Want Better Software? TEST it! (and *then* write it)

Tame defects before they appear

You Rise/Bugs Fall

Introduction

- TDD had its origins as an integral part of Extreme Programming
- TDD, BDD, DDD and the coming of age of software development
- Part of a group of good practices unifying under a common terminology

What Causes Defects

- Human error
- Requirements not well understood
- Inappropriate time assigned to the project
- Complexity of software, infrastructure or interactions
- Breaking changes in technology
- Breaking changes in the project
- Adverse environmental conditions
- Insufficient knowledge

TDD Reduces Defects

- Testing is an integral part of design and development
  - Comprehensive tests result
  - Unit tests serve as regression tests
- Improved Design helps manage complexity
  - Testable code is inherently modular and loosely coupled

Cost vs. Benefit

- Cost
  - Initially takes longer
  - Training/learning required
- Benefit
  - Faster bug detection
  - Shorter feedback loop (less effort/cost required to find and fix)

Dealing with the Pitfalls

- False Sense of security
  - Don’t assume code is correct just because it passes all the tests
  - QA and manual testing are still required
  - Don’t forget about the requirements
- Blank Slate effect
  - Deciding what to test not always obvious
  - Break down the requirement
  - Review the design

Elements

- Testing Frameworks (NUnit, VS Test, MbUnit)
- Isolation Frameworks (Rhino, JustMock, etc.)
- Code Generation within the IDE and 3rd Party Tools
- Techniques

Demo

- Developing a simple project using TDD

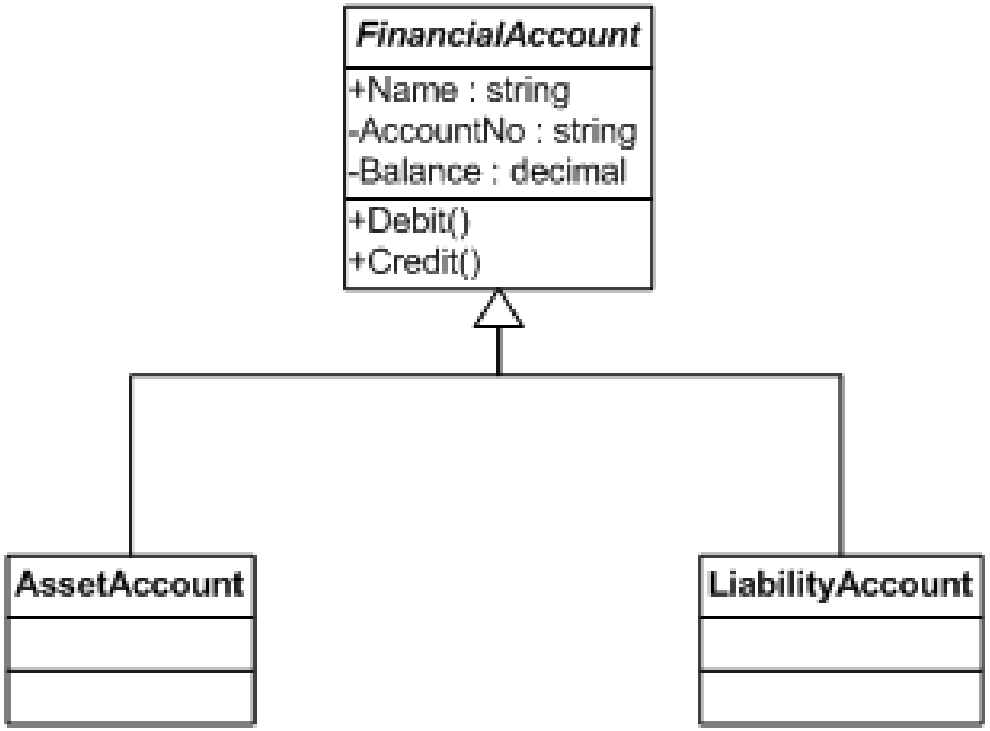
Testing Tools

- Visual Studio (C#)
- NUnit (test framework)
- Rhino Mocks (isolation framework)

Sample Application

- A simple Accounting Application
- Scenarios
  - Debit an Asset Account
  - Credit an Asset Account
  - Debit a Liability Account
  - Credit a Liability Account

Sample Application



TDD Technique

- The Three “A”s
  - Arrange
  - Act
  - Assert
- Red/Green Refactoring
  - Write a failing test
  - Write the simplest method to pass the test
  - Write another test …

Unit Tests

- Test a single unit of code
- Isolates the SUT from all other components including those it depends upon
- Does not interact with the environment
- Requires no setup
- Repeatable (does not change the state of the application)

Stubs and Mocks

- Stubs
  - Passive
  - Take the place of dependencies
  - Isolate the unit under test
- Mocks
  - Active
  - Detect interactions
  - Can tell if the unit under test is using the interface correctly

Principle/Pattern

- The Principle of Inversion of Control
- The Dependency Injection Pattern

Integration Tests

- Test two or more units of code that work together
- May interact with the environment
- May change the state of the application
- May require set-up/reset

Unit Tests vs. Integration Tests

- Important to distinguish
- Keep separate, preferably in separate libraries
- Create Tests in a Solution Folder
- Use Categories to separate tests into groups that can be run separately

Good Tests Should Be

- Trustworthy
- Maintainable
- Readable

How to Start

- Research
- Get the tools
- Practice on small systems/projects
- Measure progress and make visible
- Obtain support from management/influential people
- Don’t set yourself up for failure

Summary

- TDD reduces defects, improves design and helps make software more maintainable
- Part of a broader group of concepts
- Fairly mature, great tools and information available
- Techniques are simple, results proven
- Integration tests and unit tests are separate
- Good tests have three characteristics
- Start small, build support

Part III – Optional Topics

- Test Coverage
- Visual Studio Power Tools – Pex, Moles
  - Generating Test Code Automatically
  - Parameterised Tests – The Fourth “A”
  - Code Contracts
- Evolution: Behaviour Driven Development (BDD)
  - Ubiquitous language for software development
  - How TDD supports BDD

References

- Domain Driven Design: Tackling Complexity in the Heart of Software [Eric Evans, Adison Wesley, pp23-27]
- Extreme Programming Explained: Embrace Change 2Ed[Beck/Andres, Addison Wesley, p50]
- The Art of Unit Testing [Roy Osherove, Manning, p172-181]
- http://dannorth.net/introducing-bdd/
- http://www.blurtit.com/q834817.html
- http://tech.groups.yahoo.com/group/extremeprogramming/message/111829
- http://www.specflow.org/
- http://behaviour-driven.org/

Where to get the tools

- Nunit http://nunit.org
- MbUnit http://mbunit.com/Default.aspx
- Rhino Mocks http://ayende.com/projects/rhino-mocks/downloads.aspx