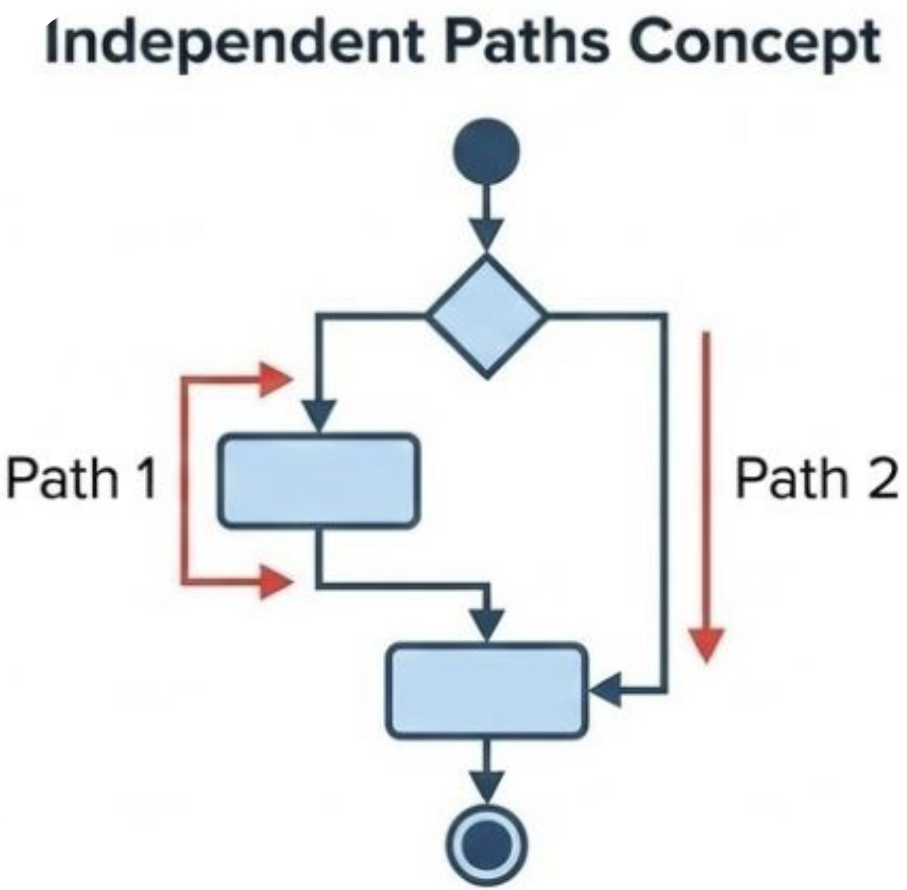
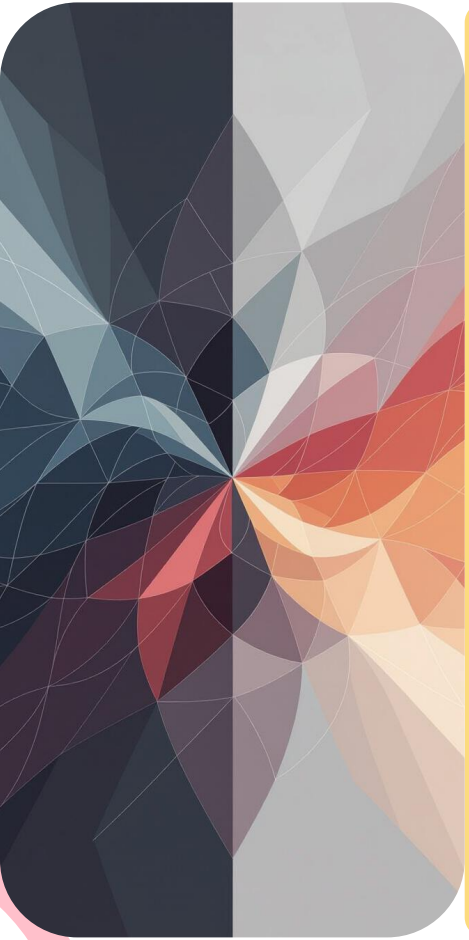
Cyclomatic Complexity

Cyclomatic complexity is a **software metric** that measures the **complexity of a program**. It counts the number of linearly independent paths through a program's source code.





Importance of Cyclomatic Complexity

Why does it matter?

- **Maintainability:** Lower complexity makes code easier to understand and modify.
- **Testing Effort:** High complexity means more test cases are needed to achieve adequate coverage.
- **Bug Detection:** Complex code is more prone to errors and harder to debug. Strive for simpler code to minimize bugs.

Calculation of Cyclomatic Complexity

Formula

M = E − N + 2P

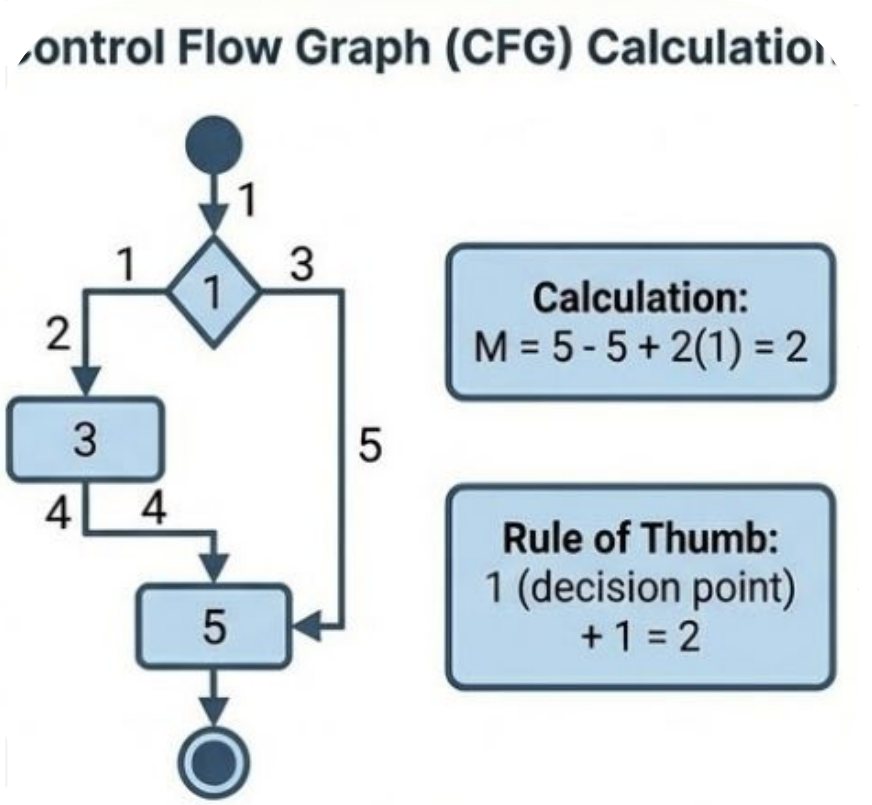
Variables

M = Cyclomatic Complexity

E = Number of edges in the control flow graph

N = Number of nodes in the control flow graph

P = Number of connected components



Rule of Thumb: Count the number of decision points(if, for,while,switch etc) and add 1

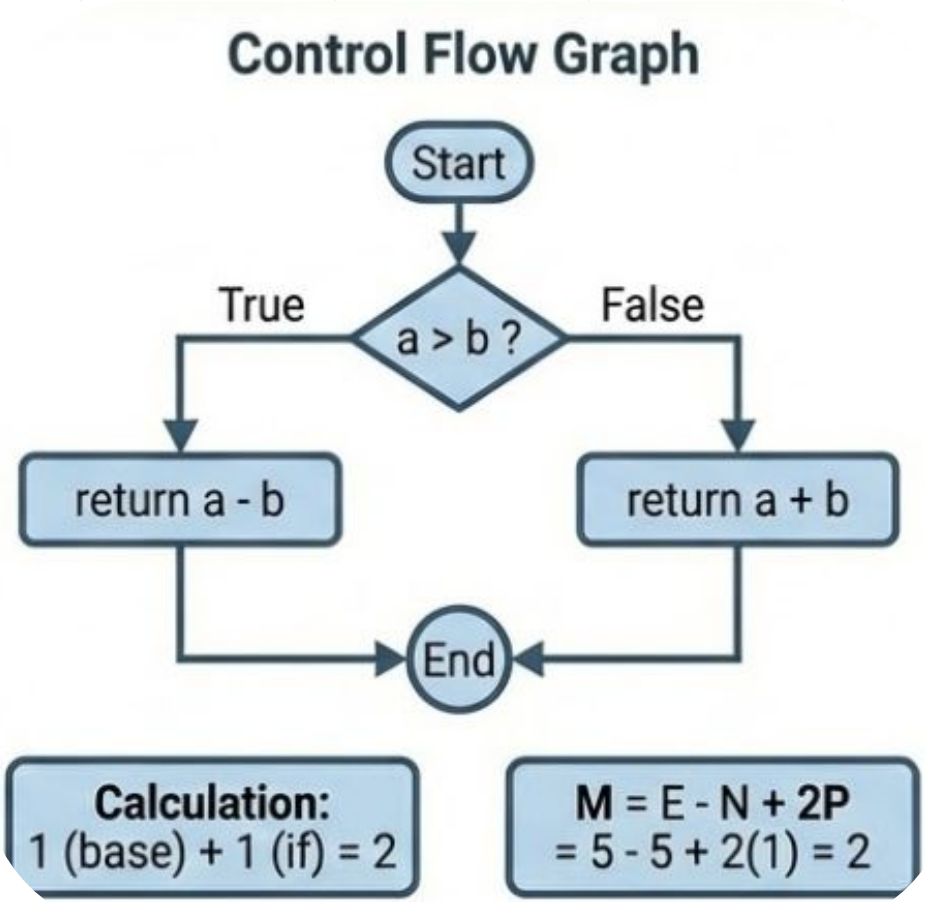
Example: Cyclomatic Complexity Calculation

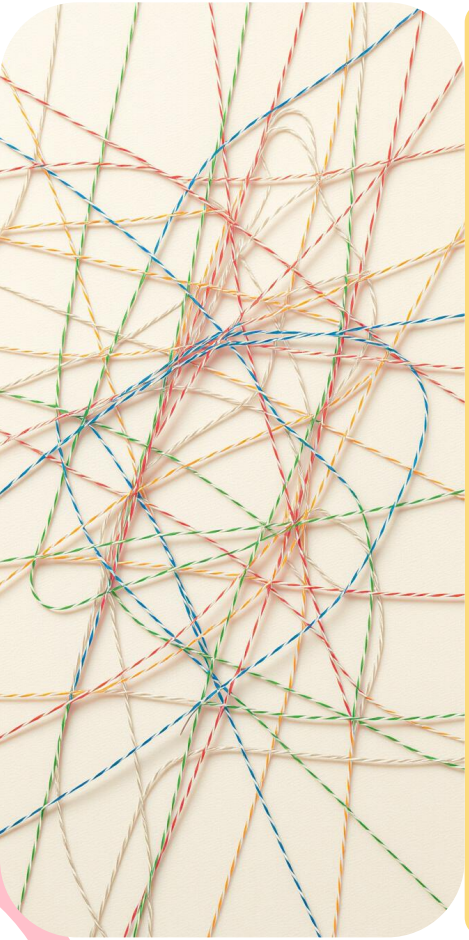
Code Snippet (Illustrative):

```
public int example(int a, int b) {
if (a > b) {
   return a - b;
}
return a + b;
}
```

Control Flow Graph: (Imagine a simple graph with nodes for each statement and edges for the flow)

Calculation: E = 5 (edges)
N = 5 (nodes)
P = 1 (component)
M = 5 – 5 + 2(1) = 2





Implications of High Cyclomatic Complexity

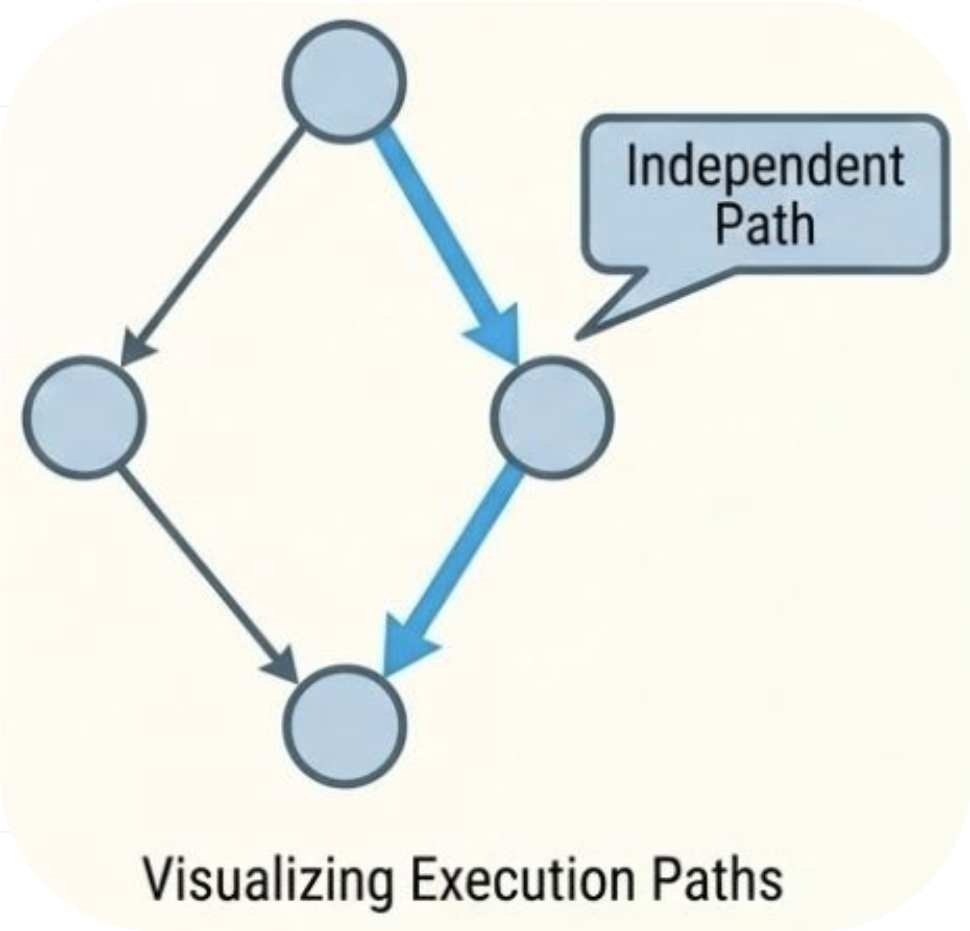
High cyclomatic complexity indicates:

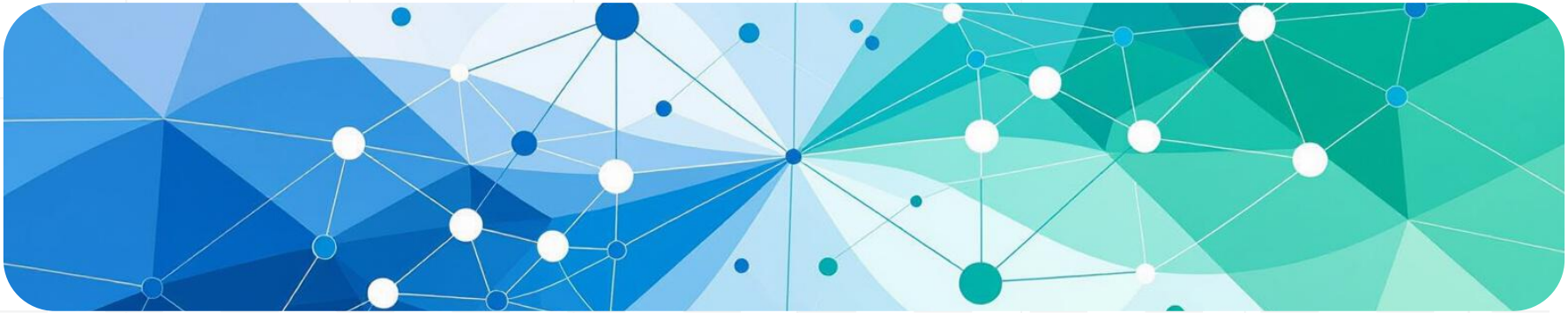
- **Increased risk of defects**: More paths, more chances for errors.
- **Challenges in code maintenance**: Difficult to understand and modify.
- **Higher testing costs**: Requires more test cases.

Aim for lower complexity to improve code quality.

Introduction to Basic Path Testing

Basic path testing is a **white-box testing** technique. It aims to execute all possible *independent* paths in a program at least once. This helps ensure that every part of the code is tested.



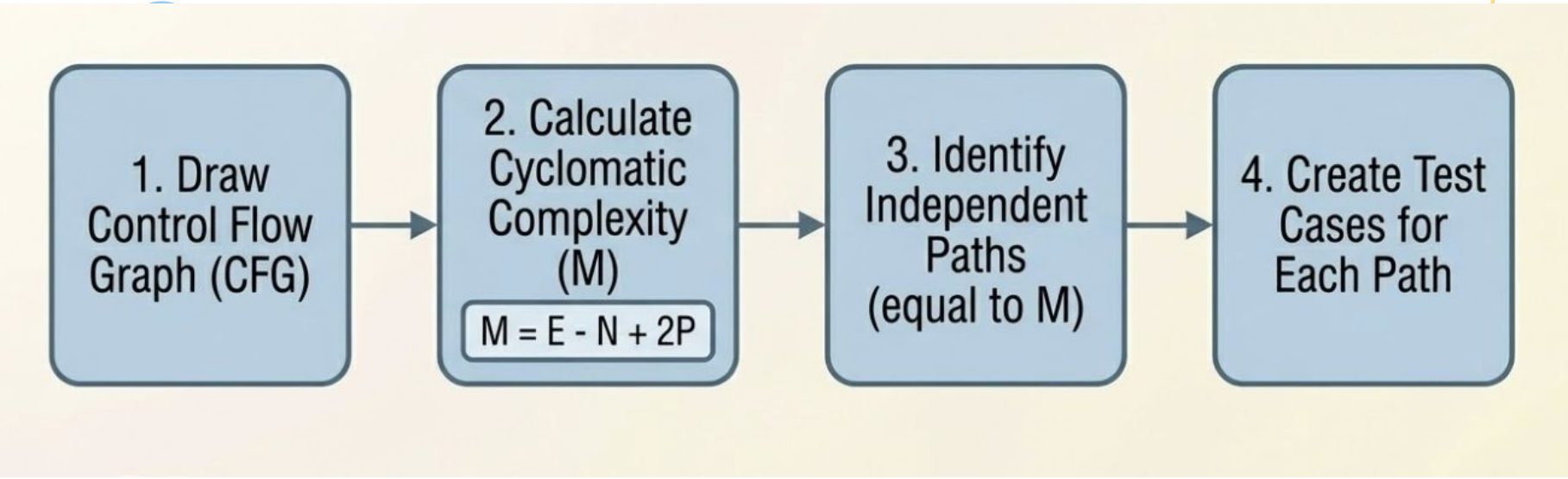


Methodology of Basic Path Testing

Key Steps:

- **Path Identification:** Identify all independent paths in the control flow graph.
- **Test Case Generation:** Create test cases to execute each path.
- **Test Execution:** Run the test cases and verify the results.

The Basic Path Testing Process



- Follows a structured approach to ensure coverage.
- The value of 'M' directly dictates the number of required test cases.

The Role of Cyclomatic Complexity

- Quantifies Test Effort: It provides the exact number of test cases needed.
- Guarantees Coverage: Ensures all statements and conditions are executed at least once.
- Identifies Complex Logic: High complexity points to code that is harder to test.