Money Retrospective

Part II: The xUnit Example

- Implementation of TDD tool will be done by test-driving
- Python is used for smooth xUnit architecture implementation
- Introduction to Python and testing framework creation included
- A more complex TDD example is provided

One Last Review

Three recurring surprises in TDD teaching:

- Three approaches to making tests work: fake it, triangulation, obvious implementation
- Using duplication removal between test and code to drive design
- Controlling gap between tests to increase traction or speed progress

Test Quality

Types of Testing Not Replaced by TDD

- Performance testing
- Stress testing
- Usability testing

Role of Professional Testing Changes

- From supervision to amplifying communication between stakeholders and developers

Measures of Test Quality

Statement Coverage

- TDD should result in near 100% statement coverage
- JProbe found only one line not covered (Money.toString())

Defect Insertion

- Changing code meaning should break tests
- Jester tool found only one line (Pair.hashCode()) could be changed without breaking tests
- Some fake implementations don't affect program meaning

Coverage Insights

- Coverage can be improved by writing more tests or simplifying program logic
- Refactoring often replaces conditionals, reducing logic complexity
- Fixed tests can cover shrinking code logic permutations effectively

Process

TDD Cycle

- Add a little test
- Run all tests and fail
- Make a change
- Run tests and succeed
- Refactor to remove duplication

Change Counts

- Number of changes to compile and run tests is small
- Number of changes per refactoring expected to have a leptokurtotic (fat tail) distribution
- Similar to natural measurements such as stock market price changes

What's Next?

- Code is not finished due to duplication between Sum.plus() and Money.plus()
- Suggestion to make Expression a class to hold common code
- TDD is a way to strive for perfection but most effective for core system parts
- Peripheral system parts can have less rigorous tests and design
- Use automated code critics to identify obsolete code and suggestions
- Identify additional tests for unexpected behaviors or known limitations
- Review design for concept coherence and duplication as symptoms of latent design

Metaphor

- Metaphor dramatically affects design structure and not just naming
- Previous money implementations used vector-like metaphors (MoneySum, MoneyBag, Wallet)
- These metaphors imply a flat collection merging same currency amounts
- Expression metaphor avoids merging issues leading to cleaner and clearer code
- Concern about expression performance but prefer to wait for usage data before optimizing
- Question about gaining insights from repeated rewrites and mindfulness in programming

JUnit Usage

- JUnit was run 125 times during coding
- Tests were run about once per minute during programming phases
- Only one unexpected test result occurred due to hasty refactoring
- Histogram of time interval between test runs shows many long intervals due to writing

Code Metrics

Statistics Overview

- Functional Test Classes: 5
- Functional Functions: 22
- Test Functions: 15
- Functional Lines: 91
- Test Lines: 89
- Cyclomatic Complexity Functional: 1.04
- Cyclomatic Complexity Tests: 1
- Lines per Function Functional: 4.1
- Lines per Function Tests: 5.9

Notes

- Ratios between test and functional code lines and functions are roughly equivalent
- Test complexity is low due to no branches or loops
- Functional complexity low due to polymorphism substituting control flow
- Lines per test function inflated due to unextracted common fixture code