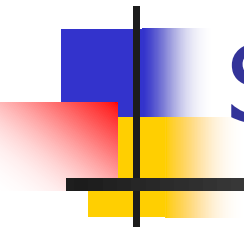
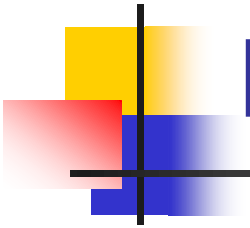


CS 575
Software Testing and Analysis

Test-Driven Development

ÖZYEĞİN UNIVERSITY



Motivation

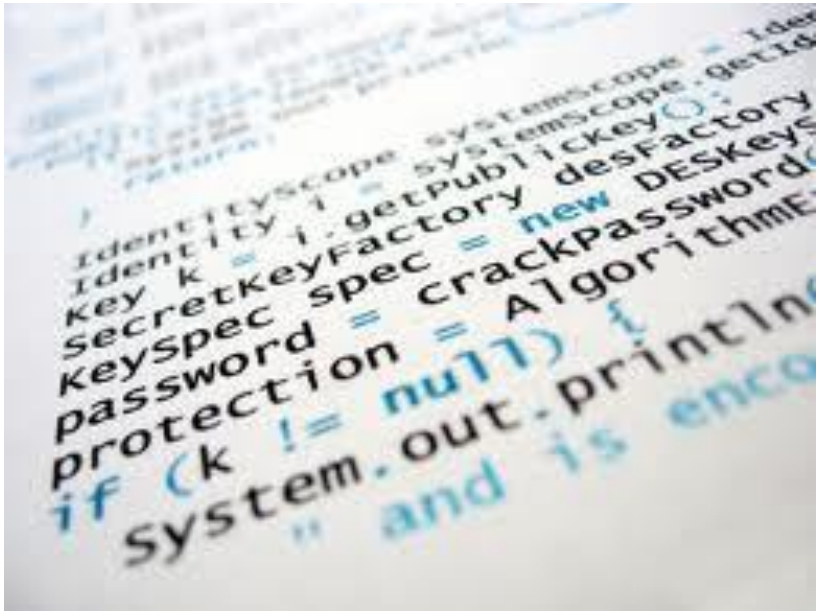
- *Common sense:*
  - **a good testing methodology** is necessary for the development of reliable software
- *Common practice:*
  - ad-hoc, unstructured
  - not repeatable
  - late

Motivation

- All the code must be tested to ensure reliability

Test-Driven Development (TDD)

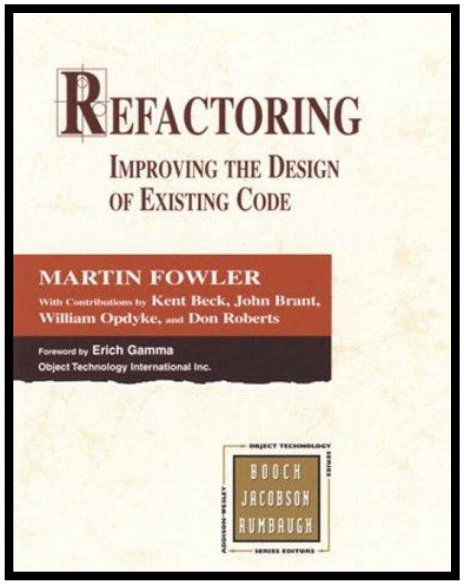
- A process with tiny steps
- main focus on
  testing and **refactoring**
- Refactoring: restructuring code without changing its external behaviour ...



Refactoring Operations

- Extract Method
- Extract Class
- Extract Superclass
- Encapsulate Field
- Decompose Conditional
- ...

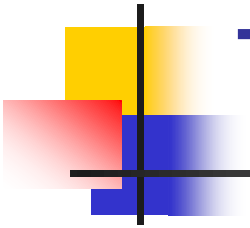
- Goal: increasing readibility and maintainability of the code



Refactoring: Improving the Design of Existing Code
Martin Fowler
Addison-Wesley, 1999
ISBN 0-201-48567-2

Origins..

- Devised in the context Agile methodologies
  - but can also be applied by itself, without Agile methods
- Agile development
  - eXtreme Programming (XP)
    - TDD
- Other methodologies, e.g., waterfall, spiral, etc.



TDD Focus

- Implementation, code-level

  (i.e., not architecture design, requirements engineering, etc.)
  - Developer = programmer

- Unit testing

  (i.e., not integration testing, acceptance testing, etc.)
  - Testing internals of a class
  - Black-box testing for objects

Tool-support for TDD

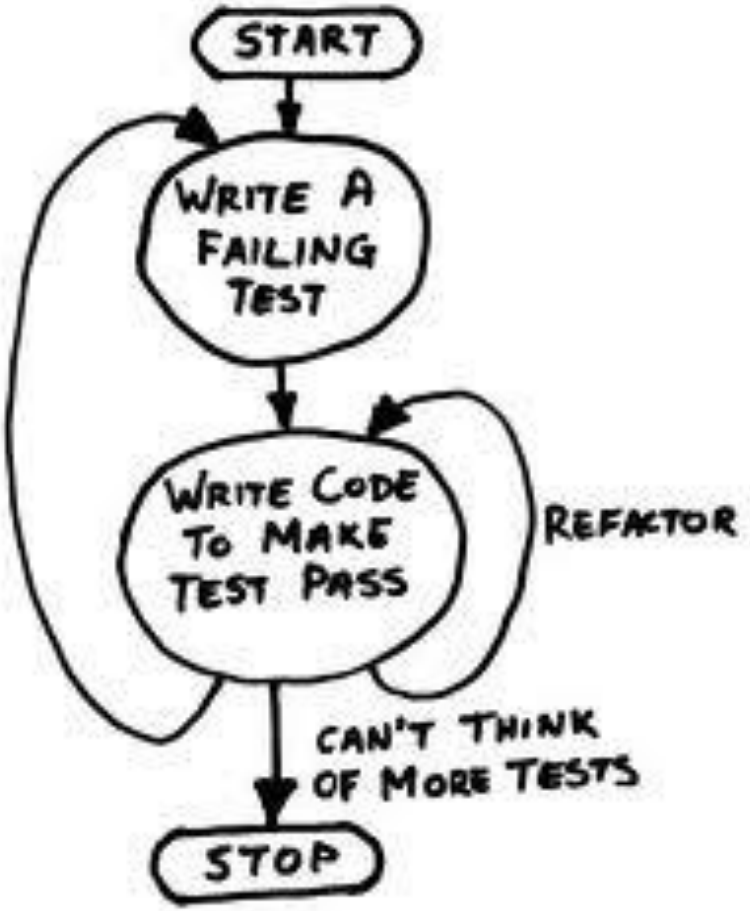
- There exists frameworks for automating the test execution
  - a must for TDD!
- Examples:
  - JUnit, CppUnit, Nunit, DUnit, VBUnit, RUnit, PyUnit, Sunit, HtmlUnit, …
  - More listed at www.xprogramming.com

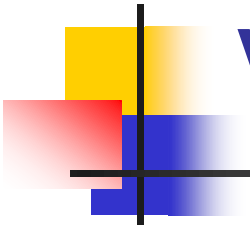
Basic principles

- **Before** writing the code, think about what it will do

- Write tests for methods that do not even exist yet

TDD Stages


START
WRITE A FAILING TEST
WRITE CODE TO MAKE TEST PASS
REFACTOR
CAN'T THINK OF MORE TESTS
STOP

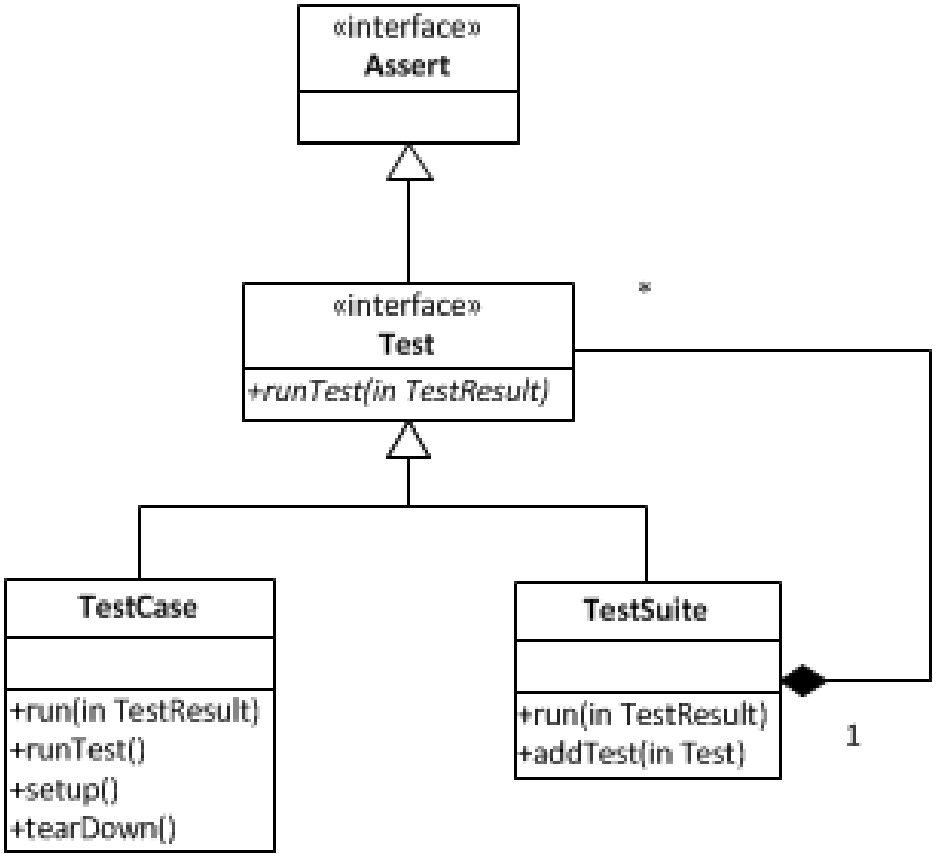


Vital Properties

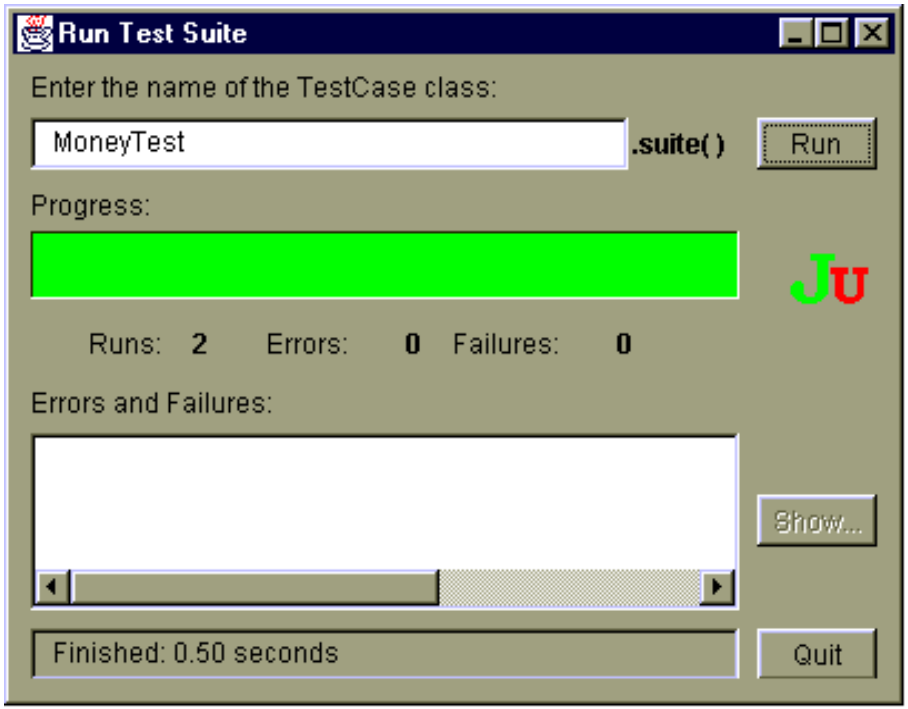
- Only **small** refactorings
  - makes it less likely to go wrong
  - system is kept fully working after each step
- Test: not an activity but a piece of **code**
  - not something you do, but something you write
- Test execution is **automated**
  - repeatedly executed after very small changes

Case: Java + Eclipse + JUnit

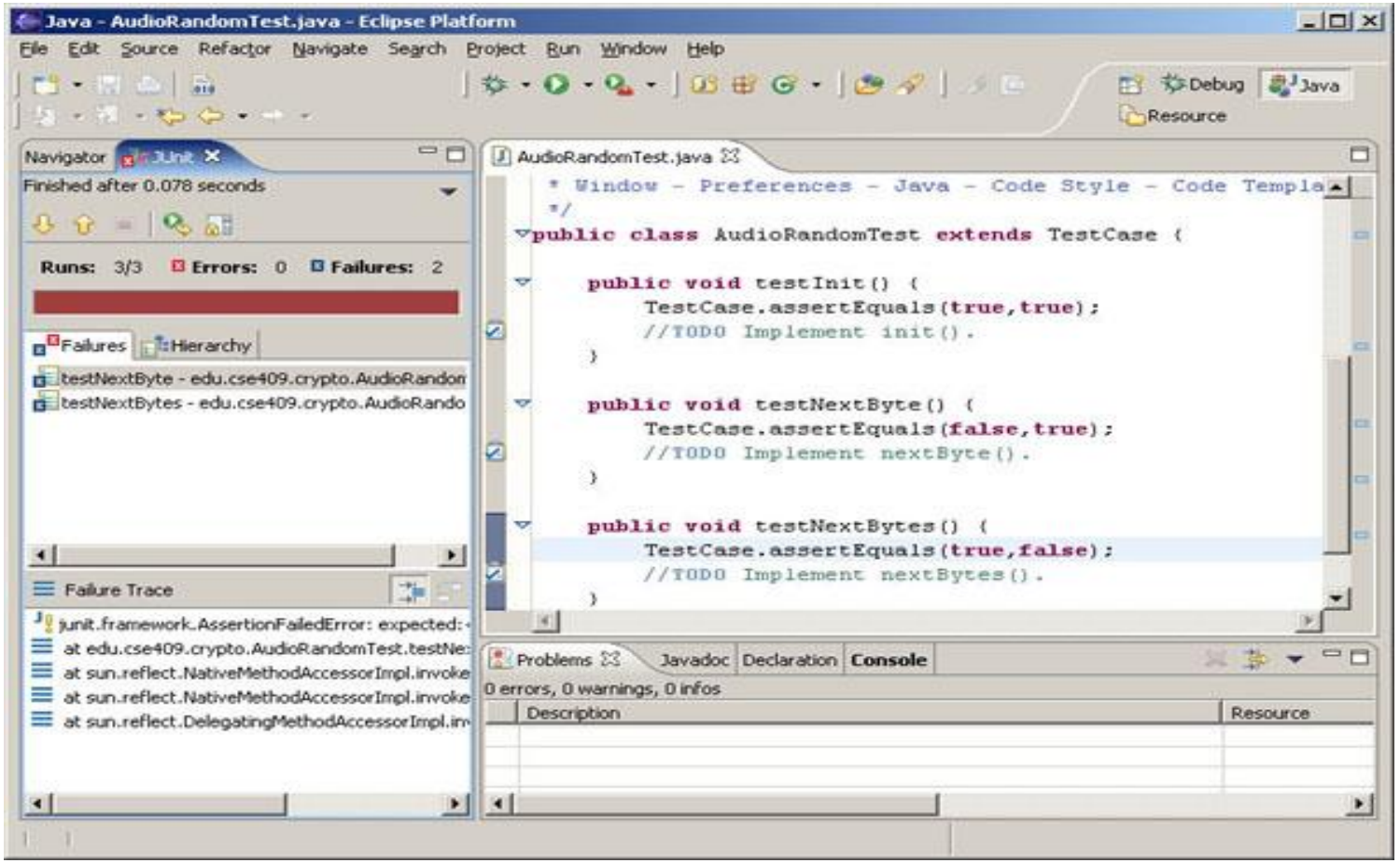
Junit Framework



Junit GUI

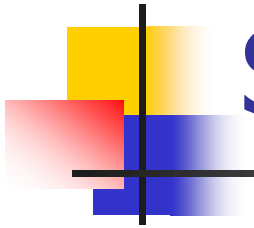

Run Test Suite
Enter the name of the TestCase class:
MoneyTest
.suite( )
Run
Progress:
Runs: 2
Errors: 0
Failures: 0
Errors and Failures:
Show...
Finished: 0.50 seconds
Quit

Junit for Eclipse

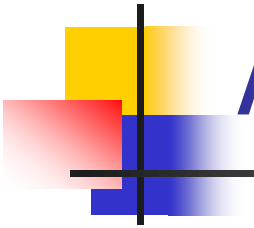


Demo..

Structure of a Test Case class

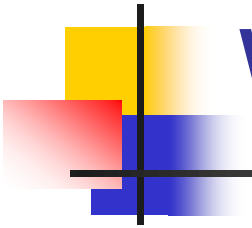


- setUp() initialize variables, claim resources
- tearDown() release the resources
- run() run a *test case/suite*
- testCase() a test case



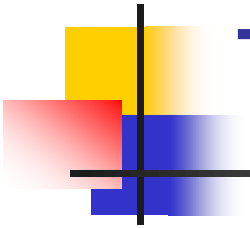
Assert Statements

- assertEquals(expected, actual)
- assertEquals(message, expected, actual)
- assertEquals(expected, actual, delta)
- assertEquals(message, expected, actual, delta)
- assertFalse(condition)
- assertFalse(message, condition)
- Assert(Not)Null(object)
- Assert(Not)Null(message, object)
- Assert(Not)Same(expected, actual)
- Assert(Not)Same(message, expected, actual)
- assertTrue(condition)
- assertTrue(message, condition)



Why TDD?

- Programmers do not like testing
  - considered to be a boring task
- TDD **encourages** and **facilitates** programmers to maintain repeatable tests
  - Tests live alongside the code
  - Test execution is automated (push-button)
  - Test after every single change increases **confidence**
- **...**

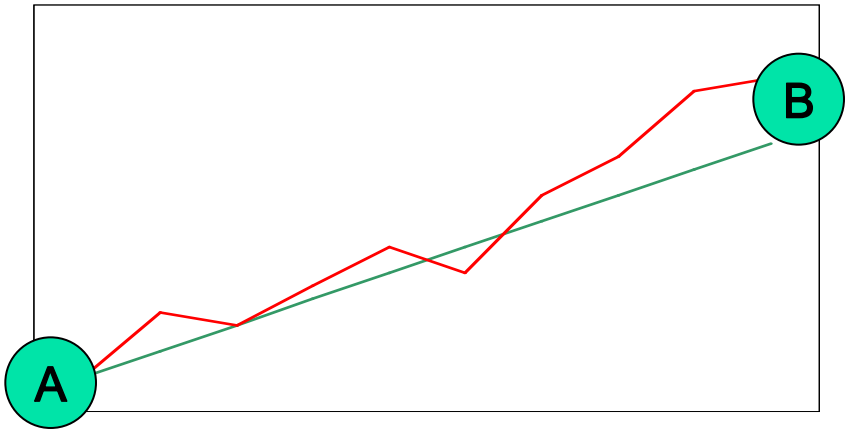


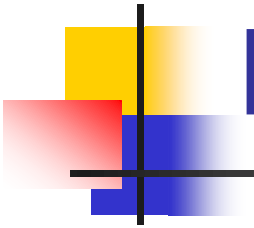
TDD ..

- shortens the programming feedback loop
  - in the order of minutes
- provides detailed specification (tests)
- promotes the development of high-quality code
- provides concrete evidence that code works
  - boosts confidence
- supports evolutionary development

Small steps

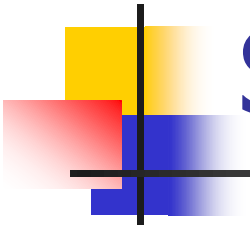
- TDD promotes "small steps"
  - Not to diverge too much from the path to the destination





Final notes

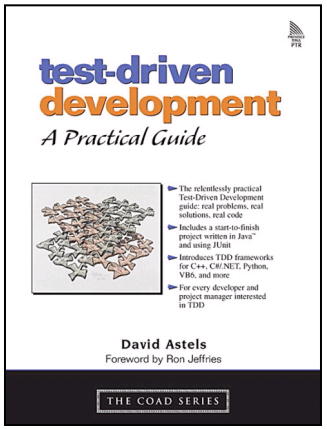
- TDD increases development speed
  - due to less time spent for debugging
- TDD decreases maintenance costs
  - due to increased modularity as a result of refactoring
- TDD does not replace traditional testing
  - just proposes a way of working
- TDD might require supporting techniques
  - creating Mock objects (stubs) and views
  - substituting visual elements during testing



Summary

- No code without tests
  - Tests verify the code, acting as documentation
- Tests dictate the code
  - Let the design emerge
- Testing and refactoring go hand-in-hand
  - Ensuring clean, modular code
- Elementary increments
  - Small, tiny, safe steps

Some Resources on TDD..

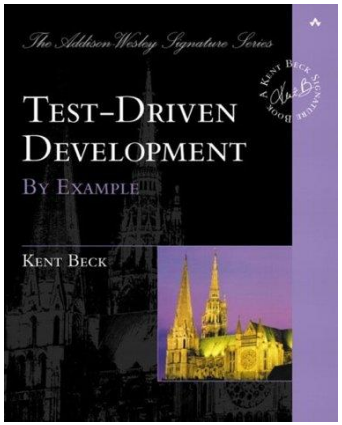


test-driven development: A Practical Guide

Dave Astels

Pearson Education, 2003

ISBN 0-13-101649-0

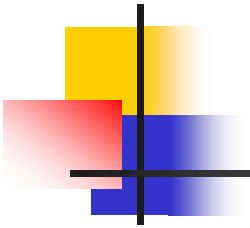


Test-Driven Development: By Example

Kent Beck

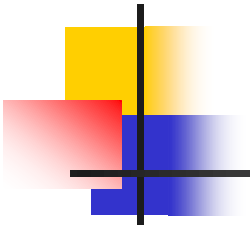
Addison-Wesley, 2003

ISBN 0-321-14653-0



Some Resources on TDD..

- Web sites
  - www.testdriven.com
  - www.xprogramming.com
  - www.agilealliance.com
  - www.refactoring.com
- Mailing lists (Yahoo groups)
  - testdrivendevelopment
  - agiledotnet



Introducing TDD in an organization...

- Start small and simple
- Adopt TDD
  - for new tasks
  - for the code that needs to be modified in small pieces
- Provide proof-of-concept results

Questions?