

07.01.23

Testing av programvare

50% praktisk del, 50% skoleeksamen

PRAKTISK DEL - frem til påske...

Her skal vi teste en ferdig java-applikasjon som krever noe spring boot / java forståelse (enhetstest). Vi skal automatisere noen tester f.eks: enhetstest, integrasjonstest og systemtesting. Vi skal også bruke et test-håndteringssystem → Microsoft Test Management system (MTM). MTM vil brukes for å dokumentere systemtest

TEORI DEL

Efter at den praktiske delen er levert vil International software testing qualification board (ISTQB) providere pensum på tre ulike nivå:

1. Foundation
2. Advanced
3. Expert

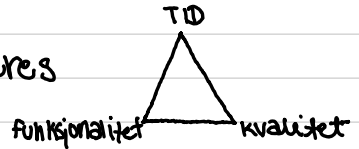
hvor vi vil dekke 1 og litt 2. Skriftlig 3 timers eksamen.

? HVORFOR TESTING

- Fordi vi mennesker gjør feil hele tiden
- vi må finne feil hele tiden
- må sikre at brukerne blir fornøyd
- sikre kvalitet i systemet
- sikre minimalt med nødvendig vedlikehold
- vi må sikre god ytelse på løsningen

PROGRAMVAREUTVIKLING

- Introdusere feil i programvaren når den utvikles
- Finne disse feilene
- Rette disse feilene
- Samtidig som nye feil introduseres
- Finne disse feilene
- Rette disse feilene
- Samtidig som nye feil introduseres
- ...



UTFORDRINGER

Testing koster mye og er ineffektivt, du kan ikke teste alt, du kan ikke bevise at ting virker, poenget er å vise at ting ikke virker og jo flere problemer du finner, jo bedre.

GRUPPEOPPGAVE

- ↳ Gruppe på 1-4, leveres for påske
- ↳ Utestet nettbankapplikasjon skal enhetstestes, integrasjonstestes og systemtestes
- ↳ Skal dokumentere dette med bruerhistorier for å kunne integrasjons- og systemteste denne
- ↳ Startes med enhetstesting

Det legges opp til å bruke automatiseringsverktøy for å teste:

- IntelliJ Spring Boot - Java - enhetstest
- SoapUI - Integrasjonstest
- Selenium - systemtest

Det legges opp til å bruke MTM for å dokumentere systemtest

- Microsoft Test Management system (MTM) under Team Foundation Server (TFS)

Enhetstesting / komponenttesting

- skal gjøres av programmereren som utvikler akkurat denne enheten (metoden)
- Det skal skrives kode for å teste annen kode
- Viktig at kodene som skal testes er små (enklere)
- Viktig at testene er uavhengige av hverandre
 - ↳ skal kjøres hver gang det er en endring → smidig utvikling
- skal ikke kjøres mot database
 - ↳ tar for lang tid
 - ↳ databasen må "resettes" hver gang
 - ↳ databasen skal aksesseres ved integrasjonstesten
- Test drevet design (TDD)
 - ↳ lag enhetstesten som feiler før du lager innholdet i metoden.

HVORFOR AUTOMATISERTE TESTER

- Det er viktig å teste så mye som mulig, så fort som mulig. Kostnaden øker vesentlig lenger ut i utviklingsløpet
- Bruk av test drevet design øker kvaliteten (TDD)
 - å lage testene før programmet
- Gir muligheter for å gjøre endringer i koden fort og sjekke om det innebærer noen konsekvenser (feil)
 - viktig i agil metodikk
- Det er en mekanisme for å beskrive funksjonalitet i programmet på to måter:
 1. I programmet selv
 2. I testen

Det gir økt kvalitet da disse må samstemme


over kontrollen bare mot

enna pædikning
naan pælekren

LAGDELING

- I større systemer vil det være krav om lagdeling
 - ↳ dette for å få systematisert og gruppere kode
 - ↳ for å få oversikt over stor mengde kode
- Det er vanlig med følgende lag:
 1. Presentasjonslag
 2. Forretningslag (business logical layer, BLL)
 3. Data aksess lag (data access layer, DAL)
- Det muliggjør bedre testing
- I den "mindre" oppgaven som Netbanken utgjør har vi valgt å ikke introdusere et BLL Lag, men har et slags "Service lag" som kan utgjøre sikkerhetsklassen.

MOCK'S

- Vi må mocke repository og da databasekoden
 - ↳ betyr å erstatte kallet til repository med en setning som værlig: 

`when (repository.hentEnkonto(anyString())) thenReturn (kontoen);`
Når kundeControlleren eller Controlleren kommer borti og skal kalle repository sin `hentEnkonto`, en eller annen String, så returnerer vi en konto.

Poenget er at vi kan skrive en setning i testen som gjør at vi ikke kaller repositoryet, men tvinger da det kallet til å returnere en verdi.

Vi går ikke ned i databasen fordi det tar for lang tid.

- Vi kunne lagt alt i minne, altså vurdere en in memory database (h2 som vi bruker er en in memory database med mindre vi gjør den fysisk, slik som den ligger nå er den i minne) det vil ikke ta vedig lang tid men det er ikke riktig måte å gjøre det på, fordi det blir mye mer kode. Vi skl teste bare kontrolleren og det er en og en metode som skal testes. Testene må være uavhengig av hverandre.

ENHETSTESTING AV NETTBANK

- Få oversikt over løsningen
- Lag nye tester og mock repository og sikkerhetskallene
- sikre at du lager alle testene som skal til for å dekke alle mulige returnverdier fra repository- et
- Det er Controlleren som skal enhetstestes
 - ↳ Det er 3 stk i nettbanken

LAG BRUKERHISTORIER / til systemtest

Brukerhistorier skal være:

- Uavhengig av andre brukerhistorier
- Små
- Mulig å teste

Følgende brukerhistorier er foreslått for Enhetstestkunde-app:

- vil jeg kunne gå til skjermbildet for å registrere en ny kunde fra skjermbildet for å registrere en ny kunde fra skjermbilde "kundeadministrasjonen"
- vil jeg kunne registrere en ny kunde i skjermbildet for "Registrer ny kunde"
- vil jeg få en feilmelding når kundenavnet ikke er mellom 2 og 30 tegn og ikke inneholder norske store og små bokstaver, mellomrom, minustegn og punktum.
- vil jeg kunne gå til skjermbildet for å endre til en eksisterende kunde fra skjermbildet "kundeadministrasjon"
- vil jeg kunne endre alle dataene til en kunde i skjermbildet "endre kunde"
- vil jeg kunne slette en eksisterende kunde fra skjermbildet "kundeadministrasjon"

MIN FØRSTE TEST

```
package test;

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

public class BilTest {

    @Test
    public void testBilFarge() {
        Bil volvo = new Bil();
        volvo.setFarge("Rød");
        String farge = volvo.getFarge();
        assertEquals("expected: 'Rød', farge");
    }
}
```

Kodene ligger ute i modul Enhetstest.

ENHETSTEST - KUNDETTEST

Det er utviklet en applikasjon for registrering og endringer av kundeinformasjon kalt KundeTest.

The assert is used to make an assertion, which is a statement that a certain condition is true. If its false, the assert statement will program to stop executing with an error message. Its typically used for debugging and testing purposes to ensure that certain assumptions about the programs state are valid.

// arrange

// act

// assert

TESTER I NETTBANKEN

Det er meningen at alle metodene i Controllerne i nettbanken skal enhetstestes.

Repository = metoder som kaller SQL

Controller = sørger for om mappingene skjer osv...

Vi skal ikke kjøre kall til repositorier, men simulere de, eller mocke de (som det heter)

NB!

video av hvordan sikkerhetsController kan testes ligger i Canvas her.

@Autowired

BankRepository rep; // kaller BankRepository og er
HTTP session

@Autowired

private HttpSession session // tar vare på session-
variabel om man er
logget på eller ikke

"Vi kan sette kode som kjøres før hver kjøring (før hver test". Yngve har kopiert fra nettet og limt inn under @Before

Hvis man ikke legger inn en @Before kan koden limes inn før hver test under @Test

Den koden er for å sette session attributtet.

22.01.23

Integrasjonstest

Integrasjonstest gjøres etter at enhetstesten er ferdig og det er behov for å teste flere "moduler" i sammenheng. Det betyr f.eks at man tester alle metoder som kaller hverandre fra et api-grensesnitt til klienten og helt til databasen.

Før man starter med å sette opp en integrasjonstest er det en del steg som bør gjennomgå først:

Teststrategi

- Hvordan teste?
- Hvilke komponenter skal testes sammen (integrasjon) (figur)?
- Hvilke verktøy skal brukes?
- Hvor mye skal automatiseres (alt?)

Testplan

- Hvilken funksjonalitet skal testes?
- Hvilke data skal brukes i testen og hvordan?
- Hva skal resultatet av testene være og hvordan skal disse dokumenteres?
- Hvilke steg skal utføres i testen?
- Hvilke steg er avhengig av hverandre (hvilke må kjøres i rekkefølge)?
- Hvordan skal databasen se ut som utgangspunkt?
- Hvordan "resette" databasen mellom ulike steg / sykler?
- Hvordan dokumentere resultatene av testene?

} Excel

Nårliggende test

vanligvis leser en nettleier med HTML/JavaScript/Ajax som kaller Controller (Java) som kaller DAL (repository metoder) som kaller database

Enhetstest

Vi skriver testene i IntelliJ med Java som kaller Controller (Java) og mocker enhetskonteksten altså mocker databasen.

Integrasjonstest

Vi har en applikasjon som vi setter opp (SOAPUI) som skal ta oss kalle med transaksjoner mot Controlleren (JAVA), akkurat som klienten kaller det samme. Dette går videre mot repositoryet og databasen. Vi får dermed svar tilbake innen SOAPUI.

gitt at det er regex på server

Inputvalidering via Reg Ex: dette skal være testet i enhetstest, her er det bare to utfall = OK eller ikke OK.

Soap er en "gammel" kommunikasjonsmåte mellom en klient og metoder eller services på server. Soap står for "Simple Object Access Protocol" og kommuniserer via XML. Vi vil bruke JSON isteden og et "REST" liknende grensesnitt, men det støtter også SoapUI.

SoapUI vil "simulere" kall til Controller-metodene og vil kunne sjekke at det som returneres er det vi forventer. Det betyr at vi kan sette opp tester der vi sender noe data til Controller-metodene og sjekker de forventede resultatene.

I integrasjonstest går vi mot databasen mens i enhetstesten måtte vi mocke/stubbe denne. Integrasjonstestene er ikke like hurtige som enhetstester nettopp fordi de går ned til databasen og har mer kode å sjekke. Enhetstester er uavhengige, men det er ikke integrasjonstestene: Man må først initiere databasen, logge inn også videre.

Vi må også **asserte**, altså sjekke om resultatene vi får tilbake er korrekte. InitDB returnerer OK eller FEIL.

Grunnleggende testprinsipper

ISTQB - International software testing qualification board

- ↳ samling av profesjonelle testere fra over 80 land som har satt ett teknisk rammeverk om hvordan man bør teste. Man kan gå opp til testeksamener for å bli sertifisert tester (finnes ulike nivåer) Vi skal se **foundation level**.

? HVORFOR TESTING

- Fordi vi mennesker gjør feil hele tiden
- vi må finne feil hele tiden
- må sikre at brukerne blir fornøyd
- sikre kvalitet i systemet
- sikre minimalt med nødvendig vedlikehold
- vi må sikre god ytelse på løsningen

Definisjon: testing er en prosess for å verifisere og validere at et produkt, system eller en enhet oppfyller spesifiserte krav og fungerer som forventet. Det kan inkludere manuelle tester utført av en tester, eller automatiserte tester som kjøres ved hjelp av en programvare.