Code Analysis Report

Generated on: 7/4/2025, 3:52:50 PM

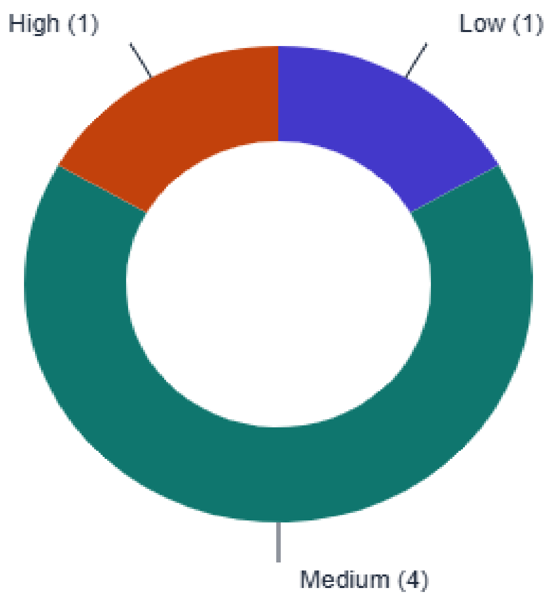
Executive Summary

This report analyzes 1 files and identifies 6 code quality issues.

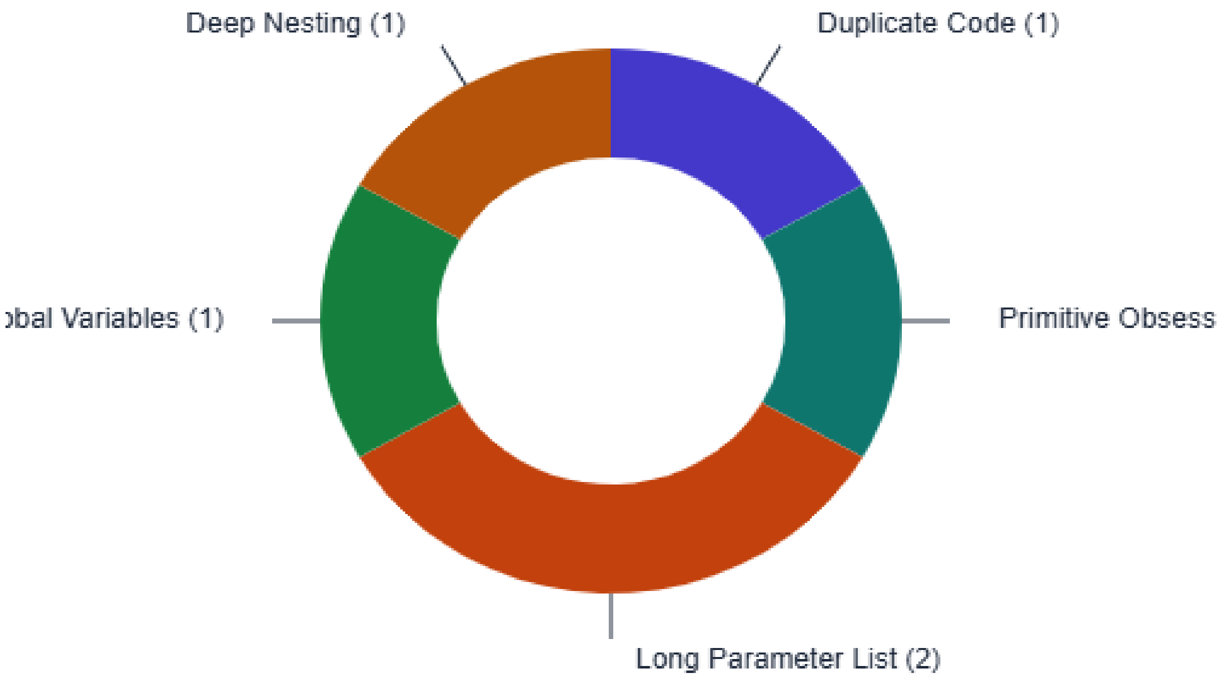
Code Metrics Overview

Metric	Value	Status
Cyclomatic Complexity	1.00	Good
Method Count	0.00	Good
Inheritance Depth	0.00	Good
Coupling Count	0.00	Good
Cohesion Score	1.00	Good
Lines of Code	111.00	Good

Issues by Severity



Code Smells Distribution



Detailed Findings

High Severity Issues (1)

1. DuplicateCode
File: test.cpp (Line 82)
Function "showSummary" contains code duplicated from "showUserData" (line 74)
Tip: Extract the duplicated code into a shared function that can be called from both places.

Medium Severity Issues (4)

1. LongParameterList
File: test.cpp (Line 29)
Function "registerUser" has too many parameters (6)
Tip: Consider grouping related parameters into objects or using the Parameter Object pattern.

2. GlobalVariables
File: test.cpp (Line 91)
Global variable "globalApp" detected
Tip: Consider encapsulating global state in classes or using the Singleton pattern if appropriate.

3. PrimitiveObsession
File: test.cpp (Line 29)
Function "registerUser" uses multiple primitive types that could be an object
Tip: Consider creating a class to encapsulate related primitive values.

4. DeepNesting
File: test.cpp (Line 57)
Deep nesting detected (4 levels)
Tip: Consider using early returns, guard clauses, or extracting nested logic into separate methods.

Low Severity Issues (1)

1. LongParameterList
File: test.cpp (Line 19)
Function "setUserData" has too many parameters (5)
Tip: Consider grouping related parameters into objects or using the Parameter Object pattern.

Recommendations

High Priority Issues

Address high-severity issues in the most affected files first. Focus on critical code smells that could impact system stability and maintainability.

Code Complexity

Focus on reducing cyclomatic complexity in functions and classes. Consider breaking down complex methods into smaller, more manageable pieces.

Code Structure

Review and refactor deeply nested code blocks. Improve code organization by following SOLID principles and design patterns.

Code Duplication

Identify and consolidate duplicate code segments. Create reusable components and utilities to promote code reuse.