Navigation

Question 1. Word Bank Matching (1 point each, 14 points total)

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

A. — A/B Testing	B. — Alpha Testing	C. — Beta Testing	D. — Competent Programmer Hypothesis
E. — Deliverables	F. — Formal Code Inspection	G. — Fuzz Testing	H. — Instrumentation
I. — Integration Testing	J. — Invariant	K. — Maintainability	L. — Mocking
M. — Oracle	N. — Pair Programming	O. — Passaround Code Review	P. — Perverse Incentive
Q. — Race Condition	R. — Regression Test	S. — Requirements	T. — Risk
U. — Spiral Development	V. — Streetlight Effect	W. — Test-driven Development	X. — Threat to Construct Validity
Y. — Threat to External Validity	Z. — Unit Testing		

Q1.1: I

Bruce is developing a video game. He creates a class Player and a class Car. His intended functionality is that the player should be able to get inside the car and drive it. He then writes a set of test cases to ensure the interaction between these two classes is functioning properly.

Q1.2: Q

Mike is writing an application that allows moviegoers to reserve seats at the theater. Unfortunately, there is a bug that allows multiple users to reserve the exact same seat at the same time. If only Mike had included safeguards to prevent multiple reservations of the same seat.

Q1.3: R

Valeria is failing a public test case provided in the project spec. She uses a debugger to identify the issue and promptly fixes it. She writes a test case afterwards to alert her if the issue resurfaces as she continues the project.

Q1.4: O

Sasha has just finished making a small change in a file within her company's codebase. Prior to pushing it to production, she emails her colleague Jean to look over her changes at his convenience. Jean gives his approval a few hours later and Sasha pushes her code, committing the change to the main repository.

Q1.5: M

Jan writes a function for sin(x). He knows sin(90) is equal to 1, so he accordingly writes a test case. In this instance, 1 is the...

Q1.6: V

After discovering that her submissions to the Autograder are failing due to a timeout, Xiaoyu realizes that she must make some runtime optimizations. She begins by looking at her recently-written functions with the most lines of code since they are fresh in her mind. Unfortunately, the real bug is in a single innocuous line in which her code calls an external library function incorrectly.

Q1.7: Y

Antoinette is interested in learning about how quickly it takes humans to understand code snippets. She conducts a study on a group of PhD students and concludes that humans, on average, take 45 seconds to understand how merge sort works. Her advisor is quick to chime in that her conclusion is flawed for this reason...

Q1.8: L

Eren is working on the backend of a website. He implements a function that makes a rather expensive database query. When writing test cases, he substitutes a hard-coded string in place of the query.

Navigation

- Question 1
- Question 2
- Question 3
- Question 4
- Question 5
- Extra Credit

Q1.9: Z

Anthony is working on his HW6 open-source contribution assignment and submits a pull request to Runelite. His code consists of many functions, but it is rejected because the functions lack test cases. If only he had implemented this quality assurance strategy...

Q1.10: U

Jordan is developing a video chat app for a client. He spends two months adding voice chat functionality and presents it to the client for feedback. He then spends two months adding live video before getting feedback from the client. He continues this iterative process.

Q1.11: J

Larry rigorously tests a mathematical function he has written and realizes that the output is always greater than or equal to the input.

Q1.12: F

Jeremy has just written a file that will be used in the software for a pacemaker, a device used for heart arrhythmias. He schedules a meeting with members of his team where they will sit down and go through his code, line by line, with the goal of identifying bugs.

Q1.13: C

Roxanne is preparing to release a new social media app. To catch bugs that may surface during common use, she hires a group of social media influencers to test out the app and report any issues.

Q1.14: A

Annie is a developer for an online retailer. She is considering changing the color of the 'BUY NOW' button from green to blue. She decides to change the color for a subset of customers and compares the difference in purchasing activity before coming to a decision.

Question 2: Coverage (18 points total)

You are given the Python function below.

```
def awesome_grizzly (j: bool, k: bool, l: bool):
    STMT_1
    if (( j or k) and (not k and l)):
        STMT_2
    else:
        STMT_3
    if ((j and l) and not (j or k) and l):
        STMT_4
    elif ((not j and l) or not (not k)):
        STMT_5
        if (k and not l):
            STMT_6

```

Q2.1 (4 points) Calculate the minimum statement coverage attainable using one test input and provide such an input (i.e., values of values of {var1}, {var2}, and {var3}).

4/6 = 66% with `j: True, k: True, l: False`

Q2.2 (6 points) Provide a single minimum set of test inputs(s) that achieves maximum statement AND maximum path coverage for this particular program. Consider only feasible paths and reachable statements. In one sentence, explain why this is the smallest number of test inputs that can maximize both statement and path coverage.

The minimum set of test inputs is `{(True, True, True), (True, True, False), (True, False, True), (False, False, False)}`. This is the smallest set of inputs to maximize path coverage because one input is necessary to cover each path, and the set of paths contains the set of paths which maximize statement coverage.

Navigation

Q2.3 (5 points total, 1 point per selection) Next, consider the C code below. Make selections for the operators P, Q, R, S, and T such that:

1. The *statement coverage* induced by executing the single test case `ecstatic_bohr(0, 1, 1)` is maximized.

```c
void ecstatic_bohr (bool j, bool k, bool l) {
    if (j  •P•  k) {
        STMT_1;
        if (k  •Q•  l) {
            STMT_2;
        } else {
            STMT_3;
        }
    } else {
        STMT_4;
    }

    if (j   •R•  l) {
        STMT_5;
    } else if (k •S• j) {
        STMT_6;
    }

    if (l  •T•  k) {
        STMT_7;
    } else {
        STMT_8;
    }
}

```

Operator	Maximize Statement Coverage
P	[✓ !=] < ≥
Q	[✓ !=] < ≥
R	[✓ !=] < ≥
S	[✓ !=] < ≥
T	[✓ !=] < ≥

Multiple answers apply. One correct answer is P, Q, R, S, T all being !=, which results in statement coverage of 4/8 = 50%

Q2.4 (3 points) Support or refute: it is harder to maximize branch coverage for code with a lower Maintainability Index. Use at most four sentences.

Likely support. The Maintainability Index (covered on slide 7+ of the Measurement slideset) involves three components: Halstead volume (operator counts are not relevant for branch coverage per se: you can have many or few operators on straight-line code), LOC (not relevant for branch coverage per se: you can have many or few branches per line), and Cyclomatic Complexity. Cyclomatic Complexity measures linearly independent paths through programs and is described on Slide 19 as relating to the number of tests to cover all branches. So higher Cyclomatic Complexity (= lower Maintainability Index) generally means code requires more tests to cover branches and thus to maximize branch coverage.

Navigation

Question 3: Short Answer and Potpourri (28 points total)

Provide answers to each question below.

Q3.1 (3 points) Generating Test Inputs

Compare and contrast fuzz testing and constraint-based solvers for generating test inputs: what aspects do they share and where do they differ? Give one example program for which we would expect a fuzzer to outperform a constraint-based solver. Give one example of a program for which we would expect a constraint-based solver to outperform a fuzzer. Use at most six sentences.

Your answer here.

Fuzz testing and constraint-based test input generation are both interested in generating test inputs (not necessarily oracles) to reach as much of the code as possible without requiring manual human effort. Fuzz testers do so by generating random inputs (e.g., random integers, random strings) and are typically "black box" analyses (they do not need to see the code). Constraint solvers do so by generating path predicates and solving them to reach particular targets and they are "white box" analyses (they do need to see the code).

A program that contains a conditional like "if (input == 12345) ..." is hard for a fuzz tester (because you are unlikely to "guess" 12345 to visit the true branch) but easy for a constraint solver. By contrast, a conditional like "if (input > length(read_file("on-disk.txt"))) ..." is likely to be hard for a constraint solver (which cannot reason about files in the disk or over the network or the like) but a fuzz tester can just guess big and small numbers. Similarly, modern constraint solvers struggle with non-linear arithmetic (e.g., "input * input > 25"). Slide 24 of the lecture gives a concrete example.

Q3.2 (2 points each; 10 points total) Software Engineering Comparisons

Consider each of the following pairs of techniques, tools, or processes. For each pair, give a class of defects or a situation for which the first does better than the second (i.e., is more likely to succeed and reduce software engineering effort and/or improve software engineering outcomes) and explain why. For full credit, each explanation must include why the second is worse in that situation (simply indicating how the first is good is not sufficient). Use at most three sentences per answer.

maximizing branch coverage vs. pair programming

Your answer here.

For defects related to control flow (such as the infinite loop leap year bug on Slide 9 of the Code Review lecture), branch coverage may reveal the defect quickly, while having a second human look at the code while it is being created may not reveal the mistake.

For a situation, consider a product with a tight time deadline: it must pass QA and ship soon. In such a setting, pair programming may be worse (because it can take longer) compared to simply generating high branch coverage (via something like AFL: note that the question asks about coverage, not about oracles). In 481 we might not favor such an argument (e.g., investing in pair programming may be better in the long term) but it can be a full-credit answer to a test question.

static dataflow analysis vs. unit tests

Your answer here.

For security bugs (e.g., information leaks), a static dataflow analysis may be better at pointing out potential defects on all paths, regardless of whether or not the unit tests happened to consider those paths. Static analysis can also give conservative answers for code involving other modules, even without calling or integrating with those modules. Finally, static analyses can be applied even if test inputs and oracles have not been written yet.

mocking vs. passaround code review

Your answer here.

Early in development if there are not many experts in a particular module or if there are no other developers who are familiar with the programming language used, mocking may be a better QA activity than code review. It only requires one developer and does not require any expensive interfaces to be in place. In the lecture we mentioned that some companies require an expert or owner of the module to be involved, and if there are no other experts available yet (perhaps they have not yet been hired this early), code review will not be as effective. Similarly, some companies require familiarity with the language (sometimes called a "badge") -- for a new module written in a new language, other language experts may not be available, so code review would not be as effective.

the Eraser dynamic analysis vs. fuzz testing

Your answer here.

Eraser does better at detecting race conditions related to misusing locking. Since such bugs tend to be influenced more by the scheduler than by the program inputs, fuzz testing (which picks program inputs) can be particularly bad at finding them.

regression testing vs. formal code inspection

Your answer here.

Regression testing is a better option for making sure that a code change does not reintroduce any bugs that have been previously fixed. It is suitable for this situation because it verifies that the code base still maintains the correct behavior, despite a code change. Formal code inspection isn't as suitable for this situation because it requires gathering the team together, which is typically too logistically difficult and expensive for such a small change.

Navigation

Q3.3 (2 points each; 6 points total) Pair programming and Process

You are a manager at WebFlix and need to decide whether or not to employ pair programming for a series of tasks. Since pair programming tends to produce code of higher quality, you are willing to opt for pair programming for a particular task so long as there is not an increase in total costs of more than 59%. The table below summarizes the various costs and benefits of using pair programming for each task.

For "Pair Programming increase in Cost per Hour (%)", 100% would mean that pair programming carries twice the cost of solo programming. Similarly, a "Pair Programming decrease in Total Hours (%)" of 40% means that a task that takes 10 hours solo would take 6 hours with pair programming.

Task	Total Hours	Cost Per Hour	Pair Programming decrease in Total Hours (%)	Pair Programming increase in Cost per Hour (%)
A	27	10	39	100
B	34	5	83	113
C	12	17	84	139

(2 points each) For each of the following tasks, decide whether to employ pair programming.

A	◯ Yes, use Pair Programming
	◯ No, do not use Pair Programming
B	◯ Yes, use Pair Programming
	◯ No, do not use Pair Programming
C	◯ Yes, use Pair Programming
	◯ No, do not use Pair Programming

ANSWER: This is a math optimization problem. We compute the original cost, subtract out the decrease in the number of hours (folding in the pair benefit), then multiply by the new cost per hour. If this decrease results in a new cost below the target acceptable percentage, then we expect to mark True. Otherwise, False.

- A: T
- B: T
- C: T

Q3.4 (3 points) Risk and Measurement

You are a software engineering manager. You are considering a proposal in which 30% of the resources currently used for integration testing would instead be reallocated and used for a different dynamic analysis (e.g., something like Chaos Monkey or Driver Verifier, etc.). Identify two risks associated with this proposal and one benefit associated with this proposal. For each, identify one associated measurement that might be taken to reduce uncertainty (i.e., to determine the degree to which that positive or negative outcome occurred).

Your answer here.

Dynamic analysis tools such as Chaos Monkey or the Driver Verifier were covered starting on Slide 52 of the Dynamic Analysis lecture (and it was remarked during the lecture that they would be fair game), as well as in some optional readings. Risks (e.g., staff illness, requirements changes, etc.) are covered in the Risk lecture and might prevent a high-quality product from shipping on time. Measurements (covered in their own lecture) help reduce uncertainty and thus help detect and manage risk.

Benefits of the proposal relate to the use of the dynamic analysis. For example, one benefit of using a tool like the Driver Verifier is that it can catch corruption bugs related to low-level systems code. One benefit of Chaos Monkey style tools is that they are particularly good at findings bugs related to resilience, redundancy or even internationalization. Students could also mention that these dynamic analyses are automated, compared to creating integration tests and oracles, so one potential benefit is that developer time and effort is freed up for other uses.

Risks. however, abound. Integration testing is particularly good at finding bugs related to two modules working together. One risk is that fewer such bugs might be detected before shipping. Similarly, dynamic analyses often suffer from soundness and completeness issues: false positives and false negatives. One risk is that the dynamic analysis will produce too many false alarms. Another is that it will miss important bugs (even of the type it is "supposed" to find). Other risks are possible: students

Navigation

- Question 1
- Question 2
- Question 3
- Question 4
- Question 5
- Extra Credit

might mention that dynamic analyses require you to already have a high quality test suite (remember: you have to run the program on something) and thus may not be workable until later in the development process when many test inputs are available.

Each associated measurement should be something that can be quantified and that could help a manager answer a question like "How big is this problem?" or "Is this really a big issue?" If one is worried that no bugs will be detected, a metric like "bugs reported per line of code" (e.g., on just one module, before deciding if the analysis should be deployed instead of 30% of integration testing) could help with that decision. Similarly, the "false positive rate", the "number of critical bugs missed", or even the "coverage requirement for the test inputs for the tool to run well" or the "weeks into development when enough test inputs will be available" could all be reasonable choices for the risks above. For benefit metrics, "bugs found" or "bugs found per lines of code" or "developer hours saved" or the like might all apply.

Q3.5 (3 points) Development Processes

In three sentences or fewer, describe the differences between spiral development and waterfall development.

Your answer here.

In the Waterfall Model, stages such as requirements elicitation, design, coding, testing, and operations are carried out in strict order. As a result, information learned during testing or operations would never influence design, for example. By contrast, in spiral development, an increasingly complex series of prototypes is constructed while accounting for risk. This allows information learned during the testing or operation of one prototype to influence the design of the next, for example. These concepts are covered in the initial slides of the Process lecture.

Q3.6 (3 points) Code Review

Identify a developer expectation of modern passaround code review that is commonly met. Identify a developer expectation of modern passaround code review that is rarely met. Describe a buggy patch that modern passaround code review is unlikely to correctly reject. Use at most six sentences.

Your answer here.

Following Bacchelli and Bird's "Expectations, outcomes, and challenges of modern code review" (also covered in Slides 32-36 of the Code Review Lecture), key expectations that are met include finding defects and code improvements. Goals that are rarely met include knowledge transfer and alternate solutions.

Consider a patch that "does what it says" (e.g., says it is removing a button and actually removes a button) but is doing the wrong thing (e.g., the customer wants the button retained, not removed). As Slide 38 and Section VI-A of the Bachhelli and Bird paper suggests, "the most difficult thing when doing a code review is understanding the reason of the change" and "the biggest information need in code review: what instigated the change". If the code reviewers do not know why the change is being made, they will not be able to assess it correctly, and may allow a patch that has no visible defects (but is ultimately doing the wrong thing). In general, students should describe a patch that has no "easy errors" but instead has a bug "beneath the surface".

Question 4. Mutation Testing (29 points)

Consider the code snippet below defining a function `foo`:

```
def foo(y):
    if (y < 0):       # Mutant 1: y <= 0
        # Invalid input
        return -1
    elif y == 0:
        return 0
    elif y == 1 or y == 2:  # Mutant 2: y == 1 and y == 2
        return 8
    else:
        return foo(y - 1) + foo(y - 2)  # Mutant 3: foo(y - 1) - foo(y - 2)

```

(a) (1 point per field, 20 points total) Complete the table below by indicating whether each test kills each Mutant. (Y for killed and N for not killed). Oracle stands for the expected output of `foo` run on the corresponding input. **Be careful**: subsequent subquestions depend on correctly understanding this subquestion.

Test #	Input (y)	Oracle (`foo(y)`)	Mutant 1	Mutant 2	Mutant 3
(Q4.a.0): Test 0	`0`	`0`	Y	N	N

Navigation

Test #	Input (y)	Oracle (`foo(y)`)	Mutant 1	Mutant 2	Mutant 3
(Q4.a.1): Test 1	1	8	N	Y	N
(Q4.a.2): Test 2	2	8	N	Y	N
(Q4.a.3): Test 3	3	16	N	Y	Y
(Q4.a.4): Test 4	4	24	N	Y	Y

(b) (1 point) What is the mutation score for tests `0-4` using Mutants 1-3?

1.0

(c) (1 point) What is the mutation score for test `0` using Mutants 1-2?

0.5

(d) (1 point) What is the mutation score for tests `0-4` using just Mutant 1?

1.0

(e) (2 points per mutant, 6 total) Make **at most one edit** each to create **THREE NEW** and **DIFFERENT** mutants of `foo`. Exactly one of your three new first-order mutants should be killed by when provided the same test input `y=2`.

(Make at most one edit to the code to create a new mutant that is different from Mutants 1-3. Repeat this process to produce a total of 3 new, mutually different mutants and make sure the kill score with input `y=2` is 1/3. For example, you might change line 2 from `if num < 0` to `if num > 0` but not to `if num <= 0` because that's already Mutant 1 in this question.)

You should not introduce any loops as part of your mutations. Make sure that your mutants correspond to valid Python3 code — syntactically invalid mutants may receive no credit. Moreover, please do not attempt to subvert this question by modifying the code to immediately return a value — you are asked to make first-order mutants.

Attempting to submit code that infinitely loops, that interacts with any I/O, that imports other libraries, or that shells out is a violation of the honor code. Doing so will result in a 0 for the entire exam.

Mutant X:

```
 1 def foo(y):
 2     if (y < 0):       # Mutant 1: y <= 0
 3         # Invalid input
 4         return -1
 5     elif y == 0:
 6         return 0
 7     elif y == 1 or y != 2:  # Mutant 2: y == 1 and y == 2
 8         return 8
 9     else:
10         return foo(y - 1) + foo(y - 2)  # Mutant 3: foo(y - 1) - foo(y - 2)
11
```

Mutant Y:

```
 1 def foo(y):
 2     if (y < 0):       # Mutant 1: y <= 0
 3         # Invalid input
 4         return -1
 5     elif y != 0:
 6         return 0
 7     elif y == 1 or y == 2:  # Mutant 2: y == 1 and y == 2
 8         return 8
 9     else:
10         return foo(y - 1) + foo(y - 2)  # Mutant 3: foo(y - 1) - foo(y - 2)
11
```

Mutant Z:

```
 1 def foo(y):
 2     if (y < 0):       # Mutant 1: y <= 0
 3         # Invalid input
 4         return -1
 5     elif y == 0:
 6         return 0
```

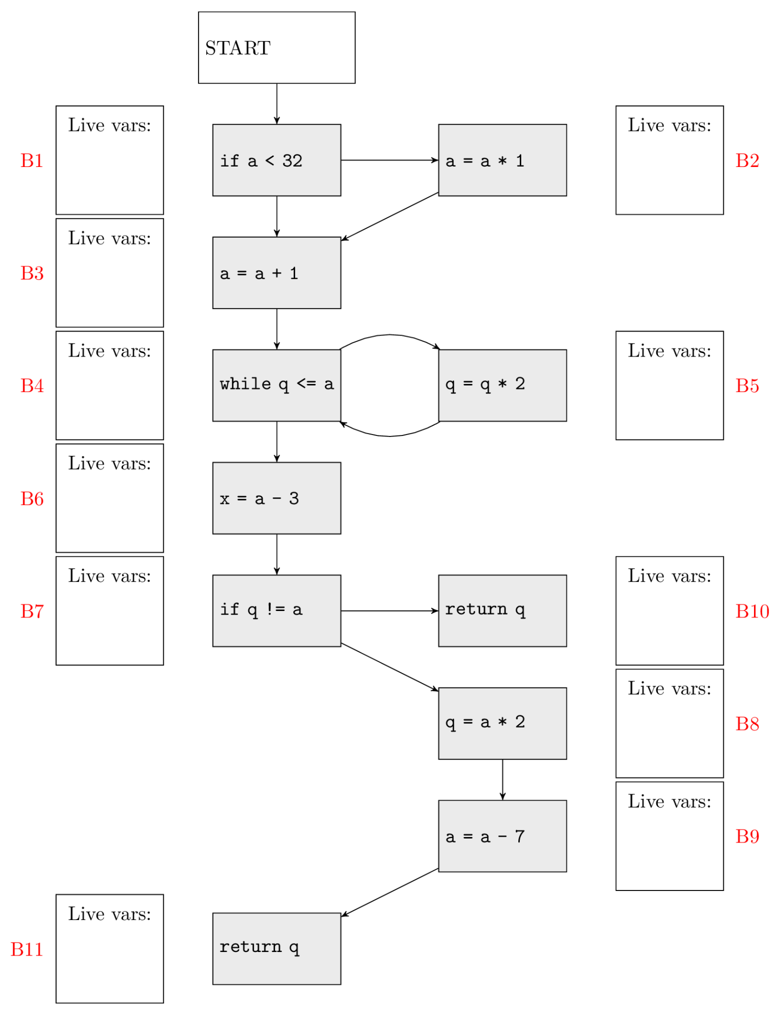
```
    elif y == 1 or y <= 2:  # Mutant 2: y == 1 and y == 2
        return 8
    else:
        return foo(y - 1) + foo(y - 2)  # Mutant 3: foo(y - 1) - foo(y - 2)

```

Navigation

Question 5: Dataflow Analysis (11 points total)

Consider a *live variable dataflow analysis* for three variables, a, x, and q used in the graph below. 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 a and x are alive but q is not, we write `{a, x}`. The special statement `return` reads, but does not write, its argument. (You must determine if this is a forward or backward analysis).



(1 point each) For each basic block B1 through B11, 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	B2	B3	B4
ANSWER: `{'a', 'q'}`	ANSWER: `{'a', 'q'}`	ANSWER: `{'a', 'q'}`	ANSWER: `{'a', 'q'}`
B5	B6	B7	B8
ANSWER: `{'a', 'q'}`	ANSWER: `{'a', 'q'}`	ANSWER: `{'a', 'q'}`	ANSWER: `{'a'}`
B9	B10	B11	
ANSWER: `{'a', 'q'}`	ANSWER: `{'q'}`	ANSWER: `{'q'}`	

Extra Credit

Each question below is for 1 point of extra credit unless noted otherwise. We are strict about giving points for these answers. No partial credit.

(1) What is your favorite part of the class so far?

Your answer here.

(2) What is your least favorite part of the class so far?

Your answer here.

Navigation

- Question 1
- Question 2
- Question 3
- Question 4
- Question 5
- Extra Credit

(3) If you read any optional reading, identify it and demonstrate to us that you have read it. (2 points)

Your answer here.

(4) If you read any *other* optional reading, identify it and demonstrate to us that you have read it. (2 points)

Your answer here.

(5) In your own words, identify and explain any of the bonus psychology effects. (2 points)

Your answer here.