1. (1pt) **Name:** _____

INSTRUCTIONS: Carefully read each question, and write the answer in the space provided. If answers to free response questions are written obscurely, zero credit will be awarded. The correct answer to a free response question with a short answer (i.e., one word or phrase) will never contain any significant words used in the question itself (i.e., "crossword rules"). You are permitted to use one 8.5x11 inch sheet of paper (double-sided) containing **hand-written** notes; all other aids (other than your brain) are forbidden. Questions may be brought to the instructor.

You have **80 minutes** to complete the exam.

For **TRUE** or **FALSE** and multiple choice questions, circle your answer.

On free response questions only, you will receive **20%** credit for any question which you leave blank (i.e., do not attempt to answer). Do not waste your time or mine by making up an answer if you do not know. (Note though that most questions offer partial credit, so if you know part of the answer, it is almost always better to write something rather than nothing.)

To get credit for this question, you must:

- Print your name (e.g., "Martin Kellogg") in the space provided on this page.
- Print your UCID (e.g., "mjk76") in the space at the top of **each** page of the exam.

Contents (blanks for graders only):

Writing your name and UCID:	1 / 1
I. Very Short Answer:	20 / 20
II. Matching:	20 / 20
III. Short Answer:	26 / 26
IV. "Your Choice" Reading Quiz	3 / 3
V. DBQs:	30 / 30
VI. Extra Credit:	x / 0
Total:	10x / 100

Clarifications posted on the screen during the exam:

- you may rip any pages out of the exam
- don't forget to put your ucid on every page with questions!. ucid = your email address, e.g. mjk76. It is not a number starting with 31...
- re:question 22: "subsume" means "contain". With respect to coverage, X coverage is said to subsume Y coverage iff 100% X coverages implies 100% Y coverage.
- you may turn in your exam as soon as you are finished
- "support or refute" means both say whether you agree with the claim or not, and explain why (note, you must choose one or the other, not both)
- in part II (matching), the key will not use any of the options more than once. You are permitted to use an option more than once if you'd like, though (but you'll definitely not get full credit...)
- in question 13, "sampled data" = "sampled inputs and outputs"
- technical interviews like the one in question 28 typically take between 45 minutes and an hour
- as you turn in your exam: you must turn in your cheat sheet. Make sure your name or UCID is on it. But your exam + cheatsheet in the bag. Take a piece of candy from the other bag. Enjoy halloweekend! (If you don't have a cheatsheet, write "no notes" on the cover sheet of your exam - i.e., where you wrote your name)

I. Multiple Choice and Very Short Answer (20pts). In the following section, either circle your answer (possible answers appear in **bold**) or write a very short (one word or one phrase) answer in the space provided. No partial credit is possible in this section.

2. (2pt) Which of the following are examples of static analyses (select all that apply)?
   - **A** linters
   - **B** build systems
   - **C** type systems
   - **D** code review

   Partial credit (1 point) for AC only on question 2.

3. (2pt) **TRUE** / **FALSE**: it is a good idea to assume that in any system with a large number of users, someone relies on every behavior of the system (intended or not) as if it were a feature.

4. (2pt) **Equivalent mutants** make interpreting an absolute mutation score (e.g., "80% of mutants killed") difficult.

5. (2pt) When a continuous integration incorrectly blocks a change (i.e., nothing in the changeset causes the CI failure), one of the most common causes is **flaky** tests, which fail nondeterministically.

6. (2pt) Modern version control systems like `git` use a **syntactic** / **semantic** analysis to decide whether two changes conflict.

7. (2pt) Which of the following are examples of quality requirements (select all that apply)?
   - **A** low cost to develop
   - **B** high availability
   - **C** easy to use
   - **D** no bugs

8. (2pt) Names for methods that return a value should be **verbs** / **nouns**.

9. (2pt) Because human attention spans are short, it's important to make sure that the edit-test-debug cycle for a project takes less than about ten **seconds**.
   - **A** milliseconds
   - **B** seconds
   - **C** minutes
   - **D** hours

10. (2pt) An advantage of which of the following is that applying it correctly guarantees that all behaviors have a regression test immediately (select all that apply)?
    - **A** test-driven development
    - **B** partition testing
    - **C** A/B testing

11. (2pt) **TRUE** / **FALSE**: When a code reviewer makes a suggestion that you disagree with, the appropriate response as a code author is to explain to the reviewer in the code review tool why they're wrong.

A. Implicit oracle	**B.** Mocking	**C.** Over-engineering	**D.** Triage
E. Formal specification	**F.** Feature request	**G.** Self-documenting code	**H.** User story
I. Differential testing	**J.** Opinionated	**K.** Dynamic type system	**L.** Greppability
M. Dataflow analysis	**N.** Hermeticity	**O.** Embarrassingly parallel	**P.** Pair programming

II. Matching (20pts). This section contains a collection of terms discussed in class in an "Answer Bank" (choices **A.** through **P.**). Each question in this section describes a situation associated with an answer in the Answer Bank. Write the letter of the term in the Answer Bank that best describes each situation. Each answer in the Answer Bank will be used at most once.

12. (2pt) **K: Dynamic type system** Alice is prototyping a new project, and values ease of implementation over speed or safety in her choice of language.

13. (2pt) **B: mocking** Thomas wants to do an integration test between his code and a system which is expensive to call, so he tests against sampled data from that system.

14. (2pt) **E: formal specification** Annie is writing code for a safety-critical system, so she writes a model of her system's behavior to detect design issues.

15. (2pt) **P: pair programming** Bud and Lou are optimizing for code quality instead of programming speed.

16. (2pt) **O: embarrassingly parallel** Frank realizes that he can rewrite his program to independently solve many sub-problems on different CPU cores, because the sub-problems don't depend on each other.

17. (2pt) **J: opinionated** Clara wants to reduce inter-team arguments about style, so she chooses an automated formatter with few configuration options.

18. (2pt) **G: self-documenting code** Bruce puts extra effort into writing good variable names, because someone reading the code might not have the documentation handy.

19. (2pt) **M: dataflow analysis** Meryl uses a tool to detect potential null-pointer exceptions in her Java code.

20. (2pt) **D: triage** Buzz prioritizes the bugs he needs to fix by severity.

21. (2pt) **A: implicit oracle** Albert uses a fuzzer to detect crashes.

III. Short answer (26pts). Answer the questions in this section in at most three sentences.

22. *Condition coverage* is defined as the fraction of the possible values of boolean sub-expressions induced by a test suite ("fraction of possible values" here means that for each boolean sub-expression, `true` and `false` are counted separately). You may assume that we're interested in a typical imperative or object-oriented programming language, like C or Java (if you know why we might want to assume this, see extra credit question 30).

    (a) (2pt) **TRUE** / **FALSE**: Condition coverage subsumes branch coverage.

    (b) (3pt) Justify your answer to part (a) with either an informal mathematical argument (if you answered **TRUE**) or a counter-example (if you answered **FALSE**). If you answered part (a) incorrectly, you cannot get credit for this question.

    Full credit for arguing that each condition being true or false guarantees that branch coverage is 100%. No need to mention short-circuit evaluation for full credit here, although that would be needed for a real proof.

    (c) (2pt) **TRUE** / **FALSE**: Branch coverage subsumes condition coverage.

    (d) (3pt) Justify your answer to part (c) with either an informal mathematical argument (if you answered **TRUE**) or a counter-example (if you answered **FALSE**). If you answered part (c) incorrectly, you cannot get credit for this question.

    `if a or b: ...` is a counterexample: you only have to make one of a or b true to get 100% branch coverage, but you need to do both for 100% condition coverage.

23. (4pt) Support or refute the following claim: on a software engineering team, it's reasonable for the roles of "engineering manager" and "project manager" to be filled by the same person. **Likely refute. The strongest answers here will cite Joel Spolsky's "How to be a Program Manager" in their argument, and repeat his claim that this should not happen because it gives the PM too much power—the PM should be a peer of the engineers, so that the PM has to be *right* to be listened to. Support answers that mention that this is common in industry and sometimes works out will get partial credit.**

24. (4pt) Support or refute the following claim: when you need to fix a bug in an unfamiliar system, the first thing that you should do is read all of that system's source code to find the bug. **Likely refute. Undirected code reading is not an effective bug-finding technique, and real systems are usually too large for this.**

25. (4pt) Support or refute the following claim: referential transparency is an advantage of programming languages with a strong, static type system such as Rust. **Must refute: referential transparency is an advantage of functional programming languages. Full credit answers must both define referential transparency and explain why it's not guaranteed by a static type system—instead, it's a consequence of the way that functional programming models state.**

26. (4pt) Consider the following proposed software engineering task from a planning document (for example, a document like your group's project plan):

> **Sprint 1, Task 3**: Implementing class/functionality (1st half)
> **Task size**: Large
> Depending on how complex the features of the game are to implement, we are not sure as to how long it would take to finish all of the functions with several people.
> **Assignee**: Everyone

Support or refute the following claim: this task is appropriately detailed for a planning document. **Must refute. This task is not good in several ways: it's vague, it's not assigned to a specific person, there is no concrete deliverable ("half done"), etc. It's based on one that was submitted by a team-to-remain-anonymous in the initial project plan.**

A. *No Silver Bullet*
B. *Syntax, Predicates, Idioms—What Really Affects Code Complexity?*
C. *Relating Reading, Visualization, and Coding for New Programmers: A Neuroimaging Study*
D. *An Experimental Evaluation of Continuous Testing During Development*
E. *Taming Google-Scale Continuous Testing*
F. *The Oracle Problem in Software Testing: A Survey*
G. *The Daikon System for Dynamic Detection of Likely Invariants*
H. *Introduction to TLA*
I. *Purposes, Concepts, Misfits, and a Redesign of git*
J. *Variability and Reproducibility in Soft. Eng.: A Study of 4 Companies that Developed the Same System*
K. *Hiring is Broken: What Do Developers Say About Technical Interviews?*
L. *Expectations, Outcomes, and Challenges Of Modern Code Review*
M. *Hints on Programming Language Design*
N. *Build Systems à la Carte*
O. *Notes on Program Analysis*
P. *Designing the WhyLine: A Debugging Interface for Asking Questions about Program Behavior*
Q. *Locating Causes of Program Failures*

IV. "Your Choice" Reading Quiz

27. (3pt) The Answer Bank on this page lists the "Your Choice" readings so far in this semester.

    (a) Write the letter of the "Your Choice" reading for which you're answering this question: **depends on student**.

    (b) Write a reading quiz question for this reading, of similar difficulty and style as the reading quiz questions in class. You may write a question in any of the following styles: a TRUE/FALSE question, a multiple choice question with 3 or 4 answers, or a fill-in-the-blank question. There is an extra credit point for the best question suggested by anyone in the class. **Answers vary.**

    (c) What is the answer to your reading quiz question? **Varies, but must be correct for credit.**

V. Document-based Questions (30pts). All questions in this section refer to a documents **A-C**. These documents appear at the end of the exam (I recommend that you tear them out and refer to them as you answer the questions).

Questions on this page refer to **Documents A**, **B**, and **C**.

Suppose that you are an engineering manager at Amazing Stuff for Websites ("ASW"), a provider of cloud computing services. Your small team is responsible for maintaining and operating Some Specific Service ("S3"). You are hiring a new junior engineer to join your team; all candidates for this positions are "new-grads"—recent graudates from university computer science programs, with little or no professional engineering experience. One of your team's senior engineers has interviewed a promising candidate, and you need to decide whether to make an offer. The interview was conducted remotely, using a shared code editor (i.e., something like Google Docs but for code). The candidate chose to write code in Java.

Document A contains the interview script that the engineer used for this candidate. **Document B** contains the interviewing engineer's contemporaneous notes (i.e., the notes that they took during the interview); this document has been edited to include the candidate's code at appropriate points. **Document C** is the candidate's final code.

Note that in a real hiring situation like the one that this question presents, you'd have feedback from several senior engineers who conducted separate interviews, and you wouldn't want to make a decision based on just one engineer's notes. Moreover, the engineers would also give you their recommendations for whether to hire this candidate. Because this is an exam, we're focusing on just a single engineer's interview notes and making you make the decision without an explicit recommendation from the interviewing engineer.

28. (30pt) Make and justify a hiring decision (either "hire" or "don't hire") based on a coherent hiring philosophy. Your justification must cite at least three distinct pieces of evidence from the interview notes that support your decision. Your justification must also cite at least one piece of evidence that does *not* support your decision, along with an explanation of why you think this evidence is not important enough to change your decision. Phrase your answer as an email to the hiring committee by filling in the template on the next page.

```
TO: s3-hiring-committee@asw.com
FROM: you@asw.com
SUBJECT: re: Hiring decision for candidate [REDACTED]

Dear hiring committee,
```

The rubric is roughly: * 3 points for giving a clear answer * 4 points for email tone and professionalism * 3 points for having a high-level hiring philosophy. The common ways to get these points are either to say that this candidate is good but has room to grow, and we want to develop good people (pro), or that hiring should avoid false positives and this candidate made some mistakes (anti). * 5 points each for the 3 pro and 1 con, with partial credit if they're poorly expressed

Either answer here is acceptable with good justifications - this candidate does pretty well, but also clearly makes some questionable decisions, and a good argument here is much more important than the actual outcome. Here are sample full-credit answers for both positions.

I recommend that we hire this candidate: while they were somewhat flustered at times and can improve some of their communication skills, they have the technical ability that we need and seem coachable. In particular, the candidate displayed good technical abilities: they showed good knowledge of algorithmic complexity, thought about important engineering concepts like testing, and perhaps most importantly got to the correct answer. They also avoided prematurely optimizing their solution by focusing on a "brute force" solution first. The candidate also took feedback from the interviewer well, showing that they can accept criticism from senior engineers: an important demonstration of teachability that bodes well for their future development, because code reviews involve similar sorts of feedback. Finally, I was impressed with the candidate's ability to perservere through adversity: they made a mistake that many junior candidates make (missing the ambiguity in the problem description), but they managed to gather themselves and still answer the question reasonably well. This candidate definitely can improve: they made a number of minor technical mistakes, didn't name their variables well, etc., but importantly these are problems that can be corrected through feedback and that might have been brought on by nerves. In summary, this is a promising candidate, and when hiring junior engineers we should focus on promise rather than how good the candidate is right now.

I recommend that we do not make this candidate an offer, in line with our hiring philosophy of avoiding false positive hiring decisions—there are too many red flags for me to be comfortable, despite the candidate's clear technical ability. First, the candidate shows an alarming tendency not to listen to senior engineers: for example, they ignored the "heavy hinting" about the ambiguity in the problem description, and they continued to focus on

the "deep-copy" issue even after they were told repeatedly that it was not relevant. Second, their technical work was sloppy—even though they got to the right answer, they made a lot of little mistakes along the way (e.g., the null-pointer issue, the lists vs sets issue, the very poor variable names...), which shows a lack of attention to detail. Finally, the candidate seemed too coached: I'm worried that the good things that we noticed might have been because they practiced a lot of interviewing, and not because they're actually a good candidate. All that said, there does seem to be some technical promise here—they knew their big-oh notation, they did get the right answer, and so on—so if the committee would prefer a hire I won't veto it.

Sincerely,

You

VI. Extra Credit. Questions in this section do not count towards the denominator of the exam score.

29. (1pt) In section II (Matching), there is a theme to the names used in the situation descriptions. What is the theme? **First names of members of the New Jersey Hall of Fame: Alice Paul, Thomas Edison, Annie Oakley, Abbott and Costello, Frank Sinatra, Clara Barton, Bruce Springsteen, Meryl Streep, Buzz Aldrin, and Albert Einstein. Any reasonable answer involving "New Jersey" gets full credit. If you named at least two of the people but got it wrong, half-credit.**

30. Question 22 specifies that the language of interest is a typical imperative or object-oriented programming language, like C or Java. There is a language feature that these languages have that is relevant to the answers for question 22.

    (a) (1pt) What is the language feature that these languages have that is relevant to the question? **Short-circuit evaluation.**

    (b) (1pt) How would the answers to question 22 change if this language feature were not present? **Condition coverage would not subsume branch coverage. For example, `if a and b` could achieve condition coverage with (true, false), (false, true) for the values of a and b, but the `if` would never be triggered.**

31. (2pt) Write a reading quiz question for another "Your Choice" reading, with the same requirements as question 27.

    (a) Write the letter of the "Your Choice" reading for which you're answering this question: **depends on student**.

    (b) Write your question in this space: **Answers vary.**

    (c) What is the answer to your reading quiz question? **Varies, but must be correct for credit.**

This page intentionally left blank (you may use it as scratch paper, and the proctor will have more scratch paper at the front if you need more).

Document A:

Phrase the question like this:

> Let's say we have a website and we keep track of what pages customers are viewing, for things like business metrics.
>
> Every time somebody comes to the website, we write a record to a log file consisting of Timestamp, PageId, CustomerId. At the end of the day we have a big log file with many entries in that format. And for every day we have a new file.
>
> Now, given two log files (log file from day 1 and log file from day 2) we want to generate a list of 'loyal customers' that meet the criteria of: (a) they came on both days, and (b) they visited at least two unique pages.

Give the candidate the following starter code (if they want to code in Java):

```
1 public Collection<CustomerId> getLoyalCustomers(Iterator<PageView> file1,
2                                                 Iterator<PageView> file2) {
3     Collection<CustomerId> loyalCustomersToReturn = // TODO: initialize this data structure
4     // TODO: Figure out your loyal customers
5     return loyalCustomersToReturn;
6 }
```

There is one important ambiguity in the problem description: is it two unique pages *per day* or *overall*. The answer is "2 unique pages *overall*", because that's a much more interesting problem. About half of candidates jump straight into coding without clarifying this, and out of those, about half will assume incorrectly that you meant "2 unique pages per day." If it's a more junior candidate, you should hint heavily before they start coding. If it's a more senior candidate, you should wait a bit and see if this comes up as they're thinking more deeply about the algorithm.

Document B:

After I introduced the problem, the candidate asked a few clarifying questions (good), but it seemed like they'd been coached—the questions were pretty generic. The best of them was "Do the log files fit in memory?" (yes); they also asked about constraints like a required run time and how much memory they're allowed to use. No good problem-specific questions, though. I hinted heavily about the per-day vs overall ambiguity, but the candidate wanted to start coding right away.

The candidate got to the code below before it was clear they'd misunderstood the question, and I corrected them about the "per-day" vs "overall" thing:

```
1 public Collection<CustomerId> getLoyalCustomers(Iterator<PageView> file1,
2                                                 Iterator<PageView> file2) {
3     Collection<CustomerId> loyalCustomersToReturn = new HashSet<>();
4     while (file1.hasNext()) {
5       PageView p1 = file1.next();
6       CustomerId c1 = p1.getCustomerId();
7     }
8     ...
9 }
```

They said aloud that they were then going to make a per-day table of how many unique pages a customer had seen using this structure, which is when I corrected them. At this point the candidate seemed flustered, and needed to sit silently for a minute or two before they started coding again. Even once they got started again, they seemed nervous for the next several minutes. They talked through their high-level plan as they typed it out, explaining to me that they were going to "brute force it" to "get a bad answer that works" first. Here's the code they came up with:

```
1  public Collection<CustomerId> getLoyalCustomers(Iterator<PageView> file1,
2                                                  Iterator<PageView> file2) {
3      Collection<CustomerId> loyalCustomersToReturn = new HashSet<>();
4      while (file1.hasNext()) {
5        PageView p1 = file1.next();
6        CustomerId c1 = p1.getCustomerId();
7        List<PageId> pages = new ArrayList<>();
8        pages.add(p1.getPageId());
9
10       // TODO: do I need to make a copy of this first?
11       while (file2.hasNext()) {
12         PageView p2 = file2.next();
13         CustomerId c2 = p2.getCustomerId();
14         if (c1.equals(c2)) {
```

```
15            pages.add(p2.getPageId());
16          }
17        }
18
19        if (pages.size() >= 2) {
20          loyalCustomersToReturn.add(c1);
21        }
22      }
23
24      return loyalCustomersToReturn;
25  }
```

The candidate volunteered that this is $O(n^2)$ and that they intended to try to optimize once they were sure it was working. I prompted them about how they were going to make sure it works, and they immediately started talking about testing. They mentioned checking for edge cases like empty log files (I told them we get enough traffic that they don't have to worry about that one), and they considered the possibility of a customer who only visits on one of the days unprompted, and convincingly argued to me that their current solution would handle that case correctly. The candidate also spent a few minutes agonizing over whether they needed to deep-copy the iterator, even though I told them not to worry about that twice—once while they were actually coding it the first time, and again when they brought it up after.

I prompted them about other edge cases and they couldn't think of any, so I mentioned duplicate page views explicitly. The candidate realized the problem with using a list almost immediately after I mentioned that, and changed their `pages` variable's type to `Set`.

At this point, the candidate started trying to optimize. They pretty quickly noticed that their initial prototype is doing quite a few redundant operations when searching through the second day's log, and decided they needed some kind of data structure to speed up searching for a particular customer. They decided on a lookup table, with the customer id as the key, pretty quickly and set to implementing that. Here's the code that they got to, after one false start where they forgot that the values in the map need to be sets of `PageId`s rather than just a single `PageId`:

```
1  public Collection<CustomerId> getLoyalCustomers(Iterator<PageView> file1,
2                                                  Iterator<PageView> file2) {
3      Collection<CustomerId> loyalCustomersToReturn = new HashSet<>();
4
5      Map<CustomerId, Set<PageId>> map = new HashMap<>();
6      while (file2.hasNext()) {
7        PageView p2 = file2.next();
8        CustomerId c2 = p2.getCustomerId();
```

```
9       if (map.containsKey(c2) {
10        map.get(c2).add(p2.getPageId());
11      } else {
12        map.put(p2.getCustomerId(), p2.getPageId());
13      }
14    }
15
16    while (file1.hasNext()) {
17      PageView p1 = file1.next();
18      CustomerId c1 = p1.getCustomerId();
19      Set<PageId> pages = map.get(c1);
20      pages.add(p1.getPageId());
21
22      if (pages.size() >= 2) {
23        loyalCustomersToReturn.add(c1);
24      }
25    }
26
27    return loyalCustomersToReturn;
28  }
```

I noticed that the types don't line up right away, but the candidate didn't seem to. They started thinking about testing again, and I hinted that they should try working through a simple example with this new code first. Once they did that, they noticed that they'd forgotten to actually initialize the inner sets of the map, and they revised the code in the first **while** loop:

```
1     while (file2.hasNext()) {
2       PageView p2 = file2.next();
3       CustomerId c2 = p2.getCustomerId();
4       if (map.containsKey(c2) {
5         map.get(c2).add(p2.getPageId());
6       } else {
7         HashSet<PageId> set = new HashSet<>();
8         set.add(p2.getPageId());
9         map.put(p2.getCustomerId(), set);
10      }
11    }
```

I encouraged them to run through the test cases they'd considered before, but they must have forgotten about the case where a customer is present on one day and not the other, because it took a bit of prompting

from me for them to get that their implementation will dereference a null pointer if that happens - I had to hint about the specific case. But, they did fix it eventually. We then had a quick conversation about the run time of this solution (they correctly said it's $O(n)$ but didn't mention that it uses more memory) before we had to wrap up for time.

Document C

```
1  public Collection<CustomerId> getLoyalCustomers(Iterator<PageView> file1,
2                                                  Iterator<PageView> file2) {
3      Collection<CustomerId> loyalCustomersToReturn = new HashSet<>();
4
5      Map<CustomerId, Set<PageId>> map = new HashMap<>();
6      while (file2.hasNext()) {
7        PageView p2 = file2.next();
8        CustomerId c2 = p2.getCustomerId();
9        if (map.containsKey(c2) {
10         map.get(c2).add(p2.getPageId());
11       } else {
12         HashSet<PageId> set = new HashSet<>();
13         set.add(p2.getPageId());
14         map.put(p2.getCustomerId(), set);
15       }
16     }
17
18     while (file1.hasNext()) {
19       PageView p1 = file1.next();
20       CustomerId c1 = p1.getCustomerId();
21       Set<PageId> pages = map.get(c1);
22       pages.add(p1.getPageId());
23
24       if (pages.size() >= 2) {
25         loyalCustomersToReturn.add(c1);
26       }
27     }
28
29     return loyalCustomersToReturn;
30 }
```