EECS 481 — Software Engineering — Exam #2 KEY

- **Write your name and UM uniqname (i.e., email address) on the exam.**
- There are ten (10) 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: KEY

NAME (print): KEY

UM uniqname: (yes, again!) KEY

Problem	Max points	Points
1 — Delta Debugging	15	
2 — Requirements Elicitation	15	
3 — Design Patterns	14	
4 — Design for Maintainability	20	
5 — Interviews	16	
6 — Other Topics	20	
Extra Credit	0	
TOTAL	100	

How do you think you did? _____

1 Delta Debugging

Consider applying the Delta Debugging algorithm to the task of skill-based personnel assignment for a development project. A finite set of skills $S = \{s_1, \ldots, s_m\}$ is necessary to complete the project. For each skill s_i, the project must be assigned at least one developer who possesses that skill s_i. There is a finite set of developers $D = \{d_1, \ldots, d_n\}$ available, and each developer has an individual set of skills given by $\mathsf{skills} : D \to S$. For example, it could be that $\mathsf{skills}(d_2) = \{s_1, s_3\}$ while $\mathsf{skills}(d_5) = \{s_3, s_6, s_8\}$. Collectively, the full set of developers has all of the necessary skills S. You are interested in finding a smaller subset of developers that also has all of the necessary skills. Formally, a candidate set of developers D' is interesting if the union of all skills held by everyone in D' is equal to S.

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

This problem formulation is **ambiguous**.

(10 pts.) Provide a simple example that shows that your chosen assumption is violated. You must do so with $n = 3$ by giving definitions for S, D and skills.

$$\begin{aligned} S &= \{s_1, s_2\} \\ \mathsf{skills}(d_1) &= \{s_1\} \\ \mathsf{skills}(d_2) &= \{s_2\} \\ \mathsf{skills}(d_3) &= \{s_1\} \end{aligned}$$

We have $\mathsf{interesting}(\{d_1, d_2\})$ and $\mathsf{interesting}(\{d_2, d_3\})$, but not $\mathsf{interesting}(\{d_1, d_2\} \cap \{d_2, d_3\})$.

(3 pts.) We believe the best possible running time to find a minimal subset of an arbitrary set with an arbitrary deterministic interesting function is $\mathcal{O}(2^N)$. Delta Debugging advertises a better running time. Explain this apparent contradiction between Delta Debugging's running time and more general theoretical bounds. Use at most three sentences.

The Delta Debugging algorithm only finds a **1-minimal** subset of an interesting set. A 1-minimal subset is not as precise as a "fully" minimal subset. Delta Debugging takes advantage of the special structure of unambiguous, monotic and consistent problem formulations and weaker output requirement. Stronger input assumptions and weaker output requirements admit a more efficient algorithm.

2 Requirements Elicitation

(4 pts.) Given an example of a *conflict* that might arise during requirements elicitation. Indicate a best practice to *resolve* that conflict.

In a **strong conflict**, statements are not satisfiable together (e.g., "the button must be entirely and exclusively red" and "the button must be entirely and exclusively blue").

In a **weak conflict** or divergence, statements are not satisfiable given a condition (e.g., "the list shall be as large as needed" and "the list shall hold at most X items").

These can be resolved by **exploring tradeoffs**.

In a **terminology conflict**, the same concept goes by different names (e.g., user vs. customer).

In a **designation conflict**, the same name denotes different concepts (e.g., Rex the dog vs. Rex the human).

In a **structre conflict**, the same conflict is structured differently (e.g., complete on Friday vs. complete by close of business on Thursday).

These can be resolved with a **glossary**.

(6 pts.) Consider the following claim: "*Validation* is less expensive then *verification*, but mistakes made during *validation* are more expensive than mistakes made during *verification*." In about three sentences, support or refute this claim.

Validation determines if requirements are correct; verification determines if software is correct. Typically one would support this claim. Significant evidence was presented during the course that post-coding maintenance activities like reading and testing code are dominant lifecycle costs. However, we also saw that the cost of fixing a bug increases the later it is caught. For example, the graph on Slide 18 of Lecture 2 explicitly shows uncaught requirements defects as the most expensive.

(5 pts.) At the end of his lecture, Jason Mars described a situation in which Clinc made a significant mistake in *stakeholder analysis*. All but two of "decision making", "different needs", "exploring alternatives", "organizational position", "personal objectives", and "traceability" were relevant in that anecdote. In at most five sentences, briefly summarize the situation and indicate the four relevant aspects.

Mars described a situation in which multiple groups with different organizational positions in a large client organization all made different demands on his company. For example, a technical group might want one feature, while a business or marketing group might want another. By failing to identify which stakeholders were decision makers, the company ran into trouble: although it would decide to complete requests for one sub-group (they all had different needs), the other sub-group would become unhappy that their demands were not met instead. Mars originally thought that the groups were all against his company, but in fact they were all in favor of his

company: they had personal objectives related to being seen to be aligned with the product's success. The concepts of **exploring alternatives** and **traceability** are not relevant here.

3 Design Patterns

Consider the following *incorrect* code for implementing a *Singleton* design pattern, adapted from James Perretta's lecture.

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

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

The defect is that lines 9–10 should be moved to appear after line 3. Without that change, `get()` will not return a valid instance, and thus `get().current_state()` will fail. (Also acceptable: `__init__` conceptually has an infinite loop.)

(4 pts.) In the *Model-View-Controller* design pattern, two pairs of components typically communicate (i.e., depend on each other, call methods from each other, etc.) but one pair does not. Identify the components that should *not* communicate. In at most three sentences, indicate some ways in which subsequent maintenance would be complicated if that pair were, mistakenly, to be tightly coupled.

The Model should not know about the View details. Indeed, multiple different views might be supported by the same MVC setup.

While it is not always bad for the View to call Model data accessor functions, it is more common, at a high level, for the View to interface with the Controller which interfaces with the Model.

If the Model were dependent on View details, refactoring the View later, or adding additional Views later, would be more complicated. In addition, changes to the View might also require changes to the Model.

4 Design for Maintainability

(4 pts.) Following Wikipedia, "*Scalability* is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged to accommodate that growth." In at most three sentences, support or refute the claim that scalability is a functional property. If possible, relate your answer to Adam Brady's lecture on Google.

Either argument can be acceptable if adequately supported.

The traditional view is **refute**: scalability is a quality property. Scalability relates to the manner or fashion in which the correct result is delivered. In that regard it is akin to performance, which is also typically viewed as a quality property.

However, you could also **support**: in the limit, scalability can be a functional property. For example, $O(2^N)$ algorithms are not practical for $N > 100$; if Google took ten thousand years to respond to each search query it would be *effectively* unusable, and thus scalability would become a functional concern. Google has set things up so that individual component failure is expected but that scalability is critical.

(8 pts.) You are considering developing a *multi-language* project. Two requirements include the *performance* of the system (i.e., how fast it runs) and also how the *debugging cost* of the system (i.e., how fast defects can be repaired). For each requirement, list both an advantage and a disadvantage of a multi-language design. Use at most four sentences (one for each pairwise consideration).

Performance advantage: using a low-level language like C or assembly for a performance-critical kernel can improve the performance of a system.

Performance disadvantage: crossing the language boundary is typically expensive (e.g., it may involve copying or marshaling data), and, if done wrong, can result in code that is actually slower than a single-slow-language design (see Slide 72).

Debugging cost advantage: typically there are very few debugging advantages to multi-language systems. However, one might argue that within a single ecosystem (e.g., Java Bytecode or Microsoft Common Language Runtime), a single debugger can help with debugging tasks across multiple abstraction layers. Alternate creative arguments are possible. For example, it might be more natural to express an critical algorithm in language A than in language B, so a multi-language design might actually make that part *more* readable.

Debugging disadvantage: as detailed in class, most coverage tools, debuggers, static analyzers, dynamic analyzers, and profilers are all language-specific and thus typically only apply to parts of a multi-language system. This can give an incomplete picture.

(8 pts.) Explain the relationship between a *verifiable* quality requirements and *measurement*. Highlight at least one *risk* associated with measurement *uncertainty*. Use at most four sentences.

A **verifiable** quality requirement is a statement using some measure that can be objectively

tested (e.g., "responds within 10 seconds" rather than simply "is fast"). Because verifiability hinges of measurement, there is a risk that measurement error may influence whether the requirement will be perceived as being met or not. For example, if the timer accuracy is ± 2 seconds and the system actually responds within 9 seconds, the developer might measure that it takes 7 seconds and meets the requirement, while the stakeholder might measure that it takes 11 seconds and fails the requirement.

5 Interviews

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:

> Two strings are said to be anagrams of one another if you can turn the first string into the second by rearranging its letters. For example, "table" and "bleat" are anagrams, as are "tear" and "rate". Your job is to write a function that takes in two strings as input and determines whether they're anagrams of one another.

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

```
1  /* Q: Do I need to worry about upper- and lower-case? A: No. */
2  /* Q: Will the strings always be the same length? A: No. */
3
4  private boolean areAnagrams (String first, String second) {
5    return areAnagramsRec ("", first, second);
6  }
7
8  /* helper function: this is slow but correct */
9  private boolean areAnagramsRec (String soFar, String remaining, String target) {
10   if (remaining.length () == 0) {
11     return soFar.equals (target);
12   }
13   for (int i = 0; i < remaining.length (); i++) {
14     String whatsLeft = remaining.substring (0, i) +
15       remaining.substring (i + 1);
16     if (areAnagramsRec (soFar + remaining.charAt (i), whatsLeft, target))
17       return true;
18   }
19   return false;
20 }
21
22 /* test 1: "able", "bale"
23    test 2: "astronomer", "moonstarer" */
```

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

The code is functionally correct. The identifier names are clear. The indenting style is consistent.

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

No internal comments or explanations of the algorithm are given. No "why" documentation is given. No indication is made of the running time. The running time is actually very, very slow: $\mathcal{O}(N!)$. The design is too complicated (see below). No attempt was made to elicit quality requirements. The test cases are very poor: no "negative" or "corner case" tests are present, and very few standard tests are present. No discussion is made of design, maintainability or similar concerns.

Ignoring any comment or elicitation concerns, consider this much simpler, and much faster, $\mathcal{O}(N \log N)$ solution:

```
1  private boolean areAnagrams (String first, String second) {
2    char[] one = Arrays.sort(first.toCharArray());
3    char[] two = Arrays.sort(second.toCharArray());
4    return Arrays.equals(one, two);
5  }
```

6 Other Topics

(3 pts.) You design a neural representation (or "mind reading") experiment to determine if patterns of neural activation in the brain are similar for "talking about code" and "talking about prose". You use an fMRI or fNIRS device to measure the blood oxygen level dependent (BOLD) signal. You randomly sample, from the Mozilla Firefox project, methods ranging in size from 10–20 lines. While measured by the device, participants read the methods and are summarize them in their own words. In at most three sentences, explain the most significant reasons why it will be difficult to use this experimental setup to answer this research question, and also why it may be difficult to use the BOLD signal to assess software engineering.

BOLD medical imaging studies require a **contrast** or **controlled experiment**: they compare X to Y. This experimental setup will not answer the research question because it never actually measures "talking about prose".

In general, it is difficult to use the such medical imaging studies to assess software engineering because of the constraints they place on the participant's environment. For example, fMRI requires the participant to hold very still, limits the display to a small screen, limits the input to a few simple buttons, is very loud, is unfamiliar to most developers, and so on. As a result, there is a threat to ecological validity: the measured activity may not resemble "real" software engineering. You could also bring up that measurement error is non-trivial and that it is very difficult to do the mathematical analysis of medical imaging data correctly.

(5 pts.) Consider a hypothetical *pair test generation* activity in which one developer constructs test *inputs* while another developer constructs test *oracles*. Support or refute the claim that this activity will reap similar benefits for testing as pair programming does for programming. Use at most four sentences.

Typically **refute**. In pair programming, both members of the pair are focused on the construction of a single artifact: the code. One may be writing while the other is observing, but they are both focused on the same screen. In pair test generation, one is focused on the test input while another is focused on the test oracle: those might be on different screens or in different files. In addition, the participant working on one (e.g., the input) does not have the benefit of a partner double-checking that work (since the partner is focused on the other aspect).

You could try to argue **support**, but it would be a much more difficult argument. For example, having one partner think about the oracle might reveal early that a test input would actually be "less useful" or "low coverage" or somesuch. Or having one partner say "we really need something that eventually leads to output X" might help drive the other partner's test input generation. However, this should remind you of using AFL or similar random test input generation approaches. Developers typically report that it is actually fairly difficult to come up with the correct oracle for a test input that someone else (or some*tool* else) has produced. This is known as the "oracle problem" in software engineering — we covered it on Slide 40 of the Inputs and Oracles lecture.

(3 pts.) Explain the relationship between *anti-patterns*, *static analysis* and semi-automated *refactoring*. Use at most three sentences.

Refactoring can be viewed as transforming code that current exhibits an anti-pattern into code that exhibits a desired pattern. Semi-automated refactoring tools use static analyses to locate code regions that currently exhibit anti-patterns. For example, a static analysis might measure the coupling, cohesion, or complexity of a class or method. One an "underperforming" area of the code has been identified, it is transformed to be more aligned with a desired pattern and thus score better on that metric.

(4 pts.) Describe a specific situation (or sketch a small method) in which a static analysis tool (such as Infer or CodeSonar) would issue a false alarm and indicate the alarm type. Then describe a situation (or sketch a small method) in which such a tool would have a false negative and indicate the defect type.

False positives are very common with such tools (see the readings, etc.); almost anything was acceptable here.

False negatives may seem a bit harder, since these tools often report every instance of a given defect class that they see (even if they rank it low). However, remember that there are large error classes that they do not even report! For example, you could list any of the CVEs in `lighttpd` that the tools did not uncover!

(5 pts.) A senior software engineer who leads your team at a large software company suggests that the team start a paired activity. In this activity, pairs email each other code and changes and agree to merge files together. Explain why this activity is pair programming, pass-around code review, both, or neither. Support or refute the claim that this paired activity would be effective at improving code *readability*. Use at most five sentences.

This activity is likely **not pair programming**, which requires *synchronous* coding and observing. Indeed, if one partner is typing and the other is doing nothing, it is viewed as a failure mode in pair programming.

This activity could be **pass-around code review**, which is distributed and asynchronous. Your argument would depend on what it means to "agree to merge files together". In pass-around code review, the reviewers are explicitly evaluating the proposed change for correctness and quality and offering feedback.

This activity probably would **not help readability** compared to standard development. Standard pair programming produces shorter programs and shorter code segments correlate with readability. However, this is not pair programming, and we have no evidence that it will produce shorter segments. If it is akin to pass-around code review, it may result in higher readability (and you could make an argument, for example using Google's notion of readability badges and coding styles and the like). However, as phrased it sounds a bit more like asking developers to manually implement the patch-merging part of Git or SVN, which is unlikely to be helpful.

7 Extra Credit

What is one thing you would tell future students who are considering taking this class?

In retrospect, what is one thing you enjoyed about this class?

In retrospect, what should be changed about this class for next year?

From Beck et al.'s *Industrial Experience with Design Patterns*, list one of the "lessons learned".

"Patterns serve as a good team communication medium. Patterns are extracted from working designs. Patterns capture the essential parts of a design in a compact form. Patterns can be used to record and encourage the reuse of best practices. Patterns are not necessarily object-oriented. The use of pattern mentors in an organization can speed the acceptance of patterns. Good patterns are difficult and time-consuming to write. PAttern practice is of utmost importance."

In Haraldsson et al.'s *Fixing Bugs in Your Sleep: How Genetic Improvement Became an Overnight Success*, what did the system do?

"...in the evening, after the last user logs out, it starts a self-analysis based on the day's recorded interactions. It generates test data from the recorded interactions for Genetic Improvement to fix any recorded bugs that have raised exceptions. The system has already been under test for over 6 months and has in that time identified, located, and fixed 22 bugs."

From Chi et al.'s *Expertise in Problem Solving*, list one way in which experts and novices "chunk" problems differently.

Page 33 (PDF page 14): "...experts store physics equations in tightly connected "chunks," whereas novices store equations individually. ...the storage of equations may indeed be different in the knowledge base of the experts and novices ..."