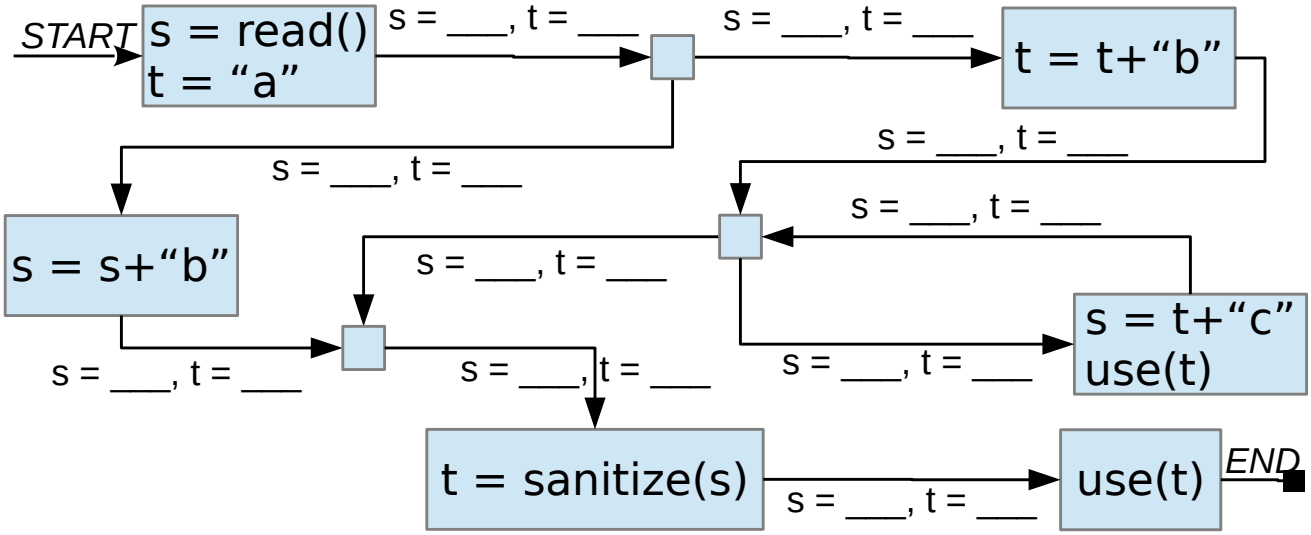
# 5 Dataflow Analysis (20 points)

Consider a *constant propagation* dataflow analysis for string values. We associate with each variable an analysis fact: either * ("the variable holds a value, but our analysis cannot be certain which value"), # ("this point has not yet been reached by our analysis") or a particular constant string ("at this point in the program, we are certain this variable holds exactly `hello`"). We also include simple string concatenation. For example, if `x = 'ada '` before the statement `y = x+'lovelace'`, we conclude that `y = 'ada lovelace'` after it.

*(10 pts.)* Complete this *forward* string constant propagation dataflow analysis below by filling in each blank for both string variables `s` and `t`.




*(10 pts.)* We also want to known if uncontrolled string values may be used un-sanitized (i.e., in an unsafe manner). It is a bug if *even part* of the string passed to the `use()` function comes from an un-sanitized, unsafe read. We associate with each variable either PU for "possibly used *unsafely* later" or NU for "not used *unsafely* later". For the same program, complete this *backward* dataflow analysis by filling in each blank for both string variables.



# 6 Quality Assurance Analyses (15 points)

*(5 pts.)* CHESS and Eraser are dynamic analysis approaches that find bugs in concurrent or multi-threaded programs. How are they affected by test quality? List one other advantage of each approach and one other disadvantage of each approach.

*(5 pts.)* List two scientific or algorithmic challenges (e.g., a theoretical or "whiteboard" concern) associated with deploying a static bug-finding tool like FindBugs at a large company like Google. List three "software engineering" (e.g., process, people, management, etc.) challenges associated with deploying such a tool at such a company.

*(5 pts.)* Support or refute the claim that an effective approach to finding security defects (e.g., leaked passwords, buffer overruns, etc.) would be to identify frequently-executed regions of code via sampling-based profiling and apply formal code inspection to them.

# 7 Extra Credit **(1 pt each; we are tough on reading questions)**

*(Feedback)* What is one thing you would change about this class for next year? What is one thing you like about this class?

*(Free/Participation)* Make up one "believable" (but possibly outlandish) claim about the class, professor or course staff. If we get enough interesting responses, we'll post them.

*(Your Choice Reading 1)* Identify one of the optional readings. Write a sentence about it that convinces us that you read it critically. (Our subjective judgment applies here — sorry!).

*(Your Choice Reading 2)* Identify a different optional reading. Write a sentence about it that convinces us that you read it critically. (Our subjective judgment applies here — sorry!).

*(My Choice Reading 1)* In "Automated Whitebox Fuzz Testing", for what sort of "well-formed input" did they need to use the whitebox information to generate inputs?

*(My Choice Reading 2)* In "A Decade of Software Model Checking with SLAM", give one key technique used by that static analysis to find API violations in software such as device drivers.