PaymentProcessor Test Design Documentation

The PaymentProcessor applies discounts (10% for first orders, 5% for credit cards, 2% for PayPal) and should apply 15% tax after discounts. Orders under $50 incur a $5 delivery fee. The critical bug is that tax is calculated as taxedAmount = amount * 1.15 but the final calculation always returns discountedAmount without tax, resulting in 15% revenue loss.

The following testing techniques have been applied: black-box testing, white-box testing and integration testing.

Black-Box Testing was implemented through three main approaches. Boundary value analysis tested critical thresholds including zero/negative amounts for input validation and delivery fee boundaries at $50. Equivalence partitioning divided inputs into valid/invalid amounts, first-time versus repeat customers, and payment methods. Decision table testing systematically examined all combinations of customer type and payment methods. These techniques revealed the tax bug by comparing expected results (including tax) against actual results (omitting tax). For example, a $100 first-order credit card payment should yield $97.75 but returns $85.00.

White-Box Testing achieved complete statement, branch, and path coverage. Statement coverage confirmed all code executes, including the problematic tax calculation line. Branch coverage tested all conditional paths, while path coverage revealed that taxedAmount is consistently calculated but never used in the final result. This systematic approach exposed that the bug affects all execution paths uniformly.

Integration Testing examined how processPayment() and calculateDeliveryFee() interact, revealing cascading effects of the tax bug. For instance, a $55 order with 10% discount should total $56.93 after tax (free delivery) but actually totals $49.50 without tax (triggering $5 delivery fee). This demonstrates the bug's broader business impact beyond individual transactions.