Deeper

- What does agile process tell us about a product built by independent (actually hostile) organizations that don't communicate well?
  - *It is a disaster in the making.*
  - *A major premise of agile process is that close communication and collaboration will solve problems faster and better than Big Design Up Front.*

Deeper

- Why is it so necessary to completely specify all components and interfaces up-front?
  - *Because the contractors don't communicate well with one another … or even with the customer for that matter.*
  - *If you can't get them to cooperate in the development, the only chance for success is to so completely specify each component that they cannot help but work when combined.*

Deeper

- What goes wrong when independently developed components are integrated?
  - *Misunderstood specifications give rise to interoperability problems.*
  - *Bugs emerge in features that could not be fully tested until the whole system was assembled.*
  - *Problems are hard to diagnose because everything is new (and flakey).*
  - *We find specification errors that necessitate the redesign of completed components.*

Deeper

- Is there any reason that most software products require an "integration phase"?
  - *Most software products are built by a single organization. There should be good communication.*
  - *The cost of automated software building and testing is nearly zero. There is no economic reason not to have daily builds.*

Deeper

- How much work is it to construct a dummy version of a component?
  - *Initially, a few minutes of work.*
  - *Later, if further dummies are needed (for top-down integration), there might be more work involved to make them work well enough to enable the higher level code to be tested.*
- Are they likely to pay for themselves?
  - *If they actually enable incremental integration.*

Deeper

- What does an automated build include?
  - *Start with a clean machine, extract all current sources from a version control system, build the entire product from scratch, and then run some level of automated testing (at least a "smoke" test).*
- Why must it be completely automated?
  - *So that it happens regularly and problems are found promptly.*

Deeper

- Why is it good to be able to build a whole system from day one?
  - *That creates a framework into which incremental check-ins can be added.*
  - *If the system builds successfully from day one, changes that break the build will immediately become obvious.*
  - *If we are (at any level) testing the system from day one, regressions will be immediately obvious.*

Deeper

- Why are problems found sooner?
  - *Because we combine components as soon as possible, giving problems an earlier opportunity to emerge.*
- Why are problems found more quickly?
  - *Incremental integration means that only a relatively small amount of code in each build is new … and problems are likely to be in the newer code (or at least in code that is being newly exercised).*

Deeper

- Why is "spreading problems out" a good thing?
  - *It is difficult to find a bug when a dozen bugs are introduced at the same time and we are confronted by the union of their symptoms.*
  - *It permits bugs to be found sooner, and hopefully before other code is based on them.*

Deeper

- How does incremental integration improve the predictability of quality and schedule?
  - *With phased integration we don't know how much trouble we are in until the start of integration.*
  - *With incremental integration we find problems sooner and have the opportunity to find problems one at a time. The amount of code successfully integrated is a good measure of progress.*

Deeper

- Why does incremental integration result in less wasteful re-engineering?
  - *If we don't find a specification problem until final integration, we may find it necessary to redesign a component that is already complete.*
  - *If we have the opportunity to combine components early in their evolution, interoperability and specification problems will be found sooner, and fixed sooner … before further work can be built on top of them.*

Deeper

- What are the advantages and disadvantages of top-down integration?
  - *+ It works well with incremental integration, since there is always something to build.*
  - *+ The completed higher level components will often act as a test harness for newly integrated lower level components.*
  - *- Making higher level components work without lower level components requires the creation of numerous test stubs.*

Deeper

- What are the advantages and disadvantages of bottom-up integration?
  - *Each class can be thoroughly tested before it is integrated (because the classes it depends on have preceded it), which means that the integrated code is more likely to be correct.*
  - *The problem is that it puts of the integration of the highest level components until last … and this is where the worst problems are likely to be.*
  - *It also necessitates the creation of numerous test drivers (analogous to the stubs in top-down integration).*

Deeper

- Why would exposing code to others ASAP be a good thing?
  - *Once you have integrated, you no longer have to worry about other people who might be working on conflicting changes. Integrating sooner means you find out about conflicts sooner.*
  - *Other people can see and work with your updated component ASAP, accelerating the finding and fixing of problems.*

Deeper

- If a product cannot integrate until it is complete, isn't this a phased integration model?
  - *The integration into the build space is incremental, and has most of the benefits of incremental integration.*
  - *The difference is not so much in when you put-back as it is in what you put-back into. The train model introduces staging areas.*

Deeper

- Why do we need to perform unit (component) testing before we start testig the whole system?
  - *Bugs can be found much faster when a smaller amount of code is being tested, directly, by the people who just finished building it.*
  - *After the components are combined together it may be much harder to figure out which one is at fault.*

Deeper

- What types of unit (component functionality) testing would require other components?
  - *Implementations of needed services.*
  - *Clients to exercise servers.*
  - *Databases.*
  - *Anything with which a component interacts.*

Deeper

- Why does it matter whether we write (or design) the test before or after the code it is to test?
  - *Designing the test cases forces us to spend a bit more time thinking about what correct functionality would be.*
  - *This exposes requirements problems sooner.*
  - *A better understanding of correctness usually leads to a more correct implementation.*

Deeper

- Why do we check to make sure that the component fails a newly written test?
  - *The new test should be testing new functionality.*
  - *If the component passes the test, before the new functionality is added, the test case is probably not valid.*

Deeper

- What do you think of TDD as a <u>design discipline</u> (vs. a coding discipline)?
  - *(a) Designing features, one at a time, is in keeping with XP's code-for-today philosophy … but encourages coding before you complete the overall design.*
  - *(b) If the test case does not fully reflect the requirements, there is a danger of coding to the test case rather than the requirements. This is why test cases must be reviewed.*

Deeper

- Why should unit tests be automated and saved?
  - *It is more work to write an automated test case that will continue working in the future.*
  - *But the accumulate to be a very complete set of functional and regression tests.*
  - *We will someday change this code, and when we do, we will be glad we still have the old unit test cases.*
  - *More (distinct and valid) testing means greater confidence.*

Deeper

- Why is it important to test the whole program, as it will be shipped?
  - *Because test harnesses for testing individual components may not work the same way as the real components.*
  - *Testing on the program we plan to ship will give us better confidence about how that program will perform for customers.*

Deeper

- Why is there <u>less</u> work building test harnesses?
  - *Because the program is the harness into which its components plug. Particularly if we do a top-down integration.*
- Why is there still <u>some</u> work building test harnesses?
  - *We are still doing incremental integration, and will be testing some parts before others are complete. The missing parts must be stubbed or simulated.*

Deeper

- What does "key internal state" mean?
  - *State we would need to query in order to determine whether or not a function worked correctly.*
- How might we expose "key internal state"?
  - *A diagnostic option could dump this information out at a specified time.*
- Would this be useful for debugging too?
  - *You're kidding, right?*

Deeper

- What do we mean by an “interesting situation”?

  Any situation that would cause the execution of the code we want to exercise.

Deeper

- How could a test harness enable us to have better control and observability of the tested component?

  Because all other components (both calling and called) are simulations that we write. As such, we can make them do anything we want.

Deeper

- If we had already done thorough (in vitro) testing of all sub-components, why would we still need to test the “whole program”?

  (a) Because the whole program is what we will be delivering.

  (b) Because our component interaction simulations may not have been faithful.

Deeper

- How is using dummy transaction generators like in vivo testing?

  Because we are exercising the whole program that we will ship.

- How is using dummy transaction generators like in vitro testing?

  Because we can completely control everything they do, and so can use them to test a wide range of normal, error and load functions.