LATE

minutes remaining

Hide Time

Question 1. Word Bank Matching

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.

A. — Alpha Testing	B. — A/B Testing	C. — Agile Development	D. — Beta Testing
E. — Constraint Solver	F. — Comparator	G. — Competent Programmer Hypothesis	H. — Confounding Variable
I. — Coupling Effect Hypothesis	J. — Dynamic Analysis	K. — Formal Code Inspection	L. — Invariant
M. — Integration Testing	N. — Maintainability	O. — Mocking	P. — Oracle
Q. — Passaround Code Review	R. — Perverse Incentive	S. — Requirements	T. — Risk
U. — Sampling Bias	V. — Software Quality Metric	W. — Spiral Development	X. — Static Analysis
Y. — Threat to Validity	Z. — Waterfall Model		

Hyakudo, Inc. is building a large Machine Learning model for predicting shopping carts. The test cases must be flexible given the variety of acceptable outputs.

Kyle is struggling to debug a seemingly uncaused segmentation fault. After a few days of working on it, she realizes that the segfault was caused by an interaction of several small typos.

You are creating a website for a client who is a bakery shop owner. She tells you about all the things the website must have: color scheme, ability to order online, etc.

Tenshi is working on a new app called *HandSonic*. The app uses an API that costs a dollar on each invocation. To facilitate testing, Tenshi decides to hard-code a few fixed values in place of invoking those APIs during testing.

Honda is making a game called Fruits Basket. She designs a test case that only passes if player's screen flashes green when they select a melon.

Whenever a developer submits a pull request to a project's Github, the other developers must examine the proposed changes and sign off on them before they are accepted into the main codebase.

Gloom is in the middle of creating their latest product: *Gloom* Video Conferencing Tool. They decide to choose a few people from the development group to test the software.

Jen is developing a fitness app that is intended to get beginners interested in fitness. Unfortunately, Jen only surveys professional weightlifters for their input. When the app was released, users complained that it was too intimidating for

LATE

minutes remaining

Hide Time

beginners.

Xiaobao notices he completes homework faster when consuming warm soda. He concludes that the only reason for his faster homework completion is the presence of soda. What oculd go wrong?

StartHart, after a development team has finished working on pacemaker firmware, the team meets for an hour to pore over s of code.

Ann Arbor HashToTable is trying to decide whether the dank reefer button should be red or green. They decide to roll out both variants to residents of the Tri-State area to determine which color button is pushed more often.

The lead engineer at *FallCrate* wants to determine how long each function takes to execute in their new storage software.

Thomas and Gordon each work on separate prototypes for a software project. After a week, Thomas's prototype has higher path coverage, but Gordon's is faster to execute.

Question 2: Coverage

You are given the C function below. For simplicity, assume that `bool` means an integer 0 for False, and 1 for True.

```
1  void high_newton ( bool x, bool y, bool z ) {
2      S_0;
3      if ((x && z || x) != (y || z && z)) {
4          S_1;
5      } else if ((x && !z || y) == (x && y || y)) {
6          S_2;
7      } else {
8          S_3;
9      }
10     if (!z == (y && !z || z)) {
11         S_4;
12     } else {
13         S_5;
```

Provide the *minimum* statement coverage that can be obtained with one test case? Express your answer as a comma-separated list of `S_n` terms (e.g., `S_0,S_1,...`). If no statements can be covered, write `N/A`.

Minimum statement coverage

Provide values for `x`, `y`, and `z` that achieves the *minimum* statement coverage for this function.

`x`

x

`y`

y

`z`

z

What is the *maximum* path coverage obtained using a suite of two test cases? Express your answer as a whole number (i.e., the *quantity* of paths covered).

Maximum path coverage

LATE

minutes remaining

Hide Time

vide two test cases in terms of `x`, `y`, and `z` that, together, yield the highest *path coverage* possible for this function. Use `1` True and `0` for False. Alternatively, indicate that it is not possible using the checkbox.

Test case 1

x

x

y

y

z

z

Test case 2

x

x

y

y

z

z

☐ Check this box if it is not possible to provide such a test case.

Now, assume that the input assumptions are relaxed: `x`, `y`, and `z` can now be any integer value. Will the total number of paths increase, decrease, or stay the same? Support your position.

Write your answer here.

Question 3: Short Answer and Potpourri

Provide answers to each question below.

You are a software engineering manager. You are considering a proposal in which 25% of the resources (i.e., developer time) currently used for integration testing would instead be reallocated and used for static analysis to find security errors. Identify two risks associated with this proposal and one benefit associated with this proposal. For each risk or benefit, 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.

Read the analysis task descriptions below. For each task, choose three components of a good analysis solutions by taking one from each column in the table below. For example, `1 a p` is valid, but not `a b 2` or `c 3`. If multiple combinations might fit, choose the *best* answer or the one corresponding to a tool or technique from class or the readings. You may use an option more than once across multiple task descriptions.

LATE

minutes remaining

Hide Time

You wish to assess the availability of a service even if its dependencies or assumptions might fail.

- (a): Automated Static Analysis
- (b): Manual Dynamic Analysis
- (c): Analysis of Non-Executable Artifact
- (d): Manual Static Analysis
- (e): Automated Dynamic Analysis
- (1): Enumerate
- (2): Replace
- (3): Track
- (4): Solve
- (5): Subjectively Assess
- (m): Abstracted Values
- (n): Syntactic Elements
- (o): Control-Flow Elements
- (p): Everything Available
- (q): Sets of Locks Held
- (r): Scheduler Interleavings

You are using Microsoft's Driver Verifier to find bugs in operating system code.

- (a): Automated Static Analysis
- (b): Manual Dynamic Analysis
- (c): Analysis of Non-Executable Artifact
- (d): Manual Static Analysis
- (e): Automated Dynamic Analysis
- (1): Enumerate
- (2): Replace
- (3): Track
- (4): Solve
- (5): Subjectively Assess
- (m): Abstracted Values
- (n): Syntactic Elements
- (o): Control-Flow Elements
- (p): Everything Available
- (q): Sets of Locks Held
- (r): Scheduler Interleavings

You apply the CHESS algorithm to a program to detect concurrency bugs.

- (a): Automated Static Analysis
- (b): Manual Dynamic Analysis
- (c): Analysis of Non-Executable Artifact
- (d): Manual Static Analysis
- (e): Automated Dynamic Analysis
- (1): Enumerate
- (2): Replace
- (3): Track
- (4): Solve
- (5): Subjectively Assess
- (m): Abstracted Values
- (n): Syntactic Elements
- (o): Control-Flow Elements
- (p): Everything Available
- (q): Sets of Locks Held
- (r): Scheduler Interleavings

You wish to determine the Halstead Volume of a method.

- (a): Automated Static Analysis
- (b): Manual Dynamic Analysis

LATE

minutes remaining

Hide Time

- (c): Analysis of Non-Executable Artifact
- (d): Manual Static Analysis
- (e): Automated Dynamic Analysis
- (1): Enumerate
- (2): Replace
- (3): Track
- (4): Solve
- (5): Subjectively Assess
- (m): Abstracted Values
- (n): Syntactic Elements
- (o): Control-Flow Elements
- (p): Everything Available
- (q): Sets of Locks Held
- (r): Scheduler Interleavings

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.

static dataflow analysis vs. **maximizing branch coverage**

static dataflow analysis vs. maximizing branch coverage

formal code inspection vs. **careful defect reporting and triage**

formal code inspection vs. careful defect reporting and triage

regression testing vs. **mocking**

regression testing vs. mocking

Consider the following method with inputs `x` (an integer) and `y` (a string). For each of the marked `print` statements, give values of `x` and `y` that force the statement to be executed (or check the box indicating no such values exist). In addition, for each fo the marked `print` statements, indicate (1) whether or not random input generation (i.e., *fuzzing*) is likely to generate test inputs to reach that statement and (2) whether or not whitebox test input generation (with *constraint* solving) is likely to generate test inputs to reach that statement.

```
1  import urllib
2  def liskov(x,y):
3      z = x * len(y)
4
5      print("statement 1")
6
7      if(x == len(urllib.urlopen("www.google.com").read())):
8          print('statement 6')
9
10
11     if (z > 26):
12         print("statement 2")
13         x = x * 26
```

LATE

minutes remaining

Hide Time

Statement#	No satisfying input?	x?	y?	Fuzzing?	Constraint?
stmt1	No values for x, y will reach "statement 1".	x	y	☐ Fuzzing will likely generate an input that reaches "statement 1"	☐ Constraint solving will likely generate an input that reaches "statement 1"
stmt2	No values for x, y will reach "statement 2".	x	y	☐ Fuzzing will likely generate an input that reaches "statement 2"	☐ Constraint solving will likely generate an input that reaches "statement 2"
stmt3	No values for x, y will reach "statement 3".	x	y	☐ Fuzzing will likely generate an input that reaches "statement 3"	☐ Constraint solving will likely generate an input that reaches "statement 3"
stmt4	No values for x, y will reach "statement 4".	x	y	☐ Fuzzing will likely generate an input that reaches "statement 4"	☐ Constraint solving will likely generate an input that reaches "statement 4"
stmt5	No values for x, y will reach "statement 5".	x	y	☐ Fuzzing will likely generate an input that reaches "statement 5"	☐ Constraint solving will likely generate an input that reaches "statement 5"
stmt6	No values for x, y will reach "statement 6".	x	y	☐ Fuzzing will likely generate an input that reaches "statement 6"	☐ Constraint solving will likely generate an input that reaches "statement 6"

In his guest lecture, Titus Winters talked about how Google addressed a number of software engineering concerns. In *two short paragraphs*, choose two examples he gave of software engineering concepts (e.g., important considerations, process decisions, measurements, code analyses or tools, differences between school projects and real-world software, etc.). For each example, identify a way it disagrees with (or at least suggests a significant alteration to) something that was discussed in the class or a reading.

Titus Winters discussion.

Question 4. Mutation Testing and Invariants

Consider the following Python function `compute_lcm` that takes as input two integers and computes the least common multiple of the integers:

```
def compute_lcm(x, y):
    lcm = 0
    greater = 0
```

LATE

minutes remaining

Hide Time

```
    if x > y:
        greater = x
    else:
        greater = y
    while(True):
        if((greater % x == 0) and (greater % y == 0)):
            lcm = greater
            break
        greater += 1
```

Given the following two suites of mutants of `compute_lcm`, provide the smallest set of inputs for `x` and `y` that kill one suite but not the other, or indicate that no such input exists. In your answer, indicate which suite, if any, is killed by your inputs.

Suite 1:

```
def suite1_mutant1(x, y):
    lcm = 0
    greater = 0
    if x > y:
        greater = x
    else:
        greater = y
    while(True):
        if((greater // x == 0) and (greater % y == 0)): # mutation performed here
            lcm = greater
            break
```

```
def suite1_mutant2(x, y):
    lcm = 0
    greater = 0
    if x <= y: # mutation performed here
        greater = x
    else:
        greater = y
    while(True):
        if((greater % x == 0) and (greater % y == 0)):
            lcm = greater
            break
```

Suite 2:

```
def suite2_mutant1(x, y):
    lcm = 0
    greater = 0
    if x > y:
        greater = x
    else:
        greater = y
    while(True):
        if(not (greater % x == 0) and (greater - y == 0)): # mutation performed here
            lcm = greater
            break
```

```
def suite2_mutant2(x, y):
    lcm = 0
    greater = 0
    if x > y:
        greater = x
    else:
        greater = y
    while(True):
        if((greater * x == 0) and (greater % y == 0)): # mutation performed here
            lcm = greater
            return lcm # mutation
```

When providing an answer, please answer either:

- `N/A` if no test case exists.
- If a test suite exists, specify it with pairs of x,y on each line:

```
x1,y1
x2,y2
...
xn,yn
```

Answer to mutation testing question here.

Now, consider the *candidate invariant* `lcm < 7`. As appropriate, provide one of the following:

- If the invariant `lcm < 7` is *falsified* by *any one* of the mutants above, indicate which test case and explain why.
- If the invariant `lcm < 7` is *not falsified* by one of the test suites above, but *is falsified* by a *single-order mutant* of the original `lcm` function, describe the line number and mutation that falsifies the candidate invariant, and explain why.
- If the invariant `lcm < 7` is *not falsifiable*, explain why.

LATE

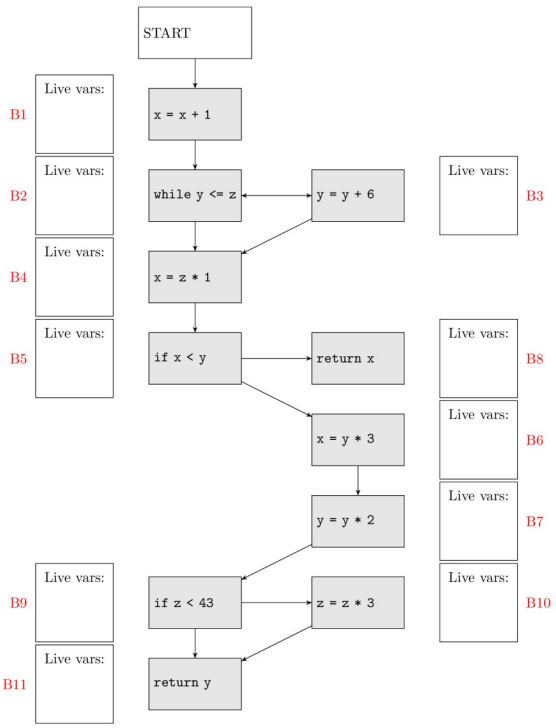
minutes remaining

Hide Time

Answer to candidate invariants question.

Question 5: Dataflow Analysis

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



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.

B1	B2	B3	B4
B5	B6	B7	B8
B9	B10	B11	

Question 6: Extra Credit.

LATE

minutes remaining

Hide Time

Each of the following questions are worth 1 point each. We are very strict about reading-related questions.

(*Feedback*) What is one thing you like about this class?

What is one thing you like about this class?

Feedback) What is one thing you dislike about this class?

What is one thing you dislike about this class?

(*Optional Psych*) Describe (or make up) a software engineering scenario in which *confirmation bias* may impact software quality.

Confirmation bias

Optional Reading 1) Identify a single optional reading. Write a sentence about it that convinces us you read it critically.

Optional Reading 1

(*Optional Reading 2*) Identify another single optional reading. Write a sentence about it that convinces us you read it critically.

Optional Reading 1

(*Statistics*) What browser and OS are you using?

Tell me about your browser.

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.*

Note that your submission will be marked as late. You can still submit, and we will retain all submissions you make, but unless you have a documented extenuating circumstance, we will not consider this submission.

Submit My Exam

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