Testing

Af Kasper

I softwareudvikling er det vigtigt at sikre sig, at det der bliver produceret også fungerer som det skal. Firmaer benytter software til både behandling af persondata, lønstyring og behandling af anden fortrolig data, og derfor er det vigtigt, at alt er sikkert og fungere efter hensigten når det bliver sendt live.

# Manuel Testing

Vi har alle i gruppen individuelt gået igennem alle funktioner i programmet, og sikret os at det hele fungerer. Dette har inkluderet at prøve at sende forkert data i input-felter, se sider, hvor man skal være logget ind som administrator, uden at være det og tjekke efter ødelagte links. Vi har derefter fikset alle fejl der blev fundet, for at levere et bedre produkt.

# Automatiseret Testing (JUnit)

Vi har Unit Tests på de mest essentielle metoder i klassen DataAccessObject, da dette er den klasse der foretager de fleste vigtige metodekald.

public class DataAccessObjectTest {

DataAccessObject instance = null;

Random r = new Random();

public DataAccessObjectTest() {

}

@BeforeClass

public static void setUpClass() {

}

@AfterClass

public static void tearDownClass() {

}

@Before

public void setUp() throws Exception {

instance = new DataAccessObjectImpl();

}

@After

public void tearDown() {

}

/\*\*

\* Test of getUserByUsername method, of class DataAccessObjectImpl.

\*/

@Test

public void testGetUserByUsername() throws Exception {

String username = "test1";

String expResult = "test1@test1.com";

User u = instance.getUserByUsername(username);

String result = u.getEmail();

assertEquals(expResult, result);

}

Her ses et eksempel på en test af DataAccessObject. Der oprettes en ny version af objektet hver gang en metode kaldes, for at sikre at en tidligere test ikke har ændret noget i objektet.

Testen verificerer at man kan logge ind som en bruger. Testen er dog ikke optimal i det den bruger den faktiske databaseforbindelse til at tjekke på en faktisk bruger, der er lagt i databasen til testbehov. Det optimale vil være at skrive en mock-database connection med for eksempel Mockito. Dette vil give den fordel at man kun tester om metoden gør det, der er meningen, ved at mocke de svar der skal sendes til metoden. Dette er kernen af en Unit-test – at teste en specifik metode, for at sikre at de enkelte dele i projektet fungerer som de skal.

Fordelen ved at teste med den faktisk database koblet på projektet er, at man samtidig metoden, også får testet om ens metode fungerer som planlagt. Dog bevæger dette sig mere over i det område det kaldes Integration Tests i stedet for Units Tests.

Det er dog vigtigt at have både Unit- og Integration tests, da begge er vitale for at se om ens projekt fungerer som planlagt.