Softwarekonstruktion og -arkitektur

August 26, 2019

1 Generelt

Bogens hjemmeside
Kap 36 er på side 486

Instruktor:

Navn: Katrine Scheel Nelkman
Github: katrinescheel
Mail: 201509419@post.au.dk
Kontor:

Virtual machine:

User: csdev pwd: csau.

2 Agile Developent Process

2.1 Software Development Methods

- A software development processes must have defined tecniquest to deal with activities or steps in developent, the steps often includes
  - **Requirements**. How do you collect the userst'and customerst'expectation to the software?
  - **Design** How do you structure and partion the software and how do you communicate it
  - **Implementation** How do developers program the software so that it fulfills the requirements and adheres to the design?
  - **Deployment** How do you ensure that the software system executes in the right environment
  - **Maintenance** How can we ensure that the soft ware is correct and enhance the defects discorvered.
- The **waterfall method** is the method where you completly finished the requirement before starting on the next one (is cheap but hard to use)

2.2 Agile Methods

- Agile Manifesto
- Examples of agile methods

  - Extreme Programming (XP)
  - Scrum
  - Crystal Clear

- Agile methods value to move towards the defined goal with speed while maintaining the ability to change to a better route without great cost
- Agile methods put a lot of emphasis on software development as a team effort
- Agile methods focuses on making the high quality code rather than and less on writing docmentation
  - Testing is central when trying to keep the code hight quality
  - Refactoring is central to improve the quality of the code
- Agile methods focuses a lot on customer collaboration
  - It is important to have customers use the product before it is finished, because it makes it easier to corect errors
  - Small releases can be used
- It is important to revisit the plan during development
  - When working on a product you learn many new things

2.2.1 Extreme Programming (XP)

- Extreme programming was one of the first agile methods
- It is important to have a good balance on the four parameters cost, time, scope and quality
  - cost, time and scope is often the fixed values from the company
  - It is better to have cost, time and quality as fixed value, because two great implemented features is better than tree poorly implemented features
- XP has four central values
  - **Communication** Good communication is a primary cure for mistakes. XP value interaction between everyone
  - **Simplicity** In XP you focus on the features to put into the next small release.
  - **Feedback** XP focus on feedback in the minutes and hours time scale
  - **Courage** It is important to have the courage to throw away bad designed and/or low quality code
- A central technique in XP is pair programming
  - One person sits at the computer and codes
  - The other person is think strategically and evaluates the design, the code etc.
  - Pairs are often dynamic
  - Collective ownership is used so everyone can change the code, if they think they can make it better
  - It focuses on quality because no code is never written without being read through
- Automated testing is vital in XP
  - It is carried out by computers
  - Computers does not make mistakes

  - The system gives feedback on its health and quality
- XP is a highly iterative development process
  - Work is organized in small iterations each with a well defined focus
- XP uses stories to describe the behavior of the system

3 Reliablity and Testing

- Reliablity is defined as maintaining a specified level of performance
  - Performance is the ability to perform the required function without failing
- Reliabilty is a highly desired quality of software
- Examples of achieving reliablity
  - Better programming language constructs e.g. local variables
  - Reviews where reviewers read source code to find defects
  - Testing to find situation where the software does not perform the required function

3.1 Testing Terminology

- A defect (or bug) is the algorithmic cause of failure
- Test case is a definition of input values and expected output values for a unit under test
  - Test cases are defined by the unit under test (some part of the system)
- A test suite is a set of test cases
  - Is often represented by a **test case table**
- A failed test can be referred to as a broken test
- Manual testing where suites of test cases are executed a verified manually by humans
- Regression testing is the repeated execution of test suites to ensure they still pass and the system does not fail after a modification
- Automated testing is a processes where the test suites are executed and verified by computer programs
- The production code is the code that defines the behavior in the software
- The test code is code that defines test cases for the production code

3.2 JUnit

Example of testing using JUnit

```
import org.junit.*;
import static org.junit.Assert.*;

/** Testing dayOfWeek using the JUnit 4.x framework. */
public class TestDayOfWeek {
    /**
    * Test that December 25th 2010 is Saturday
```

```
    */
    @Test
    public void shouldGiveSaturdayFor25Dec2010 ( ) {
        Date date = new Date( 2010, 12, 25);
        assertEquals ( "Dec 25th 2010 is Saturday" ,
                       Date.Weekday.SATURDAY, date.dayOfWeek() );
    }
}
```

Asserts in JUNIT All asserts can also take a string as the first argument if the test failes

```
/* Assert and pass if */
assertTrue(boolean b) // expression b is true
assertFalse(boolean b) // expression b is false
assertNull(Object o) // object o is null
assertNotNull(Object o)// object o is not null
assertEquals(double e, double c, dou- ble delta) // e and c are equal to within a positive del
assertEquals(Object[] e, Object[]c) //object arrays are equal
```

If a methods should throw an exception provide the exception to the @Test annotation, example:

```
@Test (expected = ArithmeticException.class)
    public void divideByZero () {
        int value = calculator.doDivide (4, 0);
    }
}
```

- To make a function run before every test use the `@Before` over the function e.g.

```
public class TestPayStation {
    PayStation ps ;
    /** Fixture for pay station testing. */
    @Before
    public void setUp() {
        ps = new PayStationImpl() ;
    }

....
}
```

To compile the JUNIT test use the following command:

```
javac -classpath .:junit-4.4.jar *.java
```

To execute a test in the terminal use the following ‘`java -classpath .: junit-4.4.jar org.junit.runner.JUnitCore TestDayOfWeek`

Jar can be found on junit side or the books webside

4 Test-Driven Development (TDD)

- Test-driven development is a part of Extreme Programming
- Programming fast is not achived being sloppy or by rushing
- In TDD Production code is driven into existence by tests
  - You cannot enter a single character into your production code unless there is a test case that demands it
- All unit tests must always pass at the end of each iteration

---

The TDD Rhythm: 1. Quickly add a test 2. Run all tests and see the new one fail 3. Make a little change 4. Run all tests and see them all succeed 5. Refactor to remove duplication

---

- Values
  - Take small steps
    - It is important because if you leap over several steps you get a lot of problems
    - Even if it means writing temperary code
  - Keep focus
    - Focus on one step, one issuem at the time
  - Speed
    - having a well structered programming process guided by sound principles
    - testing ideas early
    - keeping the code maintainable and of high quality
  - Simplicity
    - implementing the code that makes the product work and no more

---

- Principles
  - **Automated test:** use automated test to test your software
  - **Test First:** Write your test before your write the code that should be tested
  - **Test List:** Before you begin, write a list of all the tests you know you will have to write and add more later
  - **One Steps Test:** Pick a test that will teach you something and that you can implement
  - **Fake It (t'Til You Make It):** What is your first implementation once you have a broken test? Return a constant and then gradually transform it
  - **Triangulation:** Abstract only when you have two or more examples
  - **Isolated Test:** The running of tests should not affect one another
  - **Evident Data:** Make the expected and actual results relationship apparent.
  - **Representative Data:** Select a small set of data where each element represents a conceptual aspect or a special computational processing

- **Assert First:** Write asserts first
- **Obvious Implementation:** Just implement simple operations
- **Evident Tests:** To avoid writing defective tests, we keep the testing code evident, readable and as simple as possible
- **Break**: When you feel tired or stuck, take a break, ___

5 Configuration Management

- **Software Configuration Management** (SCM) is the process of controlling the evoulution of a software system
- Software systems view software a hierarchical structure of some atomic items ___
- A SCM system is a tool set that defines

  1. A central repository that stores versions of entities
  2. A schema for how to setup multiple, individual workspaces
  3. A commit and a check-out operation that transfer copies of versions between the repository and a workspace
  4. A schema for handling/defining versions identities for configuration items and configurations
  5. A schema for collaboration/concurrent access to versions

---

- The two principal responsibilities of a SCM system is

  1. Ensure uniqueness of the version identities
  2. Originize version with respect to each other ___

- A **configuration item** is the atomic building block in a SCM system.
  - The SCM system views a configuration item as a whoe
  - It is identified by name
  - Can be thought of as a file as an anology
- A **configuration** is a named hierachical structure that aggregates configuration items and configurations
  - The definition is recursive
  - It is essentially the `COMPOSITE` design pattern
  - Can be thought of as a folder as an anology

5.1 Versions

- The SCM tracks the evoulution of the file system
- Many configurations systems handle the version of a configuration item and a configuration very different ___
- A **version** v_i, represents the immutable state of a configuration item og configuration at time t_i
- A version is identified by a **version identity** v_i, that must be unique in that SCM system