1. (10pt) **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 are required to turn in your notes with your exam; they will be returned to you with your graded exam.

You have **80 minutes** to complete the exam. You may turn in your exam as soon as you are finished. There are **800 points** available on the exam, and most questions are worth a number of points that is divisible by ten. These point values are a rough guide to how long I think you should spend on each question.

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.)

You may rip out any page in this exam. If a page has questions on it, make sure your UCID is on it before you separate it from the rest of your exam.

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 top-right of **each** page of the exam with a question on it.

Contents (blanks for graders only):

Writing your name and UCID:	**10** / 10
Pages 2 and 3:	**310** / 310
Page 4 and 5:	**180** / 180
Page 6 and 7:	**300** / 300
Extra Credit:	**x** / 0
Total:	**800** / 800

I. Multiple Choice and Very Short Answer (150pts). 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. (20pt) The algorithm used by a modern, distributed version control system like Git for detecting conflicts between different changes can have:
   A false positives
   B false negatives
   C both false positives and false negatives
   D neither false positives nor false negatives

3. (20pt) As a test quality metric, **mutation score** is difficult to interpret in isolation, in part, because checking for equivalence between programs is undecidable.

4. (20pt) Many students' frustrations during IP0 were caused by Covey.Town's build not being **hermetic**.

5. Consider the following situations where we might apply delta debugging. Recall that delta debugging relies on three assumptions about the *Interesting* function: it must be monotonic, unambiguous, and consistent. For each situation below, circle the assumption(s) that are **violated** in that situation.

   (a) (30pt) A video game's rendering engine has a bug that causes visual artifacts. The truth, which the developers don't know, is that there are two independent causes: 1) Having "Shadows" set to "High" AND "Reflections" set to "Ultra", OR 2) Having "Anti-Aliasing" set to "MSAA" AND "Texture Filtering" set to "Anisotropic 16x". A developer tries to find the minimal set of settings (from a large list) that triggers the bug.

   Monotonic **Unambiguous** **Consistent**

   (b) (30pt) A static webpage renders incorrectly (e.g., a button is misaligned). The developer suspects a rule in a single, large .css file. They use delta debugging to find the minimal set of CSS rules from that file that still causes the misalignment. The "Interesting" test is an automated visual regression test that takes a screenshot and compares it to a "correct" version.

   Monotonic **Unambiguous** **Consistent**

   (c) (30pt) An application fails to start if its .xml configuration file is malformed. A developer has a large, failing config file and wants to find the minimal set of lines that causes the failure. The "Interesting" function is a script that prints the selected lines into a new file and feeds it to the application.

   Monotonic **Unambiguous** **Consistent**

A. beta test	**B.** Rice's theorem	**C.** waterfall	**D.** A/B test	**E.** regression test
F. behavioral	**G.** object-oriented	**H.** functional	**I.** logging	**J.** mocking
K. agile	**L.** imperative	**M.** quality	**N.** fuzzing	**O.** over-engineered

II. Matching (160pts). This section contains a collection of terms discussed in class in an "Answer Bank" (choices **A.** through **O.**). 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.

6. (20pt) **E: regression test** After William fixes a bug, he ensures that that specific bug won't reoccur.
7. (20pt) **H: functional** Because destructive updates are difficult to reason about, Phil chooses a programming language that yields new similar states over time.
8. (20pt) **M: quality** Harry carefully gathers requirements of this type from his customers, because he knows that to deliver a working system on time he will have to compromise on some of them.
9. (20pt) **B: Rice's theorem** Chris knows that perfect static analysis is undecidable, so he decides to use a sound but imprecise analysis, because he is working in a safety-critical domain.
10. (20pt) **D: A/B testing** While working on a web application, Woodrow deploys a change to a small percentage of traffic to test whether it impacts user behavior.
11. (20pt) **I: logging** To debug a failure in his large distributed system, Jon relies on information about the program's internal state during an outage.
12. (20pt) **K: agile** George's team strives to always have a working prototype of their system.
13. (20pt) **F: behavioral** When he interviews candidates, Richard asks them questions that are designed to test their communication ability, not just their technical skills.

III. Short answer (150pts). Answer the questions in this section in at most three sentences.

14. (50pt) Suppose that your team has decided to rewrite a Java codebase into Rust to avoid garbage-collection stalls. Describe an approach, technique, or tool that we discussed in class that you could use to increase your confidence that the rewrite will succeed. **I'm looking for differential testing.**

15. (50pt) Suppose that you are a software engineer at well-funded startup. Your boss tells you that in one week, you'll have a big demo for a potential customer. This customer would be your biggest yet, by a fair margin, so it's important that the demo goes well. Fortunately, however, you'll be the one operating your system during the demo. One of your coworkers suggests that you should use a sound static analysis to *prove* that the demo will succeed. Do you agree with this coworker? Why or why not? **Must disagree: this is a bad use of a static analysis. Static analysis is useful when you need to reason about all possible executions of the program. In this case, though, you only care about the small subset of executions that will actually be exercised during the demo. So, you can exhaustively test those.**

16. (50pt) Support or refute the following claim: if you run your CI builds on one server in a specific timezone (e.g., all your CI servers are in California), it is safe to use a build system that incrementalizes using timestamps. **Must refute. There are still subtleties with e.g., daylight savings. More importantly, though, builds need to incrementalize properly not only on the CI server but also on your engineers' machines, which may move between timezones. Full credit answers must mention both issues.**

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. *Purposes, Concepts, Misfits, and a Redesign of git*
H. *Variability and Reproducibility in Soft. Eng.: A Study of 4 Companies that Developed the Same System*
I. *Hiring is Broken: What Do Developers Say About Technical Interviews?*
J. *The Daikon System for Dynamic Detection of Likely Invariants*
K. *Introduction to TLA*
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

17. (30pt) 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) Describe a software engineering situation in which you might use something you learned from the chosen "Your Choice" reading. To get credit for this question, you must both describe such a situation and clearly explain what you learned. If the "thing you learned" is so simple that one could understand it from the title of the reading alone, you won't get credit. **Answers vary.**

V. Document-based Questions (300pts). All questions in this section refer to **Document A**, which we recommend you tear out of the exam and read before trying to answer the questions in this section. **Document A** is a blog post from an engineer about why he doesn't always do code reviews for pull requests with a lot of AI-generated code, in the open-source project that he works on in his spare time. We have redacted some parts of the document. Your job is to use your knowledge of what makes a good code review to fill in the blanks.

For five of the six "Reasons for no CR" listed in the document, argue from your understanding of the benefits of code review why or why not that reason is a valid one for not providing a code review, if the code in question is AI-generated. Fill in the number of the associated reason in each question; you may choose which you'd like to skip. For clarity, the reasons are:

- Reason **1**: "Code deletion would improve the MR a lot."
- Reason **2**: "You don't know the basics of the language you submitted."
- Reason **3**: "Documentation spam."
- Reason **4**: "Blatantly inconsistent."
- Reason **5**: "Edge case overload."
- Reason **6**: "Adding pointless or deprecated dependencies without knowing why."

18. (60pt) Support or refute the following claim: reason 1 is a valid reason not to review an AI-generated code review. **Must support. Simple problems like this should have been caught by the code's author, when they reviewed the code themselves before proposing the change. A diff with obviously-unhelpful changes that should just be deleted is not a clean diff. Refute answers that argued that "delete this" is useful feedback to the author of the CL got no credit, because that misses the point: the CL author should have already deleted something that obviously doesn't belong. It is rude and unprofessional to ask someone else to control your AI for you.**

19. (60pt) Support or refute the following claim: reason 2 is a valid reason not to review an AI-generated code review. **Likely support. The author is expected to learn something from the review. If they don't even know the basics, how will they learn anything? Moreover, if the author lacks readability, then the code is almost certainly not in a good state: 2025-level AI tools effectively lack readability in every language, and shouldn't be trusted to write high-quality code without supervision. Full credit answers for this reason usually mention readability. The strongest refute answer (40 points) argues that if the AI-generated code improves the health of the codebase overall, it may still be worth merging. But in that case, why is the "author" of the change involved at all?**

20. (60pt) Support or refute the following claim: reason **3** is a valid reason not to review an AI-generated code review. **Much easier to support. The simplest reason is that duplicate documentation doesn't make the project better, and the MR author should have caught it before sending it for review. Good refute answers need to argue that humans can make this mistake, too, and so if the docs are redundant but look human-written then review is appropriate.**

21. (60pt) Support or refute the following claim: reason **4** is a valid reason not to review an AI-generated code review. **Must support. Consistency is a core value in software engineering, and blatantly inconsistent code is a clear indicator that the submitter did not review the code themselves before asking for a review. A common problem on all of these questions, but this one especially, was saying why the thing is bad but not connecting that to anything we talked about related to code review.**

22. (60pt) Support or refute the following claim: reason **5** is a valid reason not to review an AI-generated code review. **Full credit answers for support and refute are both possible, but refute is easier. A full credit support answer is that a diligent CL author should have caught the AI over-engineering the code, and over-engineered code won't improve code health. However, a full credit support answer also needs to acknowledge that this principle in particular needs to be applied carefully to avoid false positive rejections. A full credit refute answer should argue that *sometimes* covering all edge cases carefully is a good idea, so denying a review just because of a bit of edge case over-engineering will have false positives: code that ought to have been reviewed will not be, some of the time. Ignoring a CL that was submitted in good faith is unprofessional.**

23. Support or refute the following claim: reason **6** is a valid reason not to review an AI-generated code review. **Must support. Pointless or deprecated dependencies make code health worse, almost by definition, and the CL author should have caught the problem during their own review.**

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

24. (10pt) In section II (Matching), there is a theme to the names used in the situation descriptions. What is the theme? **First names of New Jersey governors: William Livingston (1st gov. of NJ), Phil Murphy (current NJ governor), A. Harry Moore, Chris Christie, Woodrow Wilson, Jon Corzine, George McClellan, Richard Codey**

25. (a) (10pt) Why was Canvas unavailable on October 20th? **"AWS outage".**

    (b) (20pt) Describe the root cause of Canvas' unavailability on October 20th. **A cascade failure, initially triggered by a DNS misconfiguration. See https://aws.amazon.com/message/101925/. Anything mentioned in that doc that's remotely close to the correct root cause gets at least half credit.**

26. (10pt) Describe something interesting that you learned in this class that wasn't tested on this exam. **Anything reasonable counts.**

27. (10pt) Describe something interesting that you learned from a "Your Choice" reading other than the one that you used to answer question 17. Be sure to include the letter corresponding to the reading in the answer bank for question 17 prominently in your answer. **Anything reasonable counts.**

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:

Why I'm declining your AI generated MR

Published on 2025-08-26 by Stuart Spence

Sometimes a merge request (MR) doesn't merit a code review (CR) because AI was used in a bad way that harms the team or the project. For example:

(a) Code deletion would improve the MR a lot.

(b) You don't know the basics of the language you submitted.

(c) Documentation spam.

(d) Blatantly inconsistent.

(e) Edge case overload.

(f) Adding pointless or deprecated dependencies without knowing why.

If I decline your AI code MR with no further comments and send you this page then some of these conditions were met.

Despite some recent research and discussions on it I know that AI can be helpful in writing code. However AI misuse is also a new phenomenon and we need guidelines to help identify it. This page was written in 2025 and I expect the tools and guidelines to evolve.

Vocabulary

- **Merge Request (MR)**: when a programmer submits proposed changes to a project in a structured way. This makes it easy for anyone to see the differences and review the changes. Also called a **Pull Request** like on GitHub.
- **Code Review (CR)**: when another programmer reviews a MR, provides feedback or improvements, and approves or rejects the changes.

Why me?

I feel like I'm in a good position to write about this because:

(a) I'm a senior computer scientist for AI and cloud. I've been a technical supervisor to about 20 students and juniors.

(b) I have degrees and work experience in both computer science and education.

(c) I know code and AI. My AI project has a million installs and monthly income.

(d) I spend a lot of personal time enjoying, exploring, and discussing AI news and breakthroughs.[1]

(e) I don't need a job and I'm not selling anything. I'm not an investor or CEO shilling AI slop and I don't get ad revenue for flaming AI.

Why do code reviews?

There's thousands of opinions and articles. Here's a good one from Google[2]. Instead of rehashing that, let's focus on what AI misuse threatens. With good code reviews:

(a) Authors learn and improve.

(b) Reviewers learn and improve.

(c) We sanity check important changes.

(d) We minimize mental load for both humans and AI.

(e) We get consistent and simple code.

(f) Every MR makes the project better.

(g) Authors take responsibility for their code and can justify it.

Reasons for no CR

1) Code deletion would improve the MR a lot.

Can code be trivially deleted?

This violates the CR goals "sanity check" and "mental load". For example a setup script handling operating systems that we don't even have in our org. Not only should the author do this basic cleanup, but now they're placing an added burden on the reviewer to do it for them. In 2025 AI is not in a state where I'm comfortable running it in production with zero human review.

2) You don't know the basics of the language you submitted.

This violates the CR goal "authors learn". How can my feedback improve you as a software developer if you don't understand your own code? Going through you is not the best way for me to give feedback to your AI.

[1] The author linked to evidence of themselves doing these things in the real blog post, but in this exam you can take them at their word.

[2] In the actual blog post, this links to the reading that you were assigned for the "Code Review" lecture on October 13th, which is Google's code review guidelines

3) Documentation spam.

One example I've seen is two nearly identical copies of documentation in two different formats.

This violates "reviewers learn" and "make the project better".

If an author didn't trim or even read the AI generated documentation I think "they don't value my time or the time of my team". It's not the responsibility of a reviewer to edit 300 words of AI slop because the author didn't write the 3 words "keep it short" in their prompt.

4) Blatantly inconsistent.

Common examples I see are using new frameworks or styles for logging and unit tests.

This violates the CR goals "consistency" and "mental load". To understand a software project humans and AI may need to understand 50 concepts at once. Do we want them to have to consider 200 instead? Failing to manage complexity and consistency paralyzes a project once no human or AI is smart enough to improve it further.

5) Edge case overload.

This violates the CR goals "sanity check" and "make the project better". Handling many new unusual edge cases likely means the author didn't test all the code.

If we implement a feature at the cost of introducing twenty bugs with untested edge cases, that does not make the project better. It's like taking one step forward (progress) but falling into a mud pit.

AI slop may desperately catch an exhaustive list of all exceptions to "handle all cases". But the AI isn't handling the cases. It's just suppressing the valid exception or writing a non-standard error message.

6) Adding pointless or deprecated dependencies without knowing why.

This violates the CR goal "reviewers learn" and "take responsibility for your code". A reviewer might ask "why are we using this new thing here?" The author shouldn't respond "I have no idea, the AI did it". This may teach the team to use a deprecated tool, or the wrong tool for the job.

Maybe this is fine.

None of these are hard rules. I'm more inclined to accept an AI generated MR or give a CR to one if:

- The code is temporary or a one-shot analysis with no long term maintenance requirements. If it works it works!
- The MR includes an explanation of why AI was used, how much, why, and what extra steps the author took to validate it.
- This is an edge feature and not a core component.

Challenging Times

As a team lead, teacher, and I think nice guy I'm struggling with how to confront teammates when I feel their MR harms them, or the team, or the project. Why did they submit AI code? Was it a smart decision or just laziness? Do I harshly confront them and call it AI slop or do something else?

It's not always clear to me when it's a good use of AI that I should support with a full CR, or when it's a bad use of AI that I need to confront by rejecting it entirely. For me, just writing this page has helped.

Nobody has years of experience understanding how AI slop impacts technical debt or learning. If software development is changing in a good way then team leads need to change with it. If software development is changing in a bad way then we need to resist.