Deeper

- Are bugs the same thing as defects?
  - *Most defects are indeed bugs … code that does not properly handle input or situations.*
  - *There may be an argument, however, that some defects are merely elements of the problem, and not bugs. Consider a distributed locking protocol, where a host going down in the middle of a transaction can cause all kinds of problems. This is certainly a susceptibility to failure. Is it a bug?*

Deeper

- How does this notion of fault compare with the more familiar usage (an exception)?
  - *They do coincide in some situations (e.g. zero divide or following a garbage pointer).*
  - *As the words (fault and exception) are commonly used, however, they do not refer to the incident, but to the means by which the incident is reported to the program.*

Deeper

- Why is it useful to distinguish defects from the events that exercise them?
  - *Defects are susceptibilities to error. If the triggering events never happen, the defect will never give rise to an error.*
  - *We can reduce errors by redesigning a component to make it less susceptible to a particular problem.*
  - *We can also redesign the system to make the triggering events less likely (or impossible).*

Deeper

- Why might robustness be a better goal than correctness?
  - *a) Perfect correctness is difficult to achieve, and we may not be able to tell whether or not we have achieved it.*
  - *b) Not all errors originate within our components. We must also be able to deal with error situations that arise externally.*
  - *c) Investments in robustness often pay off better than investments in correctness.*

Deeper

- Can a very reliable component have poor availability?
  - *Yes, if it does occasionally fail, and once it fails, it stays broken for a very long time.*
  - *example: the primary mirror on the Hubble*
- Can an unreliable component deliver good availability?
  - *Yes, if the system automatically detects and corrects or works around those failures.*
  - *example: TCP error detection & retransmission*

Deeper

- Why do those "in the know" characterize reliability with FIT rates (vs. MTBF)?
  - *1) FIT rates from independent sources add*
    - tires: front 25,000, rear 15,000, spare 5,000
      - corresponding MTBFs are: 4.5, 7.5, 22.7 years
    - FIT rate for all five is 85,000 (/billion hours)
      - *does anybody care to calculate the MTBF?*
  - *2) MTBF is highly misleading*
    - US four year olds have a mortality of 3/10,000
    - this yields an MTBF of 3,333 years
    - vs. the FIT rate of 34 (/billion hours)

Deeper

- Where does the definition of correctness (for code) come from?
  - *It comes from the requirements, architecture, specifications, and design.*
- What form would such definitions take?
  - *pre-and post-conditions about procedures*
- When should these be specified?
  - *when we are defining the class or routine*

Deeper

- Where does the definition of data validity come from?
  - *It comes from the requirements, architecture, specifications, and design.*
- What form would such definitions take?
  - *assertions about data values and their relationships to one-another*
- When should these be specified?
  - *when we are defining the class or data structures*

Deeper

- How can pre- and post-conditions improve our implementation?
  - *If we have them in mind while writing the code, they are more likely to be met.*
- How can we ensure or verify data validity assertions?
  - *With explicit run-time checks.*
- How do we know if a request succeeded?
  - *By checking the return code.*

Deeper

- Why is it important to detect errors as quickly as possible?
  - *(a) The closer to the source we detect the problem, the more likely we are to properly diagnose it.*
  - *(b) We would like to detect and respond to the error before it has the opportunity to affect other computations.*
  - *(c) The sooner we detect a problem, the sooner we can initiate repair/recovery.*

Deeper

- Why are a smaller number of more general checks preferable to a larger number of more specific checks?
  - *(a) They may execute more quickly.*
  - *(b) They may detect a wider range of problems (more than the sum of the more specific checks).*

Deeper

- Why should error checking code be as simple as possible?
  - *Increasing the complexity of a program rarely improves its reliability.*
  - *Because complex error checking code is more likely to have bugs in it, and bugs in the error checking code can be just as severe as bugs in the code they are supposed to monitor.*

Deeper

- Give an example of an error that would be difficult to test with simple assertions?

Agreement between student programs and class rosters is difficult to test with in-code assertions because they are in different objects, updated in distinct transactions.

- How could we check for this problem?

After the completion of each enrolment or drop transaction, we could go back and recheck the consistency of the program and roster for this class.

We could periodically run an audit of the student programs and course registrations to review all of the student programs against all of the class rosters.

Deeper

- Give an example of a symptom that could have many causes?

If we don't get a response from an RPC request, the client process could be wedged, or the client host down, or there could be a network failure.

- How could we diagnose such a problem?

Each host could monitor the responsiveness of its own processes.

Host-host heartbeats could monitor the responsiveness of each host.

A cluster membership algorithm could be used to infer link failures (a can talk to b, but c cannot).

Deeper

- Why do we need to know the source, impact, and persistence of an error?

Because this information will enable us to determine how to most reasonably respond to the error (e.g. retry the operation, fail this operation, rebind to a new server, give up, etc).

Deeper

- What does it mean to say that much diagnosis can be done at design time?

In many cases, when we get an error response to a particular request, the error code provides us much of the needed information.

Even if it is not obvious what the impact or persistence of a problem is, a rational retry strategy may work in most recoverable situations, while doing little harm in unrecoverable situations.