Bugs are costly...

1. Types of bugs (What type of bugs have you had in the past?)
   a. Race conditions and deadlocks
   b. Library misuse
   c. Logical errors (off by one, null, buffer overflow)
   d. Usability
   e. Performance defects
   f. Error-handling errors
   g. Requirements bugs

2. Terminology of “bugs” and an example from Therac-25
   a. Fault is the static problem in the code (ex: when entered data is invalid, incrementing a variable called “errorCounter” instead of setting it to 1)
   b. Error is the dynamic problem at runtime (ex: the variable errorCounter overflows and goes to 0)
   c. Failure is the problem of the system (ex: the machine, Therac-25, believes that it has valid data when it does not and shoots a beam of radiation)
   d. Hazard is the resulting risk that actually happens (ex: a patient dies from radiation overdose)
   e. Faults in dead code do not cause errors
   f. Multiple faults may cause an error together
   g. An error may not always result in a failure (The Therac-20 had the same fault, and the same error, but there was a hardware override that prevented the failure).

3. The cost of defects
   a. 20% of modules cause 80% of defects, and 50% are defect free
   b. Cost of repair, from injected in requirements to detected in production, ranges from 5x to 100x (and potentially more)
   c. Rule of thumb is 3x-10x per phase in the software lifecycle.
   d. See graphics for more.

Kinds of Quality Assurance

1. Evaluative QA is detecting problems after they are in the code
   a. Testing
   b. Inspections
   c. Program analysis

2. Preventative QA attempts to stop defects in the first place
   a. Getting the requirements right
   b. Getting the design right
   c. Implementing the design properly

3. Can't detect all defects
4. Can't prevent all defects

5. Verification and Validation (V&V)
   a. Verification is checking whether the product matches its requirements
   b. Validation is checking whether the product accomplishes its goals
c. We can use testing to verify that the program matches the specified tests.
d. We must still validate that the specified tests are testing the right thing.

6. Precision of quality assurance
   a. All quality assurance techniques suffer from false positives and/or false negatives
   b. True positive: QA detects problem, and a problem exists.
   c. True negative: QA says no problems, and there is no problem.
   d. False positive: QA detects problem, but there is no problem.
   e. False negative: QA says no problems, but a problem exists.
   f. False negatives mean we miss bugs, and we never have a guarantee of no defects
   g. False positives cost developer time to investigate. Too many of them relative to true positives mean that developers stop using the QA technique.

Testing
   1. Direct execution of code on test data in a controlled environment
   2. Important: Testing only reveals inconsistencies with the specification. It does not say whether the program is wrong or whether the specification is wrong.
   3. Goals of testing
      a. Reveal specific failures
      b. Assess overall quality
      c. Verify contact (legal standards)
      d. Clarify the specifications
         i. Learn about the program
         ii. What does it do under unusual circumstances
         iii. Important for legacy systems
   4. Some forms of testing
      a. Black box and white box
         i. Black box: we do not see the internals of the code
         ii. White box: we see internal control flow and structures
         iii. Sometimes we do “grey box” as well
      b. Manual and Automated
         i. Semi-automomated techniques exist too.
      c. Unit testing is typically manual to create (then automated to run) and white box
         i. But can be black box if another developer tests to an API
         ii. Or can even be semi-automated creation with some tools
   5. Level of scale that we test at
      a. Test the entire system?
      b. Test a subsystem?
      c. Test a single class?
      d. Test a single method?
      e. For this lecture, we will look at method level, but same principles apply to all.
   6. Scaffolding a test
      a. Scaffolding provides a way to run the test.
b. Example: JUnit is a test scaffold.
c. Driver: the component that calls the test
d. Stubs: components that the test calls and return canned data
e. Test has three phases
   i. Setup phase: Create all the inputs to the test, including any needed stubs
   ii. Exercise phase: Run the unit under test with the inputs
   iii. Verification phase: Check the outputs against the expected outputs
f. Advice:
   i. Provide a different test for each different sets of input
   ii. Need to control the inputs, which includes the database state and other extraneous things
   iii. Build stubs to handle these. Stubs don’t have to actually work, they provide fake data. (mocks is a related term)
   iv. Limit exercise phase to one “unit”, whatever that is, so we can identify the faulty unit.
   v. Verify all parts of the output, including verifying for not having extraneous data
g.Mocks are related to stubs, but are not the same thing. See Fowler article in resources.

7. Regression is a major benefit to automated tests
   a. Benefits: prevent future bugs
   b. Low cost to store and run again
   c. Every time you fix a bug, write the test for it first so you can be sure that you fixed it.

8. Test driven development
   a. Write your tests first, then develop your code
      i. The tests you write should run and they should fail
      ii. Ensures that it actually tests something
      iii. Then write code that makes it pass
   b. Or maybe develop your code and write your tests very shortly after
      i. Upside: code is written first. Makes stubbing easy, and you know what the interface will be.
      ii. Downside: code is written first. Easy to write the test to pass the code.
   c. Or have a dedicated tester write the tests blackbox style
   d. Primary benefit: getting into a testing mindset and purposely trying to break your code.

9. Coverage
   a. Coverage is the proportion of code covered by your tests
   b. Common kinds of coverage:
      i. Line coverage
      ii. Branch coverage
      iii. Path coverage
      iv. Cem Kaner list of 101 types of coverage
c. 100% coverage can be difficult to achieve due to dead code or code that is not important for testing (like toString()).
d. However, coverage lets us see what code we have not yet written a test for
e. Mutation testing
   i. Another way to check coverage
   ii. Use the same tests, but mutate the code!
   iii. Check that the mutation causes some test to fail

10. Selecting inputs for test
   a. Since we can’t get complete coverage, we must select inputs that are most likely to find defects
   b. Can do a structural analysis of code and select inputs to exercise each branch or path through the code.
   c. Can use equivalence classes to select one input from each “type” of input
      i. Example with a list: list of three elements, list of one element, empty list, null list. Maybe a list of lists?
   d. Can get representative samples from practice
   e. Can automatically generate inputs across the space of inputs

Evaluating Quality Assurance Techniques
1. All quality assurance techniques have different costs and benefits.
2. Need to be able to critically analyze them.
3. Here are some questions you might consider...can you answer them for white box, method level unit testing?
   a. Overall Cost What are the costs of this technique? Consider both sunk costs and recurring costs. Sunk costs are costs for purchasing tools and setting them up. Recurring costs are the cost of using the tool and maintaining any resources it needs. Remember, software development is a dynamic process. The code will change. Consider what costs will accrue from changes or from making the technique more resilient to change.
   b. Bugs Found What classes of bugs does this technique find? What's the relative importance of classes of bugs? What is their relative expense if they leaked into production code? A security bug could be very expensive (your company can get sued!), but a minor UI flaw will only annoy customers.
   c. Bugs Missed What classes of bugs did it miss? What caused it to miss these bugs?
   d. Technique Overlap What other quality assurance techniques might find similar types of bugs?
   e. Cost of Fix What are the costs of fixing a bug once it is found with this technique? Consider how the developer will reproduce the bug, find the root cause of the bug, and then fix the issue.
   f. Lifecycle How does this technique fit into the software lifecycle? Think about who might run this QA technique, when they would do it, which resources to use the technique on, and what tools/resources they will need.
g. **Stopping Point** In any quality assurance technique, one could go on finding defects forever. With this technique, what are some ways to determine when to stop?

h. **Intangible benefits.** Sometimes there are other benefits that are intangible. A technique that has no false negatives provides a certain level of confidence in the results that cannot otherwise be achieved.

4. **Quality assurance plans**
   a. A plan for doing QA helps us determine which techniques to use, before it is too late to use them. For testing, should answer questions like:
      i. What will be testing?
      ii. How will testing take place?
      iii. When will testing take place, and when will it stop?
      iv. Who will write tests? Who will run them?
      v. Why do we believe the set of tests is good?
   b. Should also include other QA techniques that are used as well, and answer similar questions for each.
   c. Should give evidence for why these are the right QA techniques for this project.
   d. Primary point: QA (or even lack of QA) should be a conscious decision, not an accidental oversight.