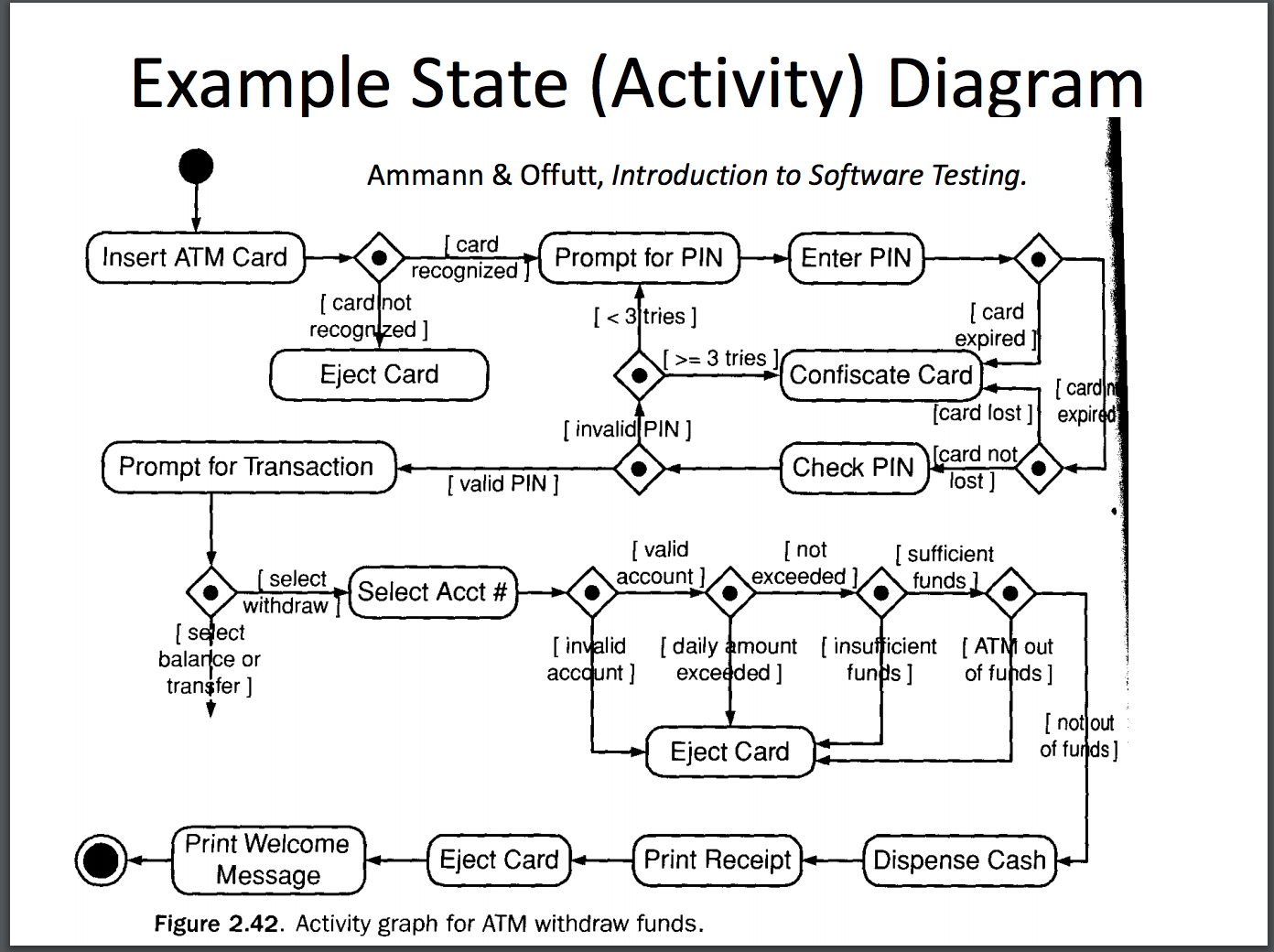
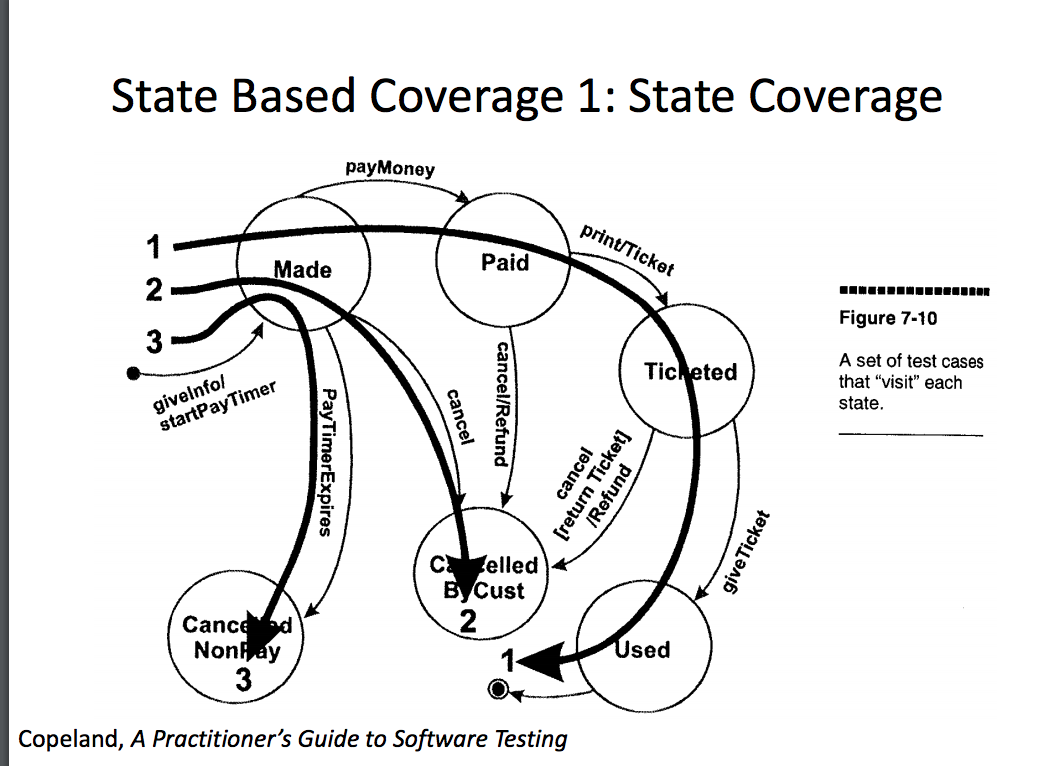
Coverage Testing: A Brief Glimpse

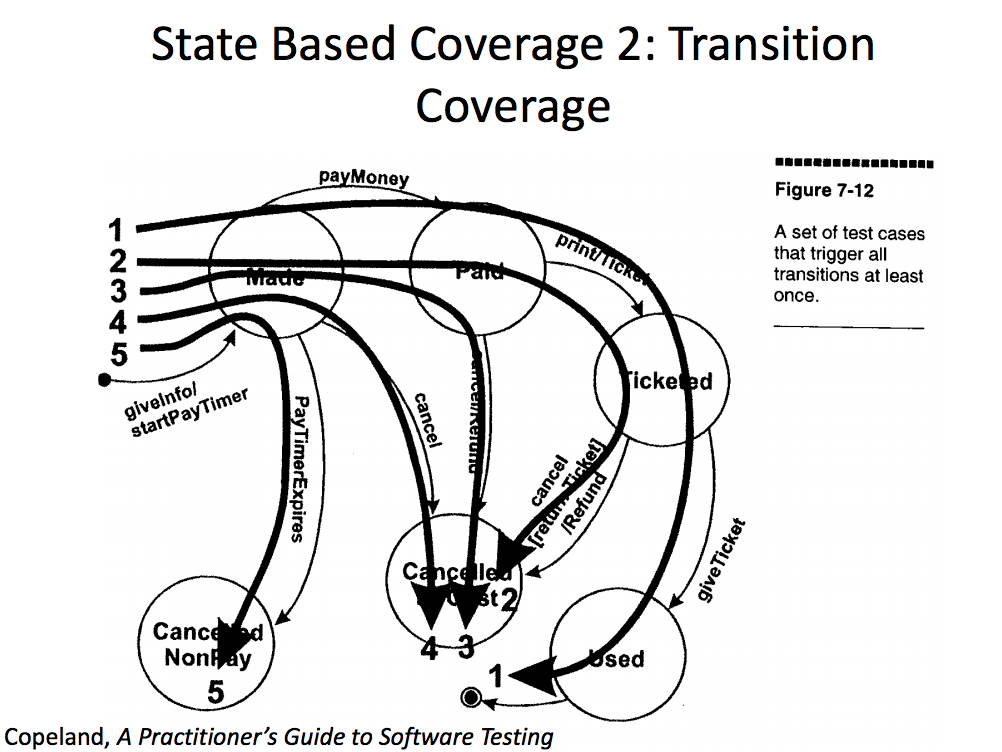
1. The many faces of “Coverage”
   1. State-based (node) coverage
   2. Transition coverage
   3. Prime Path
   4. Constraint-based coverage
      1. Checking combinations of preconditions and post-conditions
   5. Code coverage
   6. Coverage testing can occur at either the “black box” level or at the “glass box” level
2. Example State Diagram

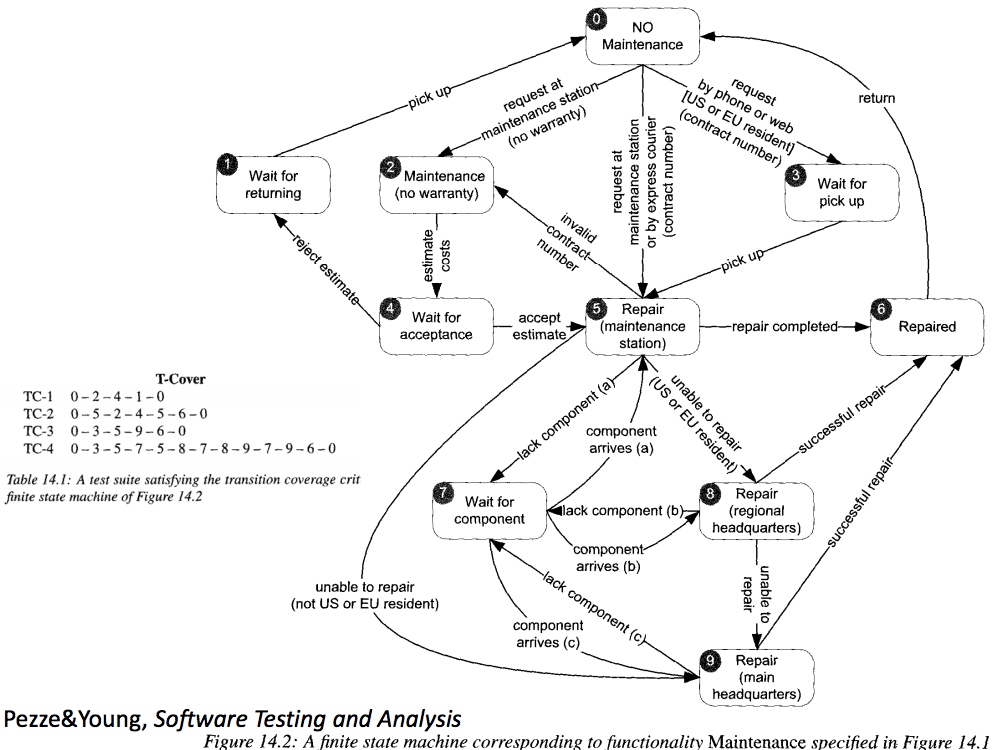


1. State based Coverage 1: State Coverage

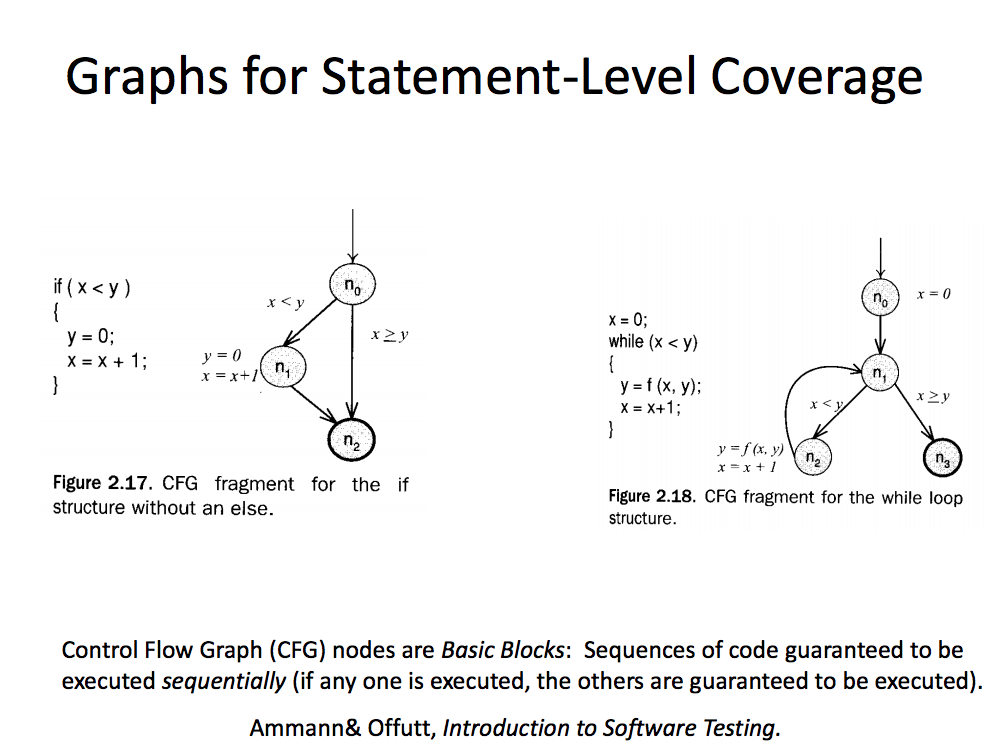


1. State based coverage 2: transition coverage

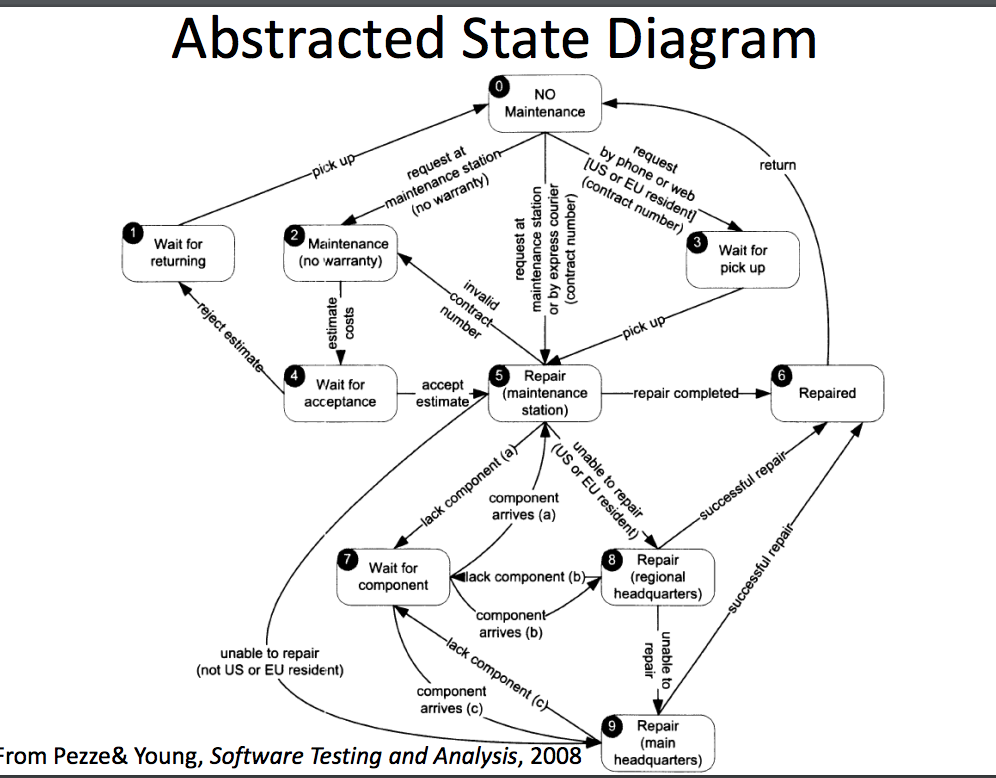




1. Graphs for statement-level coverage



1. Key Steps in Coverage Procedure
   1. We identify the set of things we need to cover (e.g., nodes, transition, prime path)
   2. We develop a set of abstract scenario (e.g. paths) that include all of the things we need to cover
   3. We develop a set of concrete test cases that cover all of these paths
      1. Note that these test cases may exercise additional paths not included in the specified path
         1. Sometimes this is required to assemble feasible test cases
      2. These are the actual tests you’d perform to realize the paths – and thereby achieve the desired converge
2. Basic of Coverage – based Path testing
   1. For a given situation, coverage techniques let you derive legal paths through the program (from start to finish) that will exercise the desired level of coverage
      1. E.g.
      2. Hitting all state
      3. Hitting all transition
      4. Hitting a distinct set of paths by which code could reach a given state
   2. Will then seek find a small set of specific test cases that will exercise these legal paths
      1. These cases use particular inputs to achieve the path
3. Advantages of Flow Graph Models in testing
   1. Standard advantages of models (clear documentation, ability to reason about and communicates)
   2. Ability to derive test cases
   3. If precise enough ability to automatically test cases against model specification
4. Abstracted State Diagram



1. Examples of Abstraction
   1. Representation of history-dependent state
   2. Omission of excessively large state space in visual description
      1. Outer behavior: represents small number of router states, ignoring message state space
   3. Omission of (important!) error transitions or (potentially important) conditions perceived as impossible
2. State Coverage
   1. Path taken reach each node in the graph at least once
   2. This is the simplest and weakest coverage criterion
   3. Testing gaps?
      1. Common coding defects: code works fine based on one path through code, but fail on another path
         1. Pointer/ref initialized on one path but not on another
         2. Variable left uninitialized on one path
3. Example of Test Gap for state coverage
   1. We could have complete state (node) coverage here but still not uncover the possible null pointer exception
4. Transition (Edge) based testing
   1. Here we are exercising all transitions at least once
   2. Many test paths may cover a particular edge
   3. For each path, we will have to find a specific situation that will exercise that path
   4. Sometimes need to watch out and reject paths that are impossible (for which one cannot find a case, due to information not in the diagram)
   5. must cover each transition between states/block
   6. Testing gaps
      1. Haven’t covered combinations of conditions (e.g. where come in via a certain path and leave via a certain path)
5. Transition-Tuple Coverage
   1. Must cover all paths of length n (involving tuples of transitions) between state/blocks
6. Functional testing
   1. Most of our examples thus far have been drawn from functional (black box) testing
   2. As discussed, such tests have many advantages, including the fact that they do not require development tools and training
   3. The same basic graph-coverage testing can be used for low level testing
7. Lower level Coverage
   1. Lower-level testing (based on control flow graphs) uses similar methods to graph-based functional testing, but tests other things
   2. Tests implementation details
      1. Uses knowledge of structure (NULL condition)
      2. Pieces of implementation of same feature may occur in multiple places – all exercised by functional testing
8. Testing coverage of functional tests
   1. Although functional tests are specified using criteria above the code level, it is worth testing the coverage of functional tests at the code level
      1. Identifying under-specified areas of functionality
         1. Code handles many cases that are not discussed in functional specification: need to improve functional specification
      2. Identifying cases where code includes many undocumented (any possibly unrecognized features)
      3. Too limited a functional test set
9. Prime Path
   1. Simple path; a path p from node a to b is simple if p does not contain any repeated node except for 9optionally) case a = b
      1. We can compose any path out of simple path
      2. This is useful, but there are typically a huge set of such simple paths present a program
      3. Key goal: reduce needless repetition
   2. Prime Path: A simple path p from node a to b is a prime path if it is not contained as a sub path of any other simple path
      1. This is a basically a maximum length path that doesn’t contain repetition of loops
10. Logic testing
    1. Logical conditions often determine state transitions / control flow
    2. When discussing path, we have abstracted away from the issue of what conditions will force different path to be taken
    3. Sometimes combinations of paths are impossible due to logical conditions
    4. In addition to confirming that the code works when different combinations of paths are taken, we also need to make sure that the appropriate paths/actions are taken under the right conditions
11. Challenges with structural logic coverage
    1. Determining logical conditions to test (see next)
    2. Reaching to place where test occurs
       1. Use understanding of path
    3. Determining test cases that will exercise the path
12. Basic terminology
    1. We are dealing with Boolean quantities b/c they form the basis for most control flow decisions
    2. Predicate: expression that evaluates to Boolean
       1. May include logical operation
          1. ((a>b)) with C) jiao p(x)
    3. clause: predicate without Boolean (logic) operators
13. Predicate Coverage
    1. Test one case when true, one when false
    2. Far too weak: not testing conditions that lead to predicate being true
    3. Similar to node coverage for control-flow: we test that a state seems to be present, but no how that comes about
14. Clause Coverage
    1. For each c in C, T R contains two requirements; c evaluates to true, and c evaluates to false
    2. Isn’t guaranteed to test effect of different values of clause in actually changing the predicates
       1. The predicate as a whole may just retain the same value due to values of other clause within the predicate