Deeper

- How might unit test cases be useful in system testing?
  - *(a) When the external interfaces to a component are directly exposed as system interfaces.*
  - *(b) To verify that a component works correctly when connected to other real (vs. simulated) components.*

Deeper

- Is system testing likely to require additional test cases and tools (beyond those used for unit testing).
  - *It is possible that all functional interfaces can be exercised by unit test cases, but system testing includes much more than functionality.*
  - *Installation testing, interoperability testing, performance testing, and stress testing, however are areas that are likely to require tools beyond those that were used in unit testing.*

Deeper

- Give a few examples of whole system behavior, and explain why they are hard to test at the component level.
  - *Performance can be measured in each individual component, but whole system performance may be driven by interconnections and interactions.*
  - *We can exercise the error handling capabilities of each component, but system robustness may be driven by interactions in these behaviors.*
  - *Security is a "weakest link" problem, and problems are as likely to be in the connections between components as in the components themselves.*

Deeper

- What are developers the "right people" to test the software?
  - *They understand it, and its weaknesses, so they can design better test cases.*
  - *They will be debugging the problems and fixing them, so they might as well find them.*
  - *They are the people who are ultimately responsible for the quality of the software.*
  - *Deferring the finding of bugs until after integration is extremely inefficient.*

Deeper

- Why are testers working with the developers the "right people" to test software?
  - *They have different skills, training, experience and tools than the developers, which should make them much more productive at developing test cases.*
  - *Since they work with the developers, they will still gain most of the advantages of in-development testing.*

Deeper

- Why is an independent test group the "right people" to test software?
  - *They have better testing skills, training, experience, and tools than the developers.*
  - *Operating in a quality-centered group, they are likely to be more systematic and disciplined in their testing and reporting processes.*
  - *Operating in an independent group, they are relatively immune from pressure by development managers to make sure that the software passes.*

Deeper D_2

- On what is our confidence about the capabilities of our software based?
  1) *Our knowledge of the design.*
  2) *Testing we have performed.*
  3) *Experience gained during integration.*
  4) *The bug discovery history of the product.*
  5) *We may have been using it ourselves.*
  6) *Knowing how it has been exercised so far.*

Deeper D_3

- What should we do if we find expectations about which we are not yet confident?
  Design a testing program to exercise those aspects of the software so that we can gain confidence about it.
- How?
  Find people who know about that area.
  Do some research into how other people test this.
  Just make up some basic tests … almost anything is better than nothing, and you will learn from your mistakes.

Deeper E_1

- Why did the first set of test cases flush out more bugs (per test case) than the second?
  (a) Because it was run sooner, and there were more bugs to find.
  (b) Because the first test cases were better at finding more of the mistakes that we actually made.

Deeper E_2

- If we continue running the second set of test cases longer, how many more bugs should we expect to find?
  If we continue the current testing activity, we should expect the bug arrival rate to follow the descending curve it us currently on.
- Does this mean we will soon have all the bugs found and fixed?
  No, just all the bugs that are likely to be turned up by these test cases.

Deeper E_3

- The second set of test cases found fewer bugs than the first. Does this mean that the third set of test cases will find fewer than the second?
  (a) Perhaps, because we are fixing bugs, leaving fewer to find.
  (b) Perhaps not, if the third set of test cases turns out to exercise buggy code that has not yet been exercised.

Deeper E_4

- Is this product ready to ship?
  Can't tell from this graph.
  Is the current bug discovery rate acceptable?
  What fraction of the planned test cases have been run?
  What fraction of the functionality has been exercised?
  How representative do we believe this testing to be of the way the product will be used?

Deeper

- We think new product is ready to expose to users for the first time, what should we expect (in terms of bug discovery).

When a product goes to users for the first time, expect a large spike in the bug arrival rate?

- Why won't the bug rate continue falling, as it has been doing recently.

Bug discovery is down because we have stopped exercising the system in new ways. Users will surely exercise the system in many new ways.

Deeper

- How do we know how long to continue running a particular test suite?

(a) As long as it is effective at finding bugs.

(b) If the tests are deterministic, multiple runs (against the same product version) shouldn't find more bugs than the first run encountered.

(c) Load and stress tests tend to provide greater confidence the longer they run.

Deeper

- How does the Pareto Principle help us?

If most of the cycles are spent in a relatively small number of places, then we may be able to greatly improve performance with relatively minor changes to the code.

Deeper

- Why does the Pareto Principle hold?

The cost of code depends on two things:

(1) how often it is executed.

(2) how long it takes to execute.

Most code is either fairly simple (and executes relatively quickly) or not called very often. Relatively few pieces of code are both expensive and frequently used.

Deeper

- Why must performance management be based on hard measurements?

(a) You cannot optimize what you cannot mesure.

(b) The measurements give us a quantitative assessment of where we are.

(c) The measurements give us objective data about where the costs are coming from.

(d) The measurements give us an objective basis for determining if our changes have made improvements.

Deeper

- Why is this a reality?

Most products have competition. In many products, performance is a critical element of competitiveness. If you are faster than someone else, you can be sure that they are working on changing that.

- Does our software actually get slower?

Yes, because people regularly make changes to the code without considering their performance implications, leading to a "slow death by a thousand cuts".

Deeper

- Why is design usually more important than code optimization?

Relatively little code is called so often that its efficiency makes a real difference in system performance.

The overheads created by bad design can easily dwarf the overheads associated with poorly written code.

Deeper

- Why is performance optimization like debugging?

The program has undesirable behavior, and you have to track down the cause and fix it.

- Why is it harder?

Because you have to understand the whole system, and everything it interacts with.

Because it will never be "good enough".

Deeper

- What adverse consequences could there be to making a poor choice for an alpha site?

(a) They might not provide useful feedback.

(b) They might be a major support burden.

(c) They might leak intelligence to your competition.

Deeper

- Why would a customer want to be an alpha site for known to be flakey software?

(a) To get a sense of how it is going to work, so they can better plan and schedule their adoption.

(b) To have the opportunity to find bugs that impact them and have them be already fixed in the official release.

(c) to get an earlier start on a development project.

Deeper

- Why is it important that ship criteria be established at the start of the project?

In the absence of established criteria, the default criteria will almost surely be:

(a) the date the customer was promised

(b) the date that secures the ship bonus

(c) the date the press was told to expect it on

If any other criteria are to stand a chance of being considered, they must be established before the pressure for delivery is felt.

Deeper

- What does it mean for a component to be ready to integrate?

(a) It is complete and fully complies with all of its specifications.

(b) It is sufficiently complete, and working well enough that it would be productive to test it with other components.

- How do we make this determination?

With unit test cases that exercise the full range of its specified functionality and correct error handling.

Deeper

- How do we test components before formal integration?
  1. *With external testing harnesses.*
  2. *With simulated versions of other components.*
  3. *In test integrations, where we build our own version of a complete product.*

Deeper

- How does this differ from "system testing"?
  - *We are still trying to determine whether or not particular components (including their interface implementations) are working.*
  - *We are exercising and observing component functionality.*
  - *We are testing it against component level specifications.*

Deeper

- Is there a "natural" group do component level functional validation?
  - *Because of the technical skill required to define the right test cases, and the interaction required to diagnose problems, it seems clear that this will be done much more efficiently in the development group.*
  - *Whether there should be dedicated test developers within that group depends on whether or not the testing tools and methodology are so complex as to require experts.*

Deeper

- Is there a "natural" group to do system level validation?
  - *If the individual components are already well tested, the code knowledge of the developers is less critical.*
  - *If the system testing tools and techniques are very different from those used in component testing, this might argue the need for different people.*
  - *The fact that this is "acceptance" testing argues for an independent group.*

Deeper

- Is there a "natural" group to do robustness and performance testing?
  - *Yes … but it is probably none of the above.*
  - *The testing tools and techniques are fairly sophisticated and require people with additional training.*
  - *The diagnosis of performance problems requires a good understanding of the software in question, at both the architectural and component levels.*
  - *People who work in these areas tend to be specialists, and are highly valued.*

Deeper

- How do we know all of the things that will be demanded of our product?
  1) *The most important things should be spelled out in the requirements.*
  2) *Interviews with and observations of customers should give us additional information on how it will be used.*
  3) *Prior experience with similar products will suggest other things that commonly go wrong.*
  4) *A knowledge of current events can also be helpful (e.g. worm attacks, privacy issues).*

Deeper

- If we this is a minor update to a mature product, and we are just finishing system test, what should we expect the bug rate to do.

If this is a mature product, we may already have very good test suites, and may not encounter many surprises at all when we give it to users. The bug arrival rate could rise or just continue on its downward trend.

Deeper

- Why shouldn't we optimize code if we are certain that it is terribly written?

(a) It may never be called, and so may not actually be impacting our performance.

(b) Performance optimization is about "bang for the buck". You want to find the simplest thing that accounts for the greatest cost. Most interesting software is far to complex for our intuition go guide this search.

Deeper

- What considerations should go into selecting (real) beta sites?

(a) A wide cross-section of users, who will exercise the system in different ways.

(b) People who will provide us with high quality feedback.