Review of SWE 261P

- To study: Slides from class can be used as a summary of the most important aspects. Notes from class can provide more context and depth.
- Exam, Tuesday, March 12. 4:00pm–5:30pm, Pacific Standard Time. Our normal classroom and time. Should not need more time than 1.5 hours.
- Paper-based, mostly multiple-choice, multiple-answer, matching, etc.
- The TAs and I will be here to

help answer any questions that you have (in case a question is confusing, for example)

<u>Basic concepts</u>

- Why test software? Why study testing?
- What is the relationship between "testing" and "quality assurance"?
- What are some types of software "quality"?
- What are the similarities and differences in the way that various software process models incorporate testing and quality assurance? E.g.,

waterfall model, v-model, spiral model.
- Identify/differentiate validation, verification

<u>Testing Fundamentals</u>

- Identify/Differentiate mistake, fault, error, failure
- What is a test case?
- What is a test suite?
- What is a test oracle?
- What is a test plan?
- Black box versus white box testing
- Why not exhaustively test, or fully prove for correctness?
- What is the basic power of

testing to find bugs? Why do we then test at all?

Principles of Quality Assurance

- Six principles: yes, memorize and be able to describe them.
- Six principles: Partitioning, Restriction, Redundancy, Sensitivity, Visibility, Feedback

Exploratory Testing, Acceptance Testing

- What is exploratory testing? What is its strengths? What is its limitations? Who would want to do it, and when, and how?
- What is an exploratory testing

"test charter"?

- What is acceptance testing? What forms can it take?
- What is smoke testing?

Test Driven Development (TDD) and Behavior Driven Development (BDD)

- What is behavior driven development and how does it relate to acceptance testing?
- How does BDD relate to TDD?

Functional Testing

- Random testing versus systematic (what is the benefit of each?)

- Partitioning
- Why do we do functional testing?
- Identify boundary values and/or representative values

Combinatorial Testing

- What is combinatorial testing?
- What is the purpose pairwise testing, n-wise testing?
- How are different levels (pairwise, 3-way, 4-way, etc.) related, in terms of testing strength and size?
- I might give you a set of categories and ask to present the combinations (exhaustive

and pairwise).

Finite Models (Functional)

- Draw a FSM
- What are three purposes for having such program models?

Finite Models (Structural)

- CFG concepts (including "peculiarities": for loops, switch, break, continue, ternary operator)
- Short-circuiting: What is it? How can it affect control flow?
- Interprocedural, intraprocedural
- Draw CFG, with/without

maximal basic blocks (remember switches, for loops, ternary operators, continue, break)

- What are three purposes for having such program models?

Structural Testing

- Why would one want to perform structural testing?
- Is structural testing enough? If we had full structural coverage, can we guarantee that the program is correct? If we don't have full structural coverage, what confidence do we have in

the uncovered parts of the program?

- What is the typical process that would utilize functional and structural testing?
- What are the types of structural coverage that we discussed? What are each of their strengths? Are some subsumed by others?
- Why might it be impossible to cover all statements in your program? How might we handle these situations?
- You do not need to remember the entire subsumption hierarchy of test adequacy

criteria. But, you should know some of the basic principles: statements to branch, method to statement.

- Can we guarantee fault-detection ability with structural coverage? If we can guarantee execution of all statements, why can we not guarantee detecting all bugs?

<u>Instrumentation</u>

- What is program instrumentation? Why would anyone want to instrument a program?
- What is the difference between

coverage, profiling, and tracing?

- Can each of these (i.e., coverage, profiles, and traces) be derived from the others? If so, which can be derived from which?
- What are the runtime costs of each coverage, profiling, and tracing?
- What is an example of an instrumentation tool?
- Does instrumentation change the behavior of the program?

Integration

- What are some strategies for

integration during development?

- What are some of the challenges of integration?
- What is a stub? Why do we need them? What are they useful for?
- What is a mock? Why do we need them? What are they useful for?
- How are stubs and mocks different?
- What are some strategies for writing your code that makes it easier to test, particularly when it comes to integration? (i.e., testable design)

Continuous Integration

- What is continuous integration?
- What is continuous deployment/delivery?
- What are some common ways that developers use continuous integration systems?

Automated GUI Testing

- What is automated GUI testing? Why do it?
- What are some ways to encode GUI interaction, and what are the strengths and limitations of those ways?

- What part of automated GUI testing (e.g., when using capture/replay tools) that almost always needs extra human specification?

<u>Code Reviews and Static Analyzers</u>

- What is a code review and how does it work? What are some of the difficulties with doing these, and what are some approaches to try to avoid such difficulties?
- What is a *static* analysis (as opposed to a *dynamic* analysis)

- What are some best practices and etiquette for code reviews?
- What are “static analyzers” and how do they relate to code reviews?

Mutation Testing

- What is mutation testing? What is the purpose of mutation testing?
- What is the difference of strong and weak mutation testing?
- Be able to define the terms: "mutant", "kill", "mutation operator".

Debugging

- What is the difference of testing and debugging?
- What are some techniques/tools/strategies that developers use to help them debug their programs?

<u>Tools</u>

- JUnit
    - Be able to write the basic structure of a test class and method. I will not be concerned with all details, like if you forgot to put a semicolon on the end of a line, but you should know the main elements that go

into writing a JUnit test case method. (e.g., "@Test" annotation, assert method, etc.).
    - Know some best practices about how to write a set of test cases (e.g., independence, set up and break down, multiple asserts, etc.).
    - Know the purpose of using the Parameterized test runner, but you do not need to remember the syntax.
- Mockito
- EclEmma, IntelliJ Coverage
- Selenium

- FindBugs/SpotBugs, Infer, PMD, CheckStyle
- TravisCI, CircleCI, GitHub Actions
- PIT