EECS 481 — Software Engineering
Winter 2019 — Exam #2

- **Write your UM uniqname and UMID and your name on the exam.**
- There are thirteen (13) pages in this exam (including this one) and six (6) 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): _____

UM uniqname: (yes, again!) _____

UM ID: (yes, again!) _____

Problem	Max points	Points
1 — Delta Debugging	20	
2 — Requirements	18	
3 — Design	21	
4 — Productivity	18	
5 — Other Topics	23	
Extra Credit	0	
TOTAL	100	

How do you think you did? _____

1 Delta Debugging (20 points)

Consider applying the Delta Debugging algorithm to the task of selecting off-the-shelf libraries and components to add to a software project. A finite set of capabilities $C = \{c_1, \ldots, c_n\}$ exists; a subset $D \subseteq C$ is necessary to complete the project. A finite set of third-party libraries $L = \{l_1, \ldots, l_m\}$ is available. Each library l_i has an associated set of positive and/or negative capabilities given by a function $\mathsf{caps} : L \to \mathcal{P}(C \times \mathcal{B})$, where $\mathcal{B}$ denotes the Booleans.

For example, it could be that $\mathsf{caps}(l_3) = \{(\text{graphics}, \text{true}), (\text{iOS}, \text{false})\}$, meaning that l_3 is a library that provides "graphics" but does not work with "iOS". Similarly, if $\mathsf{caps}(l_7) = \{(\text{authentication}, \text{true}), (\text{iOS}, \text{true})\}$, then l_7 is a library that provides "authentication" support and does run on "iOS". However, we could *not* use l_3 and l_7 at the same time because one uses iOS and the other forbids it: they *mismatch* on that capability.

We want to find a small set of libraries possible such that (1) all of the required capabilities are satisfied, and (2) there are no mismatches on any capabilities. Formally, we want to find an answer $A \subseteq L$ such that (1) $\forall c_i \in D.\ \exists l_j \in A.\ (c_i, \text{true}) \in \mathsf{caps}(l_j)$, and (2) $\forall c_i \in C.\ \forall l_j \in A.\ (c_i, \text{true}) \in \text{caps}(l_j) \Rightarrow \neg\exists l_k \in A.\ (c_i, \text{false}) \in \text{caps}(l_k)$. (That logical formulation says the same thing as the English; you can skip it if you like. There are no "tricks" in the math.) A set of libraries is *interesting* if all of the required capabilities are positively satisfied with no mismatches anywhere.

(2 pts.) In general, this problem formulation violates at least one of the fundamental assumptions of the basic Delta Debugging algorithm. Identify one such unmet assumption.

(10 pts.) Provide a simple example that shows that your chosen assumption is violated. You must do so with $n \leq 3$ and $m \leq 3$ by giving definitions for C, D, L and caps.

(4 pts.) Consider the following claim: "using Delta Debugging to minimize failure-inducing code changes is unnecessary if your development process includes continuous integration testing." In no more than four sentences, support or refute this claim.

(4 pts.) Our formulation of Delta Debugging included a check to $\mathsf{Interesting}(P \cup P_1)$ and $\mathsf{Interesting}(P \cup P_2)$. Explain why both checks will never return true at the same time. (You should walk through the reasoning or otherwise explain the contradiction that occurs if both are true; a "one word" answer will not suffice.)

2 Requirements (18 points)

(4 pts.) List an *informal goal* and a *verifiable requirement* for the same quality (i.e., non-functional) property. Why is the first "merely" informal? How would you verify the second? (A four-sentence answer should suffice.)

(6 pts.) Consider the following claim: "Branch coverage metrics simply approximate the notion of traceability from requirements elicitation." Support or refute the claim, giving very brief definitions for the key terms. (You should not need more than six sentences.)

(8 pts.) Consider the following descriptions adapted from a Mutation Testing Homework Assignment. For each one, write "**F**" if it best describes a functional requirement, "**Q**" if it best describes a quality (non-functional) requirement, and "**N**" if it does not describe a requirement. (If you think there is some overlap, pick the *best* or most precise answer.)

_____ You must implement and support three mutation operators: negating comparisons, swapping binary operators, and deleting statements.

_____ Your submission must run to completion on the autograder within two minutes.

_____ A correct implementation will distinguish test suite A from test suite B.

_____ Mutants that violate Python's indentation rules will not efficiently help mutation analysis.

_____ Your submission must not attempt to reverse engineer or in any way communicate or reveal information about the held-out autograder tests.

_____ Your solution must be deterministic.

_____ The mutants must be named 0.py, 1.py, 2.py, and so on.

_____ The ast.parse, ast.NodeVisitor, ast.NodeTransformer, and astor.to_source portions of the interface are relevant for this assignment.

3 Design (21 points)

Consider the following *incorrect* code for implementing an *Observer* (i.e., Publish-Subscribe) design pattern, adapted from James Perretta's lecture.

```
1  class Subject {
2    public static void subscribe (Observer observer) {
3      subscribers.Add (observer);
4    }
5    public static void unsubscribe (Observer observer) {
6      subscribers.Remove (observer);
7    }
8    public static void change_state () {
9      foreach (Observer observer in subscribers) {
10       observer.update ();
11       subscribers.Remove (observer);
12     }
13   }
14   private static List < Observer > subscribers = new List < Observer > ();
15 }
```

(5 pts.) In at most three sentences, indicate the bug in this code and how you would fix it.

(5 pts.) Consider the Template Method Pattern and the notion of Design by Contract. How do the use of virtual methods, non-virtual methods, and access modifiers (public, private, etc.) help enforce the pre- and post-conditions of the algorithm being implemented using this design pattern?

(5 pts.) Consider the following claim: "A tool that symbolically generates human-readable What-style documentation (e.g., for exceptional conditions or code changes) is actually a tool that generates test oracles." Support or refute the claim. (You should not need more than four sentences.)

(6 pts.) Consider the following claim: "Since reading code (e.g., in a code review or code inspection) takes time per line and since code comprehension is the dominant activity in software maintenance, the most effective design strategy for software maintenance is to write correct programs that contain fewer lines of code." Support or refute this claim. (You should not need more than five sentences. You may assume the reader is familiar with all references from the lectures or readings.)

4 Productivity (18 points)

(6 pts.) Consider the following quotation adapted from Laurie Williams et al.'s *Strengthening the Case for Pair Programming*:

> Programmer-hours do not double with pair programming as one might expect. First, collaboration improves the problem-solving process. As a result, far less time is spent in the chaotic, time-consuming compile and test phases . . . Additionally, pairs find that by pooling their joint knowledge, they can conquer almost any technical task immediately. In teaching the university class that participated in the experiment, the instructor noticed that individual workers asked two or three times more technical questions than did collaborative workers. While waiting for the answers to these questions, these students were generally unproductive . . . Pair programmers put pressure on each other to perform, keeping their partner focused and on task.

How do you reconcile this claim with Fred Brooks' law "adding human resources to a late software project makes it later" and suggestion that software engineering is not a partitionable activity? (Convince us that both claims can be true at once. This may involve bringing in knowledge of pair programming or the Mythical Man Month not mentioned here.)

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 MoveNode(), a variant on Push(). Instead of creating a new node and pushing it onto the given list, MoveNode() takes two lists, removes the front node from the second list and pushes it onto the front of the first.

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

```
1  /* Q: Can the source list be empty? A: No. */
2  /* Q: Can the lists overlap? A: Yes. */
3
4  void MoveNode(struct node** destRef, struct node** sourceRef) {
5    struct node* newNode = *sourceRef; // the front source node
6    assert(newNode != NULL);
7    *sourceRef = newNode->next; // Advance the source pointer
8    newNode->next = *destRef; // Link the old dest off the new node
9    *destRef = newNode; // Move dest to point to the new node
10 }
11
12 // test: {1,2,3} {1,2,3} -> {1,1,2,3} {2,3}
```

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

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

5 Other Topics (23 points)

(4 pts.) List and briefly explain two difficulties that may arise when attempting to use branch coverage in a multi-language project.

(4 pts.) List two things that change in the brain as humans gain expertise. Your answers can be associated with general expertise and/or computing expertise.

(4 pts.) Kate Highnam's guest lecture mentioned the possibility of changing managers multiple times while working on the same project. Choose "planning", "testing" or "requirements". Explain two aspects of your choice that would be complicated by changing your direct manager during a project.

(5 pts.) Consider the following computer science tasks:

1. Find a number in an unsorted array.
2. Find a number in a sorted array.
3. Find a number in a balanced binary tree.
4. Find a number in an unbalanced tree.
5. Reverse an unsorted array.

In one sentence, explain how experts and novices cluster problems. Then indicate how novices would likely divide the problems into two clusters (give the problem numbers the novice would put in the first partition, then the problem numbers the novice would put in the second). Finally, indicate how expert CS theoreticians would likely divide them.

(1 pt. each) Read the following narrative adapted from quotes from the Reetu Das guest lecture. Fill in each ____ blank with the single *most specific or appropriate* corresponding concept from the answer bank. (Each ____ blank does have a corresponding answer.) Each option from the answer bank will be used *at most once*.

A. Code Inspection	B. Debugger	C. Design Pattern	D. Fault Localization
E. Functional Property	F. Native Interface	G. Quality Assurance	H. Quality Property
I. Signal Coverage	J. Stakeholder	K. Technical Interview	L. Triage

____ We finished our design in May of last year but we are still verifying it.

____ Verifying hardware was difficult because it had less visibility: we did not use GDB, just signals.

____ You start with one person, but that person isn't the one you want to talk to. So you keep following pointers until you get to the right person.

____ Doctors won't use your stuff unless every bit of the output matches the expected value. Even if our output accuracy is actually better, they won't be happy.

____ One important aspect of this domain is security. We must anonymize everything; you can't share patient data on the cloud.

____ We transfer data between C and Verilog.

6 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?

(Free/Participation) If you could give one piece of advice to yourself when you were starting this class, what would it be?

(Your Choice Reading 1) 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 1) In "The Economic Value of Rapid Response Time", characterize the reported relationship between system response time and the number of transactions a user could complete in an hour.

(My Choice Reading 2) In "Industrial Experience with Design Patterns", name one of the three claims or lessons that all six companies attributed to design patterns.

(My Choice Reading 3) On the forum, Microsoft Program Manager Peter Shultz gave an industry perspective on the class material. Explain the "persuasive" relationship between Program Managers and Software Engineering Managers he detailed.