EECS 481 — Software Engineering
Fall 2019 — Exam #2

- **Write your UM uniqname and UMID and your name on the exam.**
- There are ten (10) pages in this exam (including this one) and eight (7+1) questions, each with multiple parts. Some questions span multiple pages. If you get stuck on a question, move on and come back to it later.
- You have 1 hour and 20 minutes to work on the exam.
- The exam is closed book, but you may refer to your two page-sides of notes.
- Even vaguely looking at a cellphone or similar device (e.g., tablet computer) during this exam **is cheating**.
- Please write your answers in the space provided on the exam. Clearly mark your solutions. You may use the backs of the exam pages as scratch paper. Do not use any additional scratch paper.
- Solutions will be graded on correctness and clarity. Each problem has a relatively simple and straightforward solution. We may deduct points if your solution is far more complicated than necessary.
  - *Good Writing Example:* Testing is an expensive activity associated with software maintenance.
  - *Bad Writing Example:* Im in ur class, @cing ur t3stz!1!
- If you leave a non-extra-credit portion of the exam blank, **you will receive one-third of the points for that small portion (rounded down) for not wasting time.**

UM uniqname: _____

UM ID: _____

Name (print): _____

1 Delta Debugging (22 points)

The Delta Debugging (DD) algorithm makes certain assumptions about its input. It requires that the interesting function be *monotonic*, *unambiguous* and *consistent*. Consider a set of of items $C = \{1, \ldots, 8\}$. In each of the following scenarios, we will provide the output of interesting on certain subsets of C and you must fill in the blanks for other subsets. Mark each other subset with Y for "must be yes", N for "must be no" and ? for "could be yes or no, it's unconstrained" such that the entire scenario meets DD's assumptions.

(3 pts.) Fill in the blanks for interesting:

interesting($\{3,4\}$)	=	Y	interesting($\{4,8\}$)	=	Y
interesting($\{8\}$)	=	N	interesting($\{4\}$)	=	_____
interesting($\{3,8\}$)	=	_____	interesting($\{1,3,4\}$)	=	_____

(3 pts.) Fill in the blanks for interesting:

interesting($\{3,4\}$)	=	_____	interesting($\{4,8\}$)	=	Y
interesting($\{8\}$)	=	_____	interesting($\{4\}$)	=	N
interesting($\{3,8\}$)	=	Y	interesting($\{1,3,4\}$)	=	_____

Your friend from Ypsilanti proposes an automated debugging algorithm, Ypsilon Debugging (after the Greek υ ypsilon instead of δ delta). Optimizing for the common case in which the final minimized set is small, Ypsilon Debugging (YD) starts by checking each singleton element in turn to see if it is interesting, then checks all pairs, then all triples, etc. It returns the first subset $c \subseteq C$ found with interesting(c) = Y. Fill in the blank or short response:

(1 pt.) _____ Will YD find a minimal interesting subset ("Y" or "N")?

(2 pts.) _____ Will YD find a one-minimal interesting subset ("Y" or "N")?

(1 pt.) $\mathcal{O}$(_____) What is the Big-Oh running time of YD if a singleton is interesting ($|C| = n$)?

(2 pts.) $\mathcal{O}$(_____) What is the worst-case Big-Oh running time of YD ($|C| = n$)?

(2 pts.) _____ Does YD require monotonicity ("Y" or "N")?

(2 pts.) _____ Does YD require unambiguity ("Y" or "N")?

(2 pts.) _____ Does YD require consistency ("Y" or "N")?

(2 pts.) With respect to a quality property, describe a situation in which YD outperforms DD.

(2 pts.) Describe a situation in which YD would be correct but *binary search* would not be.

2 Design Patterns (14 points)

Consider the following *incorrect* code for implementing a *Singleton* design pattern.

```
1  class Singleton:
2    @staticmethod
3    def get():
4      return Singleton._instance
5
6    _instance = None
7
8    def __init__(self):
9      self._state = 42
10
11   def current_state(self):
12     if Singleton._instance is None:
13       Singleton._instance = Singleton()
14     return self._state
15
16 def main():
17   print(Singleton.get().current_state())
```

(6 pts.) In at most four sentences, indicate the defect in this code and how you would fix it. Be specific.

(8 pts.) Consider a *Singleton* design pattern with a public "get" method and a private constructor. When using *inheritance*, which of those two can (or should) be overwritten by a subclass? Why or why not for each?

3 Requirements Elicitation (24 points)

(4 pts.) You are considering requirements for a bug-finding static analysis tool (like Infer or CodeSonar). Give an example of a useful *informal goal* for a quality requirement. Give an example of a *verifiable* quality requirement (be specific and detailed).

(8 pts.) You are the customer asking a software company to build you a program that sorts a list of numbers. You both agree that the output of sort must be in ascending order. The company delivers a program that always returns the empty list. You meet to correct this. The company then delivers a program that deterministically tries all permutations of the input list until one is found that is sorted. (This is sometimes called "bogosort" or "permutation sort".) Explain elements that may have gone wrong during requirements elicitation (at any stage) and describe best practices to avoid such mistakes.

(4 pts.) Support or refute the claim that *traceability* from requirements to tests is an important consideration when *designing for maintainability*.

(4 pts.) Using a situation from class or your own experience, give an example of a reasonable *strong conflict* in requirements. Then give an example of a reasonable *weak conflict* in the same situation.

(4 pts.) A customer cares about "analysis time" and "test suite quality". Your company understands "statement coverage", "branch coverage" and "mutation analysis". Explain *why and how* you would *explore alternatives* with the customer.

4 Design and Maintainability (12 points)

(4 pts.) Compare and contrast *what* and *why* documentation content with respect to both usefulness for *maintenance* and also *tool support*.

(4 pts.) Consider an autograder used for class assignments. (For example, consider `autograder.io` evaluating HW3 submissions in this class.) How would you design it to support testing its non-interface (i.e., non-web, non-GUI) functionality?

(4 pts.) Consider an automated fuzz-testing input generation tool (like AFL) and an automated unit test generation tool (like EvoSuite). For each, support or refute the claim that it would be useful for quality assurance for a web-based autograder (including the web interface this time).

5 Interviews (10 points)

You are responsible for *giving* a non-behavioral technical interview to job candidates; you are the interviewer. Your company views technical interviews as an assessment of software engineering skills. The programming problem you ask of candidates is:

> Write CharCount(), a function that counts the number of occurrences of a character in a string.

The candidate's complete response is below. The first two commented lines indicate questions the candidate asked you.

```
1  /* Q: Can the string be empty? A: Yes. */
2  /* Q: Should I optimize for multiple repeated calls? A: No. */
3
4  int CharCount(String str) {
5    // counting occurrence of character with loop
6    int charCount = 0;
7    for (int i=0; i<str.length(); i++) {
8      if (str.charAt(i) == 'a') {
9        charCount++;
10     }
11   }
12   return charCount;
13 }
14
15 // test: "barbara liskov" -> 3
```

(2 pts.) Identify two thing that the candidate did well.

(8 pts.) Identify and justify four significant things that the candidate did poorly.

6 Other Topics (18 points)

(4 pts.) Consider a dynamic memory analysis, such as taint tracking (to determine if data from an untrusted source ever reaches a trusted consumer) or memory leak detection (to determine if all allocated memory is eventually freed). Explain how a multi-language project involving both C++ and Python (with the associated "glue code" or "foreign function interface") complicates such an analysis. Be as specific as you can.

(6 pts.) Consider the following quotation reproduced in Cockburn and Williams *The Costs and Benefits of Pair Programming*:

> A system with multiple actors possesses greater potential for the generation of more diverse plans for at least three reasons: (1) the actors bring different prior experiences to the task; (2) they may have different access to task relevant information; (3) they stand in different relationships to the problem by virtue of their functional roles ... An important consequence of the attempt to share goals and plans is that when they are in conflict, the programmers must overtly negotiate a shared course of action. In doing so, they explore a larger number of alternatives than a single programmer alone might do. This reduces the chances of selecting a bad plan.

Relate this quotation to *Agile Development* and its result that team diversity has a positive effect on software functionality. Then relate this quotation to *conflict resolution* in requirements elicitation.

(4 pts.) Explain the relationship between coverage-based *fault localization* and *automated program repair* (APR). In that context, give one example of a bug that would be difficult for APR to fix and one example of a bug would be easy for APR to fix.

(4 pts.) You are considering a medical imaging brain study experiment aimed at understanding code readability. Participants are placed in an fMRI scanner, shown pieces of code or prose, and given questions to answer about them. Participants are shown programs with different complexities (e.g., bubblesort vs. quicksort) as well as prose snippets with different complexities (e.g., elementary vs. high school reading level). The researchers contrast the code and prose conditions and conclude that a activity in a particular part of the brain is associated with assessing readable code but not with assessing unreadable code. With respect to *construct validity* and/or *controlled experimentation*, argue for or against believing the results of this study.

7 Extra Credit **(1 pt each; we are tough on reading questions)**

(Feedback) In retrospect, what is one thing you enjoyed about the second half of this class? What should be changed about the second half of this class for next year?

(Feedback) What would you like to see changed about the discussion sections?

(Your Choice Reading) Identify one of the readings from the second half of the class that we did not cover explicitly during the lecture. Write a sentence about it that convinces us that you read it critically. (Our subjective judgment applies here — sorry!).

(My Choice Reading) In "Experts bodies, experts minds: How physical and mental training shape the brain", what change in the brain "across each domain of expertise, ... reflecting an increased level of expertise" was found?

(Guest Lecture 1) In "Medium-Scale Software Engineering", give one point raised by Dr. Leach about industrial code management at Clinc.

(Guest Lecture 2) In "Accelerated Gnome Sequencing for Precision Medicine", what did Dr. Wadden say was the critical issue with hardware debugging at Sequal?