UNIT TESTING

PBA SOFTWAREUDVIKLING/
BSC SOFTWARE DEVELOPMENT

Christian Nielsen cnls@cphbusiness.dk

Tine Marbjerg tm@cphbusiness.dk

SPRING 2019

TODAY'S TOPICS

- **Overview**
  - Study points / Peer reviews / Assignments / Hand-ins
  - Learning objectives
  - Why? / When? / What?
  - Good code + Good tests
  - Designing / Structuring / Automating
  - xUnit / JUnit
  - Features
  - Examples
  - Assignment

LEARNING OBJECTIVES?

- Produce testable code
- Design, structure, and automate tests
- Know how to apply equivalence partitioning and boundary value analysis in tests
- Know general features and usage of xUnit Framework API
- Implement high quality automated tests with xUnit Framework

WHY MAKE TESTS?

- Leads to better code and design
- Generates code targeted at tests
- Reduces manual testing
- Ensures basic functionality is independently scrutinized for proper operation regularly
- Creates control of code working as assumed throughout project process
- Requires more consideration when refactoring and redesigning is done
- Catches regression defects when they are introduced

WHEN TO WRITE TESTS?

After coding? / During coding? / Before coding?

Verify functionality

Discover problems

Clarify requirements

Explore implementation

Make sure tests are written…

WHAT CODE TO TEST?

Constructors / Getters / Setters?

Logic?

Logic is probably more interesting than constructors / getters / setters but can still contain problems…

WHAT MAKES CODE TESTABLE?

- Follow the SOLID rules
- Follow the single responsibility principle
- Create simple constructors
- Use new with care
- Use inversion of control and dependency injection
- Reduce dependencies
- Avoid hidden dependencies
- Avoid logic in constructors
- Avoid static methods
- Avoid singletons
- Avoid too many parameters in methods
- Avoid too large methods
- Favor generic methods
- Favor composition over inheritance
- Favor polymorphism over conditionals

WHAT MAKES A TEST A UNIT TEST?

Smallest parts

Individual units of work

Separate components

Class methods

No external dependencies

No communication with database, file system or network

DESIGNING TESTS…

1. Identify which conditions to test
2. Design test cases that cover conditions

Dynamic testing = Tests are executed on running code

- Black box techniques
  - Implementation is not known
  - High level / Testers / No programming knowledge
  - External / Based on requirements documentation
  - Equivalence partitioning / Boundary value analysis / Decision tables / State transition
- White box techniques
  - Implementation is known
  - Low level / Developers / Programming knowledge
  - Internal / Based on design documentation

EQUIVALENCE PARTITIONING…

Good all round technique to use first

Aims at minimizing the number of test cases

Divide set of test conditions into groups that can be considered the same and handled equivalently

Input values are partitioned into equivalence classes if they result in the same program behavior and find the same errors

Only need to test one condition from each partition, since it is assumed that all conditions in a partition will be treated the same way and either work or not work

Identify partitions and write a test case for each partition

Both valid and invalid partitions need to be considered

Invalid inputs are separate equivalence classes

Equivalence partitions = Equivalence classes

EQUIVALENCE PARTITIONING…

Example: Bank Interest

A savings account in a bank earns a different rate of interest depending on the balance in the account

- A balance in the range $0 up to $100 has a 3% interest rate
- A balance over $100 and up to $1000 has a 5% interest rate
- A balance of $1000 and over have a 7% interest rate

Partitions

- Invalid partition: -0.01$
- Valid partition: 0.00$ - 100.00$
- Valid partition: 100.01$ - 999.99$
- Valid partition: 10000.00$
- Invalid partition: >max$

BOUNDARY VALUE ANALYSIS…

Equivalence partitioning and boundary value analysis are closely related

Based on testing at the boundaries between partitions

Errors often show at the boundaries between equivalence classes

Boundary value is on the edge of an equivalence partition

Choose minimum and maximum values from an equivalence class together with first and last value respectively in adjacent equivalence classes

Two value approach is on boundary value and one side, where three value approach is on boundary value and two sides

Open boundaries are when one of the sides in a equivalence class is left open

2 or 3 value approach / Middle value

BOUNDARY VALUE ANALYSIS…

Example: Bank Interest

A savings account in a bank earns a different rate of interest depending on the balance in the account

- A balance in the range $0 up to $100 has a 3% interest rate
- A balance over $100 and up to $1000 has a 5% interest rate
- A balance of $1000 and over have a 7% interest rate

Boundary values

- -0.01$ / 0.00$ / 0.01$ / 50.00% / 99,99$ / 100.00$ / 100.01$ / 500.00$ / …

EXERCISES

- getsMortgage
- isEven
- getNumDaysInMonth

STRUCTURING TESTS…

Naming conventions

BDD:

ShouldAlertUserIfAccountBalanceIsExceeded

UNIT / ACTION / EXPECTATION:

Atm_NegativeWithdrawal_FailsWithMessage

Be sure to name tests in a consistent and meaningful manner…

STRUCTURING TESTS…

Organizing tests

Arrange-Act-Assert

1. Set up test objects and arrange data
2. Execute the test code
3. Verify the outcome and assert that result is as expected

There are other styles of structuring tests…

- Operate-Check
- Given-When-Then
- Setup-Execute-Verify-Teardown

STRUCTURING TESTS…

One assert / Many asserts

One test case – Many asserts

- Fewer test cases are needed
- Good enough for some test cases

One test case – One assert

- More test cases are needed
- If a test fails it is easier to find out why

Be sure to have a concentrated focus in each test…

AUTOMATING TESTS…

Why?

Manual testing is tiresome

Make sure new code works

Make sure new code doesn’t break existing code

Why not use system out?

Needs human interpretation

Confusing with many old messages

Slows down code

How?

xUnit frameworks and build tools can be used to automate execution of tests

JUnit / Maven

GOOD UNIT TESTS…

What makes a test a good unit test?

Must have a clear intention

Must be easy to understand

Must use descriptive test names

Must be based on good test case design

Must consider invalid conditions

Must follow the style of arranging, acting and asserting

Must only have a single assert in each test

Must not be dependent on environment

Must not be dependent on external factors

Must not be dependent on other tests

Must provide understandable feedback

Must provide same inputs and expect same result each time

Must be reliable and consistent

Must be repeatable and automated

Must be executed quickly

MAVEN…

- **Build automation tool used primarily for Java projects**
- **Standard way to build project**
- **Clear definition of what project consists of**
- **Easy way to publish project information**
- **Way to share JARs across several projects**
- **Dependencies and plugins can be added to pom.xml file**
- **Lifecycle comprises of phases such as compile, test, deploy…**

Unit tests are usually included within the build process, which means they are run by a build tool like Maven.

Used to handle project dependencies via the pom.xml and execute commands in console

JUNIT…

JUnit is a regression testing framework used by developers to implement unit tests

"Framework" because subclasses are created from abstract classes

"Regression" since tests can be kept and run frequently

Uses annotations to identify methods that specify a test

Executes code during test and *ASSERT* methods, called *ASSERTS* or *ASSERT STATEMENTS*, provided by JUnit or another assertion framework

Checks an expected result with an actual result and then either passes or fails tests, plus provides messages based on results of asserts

Architectural components

Test fixtures / Test cases / Test suites / Test runners / Assertions

JUNIT…

Usage

Create test classes with test methods, asserts and annotations

Run test classes directly from main() or with test running utility, such as JUnit TestRunners

JUnit assumes all test methods can be executed in an arbitrary order

Most IDE's has built in functionality in GUI for executing JUnit tests

Create / Update tests

Options: IntegrationTest / Levels / Generate / Comments

Naming

Mandatory prefixes and postfixes for tests to be executed during build (Test* / *Test / *TestCase)

Can be used to disable tests or might be the cause why tests are not run

JUNIT…

Maven dependencies…

JUnit4 + JUnit5:

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.3.2</version>
</dependency>
<dependency>
    <groupId>org.junit.platform</groupId>
    <artifactId>junit-platform-runner</artifactId>
    <version>1.3.2</version>
</dependency>
<dependency>
    <groupId>org.junit.vintage</groupId>
    <artifactId>junit-vintage-engine</artifactId>
    <version>5.3.2</version>
</dependency>
```

JUNIT…

Maven plugins…

JUnit4 + JUnit5:

```
<build>
    <plugins>
        <plugin>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>2.22.1</version>
            <configuration>
                <argLine>-Dfile.encoding=UTF-8</argLine>
            </configuration>
        </plugin>
    </plugins>
</build>
```

JUNIT…

Features…

Fixtures	Fixed state of a set of objects used as a baseline for running tests
Tests	Defines test cases to be executed
Suites	Several test classes can be combined into a test suite
Assertions	Communicates success or failure and expresses outcome of test
Runners	Used for executing test cases

Latest features in JUnit5…

Nesting / Tagging / Assumptions / Extensions / Conditions / Lambdas / Parameters

JUNIT…

Ressources…

JUnit.org5 user guide - https://junit.org/junit5/docs/current/user-guide/

Baeldung.com guide - https://www.baeldung.com/junit-5

Petri Kainulainen tutorial - https://www.petrikainulainen.net/junit-5-tutorial/

HowToDoInJava tutorial - https://howtodoinjava.com/junit-5-tutorial/

ASSIGNMENT

- calculateYearlyInterest
- eightDifferentAssertions