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. — Coupling Effect Hypothesis	F. — Feature Request	G. — Formal Code Inspection	H. — Fuzz Testing
I. — Instrumentation	J. — Integration Testing	K. — Invariant	L. — Milestone
M. — Mocking	N. — Pair Programming	O. — Passaround Code Review	P. — Perverse Incentive
Q. — Race Condition	R. — Regression Test	S. — Severity	T. — Spiral Development
U. — Streetlight Effect	V. — Test-driven Development	W. — Traceability	X. — Triage
Y. — Unit Testing	Z. — Waterfall Model		

Q1.1:

O

Before a proposed change can be merged into the main branch, at least 2 other developers need to look over and sign off on the changes. This process happens without much preparation; once a change is ready to be reviewed, it can be done so at developers leisure.

Q1.2:

U

LolTube's software has two components: a single-threaded program and a multi-threaded program. LolTube's testers find more bugs in the single-threaded program than in the multi-threaded program because it is easier to write tests for the single-threaded program.

Q1.3:

P

Chase, the project manager for the app development team at ElbowBump, announces that the person who resolves the most bug reports in a month would get a large bonus and promotion. This, however, leads to developers racing to clear bug reports without actually solving issues consumers are having with the app.

Q1.4:

A

Tanvas is a university management system that students use to submit assignment. They are considering adding a new feature that shows confetti to students who submit their assignments early. To evaluate whether students enjoy the new feature, they release it to half of the students, but not the other half. They also use a survey to gauge satisfaction with the candidate feature.

Navigation

Q1.5:

J

Hodaka is writing a weather prediction app that contains a web-based front-end, a payment system, and a back-end for storing user data. He has written many tests for each individual component. However, the app fails when all the components are brought together. If only Hodaka had listened to his boss, Hina, and used *this* to validate the whole app.

Q1.6:

R

After creating major changes to the code base, Herman notices that previous issues that they had resolved are showing up again. They resolve these issues and add tests such that, in the future, these issues will not arise again without alerting the developer.

Q1.7:

Z

Meimei is the new manager of a mobile app development team and has decided to apply a different process for managing effort. They will first gather requirements from customers, then design and implement the app. Only after all of the app is developed, will they test functionality.

Q1.8:

G

Every month at Whoosh Video Conferencing, the entire team gathers around to thoroughly examine all of the code to identify and discuss issues.

Q1.9:

X

Roberto's job is to assign developers to work on issues reported by customers. Typically, Roberto assigns an engineer to work on security defects within a week, but often leaves feature requests unassigned for months.

Q1.10:

C

Jaxon is developing a mobile app called Buc-ee's that finds clean public bathrooms for you. After internal testing, Jaxon releases the first prototype to a group of users who frequently roadtrip to try out the app.

Q1.11:

B

After finishing a prototype, teams at Watchflix provide the prototype to an internal Quality Assurance team to identify early issues with the prototype.

Q1.12:

L

Taki is writing Virtual Reality software for Mitsuha. They agree upon a set of measurable outcomes for the software that will be delivered at the end of each year in their three-year contract.

Navigation

- Question 1
- Question 2
- Question 3
- Question 4
- Question 5
- Question 6
- Extra Credit

Q1.13:

Y

For each function Elliot has written, they have also written test cases to ensure each individual function works as intended.

Q1.14:

D

Even though they were only passing 1 out of 10 test cases on the autograder, Bepis's code was mostly correct except for one typographical mistake.

Question 2: Coverage (20 points total)

You are given the following C functions. For simplicity, assume that `bool` means an integer 0 for False, and 1 for True. Further assume that statement coverage applies only to statements marked `STMT_#`; ignore all other statements from consideration when computing coverage.

```
1  void Blandroid(int a, int b, int c) {
2      if (wake(a, b)) {
3          return;
4      }
5      wave(a, b, c);
6  }
7
8  int wake(int a, int b) {
9      if (a < b) {
10         STMT_1;
11     } else if (a < b) {
12         return 0;
13     }
```

Q2.1 (2 points) Suppose you can choose one set of inputs (A, B, C) for the function invocation `Blandroid(A, B, C)`. True or False: *it is possible to select one set of inputs (A, B, C) to achieve 82% or greater statement coverage over the code above.* Assume that covering all of `STMT_1` through `STMT_5` counts as 100%.

○ True ○ False

ANSWER: F. There are 5 statements above; therefore, each of them covered increases statement coverage by 20%. We are asking you whether an input exists that provides more coverage than that stated.

Q2.2 (5 points) Provide values for inputs `A`, `B`, and `C` that achieves the *highest* statement coverage possible for this file by executing `Blandroid(A, B, C)`. Again, assume we only count coverage of `STMT_1` through `STMT_5`.

A

A

B

B

C

C

ANSWER: Answers will vary. Recall that each STMT_# covered increases statement coverage by 20%. Many answers apply. Note that we intended for you to provide answers with 0 or 1 as input, and this answer key reflects as much (it is possible to

exhaustively enumerate all 8 input combinations). *HOWEVER*, we gave you points if you provided an answer that used values besides 0 and 1. Example Answer: `['0', '1', '0']`

Q2.3 (3 points) Support or refute: the test input you provided above achieves the lowest possible path coverage for the given code snippet.

Navigation

- Question 1
- Question 2
- Question 3
- Question 4
- Question 5
- Question 6
- Extra Credit

Support or refute: the test input you provided above achieves the lowest possible path coverage for the given code snippet.

Answers may vary. However, we were looking for an understanding of path coverage with respect to a single input. A single input exercises a single path through the program. Thus, the input achieves the lowest possible path coverage (i.e., a single path).

Q2.4 (10 points total, 1 point per selection) Next, consider the code below. Make selections for the operators `P`, `Q`, `R`, `S`, and `T` such that:

1. The number of *paths* in the function `sharp_keller` is maximized.
2. The *path coverage* induced by executing the single test case `sharp_keller(1, 1, 0)` is maximized.

```
1  void sharp_keller (bool a, bool b, bool c) {
2      if      (a  •P•  b) STMT_1;
3      else if (b  •Q•  c) STMT_2;
4      else    STMT_3;
5
6      STMT_4;
7
8      if      (b  •R•  c) STMT_5;
9
10     STMT_6;
11
12     if      (b  •S•  c) STMT_7;
13     else if (a  •T•  b) STMT_8;
```

Operator	Maximize Number of Paths	Maximize Path Coverage
P	○ == ○ < ○ >	○ == ○ < ○ >
Q	○ == ○ < ○ >	○ == ○ < ○ >
R	○ == ○ < ○ >	○ == ○ < ○ >
S	○ == ○ < ○ >	○ == ○ < ○ >
T	○ == ○ < ○ >	○ == ○ < ○ >

ANSWER: Multiple answers apply. Based on our description of path coverage, the number of paths possible sets a maximum for the number of paths coverable. Thus, if you want to maximize path covergae with a single input, you need the *fewest* number of paths. In contrast, you can also maximize the number of paths by carefully selecting operators for P through T. In particular, setting the same operators for P and T reduces the number of paths (because they evaluate exactly the same condition). Similarly, setting different operators for all of Q, R, and S increases the total number of paths. Thus, correct answers involve setting either the same (to maximize single-input path coverage) or different (to maximize total number of paths) operators for P through T where they involve the same set of variables.

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

Provide answers to each question below.

Navigation

(6 points) Q3.1 Dynamic analysis and instrumentation.

Suppose you instrument a program to collect how long each function takes to run. The program contains 3 functions: `foo`, `bar`, and `qux`. Your instrumentation *only* records the average time elapsed between the start and end of every function. Assume for this question that the program executes `foo` exactly one time (i.e., that `main` immediately calls `foo`).

Running your instrumented program provides the following information for one run of the program:

Cumulative Time for **`foo`** (ms)	Cumulative Time for **`bar`** (ms)	Cumulative Time for **`qux`** (ms)
465	3	90

Based on the information above, explain the order in which `foo` could have invoked `bar` or `qux` to produce the data above. Multiple answers may apply. Use three sentences or fewer. Assume functions are not recursively called. *Hint: `qux` may call `bar` or vice versa; `qux` and `bar` may call each other multiple times.*

For example, if you believe that `foo` calls `bar` two times, and that `bar` calls `qux` once, you could explain how that sequence of function calls produces the data observed in the table above. There is no specific syntax here — do your best to explain and justify the order in which the functions could be called to produce the data above.

Your answer here.

ANSWER: This is a question about instrumentation for dynamic analysis. The question explains a type of analysis involving the collection of function invocations and timings. The question statement indicates that if `foo` calls `bar`, then the timing measurement for `foo` must also include the time elapsed in `bar`. As a result, the data you are given involves some amount of overlap. Recall from lecture the importance of understanding the difference between the raw data you can collect (e.g., total time elapsed per function) and the analysis you might care about (e.g., the structure of function calls). We graded generously here — we were looking for a reasonable explanation for a sequence of function calls.

Q3.2 (2 points each; 6 points total)

Read the scenario statements below. For each statement, choose the best combination of Task, Goal, and Target by taking one selection from each column in the table below. For example, `a 1 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 the class or the readings. You may use an option more than once across multiple scenario statements.

For example, given the statement: *You use a tool to enumerate thread interleavings in your multithreaded program to create tests that expose race conditions.* then you might answer "Automated dynamic analysis", "Generate", "Unit tests" if you think that scenario is best addressed by applying an automated instrumentation such as CHESS to generate test cases that reveal concurrency issues.

Multiple answers may apply.
(Q3.2.1) You write a mutation testing framework as in Homework 3.
Task:

Navigation

- Question 1
- Question 2
- Question 3
- Question 4
- Question 5
- Question 6
- Extra Credit

- (a): Automated static analysis
- (b): Manual dynamic analysis
- (c): Automated dynamic analysis
- (d): Software engineering process
- (e): Manual static analysis

Goal:

- (1): Validate
- (2): Generate
- (3): Transform
- (4): Maximize
- (5): Minimize

Target:

- (m): Integration tests
- (n): Control flow graph
- (o): Abstract syntax tree
- (p): Program source code
- (q): Unit tests

(Q3.2.2) You write a tool that discovers locations in code that unsafely use buffers.

Task:

- (a): Automated static analysis
- (b): Manual dynamic analysis
- (c): Automated dynamic analysis
- (d): Software engineering process
- (e): Manual static analysis

Goal:

- (1): Validate
- (2): Generate
- (3): Transform
- (4): Maximize
- (5): Minimize

Target:

- (m): Integration tests
- (n): Control flow graph
- (o): Abstract syntax tree
- (p): Program source code
- (q): Unit tests

(Q3.2.3) You use a compiler that examines program structure to report syntax errors.

Task:

- (a): Automated static analysis
- (b): Manual dynamic analysis
- (c): Automated dynamic analysis
- (d): Software engineering process
- (e): Manual static analysis

Goal:

- (1): Validate
- (2): Generate
- (3): Transform
- (4): Maximize
- (5): Minimize

Target:

Navigation

- Question 1
- Question 2
- Question 3
- Question 4
- Question 5
- Question 6
- Extra Credit

- ○ (m): Integration tests
- ○ (n): Control flow graph
- ○ (o): Abstract syntax tree
- ○ (p): Program source code
- ○ (q): Unit tests

ANSWER: Several answers could apply for each situation. Correct answers are highlighted in red above (your answers are marked with the dot you marked).

Q3.3 (5 points per correct row, 10 points total) Consider the following code examples taken from a binary tree implementation with containing defects. Of the options shown, explain what the defect is, or indicate None if no defect exists. Then, select which QA method(s) is/are most likely to reveal the defect or ensure the code works as intended.

Code snippet 1

```
1 def children (root):
2     # Returns any children of a binary tree
3
4 if root.right:
5     children
6 else:
7     return root.value
8
9
```

The code above may have a defect. It could be an entirely wrong implementation (e.g., it computes something different from what the comment indicates), or it could be a single syntactic problem (e.g., missing a symbol). Describe what you think the defect is. If you do not believe a defect is present, mark the space with None.

Descibe the defect, if present. Otherwise, indicate "None"

Now, consider each QA approach below. Mark the option(s) that you think would reveal the defect (or leave them blank if you think no defect exists).

- ☐ 100% Coverage
- ☐ EvoSuite
- ☐ Pex-like
- ☐ Pair Programming
- ☐ Formal Code Inspection

ANSWER: There are two components to this answer. 3 points were given for the short answer. We gave 2/5 (i.e., 0.4) points per correct checkmark. The answers are given below.

- (Short Answer): `Code does not adhere to spec`
- (Full testing?): no
- (Evosuite?): no
- (Pex-like?): no
- (Pair-programming?): yes
- (Formal Code Inspection?): yes

Code snippet 2

```
def search(root, value):
    if root.value == value:
```

```
        return true
    elif root.val != value:
        return search(root.right, value)
    else:
        return search(root.left, value)
```

Navigation

The code above may have a defect. It could be an entirely wrong implementation (e.g., it computes something different from what the comment indicates), or it could be a single syntactic problem (e.g., missing a symbol). Describe what you think the defect is. If you do not believe a defect is present, mark the space with `None`.

Descibe the defect, if present. Otherwise, indicate "None"

Now, consider each QA approach below. Mark the option(s) that you think would reveal the defect (or leave them blank if you think no defect exists).

- ☐ 100% Coverage
- ☐ EvoSuite
- ☐ Pex-like
- ☐ Pair Programming
- ☐ Formal Code Inspection

ANSWER: There are two components to this answer. 3 points were given for the short answer. We gave 2/5 (i.e., 0.4) points per correct checkmark. The answers are given below.

- (Short Answer): `Incorrect operator`
- (Full testing?): `yes`
- (Evosuite?): `no`
- (Pex-like?): `yes`
- (Pair-programming?): `no`
- (Formal Code Inspection?): `yes`

Q3.4 (2 points each; 6 points total) Pair programming; 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 42%. The table below summarizes the various costs and benefits of using pair programming for each task.

Task	Total Hours	Cost Per Hour	Pair Programming decrease in Total Hours (%)	Pair Programming increase in Cost per Hour (%)
A	28	15	25	22
B	15	8	34	100
C	21	17	82	9

(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

Navigation

Question 4. Mutation Testing (17 points)

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

```
1  def foo(a, b, c):
2      bar = -1
3      if (c >= 17):
4          bar += 4
5      elif (c < 19 and 2 > a):
6          bar -= 4
7      if (a >= b or c == 0):
8          if (b == 0):
9              bar = 0
10         else:
11             bar = bar + 4 * (b // 5)
12     if (c == 15 or b == 2):
13         bar = 15
```

Given test input `(9,8,4)`, the function produces an output of `3`.

(a) (4 points per mutant, 12 total) Make at most one edit each to create three mutants of foo such that the kill score of the suite of three mutants, when provided the same test input, is 2/3.

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 single-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 1:

```
1  def foo(a, b, c):
2      bar = -1
3      if (c >= 17):
4          bar += 4
5      elif (c < 19 and 2 > a):
6          bar -= 4
7      if (a >= b or c == 0):
8          if (b == 0):
9              bar = 0
10         else:
11             bar = bar + 4 * (b // 5)
12     if (c == 15 or b == 2):
13         bar = 15
```

Mutant 2:

```
def foo(a, b, c):
    bar = -1
    if (c >= 17):
```

Navigation

- Question 1
- Question 2
- Question 3
- Question 4
- Question 5
- Question 6
- Extra Credit

```
        bar += 4
    elif (c < 19 and 2 > a):
        bar -= 4
    if (a >= b or c == 0):
        if (b == 0):
            bar = 0
        else:
            bar = bar + 4 * (b // 5)
    if (c == 15 or b == 2):
        bar = 15
```

Mutant 3:

```
def foo(a, b, c):
    bar = -1
    if (c >= 17):
        bar += 4
    elif (c < 19 and 2 > a):
        bar -= 4
    if (a >= b or c == 0):
        if (b == 0):
            bar = 0
        else:
            bar = bar + 4 * (b // 5)
    if (c == 15 or b == 2):
        bar = 15
```

(5 points) Support or refute in four sentences or fewer: It is easier to kill higher order mutants than it is to kill single order mutants, therefore mutation testing suites should focus on higher order mutations.

Write your answer here.

Multiple answers may apply. We gave points by actually executing the mutants you provided. For part (b), Refute. The expectation here was that students would talk about what happens if all mutants are higher order and keep getting killed by test suites. What does that test us about the quality of the test suites? E.g., we might not be able to distinguish between low and high quality suites if all mutants are getting killed by all suites. Is the purpose of mutation testing really to kill mutants as quickly as possible? References to the competent programmer hypothesis, couple effect also received credit.

Question 5. Mutation Analysis and Invariants (10 points)

You are given the following snippet of C++ code:

```
int fulp(int a, int b, bool c) {
    int x = a + b;
    int y = a * b;
    int z = 0;
    for(int i = 0; i < a*b; i += a) {
        if ( i == a && c) {
            continue;
        }
        z++;
        x += a + b;
        y += b;
      //Evaluate invariants here
    }
```

Navigation

An oracle generates the following invariants for the snippet:

```
INV_1: x < y
INV_2: z < a
INV_3: c || (z > 0)
```

The invariants are evaluated at the end of each iteration BEFORE the loop increment. For each invariant, provide a set of inputs that violates the invariant or indicate that the invariant is valid. The range of the integer inputs is restricted to positive integers (`> 0`). Booleans can be `true` or `false` (please note they are all lowercase — it will help our grading). Express your answer as a Python dictionary.

For example, if you believe that `INV_1` is valid and that `INV_2` can be invalidated with inputs `[1, 3, true]`, then you would write: `{"INV_1": true, "INV_2": [1, 3, true]}` (again, note the brackets for expressing the inputs as a list).

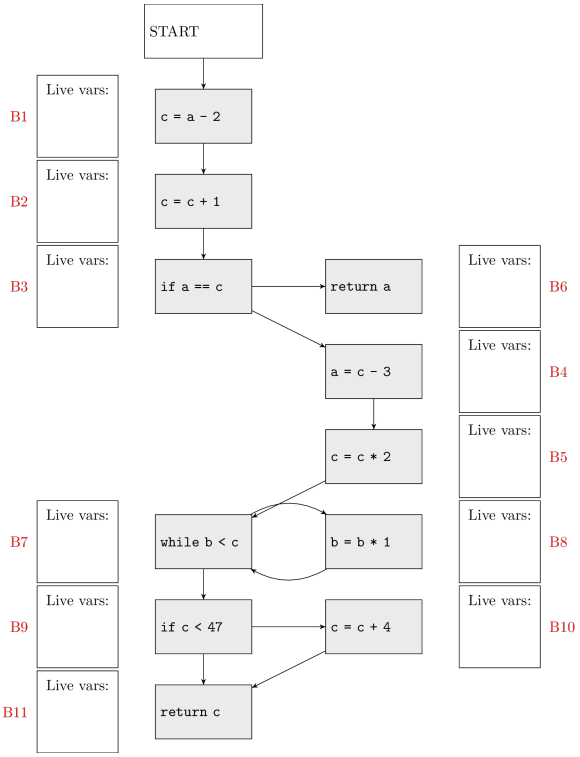
```
1
```

We gave 2 points for identifying each valid/invalid invariant (3 invariants; 1 was valid, 2 were invalid). We gave an additional 2 points for providing inputs that invalidated the invariant. We did our best to correct syntactically-incorrect answers.

Navigation

Question 6: Dataflow Analysis (11 points total)

Consider a *live variable dataflow analysis* for three variables, `a`, `b`, and `c` 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 `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).



(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., `ab` to indicate that `a` and `b` are alive before that block.

B1 ANSWER: {'b', 'a'}

B2 ANSWER: {'b', 'a', 'c'}

B3 ANSWER: {'b', 'a', 'c'}

B4 ANSWER: {'b', 'c'}

B5 ANSWER: {'b', 'c'}

B6 ANSWER: {'a'}

B7 ANSWER: {'b', 'c'}

B8 ANSWER: {'b', 'c'}

B9 ANSWER: {'c'}

B10 ANSWER: {'c'}

B11 ANSWER: {'c'}

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.

Navigation

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

Your answer here.

(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.