EECS 481 — Software Engineering
Fall 2026 — Exam #1

- There are 10 pages in this exam (including this one) and 8 questions, most with multiple parts. Some questions span multiple pages. If you get stuck on one, come back 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.
- Place all phones, watches, computers (laptops, tablets) and similar devices **face down on the floor** in front of you. You may not refer to any device or AI agent.
- Use a **dark pen or pencil** to write your answers in the space provided on the exam. If our scan does not pick up your pen or pencil you will not receive credit. You may use exam margins for scratch work. 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 more complicated than necessary.
- For each non-extra-credit full subquestion of the exam (e.g., all of subquestion 2-b) that you leave blank or draw an X through, **you will receive one-third of the points** (e.g., $4/3 = 1.33$). This saves you time by removing the incentive to spend time writing a long guess. If you don't know something, move on to the next subquestion.
- **Write your UM uniqname, copy the honor pledge, and sign your name.** "I have neither given nor received unauthorized aid on this examination, nor have I concealed any violations of the Honor Code."

Signature: _____

UM uniqname: _____

1 Word Bank Concept Matching (14 points)

(1-a, 14 pts) Fill in each ____ blank with the single *most specific or appropriate* corresponding concept from the answer bank. Multiple answers may seem appropriate, but you must select the most specific one. Each option from the answer bank will be used *at most once*.

A. Agile	B. Call-Graph Profile	C. Coupling Effect	D. Comparator
E. Dataflow Analysis	F. Dynamic Analysis	G. Flat Profile	H. Formal Code Inspection
I. Fuzz Testing	J. Integration Testing	K. Mocking	L. Oracle
M. Pair Programming	N. Passaround Code Review	O. Perverse Incentive	P. Priority
Q. Quality Property	R. Sampling Bias	S. Severity	T. Software Metric
U. Streetlight Effect	V. Threat to Validity	W. Triage	X. Watchpoint

____ This development process typically produces more concise and less buggy code but increases initial development time.

____ This dynamic approach may be useful for expensive or unavailable functionality.

____ Changes to the source code are analyzed statically.

____ A library and its client code work separately but we find they do not work together.

____ This development methodology favors team autonomy and team diversity.

____ In a dynamic analysis using this, 4.01 and 3.99 might be acceptable answers with respect to 4.00.

____ To resolve performance problems, engineers desire a nuanced report that explains the time taken by each procedure and its children.

____ Priority and severity are critical parts of this process.

____ Developers believe a bug to be caused by a race condition, so they run the program and track the set of locks held at each access to a shared variable.

____ Simpler issues can combine together to form more complicated ones.

____ Management awards a monetary bonus for each test written, hoping to improve quality. Developers check in hundreds of similar tests made by tools such as Randoop.

____ Because AI agents are efficient at analyzing text files but slower at analyzing audio files, the team focuses on analyzing source code rather than meeting recordings.

____ This generates test data, often revealing corner cases and security issues.

____ To assess the total market for a potential feature for office software, management surveys executives but not students or staff.

2 Testing and Coverage (14 pts)

Consider a small library loan system. Members borrow books. Each loan has a due date. Late returns cost a fee. The late-fee rule is specified in S1–S4 below. `lateFee(daysLate, isStudent)` returns the fee in cents. `daysLate` is zero for an on-time or early return.

- S1. A book returned on or before its due date incurs no fee.
- S2. Otherwise the fee is 25 cents per day late.
- S3. After that total fee is calculated, students pay half that amount (rounded down).
- S4. Finally, a fee never exceeds 1000 cents.

Interpreting the S1–S4 prose is part of the question. You asked a tool (AI agent, Randoop, etc.) to write tests for `lateFee`. It produced these six. All six pass.

- T1 (noFeeOnTime). `assert lateFee(0, false) == 0`
- T2 (chargesPerDay). `assert lateFee(4, false) == 100`
- T3 (studentPaysHalf). `assert lateFee(4, true) == 50`
- T4 (capsLargeFee). `assert lateFee(100, false) == 1000`
- T5 (capsStudentFee). `assert lateFee(100, true) == 1000`
- T6 (stableForLarge). `assert lateFee(365, false) == lateFee(365, false)`

(2-a, 3 pts) Identify one spec, by number, that is not (fully) covered. Write a test for it.

- Identified Specification is S_____
- New Test: `lateFee(_____, _____)` should return _____

(2-b, 4 pts) The tool-provided suite never probes when the cap (S4) takes effect. For a non-student, write one test just (strictly) below that value, one exactly at it, and one just (strictly) above it.

- Just below: `lateFee(_____, false)` should return _____
- Exactly at: `lateFee(_____, false)` should return _____
- Just above: `lateFee(_____, false)` should return _____

(2-c, 3 pts) One of the six tool-provided tests does not implicate a violated specification if it fails (sometimes called "traceability"). Which one, and why? One sentence.

- T_____
- Why:

(2-d, 4 pts) Complete the small method `foo(x)` below such that the test $\{x = 3\}$ has 50% branch coverage but 33.3% (one third) path coverage. For this question, dead code (including unreachable code or infeasible paths) does not count for coverage denominators. You need not use all available blank lines.

```
1  def foo(x):
2    if x <= 3:
3      print("stmt 1")
4    else:
5      print("stmt 2")
6                                                    # <- your code here
7                                                    # <- your code here
8                                                    # <- your code here
9                                                    # <- your code here
10                                                   # <- your code here
11                                                   # <- your code here
```

3 Short Answer (19 points)

(Review our "leave a subquestion blank, get one-third of the points" policy on Page 1.)
(3-a, 6 pts) Your company wants to build high-quality software quickly and inexpensively. The company currently does not use static analysis and is considering re-allocating 10% of its testing effort/budget to static analysis. Identify two *risks* with this proposal. For each risk, identify one associated *uncertainty* and one associated *measurement* that might be taken to reduce that uncertainty.

(3-b, 4 pts) Support or refute the claim that the number of duplicate defect reports for the same issue should influence its assessed severity. Separately, describe a situation in which a defect report might be resolved without an associated code patch.

(3-c, 3 pts) Identify a developer expectation of modern passaround code review that is commonly met. Identify a developer expectation of code review that is rarely met. Describe a buggy patch that code review is unlikely to correctly reject, and briefly explain why.

(3-d, 3 pts) Identify a bug described in any of the readings (required or optional; you should identify the reading but need not give its exact title) for which *watchpoints* would likely be more useful than traditional *breakpoints* and explain why.

(3-e, 3 pts) Give a situation from the class materials (required or optional) in which it would be "positive" or "correct" for the severity and priority of a bug report to differ (i.e., not just "management is greedy"). Explain why this difference is correct or desired here.

4 Mutation Testing (16 points)

Consider this program to compute the number of natural number squares (e.g., 1^2, 2^2, 3^2, etc.) strictly less than a given number. The original program is shown on the left; three first-order mutants are each indicated by a comment on the right.

```
1 def squaresBelow(x):
2   count = 0
3   for i in range(1,x):   # mutant 1: x becomes x-1
4     square = i * i       # mutant 2: * becomes +
5     if square < x:       # mutant 3: < becomes <=
6       count = count + 1
7   return count
```

(4-a, 12 pts) Complete the table below by indicating whether or not each test kills each Mutant. Write "K" for Killed and "N" for not killed.

	Input	Oracle	Mutant 1	Mutant 2	Mutant 3
Test 0	2	1			
Test 1	4	1			
Test 2	9	2			
Test 3	10	3			

(4-b, 1 pt) What is the mutation score for Tests 0–3 using only Mutants 1–2?

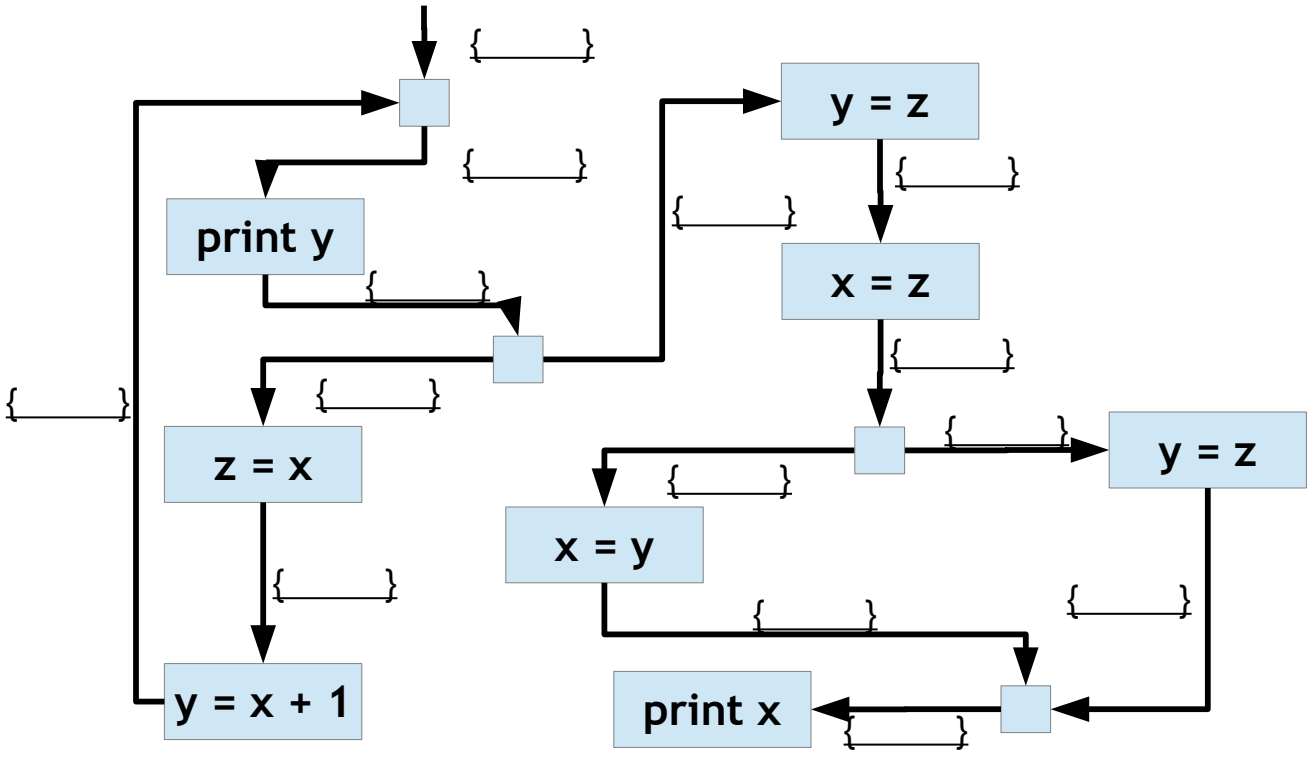
(4-c, 1 pt) What is the mutation score for only Tests 0–2 using only Mutants 2–3?

(4-d, 2 pts) Support or refute the claim that a *higher-order* mutant that combines Mutation 1 and Mutation 3 is useful. Explain.

5 Dataflow Analysis (20 pts)

Consider a *live variable* dataflow analysis for *three* variables, `x`, `y` and `z`. We associate with each variable a separate analysis fact: either the variable is possibly read on a later path before it is overwritten (live) or it is not (dead). We track the *set* of live variables at each point: for example, if `x` and `y` are alive but `z` is not, we write { `x, y` }. The special statement `print` reads, but does not write, its argument. (You must determine if this is a forward or backward analysis.)

(5-a, 14 pts) Complete this live variable dataflow analysis for `x`, `y` and `z` by filling in each blank *set* of live variables.



(5-b, 6 pts) You ask an AI agent for a one-paragraph definition of dataflow analysis and it provides the following:

> Dataflow analysis is a static program-analysis technique that computes information about how facts or properties of program variables propagate. It typically associates abstract information with program points and uses transfer functions to model how each statement changes that information. Dataflow analysis iteratively executes each path through the program to determine the information at each program point. Common examples include reaching definitions, live-variable analysis, available expressions, and constant propagation.

Support or refute the claim that this is a good definition. If you choose support, identify a common misunderstanding about dataflow analysis that this definition helps the reader avoid. If you choose refute, identify an error in the definition and correct it.

6 Quality Assurance Analyses (17 points)

(6-a, 3 pts) Support or refute the claim that the Netflix "Simian Army" (also called "Chaos Monkey") quality assurance technique is an instance of dynamic mocking.

(6-b, 4 pts) Your company acquires a new piece of software but is not given any tests for it. You are instructed to find race conditions in it. An AI agent suggests using a dynamic analysis like CHESS or Eraser, but since those require workloads, the agent suggests using AFL to generate test inputs first. Support or refute the claim that this plan is likely to identify race conditions.

(6-c, 6 pts) Choose modern passaround code review or formal code inspection. Argue that the use of agentic AI in Homework 2 and/or Homework 3 is closer to your chosen static analysis than it is to the other. As part of your answer, identify three relevant static analysis properties and, for each, argue that agentic AI reflects (or does not reflect) it.

(6-b, 4 pts) One challenge of fault localization is the large scale of modern software. An AI agent suggests that this scale is not a challenge for automated *profiling* because most of the execution time will be spent in an small number of functions or loops and human analysis can focus on those implicated regions rather than the entire code base. Support or refute this claim using two concepts or pieces of evidence from the class or readings.

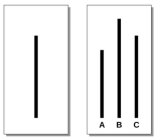
7 Honor Code

Did you observe any potential violations of the honor code during this exam (e.g., "it looked like the person on my left was using a phone")?

8 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 or a course staff member (Danni, Derek, Victoria, Wes)?

(Psych) Name this experiment. Why is it relevant to software engineering?



(Optional Reading) In Terrell et al.'s *Gender differences and bias in open source: pull request acceptance of women versus men*, how did being an "outsider" to a project impact results?

(Your Choice Reading 1) Identify any different **optional** reading. Write 1–2 sentences about it to convince us that you read it *critically*. (Our subjective judgment applies here!).

(Your Choice Reading 2) Identify any different **optional** reading or Piazza **long-instructor-post**. Write 1–2 sentences about it that convinces us that you read it critically. (Our subjective judgment applies here!).