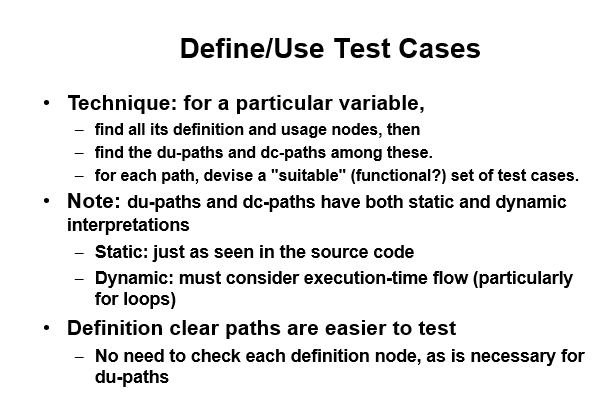
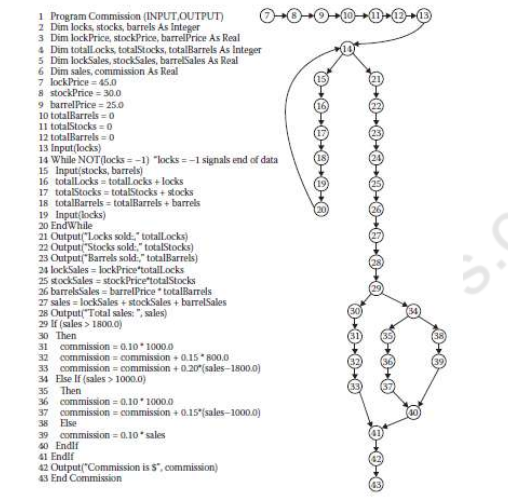
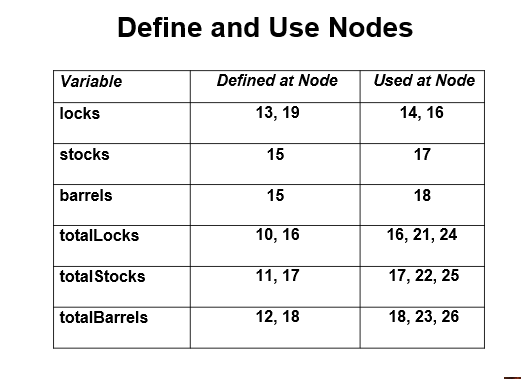
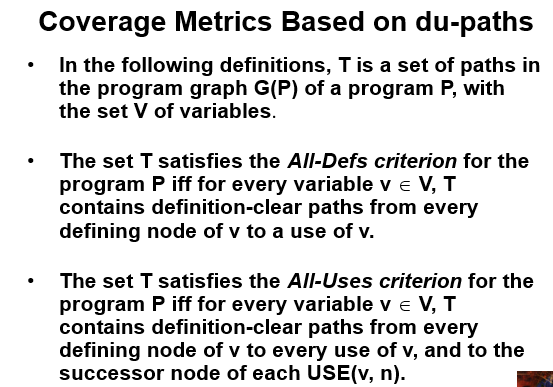
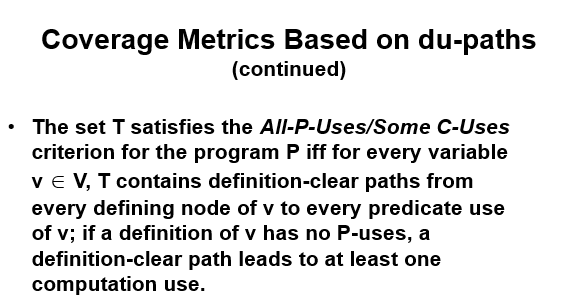
5)

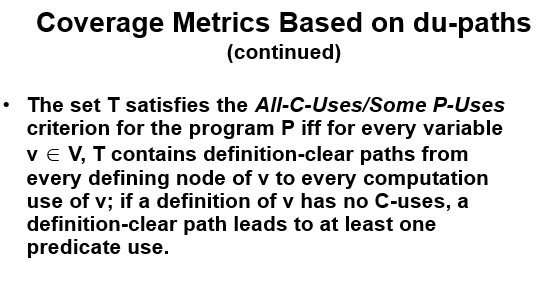


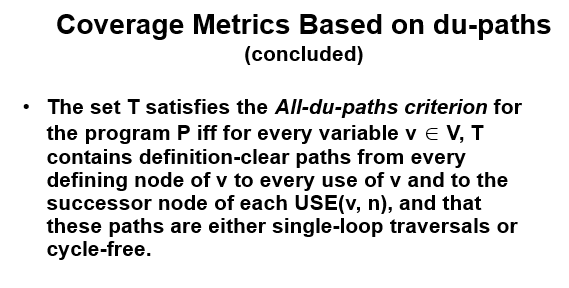


6)

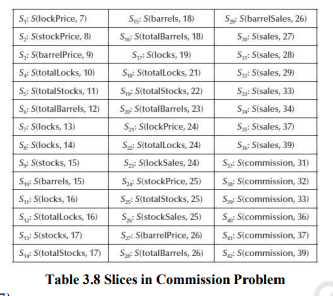


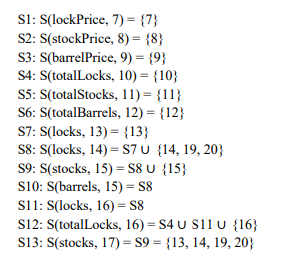


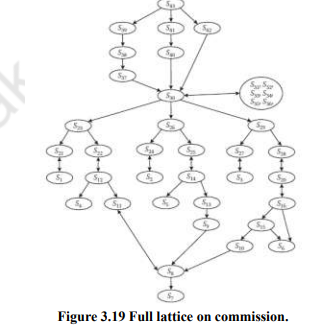
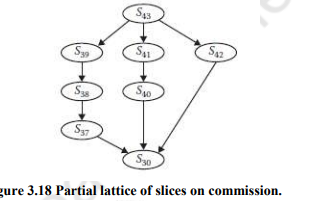
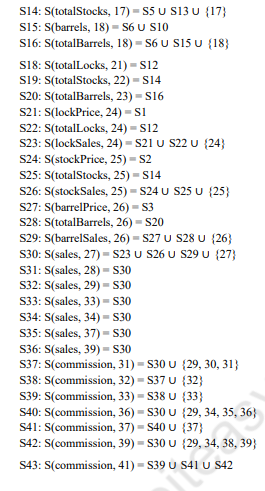




7)







4)

McCabe's insight that testing criteria serve to measure the quality of testing rather than act as procedures for identifying specific test cases highlights a crucial distinction. This observation implies that while criteria provide benchmarks and standards for evaluating the effectiveness of testing efforts, they don't prescribe the detailed steps to generate individual test cases. Let's apply this perspective:

Criteria as Quality Measures:Testing criteria, such as code coverage metrics or complexity measures like cyclomatic complexity, are essential for assessing the thoroughness and effectiveness of testing.

Role of Criteria in Assessment:Teams can use these criteria to assess the quality of their testing strategy and identify areas that may need improvement.

Flexibility in Test Case Selection:The absence of a direct link between criteria and specific test cases allows for flexibility in choosing testing approaches.

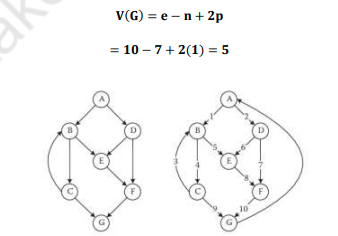
Adaptability to Changing Requirements:Test cases can be adjusted and expanded to accommodate changes without altering the fundamental criteria used to evaluate testing effectiveness.

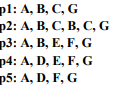
Combining Criteria for Comprehensive Assessment:Teams can use multiple criteria in combination to gain a more comprehensive view of the testing process.

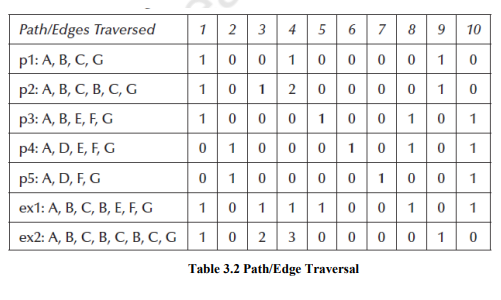
Continuous Improvement:Recognizing testing criteria as quality measures encourages a culture of continuous improvement.

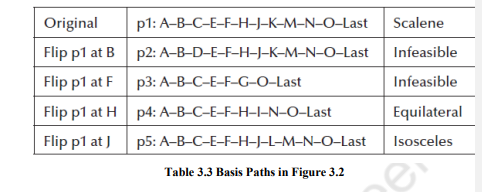
2)

McCabe's testing view is based on this, stating that the cyclomatic number of a strongly connected graph is the number of linearly independent circuits in the graph. The confusion in literature regarding the formula for cyclomatic complexity is attributed to the transformation of a directed graph to a strongly connected one by adding an edge from sink to source node. The formula discrepancy is explained as a result of counting or not counting the added edge, affecting the coefficient of the number of connected regions (p). Resolving the inconsistency, the number of linearly independent paths from the source to sink node in the original graph is crucial for understanding cyclomatic complexity.









3)

1. **Essential Complexity and Cyclomatic Complexity:**
   * McCabe's work on cyclomatic complexity contributes to both programming improvement and testing.
   * The concept of essential complexity, introduced by McCabe in 1982, is central to this discussion.
   * Essential complexity is essentially the cyclomatic complexity of a condensed graph derived from structured programming constructs.
2. **Condensation Graphs:**
   * Condensation graphs are a way of simplifying existing graphs.
   * Previous simplifications involved removing strong components or DD-paths.
3. **Structured Programming Constructs:**
   * The approach presented in Figure 3.8 involves condensing around structured programming constructs.
   * The goal is to identify and collapse the graph of each structured programming construct into a single node.
4. **Condensation Process (Figure 3.9):**
   * The process starts with the DD-path graph of a pseudocode triangle program.
   * The if–then–else construct involving nodes B, C, D, and E is condensed into node a.
   * Three if–then constructs are condensed onto nodes b, c, and d.
   * The remaining if–then–else (corresponding to the IF IsATriangle statement) is condensed into node e.
   * The result is a condensed graph with cyclomatic complexity V(G) = 1.
5. **Generalization for Well-Structured Programs:**
   * Well-structured programs, composed solely of structured programming constructs, can always be reduced to a graph with one path.
   * The graph in Figure 3.7 cannot be reduced in this way due to violations of structured programming principles.
6. **Unstructured Programming Violations (Figure 3.10):**
   * McCabe identifies elemental "unstructured" constructs that violate structured programming principles.
   * These violations, shown in Figure 3.10, contain three distinct paths, in contrast to the two paths in corresponding structured programming constructs.

