Software Engineering I (02161) Week 3

Assoc. Prof. Hubert Baumeister

DTU Compute Technical University of Denmark

Spring 2016



Recap

Requirements Engineering

- functional / non-functional requirements
- Elicitation, Documentation, Validation
- Glossary
- Use Cases
 - use case diagrams
 - detailed use cases descriptions
- User Stories

Use Case Diagram



Notation is important

- Actor: Stick figure
- Relationship actor, use case: solid line, no arrow head /
- Relationship use case, user case: broken line with arrow A and <<extends>> or <<iincludes>>
- Relationship actor, actor: Generalization: solid line with closed arrow head
- System boundary: Box

Contents

Software Testing

Acceptance tests

JUnit

Test Driven Development

How calendars and dates work in Java

Mock objects

Purpose of tests

Goal: finding bugs

Edsger Dijkstra

"Tests can show the presence of bugs, but not their absence."

- Types of bugs: requirement-, design-, implementation errors
- Types of testing:
 - validation testing
 - Does the software conform to the requirements?
 - Have we built the right system?

defect testing

- Does the software has any unexpected behaviour (e.g. crashes)?
- Have we built the system right?

Validation testing vs defect testing

Validation Test (Quality Assurance (QA))

 Start city is Copenhagen, destination city is Paris. The date is 1.3.2012. Check that the list of availabe flight contains SAS 1234 and AF 4245

Defect Test (QA and stress tests)

- Start city is Copenhagen, the name of the destination city contains the Crtl-L character.
- Check that the software reacts reasonable and does not crash

Types of tests

- 1. Developer tests (validation testing)
 - a Unit tests single classes and methods)
 - b) Component tests (single components = cooperating classes)
 - c) System tests / Integration tests (cooperating components)
- 2. Release tests (validation and defect testing, QA)
 - a) Scenario based testing
 - b) Performance testing
- 3. User tests (validation tests)

a) Acceptance tests

Contents

Software Testing

Acceptance tests

JUnit

Test Driven Development

How calendars and dates work in Java

Mock objects

Acceptance Tests

Tests defined by / with the help of the user

- based on the requirements
- Traditionally
 - manual tests
 - by the customer
 - after the software is delivered
 - based on use cases / user stories
- Agile software development
 - automatic tests: JUnit, Fit, ...
 - created before the user story is implemented

Example of acceptance tests

Use case

name: Login Admin actor: Admin precondition: Admin is not logged in main scenario

- 1. Admin enters password
- 2. System responds true

alternative scenarios:

- 1a. Admin enters wrong password
- 1b. The system reports that the password is wrong and the use case starts from the beginning

postcondition: Admin is logged in

Manual tests

Successful login

Prerequisit: the password for the administrator is "adminadmin"

Input	Step	Expected Output	Fail	OK
	Startup system	"0) Exit"		
		"1) Login as administrator"		
"1"	Enter choice	"password"	1	
"adminadmin"	Enter string	"logged in"	X	

Failed login

Prerequisit: the password for the administrator is "adminadmin"

Input	Step	Expected Output	Fail	OK
	Startup system	"0) Exit"		1
		"1) Login as administrator"		
"1"	Enter choice	"password"		V
"admin"	Enter string	"Password incorrect"		/
		"0) Exit"		
		"1) Login as administrator"		

Automatic test for the main scenario

Manual vs. automated tests

- Manual tests should be avoided
 - They are expensive (time and personal) to execute: Can't be run often
- Automated tests
 - Are cheap (time and personal) to execute: Can be run as soon something is changed in the system
 - \rightarrow immediate feedback if a code change introduced a bug
 - \rightarrow Regression tests
 - More difficult (but not impossible) when they include the UI
 - → Solution: Test under the UI
- Robert Martin (Uncle Bob) in
 - http://www.youtube.com/watch?v=hG4LH6P8Syk
 - manual tests are immoral from 36:35
 - how to test applications having a UI from 40:00

Testing under the UI



Automatic tests

Successful login

```
@Test
public void testLoginAdmin() {
   LibraryApp libApp = new LibraryApp();
   assertFalse(libApp.adminLoggedIn());
   boolean login = libApp.adminLogin("adminadmin");
   assertTrue(login);
   assertTrue(libApp.adminLoggedIn());
}
```

Failed login

```
@Test
public void testWrongPassword() {
  LibraryApp libApp = new LibraryApp();
  assertFalse(libApp.adminLoggedIn());
  boolean login = libApp.adminLogin("admin");
  assertFalse(login);
  assertFalse(libApp.adminLoggedIn());
}
```

Contents

Software Testing

Acceptance tests

JUnit

Test Driven Development

How calendars and dates work in Java

Mock objects

JUnit

- Framework for automated tests in Java
- Developed by Kent Beck and Erich Gamma
- Unit-, component-, and acceptance tests
- http://www.junit.org
- ► *x*Unit

JUnit and Eclipse

► JUnit 4.x libraries



New source directory for tests

pe fiter text	Java Build Path	(⇒ × c) ×	¥
Resource Builders Java Build Path	Cource Epoyects Libraries & Order and Export Source folders on build path:		í
Java Code Style	> 🧀 library/src	Add Folder	
Java Compiler	va Compiler > @ library/test		
Javadoc Location		<u>E</u> dit	
JML2 Plug-in Project References Refactoring History Run/Debug Settings Task Repository Task Tags		Bemove	
Validation WikiText	Allow output folders for source folders Default output folder:		
	library/bin	Browse	

JUnit 4.x structure

```
import org.junit.Test;
import static org.junit.Assert.*;
public class C {
  @Test
  public void m1() {..}
  @Test
  public void m2() throws Exception {..}
  ...
}
```

Independent tests

No try-catch blocks (exception: checking for exceptions)

JUnit 4.x structure (Before and After)

public class C {

$$(aAfter)$$

public void n2() {...}
 $(Before)$
public void n1() {...}
 $(Test)$
public void m1() {...}
 $(Test)$
public void m2() {...}
...}
 $(M_1(), M_2(), M_2(), M_2(), M_2())$
 $M_2(), M_2()$
 $M_2(), M_2(), M_2()$
 $M_2(), M_2()$
 $M_2(), M_2(), M_2($

Struture of test cases

- Test class = one use case
- Test method = one scenario
- Use inheritance to share sample data between use cases

public class TestBorrowBook extends SampleDataSetup {..}

JUnit assertions

General assertion

import static org.junit.Assert.*;

assertTrue (bexp) assertTrue (msq, bexp)

Specialised assertions for readability

- assertFalse(bexp)
- 2. fail()
- 3. assertEquals(exp, act)
- assentime (false) lais (exp, act) as sent True (exp. equals (act))
 - assertNull(obj)
 - assertNotNull(obj)
 - . . .

JUnit: testing for exceptions

Test that method m() throws an exception MyException

```
@Test
public void testMThrowsException() {
    ...
    try {
        m();
        fail(); // If we reach here, then the test fails because
        // no exception was thrown
    } catch(MyException e) {
        // Do something to test that e has the correct values
    }
} Alternative
    @Test(expected=MyException.class)
    public void testMThrowsException() {..}
```

Contents

Software Testing

Acceptance tests

JUnit

Test Driven Development Test Driven Development Example of Test-Driven Development Refactoring

How calendars and dates work in Java

Mock objects

- Test before the implementation
- Tests = expectations on software
- All kind of tests: unit-, component-, system tests

Traditional testing





Moving to TDD





Real TDD



TDD cycle

- Repeat for functionality bug, ...
 red: Create a *failing* test green: Make the test pass refactor: clean up your code
 - Until: no more ideas for tests
 - Important:
 - One test at a time
 - Implement only as much code so that the test does not fail.
 - If the method looks incomplete,
 - $\rightarrow~$ add more failing tests that force you to implement more code

Ideas for tests

- 1. Use case scenarios (missing functions): Acceptance tests
- 2. Possibility for defects (missing code): Defect tests
- 3. You want to write *more* code than is necessary to pass the test
- 4. Complex behaviour of classes: Unit tests
- 5. Code experiments: "How does the system behave, if ... "
- \rightarrow Make a list of new test ideas

TDD example: Borrow Book

Use case

name: borrow book **description:** the user borrows a book actor: user should be more! User identifies itself

main scenario:

1. the user borrows a book 🥣

alternative scenario

- 1. the user wants to borrow a book, but has already 10 books borrowed
- the system presents an error message

Create a test for the main scenario

test data:

- a user with CPR "1234651234" and book with signature "Som001"
- Test case
 - Retrieve the user with CPR number "1234651234"
 - Retrieve the book by the signature "Som001" ///
 - The user borrows the book
 - The book is in the list of books borrowed by that user

Create a test for the main scenario

```
@Test
public void testBorrowBook() throws Exception {
   String cprNumber = "1234651234";
   User user = libApp.userByCprNumber(cprNumber);
   assertEquals(cprNumber, user.getCprNumber());
   String signature = "Som001";
   Book book = libApp.bookBySignature(signature);
   assertEquals(signature, book.getSignature());
   List<Book> borrowedBooks = user.getBorrowedBooks();
   assertralse(borrowedBooks.contains(book));
   user.borrowBook(book);
   borrowedBooks = user.getBorrowedBooks();
   assertEquals(1,borrowedBooks.size());
   assert frue (borrowedBooks.contains (book));
```

Implement the main scenario

public void borrowBook(Book book) {
 borrowedBooks.add(book);
}

Create a test for the alternative scenario

- test data:
 - a user with <u>CPR "1234651234</u>", book with signature "<u>Som001</u>", and 10 books with signatures "book1", ..., "book10"
- Test case
 - Retrieve the user with <u>CPR number "1234651234"</u>
 - Retrieve and borrow the books with signature "book1", ..., "book10"
 - Retrieve and borrow the book by the signature "Som001"
 - Check that a TooManyBooksException is thrown

Implementation of the alternative scenario

```
public void borrowBook(Book book) throws TooManyBooksException
    if (borrowedBooks.size() >= 10) {
      throw new TooManyBooksException();
    }
    borrowedBooks.add(book);
}
```

More test cases

- What happens if book == null in borrowBook?
- Test Case:
 - Retrieve the user with CPR number "1234651234"
 - Call the borrowBook operation with the null value
 - Check that the number of borrowed books has not changed

Final implementation so far

```
public void borrowBook(Book book) throws TooManyBooksException
    if (book == null) return;
    if (borrowedBooks.size() >= 10) {
      throw new TooManyBooksException();
    }
    borrowedBooks.add(book);
}
```

Another example

- Creating a program to generate the n-th Fibonacci number
- → Codemanship's Test-driven Development in Java by Jason Gorman

http://youtu.be/nt2KKUSSJsY

- Note: The video uses JUnitMax to run JUnit tests automatically whenever the test files change (junitmax.com)
- > A tool with similar functionality but free is Infinitest
 (https://infinitest.github.io)

Refactoring and TDD

- Third step in TDD
- restructure the system without changing its functionality
- Goal: *improve* the design of the system, e.g. remove code duplication (DRY principle)
- Necessary step

Requires good test suite

 \rightarrow later in the course more about refactoring mechanics

TDD: Advantages

Test benefits

- Good code coverage: Only write production code to make a failing test pass
- Design benefits
 - Helps design the system: defines usage of the system before the system is implemented
 - \rightarrow Testable system

Contents

Software Testing

Acceptance tests

JUnit

Test Driven Development

How calendars and dates work in Java

Mock objects

How to use Date and calendar (I)

- Date class deprecated
- Calendar and GregorianCalendar classes
- An instance of Calendar is created by new GregorianCalendar() // current date and time new GregorianCalendar(2011, Calendar.JANUARY,10)
- Note that the month is 0 based (and not 1 based). Thus 1
 February.
- Best is to use the constants offered by Calendar, i.e.
 Calendar.JANUARY

How to use Date and calendar (I)

- One can assign a new calendar with the date of another by newCal.setTime(oldCal.getTime())
- One can add years, months, days to a Calendar by using add: e.g.

cal.add(Calendar.DAY_OF_YEAR,28)

- Note that the system roles over to the new year if the date is, e.g. 24.12.2010
- One can compare two dates represented as calendars using before and after, e.g.

currentDate.after(dueDate)

Contents

Software Testing

Acceptance tests

JUnit

Test Driven Development

How calendars and dates work in Java

Mock objects

Problems

- How to test that a book is overdue?
 - Borrow the book today
 - Jump to the data in the future when the book is overdue
 - Check that the book is overdue



- How do we jump into the future?
- → Replace the GregorianCalendar class by a *mock* object that returns fixed dates
 - Problem: Can't replace GregorianCalendar class

Creating a DateServer class



Creating a DateServer class

The DateServer can be mocked



How to use

Import helper methods

import static org.mockito.Mockito.*;

Create a mock object on a certain class

SomeClass mockObj = mock(SomeClass.class),

return a predefined value for m1(args)

when(mockObj.m1(args)).thenReturn(someObj);

verify that message m2(args) has been sent

verify(mockObj).m2(args);

Mock Example 1: Overdue book

```
@Test
public void testOverdueBook() throws Exception {
    DateServer dateServer = mock(DateServer.class);
    libApp.setDateServer(dateServer);
    Calendar cal = new GregorianCalendar(2011,Calendar.JANUARY,10);
    when(dateServer.getDate()).thenReturn(cal);
    ...
    user.borrowBook(book);
    newCal = new GregorianCalendar();
    newCal.setTime(cal.getTime());
    newCal.add(Calendar.DAY_OF_YEAR, MAX_DAYS_FOR_LOAN + 1);
    when(dateServer.getDate()).thenReturn(newCal);
    assertTrue(book.isOverdue());
}
```

LibraryApp Code



Testing for e-mails



```
@Test
public void testEmailReminder() throws Exception {
   DateServer dateServer = mock(DateServer.class);
   libApp.setDateServer(dateServer);
   MailService mailService = mock(MailService.class);
   libApp.setMailService(mailService);
   ...
   libApp.sendEmailReminder();
   verify(mailService).send("..","..","..");
}
```

Check that no messages have been sent

verify(ms,never()).send(anyString(), anyString(), anyString());

Mockito documentation: http://docs.mockito. googlecode.com/hg/org/mockito/Mockito.html

Exercises and Next Week

- Exercises
 - Programming exercise number 3
 - Exercise 3: Acceptance Tests and TDD
- Systematic tests and code coverage