You don't have a submission that is not late.

minutes remaining

Hide Time

Manual Save

Navigation

General Reminders

- You **may** use *free* generative AI tools on this exam.
- The exam is open note, open lecture recordings, open course webpage, open computer (including Visual Studio, Python, etc.), and open Internet in general. You may use general webpages (e.g., Stack Overflow, Wikipedia, etc.).
- You may **not** use live help from another human.
- If you leave a sub-question entirely **blank** then you will receive one third of its points rounded down. For example, if a sub-question is marked "(4 points)", you would receive 1 point if you left it entirely blank.
- If you are asked to support or refute a position, indicate which option you selected. One word can suffice.
- If you are asked to provide a quote from multiple possible sources (e.g., either the lecture slides or a reading), indicate which source you selected.
- If you are asked to provide a quote to help support or justify your arugment, your quote must be directly relevant to your argument.

Question 1. Word Bank Matching (1 point each, 18 points)

For each statement below, input the letter of the term that is *best* (e.g., most specifically) described. Note that you can click each word (cell) to mark it off. Each word is used at most once.

A. — A/B Testing	B. — Agile development	C. — Alpha Testing	D. — Beta Testing
E. — Competent Programmer Hypothesis	F. — Constructive Cost Model	G. — Dynamic Analysis	H. — Formal Code Inspection
I. — Fuzz Testing	J. — Integration Testing	K. — McNamara Fallacy	L. — Milestone
M. — Mocking	N. — Oracle	O. — Pair Programming	P. — Pass Around Code Review
Q. — Priority	R. — Race Condition	S. — Regression testing	T. — Risk
U. — Spiral Development	V. — Streetlight Effect	W. — Triage	X. — Uncertainty
Y. — Unit Testing	Z. — Waterfall Model		

Q1.1:

Q

Yushu reports an issue that must be addressed before any other work is done.

Q1.2:

Y

MunchyRoll corporation has a policy that for every created function, there must be corresponding inputs, outputs and oracles to assess that function.

Q1.3:

F

Wes is a project manager for Veecsa. Wes wants to make use of historical company data to predict how long the next development activity will take.

Q1.4:

U

Developers at Bank of Michigan deliver an increasingly-complete series of prototypes, while accounting for risk.

Q1.5:

minutes remaining
Hide Time

Manual Save

Navigation

- Question 1
- Question 2
- Question 3
- Question 4
- Question 5
- Extra Credit
- Pledge & Submit

O

Leena decides to employ a software development approach usually takes longer but results in fewer defects.

Q1.6:

T

Software development processes can mitigate, but not remove, *this*.

Q1.7:

R

In Peter's banking application, if a withdrawal and a deposit are made at almost the same time, neither transaction completes.

Q1.8:

J

Developers at Miscord find that modules that work well separately do not always work well when put together.

Q1.9:

X

Gwen is a project manager for Boogle. Gwen wants to make use of historical company data, but does not know if those past numbers exactly match current realities.

Q1.10:

I

This dynamic analysis creates random test inputs and can be useful even when oracles are not available.

Q1.11:

E

Developers at GlazeBook complete an internal study and find that most defects are caused by simple typos.

Q1.12:

P

Devforce corporation requires that at least one developer familiar with the language and at least one developer familiar with the module approve all changes to that module's code.

Q1.13:

K

Managers at 481andMe are focusing exclusively how many users they have for a newly-deployed game. They believe that having more users means they are doing better than their competitors and that their game will win quality awards and turn a profit.

Q1.14:

W

This software development process allocates resources to issues based on their impacts and their urgencies.

Q1.15:

N

Aamir is tasked with determining what the correct chart produced by a jFreeChart test *should* look like.

Q1.16:

H

minutes remaining
Hide Time
Manual Save
Navigation
- Question 1
- Question 2
- Question 3
- Question 4
- Question 5
- Extra Credit
- Pledge & Submit

This software development process mitigates certain biases by having different people write the code, present the code, and analyze the code.

Q1.17:

M

Yuxuan is developing a database backend. To allow others to test that interface before it is finished, a simple implementation that stores data in text files is created.

Q1.18:

V

Momo is familiar with the OS abstraction layer but has recently struggled with a low-priority issue where text is displaying in red instead of green. It was easy to examine the system calls responsible for rendering text, but Momo was still unable to fix the bug.

Question 2. Testing (25 points)

Answer the following questions about testing.

(a) (5 points)

In Tillmann and de Halleux's *Pex — White Box Test Generation for .NET*, why do the authors introduce the notion of "reachable statements"? Support your answer with a quote from the paper. (After copying the quote, use at most four sentences.)

Your answer here.

ANSWER: The notion of reachable statements is introduced because their target language (.NET) supports dynamically loading code at run time. As a result, it can be trick to know the total number of statements or branches in advance. In Section 2.3, they note "In a system with dynamic class loading such as .NET, it is not always possible to determine the statements of the programs ahead of time. In the worst case, the only way to determine all reachable statements is an incremental analysis of all possible behaviors of the program." An answer that only talks about "dead code" or "unreachable code" is likely not correct here based on the document (neither "dead" nor "unreachable" appear in the paper).
Grading Rubric (Total: 5 points):

- +2 pts — Identifies the core reason: .NET allows dynamic class loading at runtime, making it impossible to know all statements or branches in advance.
- +1 pt — Explains why this motivates the notion of "reachable statements" (reachability must be determined incrementally as code loads).
- +1 pt — Includes a relevant, accurate quote from the paper.
- +1 pt — Demonstrates understanding in the student's own words (coherent paraphrase; not just copying).

Deductions:

- −1 pt — Mentions "dead code" or "unreachable code."
- −2 pts — No quote.
- 0 pts total — Quote is fabricated or from the wrong paper.

(b) (5 points) What are the two specific relationships between code review and testing described in Sections 2.0-2.5 of *Software Engineering at Google* by Henderson? Support your answer with a quote from the paper. (After copying the quote, use at most four sentences.)

Your answer here.

ANSWER: The reading highlights two relationships: (1) the code review tool requires new source files to have unit tests, and (2) reviewers will require that changes that add functionality also add tests. In Section 2.4, they note "All code used in production is expected to have unit tests, and the code review tool will highlight if source files are added without corresponding tests. Code reviewers usually require that any change which adds new functionality should also add new tests to cover the new functionality." Generic answers that only restate the definitions of testing and code review (such as "testing increases confidence that code behaves correctly, while review provides human insight into design, readability, and subtle defects that tests may miss") are not supported by the paper.

minutes remaining
Hide Time
Manual Save
Navigation
- Question 1
- Question 2
- Question 3
- Question 4
- Question 5
- Extra Credit
- Pledge & Submit

Grading Rubric (Total: 5 points):

- +2 pts — Identifies both relationships between code review and testing:
  - (1) The code review tool enforces that new source files include unit tests.
  - (2) Reviewers require tests for new functionality.
- +1 pt — Includes an accurate, relevant quote from the paper supporting one or both relationships.
- +2 pts — Explains the relationships clearly in the student's own words (shows understanding beyond quoting).

Deductions:

- −1 pt — Provides only vague or generic statements about testing or code review.
- −2 pts — Quote is missing.
- 0 pts total — Quote is fabricated or from the wrong paper.

(c) (5 points) Support or refute the position that the particular use of Randoop in the context of our Homework #2 assignment is *black box testing* (i.e., does not require peering into jfreechart's internal structure). Your answer must include a relevant quote from the HW2 webpage, Randoop documentation, or another class reading (name the reading when providing the quote). (After copying the quote, use at most four sentences.)

Your answer here.

ANSWER:
The most technically correct answer is "refute": Randoop is a white box testing approach that must look at the code's internals to generate tests. The Randoop manual describes it as "feedback-directed random test generation". Even though the human developer does not need access to the source code per se, the tool does, and thus the entire process does not qualify as black box. A company that did not have access to the source code of something (e.g., a third-party library) could not use Randoop.

A student could earn high partial credit by supporting the claim that the user of Randoop does not need to peer into jfreechart's internals and is thus doing black box testing. However, our specific use of Randoop in HW2 still required a list of classes as an input: a white box detail.

Grading Rubric (Total: 5 points):

- +2 pts — Correctly refutes the claim that Randoop is black box testing, explaining that it uses source code access and feedback-directed test generation (i.e., a white box approach).
- +1 pt — Includes a relevant, accurate quote from the HW2 webpage, Randoop documentation, or another approved class reading (and names the source).
- +2 pts — Provides a clear, coherent explanation in the student's own words showing understanding of why Randoop requires internal access (e.g., uses bytecode or class structure to guide test generation).

Partial Credit:

- +1–2 pts — Earned if the student argues for the "support" position but does so with logical reasoning (e.g., focusing on the user's perspective rather than the tool's internals).

Deductions:

- −1 pt — Provides only vague or generic statements about black vs. white box testing.
- −2 pts — Quote is missing or not clearly attributed.
- 0 pts total — Quote is fabricated or from the wrong source.

(d) (5 points) Consider the Oracle-Comparator Model of testing. Consider the use of AFL in Homework #2 and the use of Randoop in Homework #2. For each of those two activities, indicate whether it involves explicit oracles, implicit oracles, or no oracles. For each activity, provide a sentence elaborating or explaining your answer. Use at most four sentences total.

Your answer here.

ANSWER:
The use of AFL in Homework 2 does not involve oracles. Both valid and invalid PNGs may be created, but AFL only considers covering new parts of the code, not whether the code corresponds to a human notion of a good PNG or not.

The use of Randoop in Homework 2 uses explicit oracles in the form of logical invariants over program expressions. Lines such as `org.junit.Assert.assertNotNull(shape5);` are created to assess semantics, and the generated tests are divided into those expected to pass and those expected to fail.

minutes remaining
Hide Time

Manual Save

Navigation

- Question 1
- Question 2
- Question 3
- Question 4
- Question 5
- Extra Credit
- Pledge & Submit

An answer suggesting that Randoop only involves implicit oracles is not correct. We discussed implicit oracles (such as avoiding segmentation faults in Fuzz Testing) in class and explicitly contrasted that to the generation of oracles with assertions or invariants.

Grading Rubric (Total: 5 points):

- +2 pts — Correctly identifies that AFL involves no oracles (it only measures coverage, not correctness).
- +2 pts — Correctly identifies that Randoop uses explicit oracles (assertions or invariants that encode expected outcomes).
- +1 pt — Clearly explains the reasoning for each case in the student's own words (e.g., AFL focuses on coverage; Randoop generates explicit assertions).

Deductions:

- 0 pts total — Misidentifies both oracle types or provides irrelevant responses.

(e) (5 points) Describe a scenario in which a test suite that achieves 100% statement coverage might miss a bug in a program. Then describe what other approach (testing, coverage, analysis, etc.) could find that bug. Include a quote from the slides or readings (your choice, but indicate which one) to support your argument. (After copying the quote, use at most four sentences.)

Your answer here.

ANSWER:
Slide 19 of https://eecs481.org/lectures/se-05-testquality.pdf contains an directd example of this with a potential division by zero bug ("Note that you could test line X and still have a bug on line X:"). Executing `print(6/x)` on input `x=2` gets full statement coverage but does not uncover the potential division by zero bug. For this particular example, a Dataflow Analysis (like a Potentially-Null analysis) or Fuzz Testing (test input generators like AFL love to try zero as an input) would likely reveal the bug quickly.

By contrast, an answer such as "an `if` block like `if (user_is_admin && password_correct) { grantAccess(); }`, where tests only exercise the true branch when both conditions are true and never test combinations where one is true and the other is false" is not a good example here. Some Generative AI responses favor examples like these and push for condition coverage, which could be useful in general, but this particular sort of example is actually totally safe if one of the conditions is false (it doesn't grant access, so there's no bug).

Grading Rubric (Total: 5 points):

- +2 pts — Describes a valid scenario where 100% statement coverage misses a bug (e.g., division by zero, cross-site-scripting attack (data flow), untested input values).
- +1 pt — Identifies a reasonable alternative approach (e.g., dataflow analysis, fuzz testing, mutation testing) that could detect the bug.
- +1 pt — Includes an accurate, relevant quote from the slides or readings and identifies the source if applicable.
- +1 pt — Explains the reasoning clearly in the student's own words (shows understanding beyond copying or generic examples).

Deductions:

- −1 pt — Provides only a generic or logically incorrect example (e.g., "if" statement with safe branches).
- −2 pts — Quote is missing or not identified by source.
- 0 pts total — Quote is fabricated or from the wrong source.

Question 3. Mutation and Measurement (20 points)

Answer the following questions about mutation analysis and measurement.

(a) (5 points) In the `floorSqrt` example near the end of https://eecs481.org/lectures/se-06-inputsoracles.pdf , what is a first-order mutation that falsifies the invariant `x >= retval*retval` with respect to the four tests mentioned there? Indicate your mutation (e.g., "change xyz to abc"). Indicate one of those four test inputs that falsifies the invariant for your mutant. Indicate what value the original returns and what value your mutant returns.

Your answer here.

minutes remaining
Hide Time

Manual Save

Navigation

- Question 1
- Question 2
- Question 3
- Question 4
- Question 5
- Extra Credit
- Pledge & Submit

ANSWER: One example is changing `return i - 1;` to `return i + 1;`. On the tests 1, 9, 16, 30, the original program returns 1 3 4 5. The mutant returns 1 5 6 7. Specifically, the mutant floorSqrt called on test input 30 yields retval 7. But 30 >= 7*7 is false. Note that answers that try to mutate *the invariant* are not correct: mutations must be applied to the source code.
Grading Rubric (Total: 5 points):

- +2 pts — Provides a valid first-order mutation that changes the program's behavior (e.g., altering an arithmetic operator or return value).
- +1 pt — Correctly identifies one of the four test inputs (1, 9, 16, 30) that falsifies the invariant for the mutant.
- +1 pt — States both the original and mutant return values for that input accurately.
- +1 pt — Clearly explains or demonstrates that the invariant `x ≥ retval*retval` is false for the mutant on that input.

Deductions:

- −1 pt — Mutation does not alter behavior or cannot falsify the invariant.
- −1 pt — Test input or output values are incorrect or missing.
- −2 pts — Explanation of invariant falsification is unclear or incorrect.
- 0 pts total — No mutation provided.

(b) (5 points) Suppose that a group of researchers find very convincing evidence *against* the competent programmer hypothesis but also find very convincing evidence *for* the coupling effect hypothesis. What would this mean for mutation testing? Include a direct quote from *An Analysis and Survey of the Development of Mutation Testing* (part of the HW3 spec) to support your argument. (After copying the quote, use at most four sentences.)

Your answer here.

ANSWER:
If the Competent Programmer Hypothesis is false, first-order mutations are unlikely to be relevant, making mutation testing much more difficult and expensive to apply (since higher-order mutations would have to be used). A relevant passage from that reading is "Therefore, in Mutation Testing, only faults constructed from several simple syntactical changes are applied, which represent the faults that are made by "competent programmers"."

Unfortunately, the Coupling Effect Hypothesis does not help us in this hypothetical. An indicative quote is: "Complex mutants are coupled to simple mutants in such a way that a test data set that detects all simple mutants in a program will also detect a large percentage of the complex mutants". Logically, the CEH is saying that **if** you have test data that detects simple mutants, **then** something good happens. Unfortunately, without the CPH, you **don't have** tests that detect simple mutants in the first place (because in that world humans don't make simple typos, so simple first-order mutants aren't good enough to make programs go wrong).

Grading Rubric (Total: 5 points):

- +2 pts — Explains that if the Competent Programmer Hypothesis (CPH) is false, mutation testing becomes less effective because first-order mutants no longer represent realistic faults.
- +1 pt — Explains that the Coupling Effect Hypothesis (CEH) does not offset this problem (it assumes simple mutants are meaningful, which depends on CPH).
- +1 pt — Includes a relevant, accurate quote.
- +1 pt — Provides a coherent explanation in the student's own words connecting CPH, CEH, and mutation testing outcomes.

Deductions:

- −1 pt — Mentions confuses the roles of the hypotheses.
- −2 pts — Quote is missing or not clearly related to mutation testing theory.
- 0 pts total — Quote is fabricated or from the wrong source.

(c) (5 points) Consider Buse and Zimmermann's *Information Needs for Software Development Analytics*. First, according to that report, which group makes the most use of code coverage information: developers, managers, customers or students? (Use at most one sentence.) Second, consider the report's section on "Targeting Testing" and its discussion of re-targeting testing. Support or refute the claim that mutation testing would be a good fit for re-targeting based on code churn. Include a quote from the reading to justify your argument. (After copying the quote, use at most four sentences.)

Your answer here.

minutes remaining
Hide Time

Manual Save

Navigation

- Question 1
- Question 2
- Question 3
- Question 4
- Question 5
- Extra Credit
- Pledge & Submit

ANSWER:
In Figure 5 of that reading, **managers** (the bottom, darker bars) make more use of Test Coverage information.

The best answer is probably "refute". One relevant quote is that testing is favored because it is "straightforward to deploy". Mutation testing is not generally consider straightforward. More directly, it is not clear how mutation testing would be "re-targeted" to favor a specific line. If code churn indicates that line 100 has been changed, the path constraint approach discussed in class could be used to try to generate test inputs to coverage that line, so re-targeting for standard coverage would be direct. By contrast, it's not clear how one would tweak mutation operators or probabilities in a more automated manner given a desire to target line 100 (cf. manual effort to do so in HW3).

Grading Rubric (Total: 5 points):

- +1 pt — Correctly identifies that **managers** make the most use of code coverage information (as shown in Figure 5).
- +2 pts — Correctly supports or refutes the claim about mutation testing and re-targeting, with reasoning grounded in the reading (e.g., mutation testing is not straightforward to deploy or automate for specific lines).
- +1 pt — Includes a relevant, accurate quote.
- +1 pt — Provides a clear explanation in the student's own words showing understanding of re-targeting concepts and their relation to mutation testing.

Deductions:

- −1 pt — Gives vague or incorrect reasoning about testing or code churn.
- −1 pt — Misidentifies the group that uses code coverage most frequently.
- −2 pts — Quote is missing or unrelated to re-targeting or test deployment.
- 0 pts total — Quote is fabricated or from the wrong paper.

(d) (5 points)

Consider Anda et al.'s *Variability and Reproducibility in Software Engineering: A Study of Four Companies that Developed the Same System*, as described in the second half of https://eecs481.org/lectures/se-02-risk.pdf. First, among the 35 companies that responded with bids, what relationship was found between the price a company would charge and that company's emphasis on analysis and design (development process). (Use at most one sentence.) Second, support or refute the claim that the results from that study argue *against* the use of constructive cost modeling (cocomo). Include a quote from the slides or reading to justify your answer. (After copying the quote, use at most four sentences.)

(Excerpts of this paper were described in the lecture and the slides, but the full paper was also an optional reading. You can answer this question using only the information from lecture, but you are also welcome to use the full text.)

Your answer here.

ANSWER:
First, there was *no consistent relationship* between the price a company would charge and that company's emphasis on analysis and design. This was discussed in Lecture and can be concluded by skimming the first excerpted table. Section 6.1.1 also addresses this directly: "Correspondingly, there was no relationship between the firm price and the process measures."

Second, the best answer is likely to refute that claim! While the study does show a high variability between various factors (like company size and development process) and the final schedule time, there are two reasons why that lack of a relationship is not directly relevant to cocomo. The first is that the variation reported is **between different companies**, while cocomo is done within one company based on that company's historical data. Relevant quote from the slides: "based on project history ... This requires experience with similar projects." The study may suggest that Company A shouldn't use Company B's data for Company A's modeling, but that's not how standard cocomo works anyway. The second is that the factors considered in the study (size, inheritance, encapsulation, etc.) aren't really the same as the ones considered in the cocomo processes we studied (e.g., VM experience, complexity of product, memory constraints). As an alternative phrasing, consider that for all of the companies in the study, since they were all asked to be write exactly the same program, many of the cocomo features (e.g., complexity of the product, performance constraints, memory constraints, etc.) would be the same across all companies. Ultimately, the study does show high variation *between* companies on the *same* project, but that does not mean that cocomo should not be used *within* a single company to predict different projects.

Grading Rubric (Total: 5 points):

- +2 pts — Correctly states that there was no consistent relationship between price and emphasis on analysis and design among the companies.
- +2 pts — Correctly refutes the claim that the study argues against using constructive cost modeling (CoCoMo), explaining that CoCoMo is based on within-company historical data, not comparisons between different companies.
- +1 pt — Includes a relevant, accurate quote from the slides or reading and ties it clearly to the reasoning (e.g., "based on project history ... requires experience with similar projects").

Deductions:

- −1 pt — Provides only vague or incorrect reasoning about the relationship between price and process emphasis.

- −1 pt — Misinterprets the study as directly invalidating COCOMO without distinguishing between within company and cross company variation.
- −2 pts — Quote is missing or unrelated .
- 0 pts total — Quote is fabricated or from the wrong source.

minutes remaining

Hide Time

Manual Save

Navigation

Question 4. Dynamic Analyses and Human Processes (24 points)

Answer the following questions about dynamic analyses or human processes in software development.

You are instrumenting the program below for a dynamic analysis. You want to be able to analyze the instrumented log later to determine the set of values that were passed to the `external_api()` function. At each possible instrumentation point you can either add instrumentation code to log (i.e., print) the value of a **single variable** or you can add nothing.

The next questions ask you to add the minimum amount of logging possible so that the values that were passed to `external_api()` can still be determined from the log. (We want the minimum amount to avoid slowing the program down too much).

For each possible instrumentation point below, *either* indicate that you must log a variable's value there and indicate which one *or* explain why you need not log any variable's value at that point.

```
1  import random
2
3  def ant(a):
4    return 0
5
6  def bat(b):
7    return b + random.randint(1,4)
8
9  def michigan(a,b,c):
10   # possible instrumentation point 1
11   external_api(a)
12
13   # possible instrumentation point 2
14   d = ant(a)
15   external_api(d)
16
17   if (a < 10):
18     a = bat(b)
19     # possible instrumentation point 3
20     external_api(a)
21
22   elif (a == b):
23     # possible instrumentation point 4
24     external_api(b)
25
26   a = 7
27   # possible instrumentation point 5
28   external_api(c)
29
30   # possible instrumentation point 6
31   external_api(a+b)
32
33 def main():
34   x = random.randint(1,100)
35   y = random.randint(1,100)
36   z = random.randint(1,100)
37   michigan(x,y,z)
38
39 main()
40
```

(a) (1 points) At possible instrumentation point 1

Your answer here.

ANSWER: log the value of a

minutes remaining

Hide Time

Manual Save

Navigation

(b) (1 points) At possible instrumentation point **2**

Your answer here.

ANSWER: do nothing: the value of d is always 0

(c) (2 points) At possible instrumentation point **3**

Your answer here.

ANSWER: log the value of a

(d) (2 points) At possible instrumentation point **4**

Your answer here.

ANSWER: do nothing: b == a, and a was logged at point 1 and has not changed along this path

(e) (2 points) At possible instrumentation point **5**

Your answer here.

ANSWER: log the value of c

(f) (2 points) At possible instrumentation point **6**

Your answer here.

ANSWER: log value of b

You are a software engineering manager who must decide whether or not to employ pair programming. Use the same sort of mathematical reasoning generally described in slides 18 and 19 of the lecture (https://eecs481.org/lectures/se-20-pairskill.pdf). Do not double-count factors. For each task, write the solo programming cost (as a number of hours), and then a newline, and then the pair programming cost (as a number of hours). This may be graded automatically, so please follow the format of one number (solo cost), and then a newline, and then the second number (pair cost). If relevant, use two figures after the decimal point (e.g., 1.23). The interpretation of the various features (e.g., "fewer total lines" or "fewer total defects") is as covered in class.

(g) (2 points) Task 1: 50,000 LOC program, Coding at 25 LOC/hour, Defect rate of 10 defects / KLOC, Defect fix time of 20 hours / defect, Pair programming is 20% slower for code writing, Pair programming results in 15% fewer total defects, Pair programming results in 15% fewer total lines of code.

Your answer here.

ANSWER:
12000
10900

The solo effort requires 2000 hours of coding and produces 500 defects, which then require 10000 hours of fixing. The total is 12000 hours.

The pair effort requires 2400 hours of coding (20% slower) and produces 425 defects (15% fewer), which then require 8500 hours of fixing. The total is 10900.

The ratio is 10900/12000 = 0.91. (Answers that mistakenly double-count factors, such as suggesting that the pair effort only requires 2041 hours of coding, are not correct here.)

minutes remaining

Hide Time

Manual Save

Navigation

(h) (2 points) Task 2: 10,000 LOC program, Coding at 100 LOC/hour, Defect rate of 20 defects / KLOC, Defect fix time of 15 hours / defect, Pair Programming is 30% slower for code writing, Pair programming results in 10% fewer total defects, Pair programming results in 20% fewer total lines of code.

Your answer here.

ANSWER:
3100
2830

The solo effort requires 100 hours of coding and produces 200 defects, which then require 3000 hours of fixing. The total is 3100 hours.

The pair effort requires 130 hours of coding (30% slower) and produces 180 defects (20% fewer), which then require 2700 hours of fixing. The total is 2830.

The ratio is 2830/3100 = 0.91. (Answers that mistakenly double-count factors, such as suggesting that the pair effort only requires 104 hours of coding, are not correct here.)

(i) (5 points) Identify two potential human biases that are avoided by formal code inspection (when done correctly) but may be present in passaround code review. For each one, support it with a quote from the slides or reading (the quote could either show the human bias if things are done badly or show the good thing that happens if things are done well). Focus on aspects specific to code inspection. (After any quotes, use at most four sentences total.)

Your answer here.

ANSWER:
Key human biases that are avoided by the structure of formal code inspection include: the bias of the author to explain or defend the code; the bias of the author to link self-worth to the code's worth; the bias of the reviewers to "show off" as smart by finding fixes; the bias of the author to avoid responsibility.

In code inspection, the structure enforces that "Authors do not explain or defend", mitigating that bias. Similarly, the structure encourages us to "separate self-worth from code", including by having a separate reader role to explain the code (positioning it as an independent artifact, not part of the author's self-worth). Because reviewers do not suggest fixes in code inspection, the structure helps mitigate the bias of wanting to "show off" as smart by finding fixes. Finally, the potential "why fix this, reviewers will find it" bias is mitigated because the responsibility for qulaity lies with the author.

A more generic answer, such as suggesting that confirmation bias is mitigated because code inspection usually involves a diverse and independent inspection team, would likely not receive credit without a significant explanation of how that is not also true in passaround code review (where many companies require reviews from multiple people and structurally enforce expertise along a diversity of dimensions including language familiarity and code module familiarity).

Grading Rubric (Total: 5 points):

- +2 pts — Identifies two specific human biases that formal code inspection avoids (e.g., author defensiveness, linking self-worth to code, reviewers showing off, avoiding responsibility).
- +1 pt — Includes relevant quotes from the slides or readings supporting each identified bias or showing how inspection prevents it.
- +1 pt — Clearly explains how formal inspection mitigates each bias (e.g., author does not defend code, reader role separates self-worth, reviewers do not propose fixes).
- +1 pt — Contrasts inspection with pass-around review to show why the same biases may still occur there.

Deductions:

- −1 pt — Provides only vague or generic psychological terms (e.g., "confirmation bias") without connecting them to inspection structure.
- −1 pt — Per bias mentiones that bias that was not descussed in class
- −2 pts — Missing or irrelevant quotes.
- 0 pts total — Quotes are fabricated or from unrelated sources.

minutes remaining

Hide Time

Manual Save

Navigation

(j) (5 points) Consider the Equifax breach timeline from Slide 5 of https://eecs481.org/lectures/se-03-measure.pdf. Consider defect report Severity and Priority as part of triage; for simplicy, each may separately either be *low*, *medium* or *high*. Note that the vulnerability was found in February but the patch was not deployed until just before August. Support or refute the claim that its internal Priority (for developers writing the patch) was likely high. Then support or refute the claim that its internal Severity (for developers writing the patch) was likely high. Justify one part of your argument with a quote from the reading or slides. After copying the quote, use at most four sentences.

Your answer here.

ANSWER:
For Priority, the best answer is **supporting** the claim that it was **high**. It is tempting to conclude that the Priority was low or medium because it took so long for the patch to be deployed, but a high defect Priority means "Someone is working or planning to work on this task soon", which did happen (by March). Whether or not the patch was deployed is a separate matter from whether developer effort was put into producing a patch early on. While accounts explaining the deployment delay vary (e.g., some sources suggest "Equifax administrators were told to apply the patch to any affected systems, but the employee who should have done so didn't.", another common story is that company admins were worried about public perception and quashed it, etc.), this question asked you to consider the defect reporting and triage process "for the developers writing the patch". From that perspective, the Priority is best described as High (someoned did work on the task very soon and produced the patch).

(Some AI tools favor the explanation that Priority was low because the patch was not deployed quickly, but that is not what this question asked about.)

For Severity, the best answer is **supporting** the claim that it was **high**. Security issues, including those that result in the loss or exfiltration of data from a "widely used and important component", are typically high Severity. While some exact outcomes (e.g., class action lawsuits, executives stepping down, etc.) wouldn't have been precisely knowable at the time, the severe outcome of breaching the data of millions of users was."

Grading Rubric (Total: 5 points):

- +2 pts — Correctly supports that the Priority for developers writing the patch was high, explaining that high Priority means developers are actively or imminently working on the issue (regardless of deployment delay).
- +2 pts — Correctly supports that the Severity was high, justified by the nature of the vulnerability (major security risk, sensitive data exposure, or impact to millions of users).
- +1 pt — Includes a relevant, accurate quote from the slides or readings and clearly ties it to the reasoning for either Priority or Severity.

Deductions:

- −1 pt — Confuses Priority with deployment timing or management decisions rather than developer triage effort.
- −1 pt — Provides vague or unsupported claims about Severity or Priority without clear reasoning.
- −2 pts — Quote is missing or irrelevant.
- 0 pts total — Quote is fabricated or from an unrelated source.

Question 5: Dataflow Analysis (13 points total)

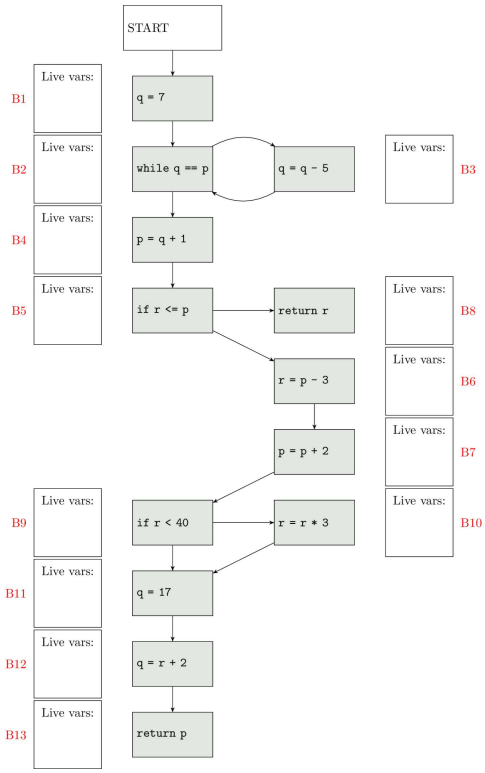
Consider a *live variable dataflow analysis* for three variables, `p`, `q`, and `r` used in the control-flow graph below. We associate with each variable a separate analysis fact: either the variable is (1) possibly read on a later path before it is overwritten (live), or (2) it is not (dead). We track the set of live variables at each point: for example, if `p` and `q` are alive but `r` is not, we write `{p, q}`. The special statement `return` reads, but does not write, its argument. In addition, `if` and `while` read, but do not write all of the variables in their predicates. (You must determine if this is a forward or backward analysis.)

minutes remaining

Hide Time

Manual Save

Navigation



(1 point each) For each basic block `B1` through `B13`, write down the list of variables that are live *right before* the start of the corresponding block in the control flow graph above. Please list only the variable names in lowercase without commas or other spacing (e.g., use either `ab` or `ba` to indicate that `a` and `b` are alive before that block).

`B1`
ANSWER: `{'r', 'p'}`

`B2`
ANSWER: `{'r', 'q', 'p'}`

`B3`
ANSWER: `{'r', 'q', 'p'}`

`B4`
ANSWER: `{'r', 'q'}`

`B5`
ANSWER: `{'r', 'p'}`

`B6`
ANSWER: `{'p'}`

`B7`
ANSWER: `{'r', 'p'}`

`B8`
ANSWER: `{'r'}`

`B9`
ANSWER: `{'r', 'p'}`

`B10`
ANSWER: `{'r', 'p'}`

`B11`
ANSWER: `{'r', 'p'}`

`B12`
ANSWER: `{'r', 'p'}`

`B13`
ANSWER: `{'p'}`

Extra Credit

(1) What is your favorite part of the class so far? Alternatively, name a course staff member and list something you have enjoyed about that person. (1 point)

Your answer here.

(2) What is your least favorite part of the class so far? Alternatively, what improvement would you suggest for future semesters? (1 point)

Your answer here.

(3) In the context of your own experience in computing courses at Michigan, how do you think CSE should apply the lessons from Lewis and Shah's *How Equity and Inequity Can Emerge in Pair Programming* to maximize the benefits of working with partners but mitigate the risks they identified? Demonstrate that you have read the paper critically and tie it in to one of your personal experiences (e.g., what is something that you think would work specifically here at UM, and what is something they talk about that you think specifically would be hard to do here at UM), going beyond a Generative AI summary. (2 points)

Your answer here.

(4) If you read any *other* optional reading (i.e., not Lewis and Shah), identify it and demonstrate to us that you have read it critically, going beyond a Generative AI summary. (2 points)

Your answer here.

(5) Which Generative AI tool(s) and version(s) did you use on this exam, if any? For which parts of the exam did you find them (or speculate that they would be) helpful or unhelpful? (Remember, free GenAI is allowed, so this is not cheating. This is to help us improve the course, not to get you in trouble.) If you used any other sources that you are uncertain about, you can cite them here. (1 points)

Your answer here.

minutes remaining

Hide Time

Manual Save

Navigation

Honor Pledge and Exam Submission

You must check the boxes below before you can submit your exam.

- [ ] I have neither given nor received unauthorized aid on this exam.
- [ ] *I am ready to submit my exam.*

Submit My Exam

Once you submit, you will be able to leave the page without issue. Please don't try to mash the button.

The exam is graded out of 100 points.