EECS 481 — Software Engineering
Fall 2026 — Exam #1 ANSWER KEY v1

- 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: ANSWER KEY

UM uniqname: ANSWER KEY

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

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

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

N Changes to the source code are analyzed statically. Note "changes".

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

A This development methodology favors team autonomy and team diversity. Slide #10.

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

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

W Priority and severity are critical parts of this process.

F 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.

C Simpler issues can combine together to form more complicated ones. Slide #62.

O 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.

U 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.

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

R 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_3_ . It may be tempting to answer S1, but the last sentence in the text indicates that zero is used for on-time *or early* returns, so T1 covers S1.
- New Test: `lateFee(_1_,_true_)` should return _12_ . The new test must show the "rounded down" aspect.

Rubric example. If you identified the wrong specification, but gave a reasonable testcase for the one you identified: 1 point.

(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(_39_, false)` should return _975_
- Exactly at: `lateFee(_40_, false)` should return _1000_. Because the question says "exactly at it", this must be (40,1000) to reflect S3.
- Just above: `lateFee(_41_, false)` should return _1000_

(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_6_
- Why: If T6 fails, it means that lateFee is non-deterministic. However, no specification S1–S4 mentions that, so no specification is directly implicated. By contrast, T1 implicates S1, T2 implicates S2, T3 (with T2) implicates S3, T4 implicates S3 and S4 (depending on whether it fails by being too high or too low), and T5 implicates S3 and S4 as well.

Rubric example. If you identified the wrong tool-provided test but gave a good try-explanation (in our assessment): take off point for wrong answer, give 1 point for explanation.

Rubric example. If the tool provided test is correct, but the explanation is bad or does not talk about how the test doesn't implicate a violated specification, take away 0.5 point.

(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
7    if x == 3:              # multiple answers are possible
8      print("stmt 3")
9    else:                   # not needed, but adds clarity
10     print("stmt 4")       # not needed, but adds clarity
```

The key here is that only feasible paths count for the denominator. Students explored this during the in-class participation activity for slide #34 of https://eecs481.org/lectures/se-06-inputsoracles.pdf . In general, every "if" statement would add another two paths. To get three paths, we'll need to use that feasibility. In the example answer above, it may look like there are four paths, but only three are executable at run-time: (1,3), (1,4), and (2,4) are all possible. However, (2,3) is not, because it would require x > 3 to reach 2 and then also x == 3 to reach 3, but there is no number that is both greater than 3 and also equal to 3.

One possible source of confusion is thinking that the infeasible path means that the branch coverage calculation is also out of "3". However, note that stmt 3 and stmt 4 are both reachable, as are stmt 1 and stmt 2. So branch one *can* go both true and false, and branch two *can* also go both true and false, so branch coverage is out of 4. Indeed, this very distinction highlights the "simplicity" of branch coverage compared to path coverage.

Rubric example. If you give a real answer, but do not correctly identify the correct if branch: give half credit. Rubric example. If your answer is only a print statement or equivalent: no credit.

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.

This is a direct copy of Question 3-b from Winter 2020 — the first practice exam listed on the webpage. Among other things, reusing questions rewards students who studied from the recommended materials.

In general, risks and uncertainties often feel very similar. Examples:

- SA may be less effective at finding bugs than testing — will SA be effective on our software? — average number of bugs found per line of code
- SA may find unimportant bugs compared to testing — do SA-found bugs matter to management? — average assigned priority of SA-reported defects
- SA may not fit with our continuous integration framework — will SA provide useful information rapidly? — average seconds for SA to analyze our software after a code check-in
- SA may not always be available — what if SA requires one particular expert developer to get it to run and that person gets sick? — fraction of developers who can run SA on hello-world after 30 minutes of training
- SA's reports may be ignored — will developers trust the SA? — fraction of SA reports that are false positives (since developers hate those in bug-finding tools)

Rubric example: 1 point for each correct risk, uncertainty, and measurement (sums to 6 points).

(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.

This is a direct copy of Question 3-b from Fall 2019.

Likely "refute". Severity relates to the cost of not fixing the bug: the risk or damage in the real world. Risk is the frequency of incidents times the cost per incident. If there are many duplicate bug reports, it means many people are seeing the same bug, so the frequency is high. However, the cost could still be very low (e.g., everyone could be reporting the same unimportant spelling error).

However, "support" is possible. You could argue that user-facing bugs are the ones that matter most. The challenge here would be to add one more link: for example, to suggest

that if the bug is important enough that many people are taking the time to report it, it must be worth time or money to them.

https://eecs481.org/lectures/se-10-bugreport.pdf : Answers referencing potential severities or priorities, like slides 35–36, are more likely to receive full credit.

For the second part, many defect resolutions (e.g., WONTFIX, WORKSFORME) officially *resolve* the issue report without a code patch. See slide 48 of https://eecs481.org/lectures/se-10-bugreport.pdf .

Rubric example for Refute. 1 point for a valid stance and a surface-level explanation. 2 points for an in-depth explanation about the meaning of severity in the context of duplicate defectg reports.

Rubric example for Support. 3 points for an extensive explanation (e.g., if reporters take the time to report than it is likely an important defect, see sample answer above).

Rubric example. 1 point for describing a valid 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.

This is a direct copy of Question 3-d from Fall 2019.

Following *Expectations, Outcomes, and Challenges Of Modern Code Review*, we see that developers expect "finding defects", "code improvement" and "alternative solutions". However, the big outcomes are "code improvement", "understanding", "social communication", and "defects".

So "finding defects" and "code improvement" are expectations of passaround code review that are met. And "alternate solutions" is an example of an expectation that is rarely met. There are many types of buggy patches that human inspection may fail to warn about.

Classic examples involve "invisible" control flow or variable changes, such as those caused by threads or exceptions. So a bug involving a race condition may be hard to spot manually. A bug involving failing to release a resource on an exceptional situation may be hard to spot manually. Finally, bugs like the "goto fail" example that "look like reasonable code" may be hard for humans to catch (but easy for automated analyses to catch).

Rubric example. 1 point for a commonly-met expectation, 1 point for a rarely-met expectdation, 1 point for a buggy patch.

(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.

Watchpoints are useful when you care about the value at a particular memory location changing but when you don't know exactly which instruction (or line of code) causes that change. By contrast, breakpoints require you to identify an instruction (or line of code) at which execution should stop so that values can be inspected.

Multiple answers are possible here, so long as the student clarifies why watchpoints are more relevant than breakpoints.

Example: https://eecs481.org/readings/debugging-helping.pdf "Are Automated Debugging Techniques Actually Helping Programmers?" — The first task used in this paper (debugging a Tetris game) is a reasonable example. The paper describes it as "The rotation of a square block causes unusual behavior: The square block will rise upwards instead of rotating in place (which would have no observable effect). If needed, you can easily reproduce the failure by running Tetris and trying to rotate a square block." In this setting, the programmer specifically does not know *where* incorrect height assignment is taking place (no line of code known in advance for a breakpoint) but could put a watchpoint on the memory location storing the y-coordinate of the piece. In addition, the task description indicates that it is easy to reproduce with a trivial input, which is relevant for this dynamic analysis.

We note that "race condition" (in the style of the Eraser tool or paper from the dynamic analysis lectures) is likely not a full-credit answer without significant explanation. In an example like Figure 3 of https://eecs481.org/readings/eraser.pdf , the two lines that change the shared variable are known, so a breakpoint might work just as well. It is true that many race conditions involve "another thread scribbling over your important variable", so a watchpoint would help, but you need to also make the argument that breakpoints would not be as useful, and there are typically only a small number of critical sections, so students would have to explain why breakpoints do not work well in their particular examples for full credit.

Rubric example. 0.5 point for teh class, 2.5 point for the explanation, 0 points if the situation is not a dynamic analysis, 1.5 points for the race condition style of answer (see above), 0 points for the wrong concept.

(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.

https://eecs481.org/lectures/se-10-bugreport.pdf , near slide 35, details severity and priority. Informally, severity is the (actual) impact of the bug or cost of not fixing the bug, while priority is the (management-assigned) urgency in fixing it.

Slide 38 directly lays out a potential answer: "Severity and Priority are often correlated, but are officially independent. A "Normal" Severity issue could currently be "Low" Priority if there are many outstanding "Critical" Severity issues, etc." A student could fully complete the answer by bringing in the reasoning from Slide 37: "Limit High priority task assignments for a single person to three, five in exceptional times." Suppose there are six high severity issues. Management might legitimately pick the five most severe and give those high priority, and then give the sixth medium priority, to avoid overwhelming developers.

Other well-supported answers are possible. For example, consider legal obligations. https://eecs481.org/readings/coverage-satellite.pdf describes how "To be certified by the FAA, aviation software must satisfy a standard labelled DO-178B". Consider two bugs, one of which is of Major Severity (loss of functionality in an important area, like astronomy data collection) and one of which is technically Minor (easy to address in terms of computer science) *but* would lead to some law (e.g., from the FAA) being violated if it is not fixed. Management might legitimately prioritize legal compliance over shipping full functionality (e.g., better to ship 2 working modules that comply with the law than to produce 3 modules but not be able to deploy anything because of legal challenges).

Rubric example. 0.5 points for referencing class materials, 2.5 points for the explanation, 0 points for incorrectly selecting the Equifax example, 1.5 points for unclear or confusing descriptions.

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
```

In Python, range(1,3) is [1,2], but students were instructed during the exam that they could consider it to be [1,3] if they desired. This was because we did not want the exam to depend on specific knowledge of Python. Here we call the former interpretation "Standard" and the latter "Simplified".

Students could use *either* interpretation for full credit for Question 4 (but can't switch back and forth across sub questions).

(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.

Standard Python answer follows. For clarity, for Killed, we list the incorrect test answer produced by the mutant in parentheses.

	Input	Oracle	Mutant 1	Mutant 2	Mutant 3
Test 0	2	1	K (0)	K (0)	N
Test 1	4	1	N	N	K (2)
Test 2	9	2	N	K (4)	K (3)
Test 3	10	3	N	K (4)	N

Simplified "range(1,3) is [1,2,3]" answer follows. For clarity, for Killed, we list the incorrect test answer produced by the mutant in parentheses. Only one cell differs from the standard answer.

	Input	Oracle	Mutant 1	Mutant 2	Mutant 3
Test 0	2	1	N	K (0)	N
Test 1	4	1	N	N	K (2)
Test 2	9	2	N	K (4)	K (3)
Test 3	10	3	N	K (4)	N

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

The mutation score is the number of mutants killed divided by the total number of mutants. A mutant is killed if it fails at least one test.

Standard answer: 2/2

Simplified Answer: 1/2

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

Standard answer and Simplified answer are both 2/2.

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

In the standard interpretation, that higher-order mutant results in K (0), K (2), K (3), N. Simplified, that higher-order mutant produces N, K (2), K (3), N. That is, in both

interpretations, the higher-order mutant is killed by the union of the tests killed by its first-order components.

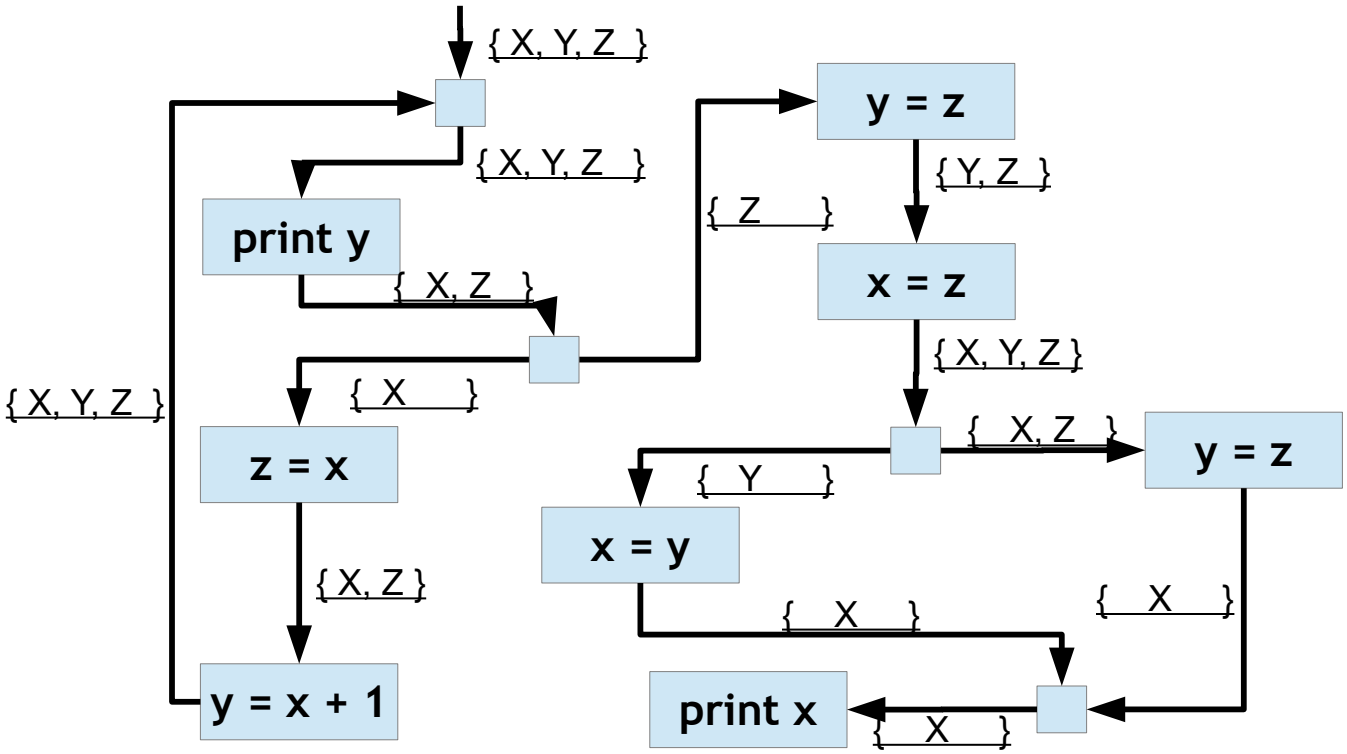
In either case, the best answer is "refute": the higher-order mutant does not provide new information. The purpose of mutation testing is to assess the adequacy of a test suite. From mutants 1 and 3 separately, we already know that the test suite is fairly good: tests 0, 1 and 2 probe enough functionality to detect those mutations. Another mutant that kills the same tests does not help. Students experienced this directly in HW3, where only a certain number of mutants could be evaluated. The new higher-order mutant 1+3 does not kill (much less "selectively kill", as in HW3) anything new.

Students could also arrive at this using the theory of mutation analysis. For example, the Jia and Harman reading for HW3 lays out one situation in which a higher-order mutant (HOMs) would be useful https://eecs481.org/readings/mutation-testing.pdf : "A subsuming HOM is harder to kill than the FOMs from which it is constructed. As a result, it may be preferable to replace FOMs with the single HOM to reduce the number of the mutants." However, in this situation the 1+3 HOM is not any harder to kill than its component first-order mutations. Students could also reason from the *coupling effect* (slide 62 of https://eecs481.org/lectures/se-05-testquality.pdf) to the same conclusion.

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.



Rubric example. -1 point for blank, -1 point for no relevant variables, -1 to -3 points for each incorrectr variable.

(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.

1

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.

"Refute". The definition explicitly contains an incorrect sentence: dataflow analysis does not *execute* each path. Indeed, it does not execute anything at all. Dataflow analysis is a static analysis. Dataflow anlaysis tracks information, analyzing an abstraction of the state space of a program. It does not execute or run the program. This is the critical distinction between static and dynamic analyses.

Rubric example. 3 points for correctly refuting, 3 points for a well-argued explanation, 2 points for an explanation that is relevant but not fully accurate, 1 point for a lacking explanation.

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.

"Support". https://eecs481.org/lectures/se-04-qatesting.pdf slide 48 introduces dynamic mocking, noting that "dynamic mocking libraries that allow you to substitute objects and functions at runtime". Slide 57 of https://eecs481.org/lectures/se-08-dynamic.pdf introduces the Chaos Monkey analysis. It "induces artificial delays" and "removes instances". This exactly aligns with the dynamic mocking example of changing a function call to always fail (i.e., removing that functionality).

Rubric example. If your answer correctly supports, but does not give a sufficient explanation, -1.5 points.

(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.

Likely "Accept" (although the quality might not be high). In https://eecs481.org/readings/eraser.pdf the Eraser paper describes doing exactly this in Section 4.1: "We tested Ni2 separately using a utility called ft that submits a series of random requests" and again in Section 4.2: "We tested the cache server using the TestCache utility that issues a stream of concurrent random requests." So random testing (fuzz testing) explicitly works for such tools.

Students could also reason to this from the slides (slide 54), noting that Eraser just needs to execute a line accessing a shared variable once or twice (with different sets of locks or an empty lockset) to identify a likely race condition. Coupled with our experience in HW2 and its writeup, where students found that AFL tests often executed the same core of lines while also exercising some corner cases, AFL would be sufficient to meet Eraser's input requirements.

(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.

Likely "modern passaround code review". For example, students could argue that the use of Agentic AI in HW2 is similar to modern passaround code review because the specification document constructed with the AI is repeatedly passed back and forth for explicit review by different agents. There is even an explicit "Goldfish — Critic Review" step calling out code

review. Similarly, in HW3, students are explicitly given "Candidate Pull Requests" and are explicitly asked to analyze them, think about comments, and select a subset to apply. This is almost exactly the definition of the modern code review process.

Students could aruge for "formal code inspection" for HW2 or HW3 because the Agentic AI reads the entire available code "all at once". For example, in HW2, the Agentic AI may read all of the source code for libpng in the current folder. In that light, the AI's actions may be seen as closer to formal code inspection (which does not work on deltas). However, this would be a harder argument to make, since (a) even formal code inspection typically only reads a few hundred lines at a time, and (b) formal code inspection involves multiple independent people with different roles. Students could perhaps argue that the libpng authors, the student and the AI are three different people with three different roles, but this would have to be elaborated clearly.

Students could pick many analysis properties: conservative, sound, complete, termination, severity, priority, etc. For example, the agentic AI use in HW3 is explicitly neither sound nor complete, and the task asks you to assign priority (rank) to them.

Rubric example. If your answer correctly indicates "modern passaround code review" but does not provide a sufficient explanation or does not include three static analysis properties correctly, -3 points.

(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.

Likely "refute". Profiling can still face the scale problem because a flat profile emphasizes where execution time is spent, not necessarily where a fault is located; the class slides covering profiling illustrate this distinction between performance hot spots and the code responsible for a particular problem.

Multiple pieces of evidence are possible. Examples:

https://eecs481.org/lectures/se-11-faultloc.pdf slide 48: a large piece of software could still suffer from a high error rate if it exhibits periodic behavior.

https://eecs481.org/readings/gprof.pdf Section 3: the authors claim "Our solution is to gather profiling data in memory during program execution and to condense it to a file as the profiled program exits", but the larger the program is (think "all of Google"), the less feasible it is to gather all of the profile data in memory first. So the large scale of modern software may be a challenge for gathering profile data.

Same paper, Section 3.2: "We make the simplifying assumption that all calls to a specific routine require the same amount of time to execute. This assumption may disguise that some calls (or worse, some call sites) always invoke a routine such that its execution is faster (or slower) than the average time for that routine." As the authors note, this is a simplifying assumption. Larger-scale modern software is unlikely to be simple. If some invocations of the same method are much faster or slower than others (varying network latency? cache

misses?), that would be a challenge for profiling modern large-scale software.