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 **150 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:	33 / 33
II. Matching:	24 / 24
III. Short Answer:	45 / 45
IV. "Your Choice" Reading Quiz	3 / 3
V. DBQs:	44 / 44
VI. Extra Credit:	x / 0
Total:	15x / 150

I. Multiple Choice and Very Short Answer (33pts). 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.

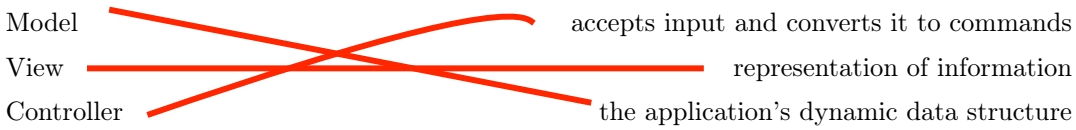
2. (2pt) **TRUE** / **FALSE**: software engineers typically assume that end users will describe bugs they encounter in a helpful manner.

3. (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

4. (2pt) **TRUE** / **FALSE**: it is possible to "foreclose" on a project that has too much technical debt by taking ownership of it in exchange for forgiving the technical debt.

5. (2pt) Which of these programming paradigms' key mathematical formalism is a Turing machine?
   A functional
   B object-oriented
   C imperative

6. (3pt) Match the terms in the right-hand column with the definitions in the left-hand column, in the context of a model-view-controller software architecture, by drawing a line between the term and its matching definition.



7. (2pt) **TRUE** / **FALSE**: the type system in every statically-typed language proves the same property.

8. (2pt) **TRUE** / **FALSE**: path coverage subsumes branch coverage.

9. (2pt) Which of the following testing techniques might be especially useful while refactoring? (Select all that apply.)
   A partition testing
   B regression testing
   C differential testing
   D stress testing

10. (2pt) Before bugs are assigned to developers, they need to be **triaged** to determine which bug reports have the highest priority.

11. (2pt) Good requirements are **testable/falsifiable/verifiable**, rather than relying on informal and ambiguous natural language.

12. (2pt) Version control systems' "conflict-free" detection can have:
    - **A** false positives
    - **B** false negatives
    - **C** both of the above
    - **D** none of the above

13. (2pt) The Liskov Substitution Principle states that "any **subclass** object should be safe to use in place of a **superclass** object at run time."

14. (2pt) In the **cathedral** / **bazaar** model of open-source development, users are treated as co-developers and projects are democratically organized.

15. (2pt) Which of the following are common causes of emergencies in deployed services? (Select all that apply.)
    - **A** incorrect error handling
    - **B** configuration changes
    - **C** hardware failures
    - **D** human/process error

16. (2pt) Netflix's Chaos Monkey testing system randomly chooses a server and **disables** it during its usual hours of activity.

17. (2pt) **TRUE** / **FALSE**: technical interviews like those commonly conducted by tech companies when hiring software engineers have been shown by academic researchers to correctly predict employee's job performance.

A. Availability	**B.** A/B testing	**C.** Stress testing	**D.** Pair programming
E. Factory pattern	**F.** Adapter pattern	**G.** Code review	**H.** Static analysis
I. Microservice	**J.** Cohesion	**K.** Type system	**L.** Partition testing
M. Canary testing	**N.** Hard fork	**O.** Anti-pattern	**P.** Pipe-and-filter
Q. Regression testing	**R.** Hermetic build	**S.** Project manager	**T.** Industrial researcher

II. Matching (24pts). This section contains a collection of terms discussed in class in an "Answer Bank" (choices **A.** through **T.**). 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.

18. (2pt) **B: A/B testing** Andreas has implemented a new login feature for their web application. They deploy it to a subset of users for two weeks and record what percentage of signups succeed in both the new and old login flows.

19. (2pt) **R: Hermetic build** Joel insists that his team's code can be shipped to a fresh VM without any manual setup.

20. (2pt) **M: Canary testing** When deploying a change, Jeff uses a tool to automatically only update a small set of servers before the change is deployed to the whole fleet.

21. (2pt) **G: Code review** Martin's team insists that he simplifies code or adds a code comment instead of just explaining complexity to them.

22. (2pt) **J: Cohesion** Charity decides to do a major refactoring, because there are several modules in her team's system that seem to have several jobs, and she wants to split them apart.

23. (2pt) **A: Availability** Hillel writes a script that automatically checks that his team's webservice is responding to calls once every minute.

24. (2pt) **I: Microservice** Management at Mike's company organizes each team around a single business capability, and Mike's team's software architecture reflects that.

25. (2pt) **E: Factory pattern** Jesse implements a method that returns a generic interface rather than a specific subtype, to hide implementation details from callers.

26. (2pt) **D: Pair programming** Ken is having trouble with a particularly difficult technical problem, so he schedules a call with one of his coworkers. Together, they write the difficult code.

27. (2pt) **O: Anti-pattern** Eric notices several copies of the same duplicated code in his team's codebase.

28. (2pt) **T: Industrial researcher** Mary is working on an AI product that requires her to stay aware of the latest work in AI.

29. (2pt) **L: Partition testing** Goran chooses inputs that represent specific sub-domains of the inputs he expects his program to take.

III. Short answer (45pts). Answer the questions in this section in at most three sentences.

30. (4pt) Support or refute the following claim: modern code review is useful even for legacy systems whose absolute quality is poor. **Likely support. Modern code review relies on an inductive argument for code quality: each change should make the system no worse. And, code review has other benefits besides code quality, like knowledge transfer, that might be important to teams working on legacy systems.**

31. Consider the following code snippet:

```
1    static boolean checkInBox(int x, int y) {
2      return (this.x < x && x < this.x + 10) && \\
3          (this.y < y && y < this.y + 10);
4    }
```

What are two code-level design improvements that you could make to this method? The two improvements cannot both be the same kind of change.

(a) (3pt) **replace the "10" magic numbers with named constants**

(b) (3pt) **rename "x" or "y" to avoid the shadowing, or rename "checkInBox" to "inBox" or similar.**

(c) (3pt) Select one of your answers to the previous question. Give a one-sentence justification for why your change improves the method's code-level design. **For 1. above: "avoid magic numbers" or "more self-documenting". For 2. above: shadowing is bad, or "check" adds no value to the name.**

32. (4pt) Support or refute the following claim: the distinction between "free software" and "open-source software" is no longer important in modern software engineering. **Likely refute. Free software is typically copyleft-licensed, so it can't be directly depended on in corporate environments, unlike OSS.**

33. Consider the following pairs of italicized tools, techniques, or processes. For each pair, give a class of defects or a situation for which the first element performs better than the second (i.e., is more likely to succeed and reduce software engineering effort and/or improve software engineering outcomes) and explain both why the better choice performs better and why the worse choice performs worse.

    (a) (3pt) *mutation testing* better than *statement coverage* **Mutation testing checks not only whether tests can reach bugs, but also checks the quality of oracles.**

    (b) (3pt) *constructive cost model* better than *tshirt size estimation* **Cocomo is based on data, so it can be more accurate.**

34. (4pt) You are a junior software engineer at GetHub, an e-commerce marketplace, on the team that handles payments. Through a series of unfortunate events, you discover that starting tomorrow, you will be the only person on your team who is pageable: everyone else is either sick or on vacation. However, one senior engineer is available this afternoon to help you prepare. What is the most important thing that you can ask your senior colleague to do to help you prepare, and why? **Almost certainly update the runbooks for emergencies. You'll just be trying to keep the lights on until your more senior colleagues return. 3 points for "how to rollback", which should be part of the playbook/runbook.**

35. (6pt) Support or refute the following claim: choosing test inputs by hand is a form of toil. As part of your argument, you should analyze each of the six aspects of toil that we discussed in class. **Either answer is acceptable if well-supported. Support answers should argue that test input generation tools are effective and that it is easy to automate. Strong answers will mention fuzzing. Refute answers should argue that selecting test inputs inherently requires human effort. A very strong support answer might argue that an LLM can probably do this job. Full credit answers must mention all six aspects of toil. A common error on this question was to choose a side and then argue that all six toil aspects support that side: for example, trying to argue that writing tests is devoid of long-term value (it is not).**

36. (12pt) Suppose that you are friends with the following set of people: Alice, Bob, Charlie, Dave, Eve, Faythe, Grace, and Harry. You hold a party, and discover that the night ends in drama when you invite everyone. You decide to hold a series of smaller parties to figure out who doesn't get along with whom. To decide who to invite to each smaller party, you decide to use the delta debugging algorithm that you learned about in CS 490. Suppose further that the ground truth is that any party involving both Alice and Eve and some third participant will end in drama, because Eve will eavesdrop on a conversation between Alice and some counterparty, but Eve is terrible at avoiding detection. Moreover, you cannot attend any of these parties, because as a neutral scientific observer you don't want to disturb the experiment. Execute the delta debugging algorithm to determine the set of parties that you will throw, showing your work at each step. You may abbreviate all of your "friends" by the first letter of their first names in your answer (e.g., you may write "A" for "Alice", "B" for "Bob", etc.). **Start with $ABCD|EFGH$.**

Neither of the sides of this split is "interesting", so we need to deal with the interference by finding the minimal part of the left-hand set that, combined with the right-hand set, causes the problem.

Next, we'd try $ABEFGH$, which is interesting. Since this is interesting, we can skip testing C and D. We'll next test $AEFGH$, which is also interesting. This means that the interesting subset of the left-hand side is just A.

Next, we'll repeat the same process on the right-hand side. We'd therefore next try $ABCDEF$, which is again interesting. So, we'd similarly recurse and try $ABCDE$, which is also interesting. The result, then, is that the interesting part of the right-hand subset is just E.

We would be tempted to return the set containing only A and E. However, if we test that we'll discover that it is not interesting: the problem statement says that a subset is interesting if it contains A, E, and any third element. This is a violation of the "unambiguous" property: the intersection of any two interesting sets must itself be interesting! But, the intersection of $ABCDE$ and $AEFGH$ (both of which are interesting!) is just AE, which is not interesting. Full credit answers must reach this conclusion and explain that the problem is the ambiguity in the interesting function. Parital credit (10 points) for AE for faithfully following the DDB algorithm. 3 points for correctly taking the first DDB step (i.e., split in half), followed by incorrect steps. 0 points if first step is incorrect. Many students would have been well-served by leaving this question blank.

Table 1: **"Your Choice" Readings:**

A. Garlan's *Software Architecture*
B. Kellogg et al.'s *Verifying Object Construction*
C. Kim et al.'s *A Field Study of Refactoring Challenges and Benefits*
D. Malkawi's *The Art of Software Systems Development: Reliability, Availability, Maintainability, Performance*
E. Scully et al.'s *Machine Learning: The High-Interest Credit Card of Technical Debt*
F. Dean and Barroso's *The Tail at Scale*
G. Xu et al.'s *Do Not Blame Users for Misconfiguration*
H. Terrell et al.'s *Gender Differences and Bias in Open Source: Pull Request Acceptance of Women versus Men*

IV. "Your Choice" Reading Quiz

37. Answer the question below about one of the "Your Choice" readings from the list above.

    (a) Write the letter of the "Your Choice" reading about which you are answering the next part of the question: **A-H.**

    (b) (3pt) Write something you learned from the article that is not obvious from reading the title. To receive credit, whatever you answer must be *explicitly discussed* in the article. **Answers vary.**

V. Document-based Questions (44pts). 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 and the next refer to **Documents A**, **B**, **C**, **D**, and **E**.

Suppose that you a software engineer at Amazing Instant Messaging, Ltd. ("AIM"), a small company that provides instant messaging services over the internet. Your main instant messaging product uses the NSS cryptography library from Mozilla to encrypt communications between your users. **Document A** is an excerpt from a blog post describing a security vulnerability in NSS. **Documents B**, **C**, and **D** are supporting documents for the excerpt. **Document E** is an email from the CEO of AIM to you about the vulnerability.

38. (6pt) Support or refute the following claim using evidence from the documents: a fuzzer should be able to discover this vulnerability. **No complicated state is required, and a fuzzer generating input signatures should have had no problem finding the bug. The easiest argument is to use Document C's existence to justify this claim, but there's a lot of evidence scattered throughout. For full credit, must mention that the bug's symptom is a crash/segfault, which is an implicit oracle that a fuzzer can detect.**

39. (6pt) Suppose that this bug existed for a long time (in real life, it existed for at least 9 years before being discovered!), even though the code in question was actively being fuzzed regularly by Google's `oss-fuzz` project. Give a plausible explanation for why a fuzzer might not have discovered the bug, despite this code being "covered" by the fuzzer. **The actual reason is twofold: the limit on the size of fuzzer inputs was 10k bits (but about 16k bits are required to trigger the bug), and a fuzzer that "covers" the target location *uses* this code, but does not actually manipulate the signature (the signature is constant, and something else is being fuzzed). Any plausible answer here is fine.**

40. (4pt) Support or refute the following claim: even if none of your customers were impacted, you need to write a postmortem about this bug. **Likely refute. Your company was following the best practice here (don't roll your own crypto) and you got unlucky. As long as no customers were impacted, you can upgrade the library and move on. A support answer is possible, but needs to bring up some other kind of impact as a justification. For example, if AIM is available as an app, customers may not have been impacted *yet*, but this vulnerability means we need them to do so ASAP. Support with "builds trust" as the reason is 2 points: it's okay, but not a great reason given that this isn't your bug.**

41. (16pt) **Document E** contains an email to you from the CEO of AIM. Write an appropriate, professional email response on the rest of this page. **Answers were judged holistically, taking into account the following requirements (each of which was a deduction):**

    - **must be written from the right perspective: if it's from Mozilla's perspective, that's wrong**
    - **must say "outside our control" or similar, to ack that it's in a dependency and not in AIM's own system**
    - **must not suggest that we were responsible for the bug**
    - **must not say that anything is currently broken (it isn't in this scenario)**
    - **must say that we used best practices or something similar, and that this kind of bug is fundamentally unavoidable**
    - **must be professional**
    - **must emphasize that no customers were impacted**
    - **must generally having a calming tone, and not feed into Mr. Obi's overreaction**
    - **shouldn't give too much technical detail about the bug, because Mr. Obi isn't an engineer**

    A full credit answer:
    Hello Sir Obi,
    It is true that we use Mozilla's NSS library for AIM's cryptographic capabilities, and yes, a serious vulnerability has been discovered in it. It is important to note that this vulnerability has only recently been found, and we immediately suspended our services as soon as we heard about it until we could integrate a fix. Thankfully, the fix was released at the same time that the vulnerability was disclosed, so customer impact was limited.
    I assure you that we have not been hacked or otherwise compromised because we only use the NSS library for user services, not for our internal communications.
    Although the library was extensively tested, this vulnerability went undetected by security professionals across the industry, not just by us. Given how many people missed it, I doubt that any bad actors would have had the chance to exploit this; even if they did, there are

much bigger targets than us (e.g., anyone using Firefox!).

Best regards,

You

Questions on this page refer to **Document F**, which contains some simplified (Java) code for a weather monitoring and display system.

42. (4pt) Support or refute the following claim: the `WeatherStation` and `Display` classes are tightly-coupled. **Likely support. Any change to e.g., add support for multiple displays would also require a change in `WeatherStation`.**

43. (8pt) Describe a refactoring that you could make to this code to de-couple these classes. (Hint: consider using a design pattern.) **The best answer here is "use the observer pattern".**

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

44. (1pt) In section II (Matching), there is a theme to the names used in the situation descriptions. What is the theme? **First names of authors of mandatory readings from this semester.**

45. (1pt) In the engineer panel, one of the engineers interrupted another to make a specific point about postmortems (which they also called "CoE"s, because that's what Amazon calls them). What was the specific point? **Postmortems need to be blameless!**

46. (1pt) If you would be willing to serve on an engineer panel in the future, write an email address that you will continue to monitor after you graduate in this space. If you're unwilling, you can write "no" in this space to recieve full credit on this question. **Varies.**

47. (1pt) What was something that surprised you about CS 490? **Anything goes here.**

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

Network Security Services (NSS) is Mozilla's widely used, cross-platform cryptography library. When you verify an ASN.1 encoded digital signature, NSS will create a `VFYContext` (see **Document B**) structure to store the necessary data. This includes things like the public key, the hash algorithm, and the signature itself.

The maximum size signature that this structure can handle is whatever the largest union member is, in this case that's RSA at 2048 bytes. That's 16384 bits, large enough to accommodate signatures from even the most ridiculously oversized keys.

Okay, but what happens if you just....make a signature that's bigger than that?

Well, it turns out the answer is memory corruption. Yes, really.

The untrusted signature is simply copied into this fixed-sized buffer, overwriting adjacent members with arbitrary attacker-controlled data.

The bug is simple to reproduce and affects multiple algorithms. The easiest to demonstrate is RSA-PSS. In fact, just the three commands in **Document C** work!

The actual code that does the corruption varies based on the algorithm; **Document D** is the code for RSA-PSS. The bug is that there is simply no bounds checking at all; `sig` and `key` are arbitrary-length, attacker-controlled blobs, and `cx->u` is a fixed-size buffer.

This wasn't a process failure, the vendor did everything right. Mozilla has a mature, world-class security team. They pioneered bug bounties, invest in memory safety, fuzzing and test coverage.

NSS was one of the very first projects included with oss-fuzz, it was officially supported since at least October 2014. Mozilla also fuzz NSS themselves with libFuzzer, and have contributed their own mutator collection and distilled coverage corpus. There is an extensive testsuite, and nightly ASAN builds.

- Did Mozilla have good test coverage for the vulnerable areas? YES.
- Did Mozilla/chrome/oss-fuzz have relevant inputs in their fuzz corpus? YES.
- Is there a mutator capable of extending ASN1_ITEMs? YES.
- Is this an intra-object overflow, or other form of corruption that ASAN would have difficulty detecting? NO, it's a textbook buffer overflow that ASAN can easily detect.

Document B

The definition of the `VFYContext` structure from NSS:

```
struct VFYContextStr {
   SECOidTag hashAlg; /* the hash algorithm */
   SECKEYPublicKey *key;
   union {
       unsigned char buffer[1];
       unsigned char dsasig[DSA_MAX_SIGNATURE_LEN];
       unsigned char ecdsasig[2 * MAX_ECKEY_LEN];
       unsigned char rsasig[(RSA_MAX_MODULUS_BITS + 7) / 8];
   } u;
   unsigned int pkcs1RSADigestInfoLen;
   unsigned char *pkcs1RSADigestInfo;
   void *wincx;
   void *hashcx;
   const SECHashObject *hashobj;
   SECOidTag encAlg;    /* enc alg */
   PRBool hasSignature;
   SECItem *params;
};
```

Document C

Reproducing this vulnerability in three easy commands.

```
# We need 16384 bits to fill the buffer, then 32 + 64 + 64 + 64 bits to overflow to hashobj,
# which contains function pointers (bigger would work too, but takes longer to generate).
$ openssl genpkey -algorithm rsa-pss -pkeyopt rsa_keygen_bits:$((16384 + 32 + 64 + 64 + 64)) -
    ↪ pkeyopt rsa_keygen_primes:5 -out bigsig.key
# Generate a self-signed certificate from that key
$ openssl req -x509 -new -key bigsig.key -subj "/CN=BigSig" -sha256 -out bigsig.cer
# Verify it with NSS...
$ vfychain -a bigsig.cer
Segmentation fault
```

Document D

The signature size must match the size of the key, but there are no other limitations. `cx-u>` is a fixed-size buffer, and `sig` is an arbitrary-length, attacker-controlled blob.

```
1           case rsaPssKey:
2                sigLen = SECKEY_SignatureLen(key);
3                if (sigLen == 0) {
4                    /* error set by SECKEY_SignatureLen */
5                    rv = SECFailure;
6                    break;
7                }
8
9                if (sig->len != sigLen) {
10                   PORT_SetError(SEC_ERROR_BAD_SIGNATURE);
11                   rv = SECFailure;
12                   break;
13               }
14
15               PORT_Memcpy(cx->u.buffer, sig->data, sigLen);
16               break;
```

Document E

Hi engineering team,

I heard about this giant security bug in NSS from someone I follow on Twitter - don't we use that library? I remember you all telling me that one time like three years ago, right? Does this means we've been 100% pwnd? I'm also cc'ing marketing to do damage control if needed. Pls fix asap.

How could you all let me and our valued customers down this way? I trusted you.

Sir Ken Obi, OBE
President & CEO, AIM Ltd.

Sent from my iPhone

Document F

```
1  class WeatherStation {
2      private Display display;
3
4      public WeatherStation(Display display) {
5          this.display = display;
6      }
7
8      public void updateWeather(Weather weather) {
9          display.showWeather(weather);
10     }
11 }
12
13 class Display {
14     public void showWeather(Weather weather) {
15         System.out.println("Current weather: " + weather.toString());
16     }
17 }
```