Deeper

- How could a test case not be dispositive?
  - *(a) If it reported what the component did, without labeling that performance as correct or incorrect.*
  - *(b) If it measured only a subset of the component's behavior, and it was not possible to determine (from that alone) whether or not the component was working.*

Deeper

- How could a test case not be valid?
  - *(a) If the behavioral expectations measured by the test case did not align with the requirements.*
  - *(b) If the manner in which the component exercised the component was incorrect.*
  - *(c) If the test case interpreted the component's responses incorrectly.*

Deeper

- Why might a test case yield non-deterministic results?
  - *(a) Because the results were dependent on the state of the component at the start of testing, and the test case did not fully/properly initialize the component's state.*
  - *(b) Because of legitimate variability in component performance that the test case was un-prepared to correctly measure and analyze.*

Deeper

- Why should test cases be isolated?
  - *(a) to make it possible to run only selected test cases.*
  - *(b) So that changes made to one test case cannot break others.*
- How does one isolate test cases?
  - *Any test case (whose results depend on the state of the tested component) must (as part of its set-up procedure) explicitly put the component into the required state.*

Deeper

- Why should test cases be automated?
  - *(a) So that they are always run the same way (correctly).*
  - *(b) So that they can be (inexpensively) re-run regularly (e.g. after every change).*
- What would make something hard to automate?
  - *(a) Depending on input that is difficult to simulate (e.g. mouse motion).*

Not just execution, but reporting as well!

Deeper

- What is the problem with a test that is not complete or self-contained?

If additional components need to be installed, configured, and run …

- *(a) it is more difficult to run the tests, and so they will be run less often.*
- *(b) there is a chance those steps will be done improperly, invalidating the results of the test.*
- *(c) determinism of results may depend on consistency of the tools and their configuration.*

Deeper

- Black Box testing is like testing blind-folded. Why does this make sense?
  - *(a) It is testing against the requirements, which are our best definition of correctness.*
  - *(b) Focusing on how a component will be used may generate very different test cases than focusing on how it operates.*
  - *(c) It particularly makes sense if testing is being done by an Independent Verification and Validation group.*

Deeper

- Is it possible to completely test a component using black-box techniques?

If both of the following are true …

(a) If the specifications are complete (in that they cover all operational situations).

(b) If the test cases are complete (in that they test the full range of all specifications).

Then it could reasonably be argued that any additional test cases derived by white-box techniques would be redundant or unnecessary.

Are ya feelin lucky?

Deeper

- Why are boundary values interesting?

They explore the equivalence partitions defined by the specifications.

Many bugs are found near the transitions from one part of the parameter range to another.

- How can white box testing improve our boundary value analysis?

Rather than merely testing subsets of the parameter domain, we can try to exercise every side of every internal test.

Deeper

- Why do we care what the equivalence partitions are?

In principle, we only need to test one combination of parameter values from each equivalence partition.

- How does white box analysis help us identify the equivalence partitions?

Tests in the code make equivalence partitions easy to recognize. Specifications may not.

Required Test Cases

- insertValue
  - empty list
  - before 1st element of list
  - between existing nodes
  - after last element of list
  - value already in list
- deleteNode
  - value present
    - first value
    - between two values
    - last value
    - only value
    - value occurs twice
  - value not present
    - empty list
    - lower than first
    - between two values
    - after the last

Deeper

- How does automated code coverage instrumentation work?

Code is broken up into execution "segments".

A counter is defined for every code segment.

A preprocessor inserts code in front of each code segment to increment the associated counter each time that segment is implemented.

An instrumentation package saves the counters to a file and permits the results to be browsed along with the corresponding source code.

Deeper

- What does it mean to verify coverage and result?
  - *Verifying coverage means verifying that we did indeed execute the code in question.*
  - *Verifying result means verifying that the code did the right thing when we executed it.*
- How cold we verify coverage?
  - *With a code coverage tool, or by the program's behavior (if detectable operations were performed by the exercised code).*

Deeper

- Justify each of these metrics:
  - ***code segments***
    - *minimum number of required test cases*
  - ***code paths***
    - *mirrors conceptual combinatorics of program logic*
  - ***call fan-out and depth***
    - *deeper systems may be harder to understand*
  - ***number of interfaces and parameters***
    - *variations & parameters complicate interfaces*

Deeper

- What kinds of output, how to observe?
  - *return values — we can capture them*
  - *output messages — we can put in a file*
  - *database — we can read*
  - *files, directories — we can browse*
  - *messages — ?simulated partner?*
  - *other system services — ???*
  - *other internal state — ???*

Deeper

- What factors, and how can we drive?
  - *parameters — we can pass them*
  - *input data — we can get from a file*
  - *database — we can initialize*
  - *files, directories — we can create*
  - *messages — ?we can simulate?*
  - *signals — ?we can simulate?*
  - *errors — ?we can simulate?*
  - *other internal state — ???*

Deeper

- How could interaction or error simulations be unrealistic?
  - *Simulated interactions might not correspond to realistic sequences of operations.*
  - *Simulated error returns might not be accompanied by a consistent set of symptoms.*
  - *In either case, an unrealistic simulation might not reasonably exercise the tested component.*