Cpt S 422: Software Engineering Principles II

Deliverable 3: Black-Box Testing & Evaluation Report

Abylay Dospayev

Cpts 422

Fall 2025

https://github.com/abylaydospayev/checkstyletestproject

Black-Box Testing and Fault Models

Black-Box testing was conducted using the custom BlackBoxTestEngine.java, which simulates the Checkstyle runtime environment by reading actual Java source files (fault models) and asserting the logged metric output.

Fault Models & Test Cases

Metric	Fault Model Scenario (Test Purpose)	Input File
Operator	Verify exclusion of symbols in strings; confirm accurate count of compound (+=) and unary (++) operators.	OperatorInput.java
Loop	Verify correct counting of nested loops (For inside While) and confirmation that do-while counts as a single instance.	LoopInput.java
Comment	Verify correct identification of both single-line and multi-line comments while ignoring identical characters inside string literals.	CommentInput.java

Test Execution Results

All black-box tests passed, confirming the functional correctness of the metrics against real code segments.

Metric	Expected Count	Actual Log Output	Status
LoopCountCheck	4	Total number of loops: 4	**PASS**
OperatorCountCheck	7	Total number of operators: 7	**PASS**
CommentCountCheck	4	Total number of comments: 4	**PASS**

Discussion: Impact of Class Testing

Class Testing evaluates the stability of a class when its instance is reused across multiple operations, which is the standard behavior in Checkstyle's runtime environment.

- Key Insight: Our custom checks maintain state using class-level counters (operatorCount, loopCount). If the mandatory reset (counter = 0;) inside the beginTree() or finishTree() methods were missing, the metrics would fail silently by accumulating counts from previous files processed by the same instance.

- Impact on Testing: Class Testing directly validated the state reset mechanism. The BlackBoxTestEngine.java successfully processes its file-based tests sequentially, implicitly confirming that the finishTree() method is robust in resetting the internal state, an essential step not directly enforced in decoupled unit tests.