Verifikasjon og validering

16. september 2004 - INF3120

Nils Christian Haugen & Stein Grimstad

[simula . research laboratory]

Hvem er vi?

- Nils Christian Haugen
  - Utvikler hos ThoughtWorks
  - Ansatt ved London-kontoret
  - Utdannet ved NTNU
  - E-post: nhaugen@thoughtworks.com

- Stein Grimstad
  - Doktorgradstipendiat ved Simula/UiO
  - Seniorrådgiver i Objectnet
  - Utdannet ved Universitet i Oslo
  - E-post: sg@objectnet.no

Agenda

- Introduksjon
- Validering og verifikasjon
- Prototyping
- Inspeksjon
- (pause)
- Testing

Mål med forelesningen

- Kjenne til og forstå viktige begreper
- Kjenne til og forstå viktige teknikker
- Forstå at verifikasjon og validering er viktig
- Ha et utgangspunkt for å planlegge og gjennomføre verifikasjon og valideringsaktiviteter i prosjektoppgaven (dog ikke vektlagt tema i prosjektoppgaven)

Pensum

- Pensum er 22, 23 og 24 i "Ian Sommerville, 'Software Engineering', 7th edition 2004".
- Vi vil ikke gjennomgå hele pensum
  - Selvstudium
- Materialet som presenteres er
  - Litt fra Sommerville
  - Litt fra tidligere års forelesninger
  - Litt eget
- I tillegg anbefales det å lese:
  - Myers : ”The art of software testing” ($$$$$)
  - Beck: ”Test-driven Development: By Example”
  - http://www.junit.org/

[simula . research laboratory]

Agenda: Validering og verifikasjon

- Introduksjon
- Validering og verifikasjon (V & V)
  - Definisjoner
  - Motivasjon
  - Feil
- Prototyping
- Inspeksjon
- Testing

V & V: Definisjoner

- Validation: “Are we building the right product?”
- Verification: “Are we building the product right?”

(Sommerville)

- Hvem definerer hva som er riktig?
- Arbeidet med å utbedre avvik som avdekkes er ikke en del av validerings- og verifikasjonsaktivitetene

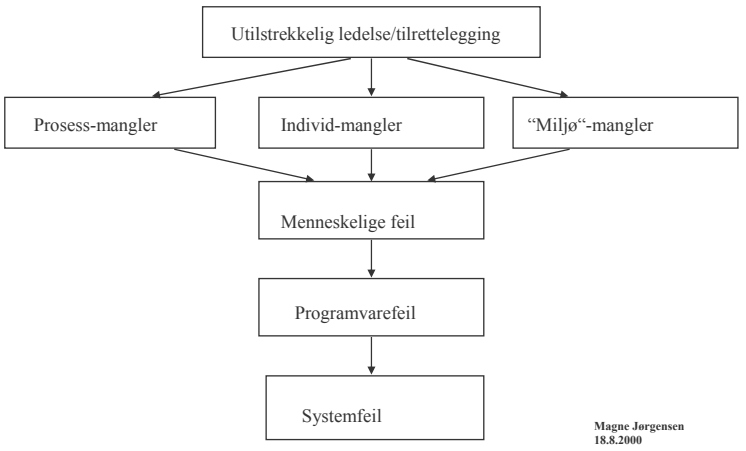
[simula . research laboratory]

V & V: Motivasjon

- Mennesker gjør feil
  - Det vi lager er ikke det vi burde laget
  - Det vi lager har defekter
- Sikre riktig kvalitet i produktet/tjenesten som lages
  - Ikke nødvendigvis ”best mulig” kvalitet: Godt nok
- Jo senere en feil oppdages, desto mer alvorlig er det
  - Typisk regner man en faktor i kostnad på 10 mellom de forskjellige fasene
  - Kan være forretningskritisk (i ytterste konsekvens kan menneskeliv gå tapt)

V & V: Eksempel på feil

V & V: Hvordan oppstår feil?



Magne Jørgensen
18.8.2000

V & V: Hvorfor finnes feil selv etter lansering?

- Det som er laget er feil (ikke det kunden vil ha)
- Ikke alle feil prioriteres rettet
  - Kategori 1: Kritiske feil som MÅ rettes (lansering holdes igjen til feilen er rettet)
  - Kategori 2: Kritiske feil som bør rettes (betydelig reduksjon av kvaliteten)
  - Kategori 3: Ikke-kritiske feil (kosmetiske feil)
- I praksis umulig å få verifisert alle kombinasjoner av input til et software system
- "Confirmation bias", dvs. en tendens til å teste og vektlegge bekreftelser, i motsetning til å prøve å falsifisere.
  - NB: Dette peker på en vesentlig forskjell i holdning mellom det å utvikle og det å teste. En god utvikler er "konstruktiv", mens en god tester er "destruktiv". Mange organisasjoner velger derfor å skille rollene, dvs. å ha egne testere.
- Tendens til å tro at sist funnet feil er "siste feil".

[simula . research laboratory]

V & V: Tre viktige former for V & V

- Prototyping
  - Lage et utkast til hvordan det endelige systemet kan se ut.
  - Benyttes tidlig i utviklingsprosessen
  - Testes mot sluttbruker
- Inspeksjoner
  - Gjennomgang av leveranser
  - Benyttes i alle faser av utviklingsprosessen
  - Utføres hovedsakelig av mennesker, men kan være programmer
- Testing
  - Sjekke hvor godt systemet oppfyller kravene
  - Gjøres sent i utviklingsprosessen
  - Resultatet av kjøringer sammenlignes med forhåndsdefinert fasit

Agenda: Prototyping

- Introduksjon
- Validering og verifikasjon
- Prototyping
  - Formål
  - Verktøy
  - Eksempler
- Inspeksjon
- Testing

[simula . research laboratory]

Prototyping: Formål

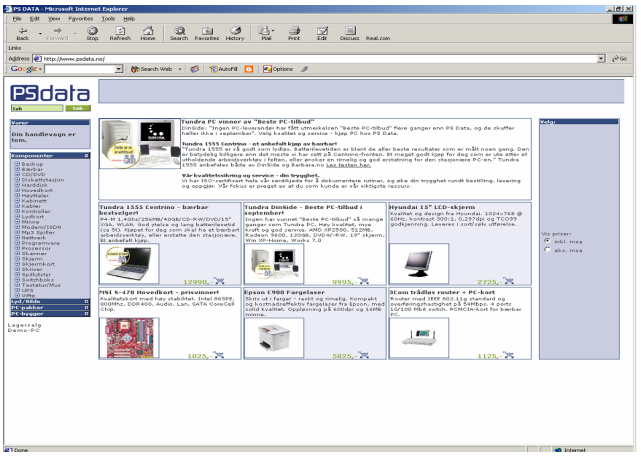
- Lager utkast til hvordan systemet kan se ut
  - Tegninger (storyboarding)
  - Skjermbilder uten funksjonalitet
- Brukes til validering av systemet i krav/analyse-fasen
  - Enkelt for brukere å forholde seg til (i forhold til modeller og dokumenter)
  - Avdekker effektivt manglende/feil krav og forretningsregler
- Ulemper:
  - Gode prototyper kan gi falske forventninger i forhold til hvor langt prosjektet har kommet
  - Brukere kan bli opphengt i irrelevante detaljer

Prototyping: Verktøy

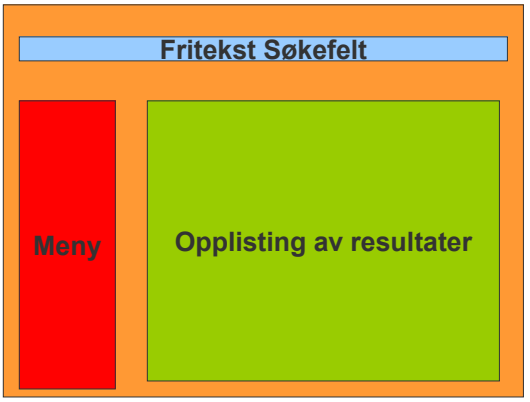
- Penn og papir
- Whiteboard
- Tegneprogrammer
- PowerPoint
- HTML-editorer
- GUI-byggere
- Modellbaserte prototypingsverktøy
- Programmeringsspråk

[simula . research laboratory]

Prototyping: Eksempler (1)



Prototyping: Eksempler (2)



Agenda: Inspeksjon

- Introduksjon
- Validering og verifikasjon
- Prototyping
- Inspeksjon
  - Dokumentgjennomgang
  - Kodegjennomgang
  - Automatisert statisk kodeanalyse
  - Matematisk verifikasjon
- Testing

Inspeksjon: Dokumentgjennomgang

- Gjennomgang av dokumenter med formål å finne feil og mangler
- Forskjellige teknikker kan benyttes
  - ”parprogrammering” (kontinuerlig inspeksjon)
  - Forskjellige typer gjennomgangsmøter (dokumentet presenteres i møtet, distribueres på forhånd, forskjellige typer fasilitatorer...)
  - Aspektbasert inspeksjon
- En typisk fremgangsmåte
  - dokumentet distribueres for gjennomlesning
  - arrangerer gjennomgangsmøte
- Hvem bør gjennomgå dokumentet?
  - De som skal ha systemet
  - De som skal bruke dokumentet
  - De som har vært delaktige i utformingen av dokumentet
  - Eksperter (rollen / forretningsområdet)
  - Personer du av ulike årsaker ønsker en godkjennelse fra

[simula . research laboratory]

Inspeksjon: Kodegjennomgang

- Gjennomgang av kildekoden for å finne feil og mangler
  - Funksjonelle feil (avvik fra design og kravspesifikasjon)
  - Avvik fra kodestandard og retningslinjer (for eksempel navngiving av klasser)
  - Tekniske feil (for eksempel feil i en hashmap-funksjon)
  - Testbarhet (for eksempel dokumentasjonsmangler, muligheten for isolasjon, etc)
- Forskjellige teknikker kan benyttes
  - Parprogrammering (kontinuerlig kodegjennomgang)
  - Distribusjon til en eller flere for gjennomlesning
  - Koden gjennomgås av en review-gruppe i et møte
  - Aspektbasert inspeksjon
- Hvem bør gjennomgå kode?
  - Sjefsprogrammerer/designere
  - Juniorprogrammerere (opplæring)
  - Testere
  - Tekniske eksperter
  - Arkitekter

Inspeksjon: Automatisert statisk kodeanalyse

- Programmer som analyserer kildekode og avdekker avvik fra spesifiserte krav
  - Sjekker at alle klassenavn begynner med store bokstaver
  - Sjekker at metoder ikke er for lange
  - Sjekker at input-paramtere valideres
  - Sjekke at alle metoder er dokumentert
  - Ikke-eksekverbar kode
  - Variabler som aldri brukes
  - Minnehåndtering
- Kan integreres i utviklingsmiljøet, versjonshåndteringssystemet eller byggesystemet
  - For eksempel kan kode hvor verktøyene rapporterer feil i nektes innsjekking
- Raskt å kjøre (billig) i motsetning til inspeksjon
  - Men mange ting fanges ikke opp her....
- Eksempel på verktøy:
  - Kompilatorer, IntelliJ, jDepend, jMetric

[simula . research laboratory]

Inspeksjon: Matematisk verifikasjon

- Matematisk bevis for at programmene oppfører seg i henhold til kravspesifikasjon
  - Matematisk bevise konsekvensen av hver programmeringsinstruks
  - Fordrer at kravspesifikasjonen er uttrykt på et egnet format:

    void Sort (A,n);
    Inndata: n tall A[0..n-1] vilkårlig rekkefølge
    Utdata: A'[i-1] <= A'[i], 0< i < n
- Problem
  - Programmeringsspråk er ikke nødvendigvis veldefinerte
  - Komplisert for avanserte strukturer
  - Bevisene er i seg selv kompliserte og sjansen for feil er stor
- Veldig, veldig dyrt (typisk $1000 per kodelinje)
  - Romferger
  - Atomreaktorer
  - T-banen i Paris
- Brukes lite i industrien
- For mer informasjon: Ta kurset IN217 (INF 4230)

Agenda: Testing

- Introduksjon
- Validering og verifikasjon
- Prototyping
- Inspeksjon
- Testing
  - Innledning
  - Testing vs. debugging
  - Typer av test
  - V-modellen
  - Testplan
  - Testdata
  - Ekvivalensklasser
  - Enhetstesting (med verktøy og eksempler gjerne relatert til XP)
  - Testdrevet utvikling
  - Gjenbruk

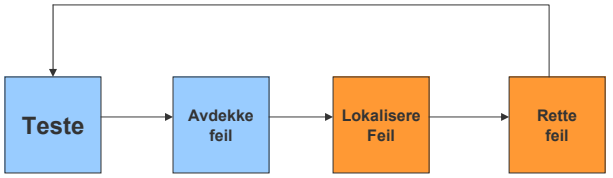
[simula . research laboratory]

Testing: Innledning

- En form for ”induktiv bevisføring”, dvs. å vise (sannsynliggjøre) at alle svaner er hvite ved å observere et representativt utvalg av alle svaner.
- Test er både verifikasjon og validering
- Testing kontrollerer ny funksjonalitet, samt at eksisterende funksjonalitet fortsatt virker (regresjonstesting)
- Effektiv testing gjøres ved:
  - Testbare krav
  - Testbare programvarestrukturer
  - Bruk av ekvivalensklasser
  - Testverktøy
  - Gjenbruk

Testing: Testing vs. debugging

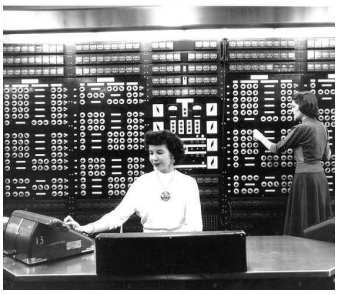
- Testing er ikke det samme som debugging
- Testing er å finne feil
- Debugging er å rette feil
- ...men begge deler gjøres ofte i test-fasen av prosjektet (noe uklare grenser)

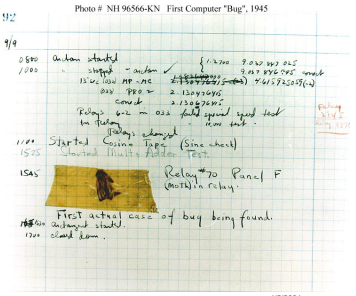


[simula . research laboratory]

Testing: Det første virkelige tilfelle av ”bug”

- 9. september 1945, kl. 3:45 p.m., fant forskere ved Harvard-universitetet årsaken til at Mark II Aiken Relay kalkulatoren oppførte seg merkelig
- En møll ble funnet fanget mellom punkter på relé #70, panel F
- Maskinen ble ”debugget” med en pinsett!
- Dokumentert i loggen som ”First actual case of bug being found.” med møllen tapet inn ved siden av





Testing: Typer av test (1)

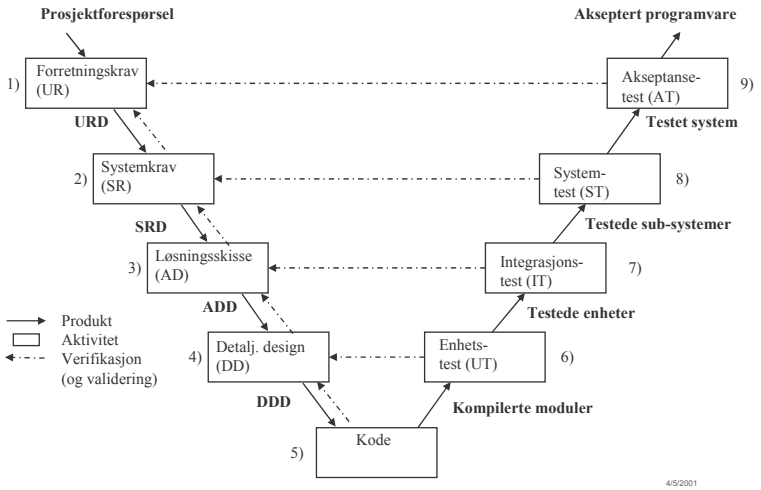
- Betatest
  - Utvalgte kunder tar i bruk systemet før offisiell lansering
- Akseptansetest
  - Tester at systemet lar brukerne gjøre det de trenger
  - Tester med reelle data
  - Utføres gjerne i samarbeid mellom kunder og testere
- Systemtest
  - Tester at systemet oppfører seg korrekt i samspill med omgivelsene
  - Testdata konstrueres ofte
  - Utføres ofte av en egen testgruppe
  - Inkluderer også ikke-funksjonelle tester
    - Ytelsestest (tester ytelse = hastigheten på én transaksjon)
    - Stresstest (tester skalerbarhet = hastigheten på mange samtidige transaksjoner)
    - Recoverability-test (tester systemets håndtering av uforutsette avbrudd)

[simula . research laboratory]

Testing: Typer av test (2)

- Integrasjonstest
  - Tester at komponenten (klassen) virker sammen med andre komponenter
  - Simulerer ofte andre sub-systemer
  - Bruker konstruerte testdata
  - Utføres ofte av utviklere
  - Gjøres ofte manuelt, men kan med fordel automatiseres
- Enhetstest
  - Tester at komponenten (klassen) virker isolert
  - Simulerer omgivelsene til komponenten
  - Utføres av utviklere
  - Skrives ofte som automatiske tester

Testing: V(erifikasjons)-modellen



[simula . research laboratory]

Testing: Testplan

- V&V-plan (kvalitetssikringsplan) er en del av prosjektplanen
- Testplanen er en del av en V&V-plan
  - Fokus på feilavdekking/-retting ved kjøring av programmer
- Testplanen bør inneholde beskrivelse av
  - enhetstest (UT)
  - integrasjonstest (IT)
  - systemtest (ST)
  - akspetansetest (AT)
- Stor sannsynlighet for at man må re-planlegge som en følge av forsinkelser og endring av ”scope” under utvikling

Testing: Eksempel på innhold i testplan

- Generelt:
  - Tidsplan (hvilke tester når)
  - Ressursbruk (mennesker/systemer/maskiner/data)
  - Testleveranser (testscript, testrapporter, dokumentasjon, etc.)
  - Risikofaktorer
- For hvert testdesign
  - Gjennomføring av test for en eller flere "features"
  - Relatert til kravspesifikasjon
  - "Features" som skal testes/ikke testes
  - Hvilke "test-cases" inngår
- For hver test-case
  - Situasjoner som tester "features" til programvaren
  - Input og forventet output
  - Eksekveringsbetingelser (f.eks. hvilke kundedata som skal ligge i databasen)
  - Testprosedyre (detaljert steg-for-steg beskrivelse for gjennomføring av testen)

[simula . research laboratory]

Testing: Testdata

- Eksempel:
  - Skal teste at en kunde flytter fra Oslo til Svalbard
    - Skal ikke betale moms
  - Bruker Stein som testdata
    - Ok første gang
    - Andre gang feiler testen siden Stein ble flyttet i test nr 1

==> Testdata må resettes mellom hver test

  - Dette er vanskelig
- Vanskelig:
  - å finne komplette/realistiske testdata
  - å isolere testdata mellom tester
  - å teste konfidensielle data
- Testdatabaser får fort dårlig datakvalitet
  - Blir ikke vedlikeholdt på linje med produksjonsdatabaser
  - Data blir herjet mer med enn data i produksjon
  - Feil i kode kan føre til inkonsistente tilstander

Testing: Ekvivalensklasser og grensesnittsverdier

- Hver klasse av input-data som fører til samme oppførsel tilhører samme “ekvivalensklasse”.
- Testing gjøres ikke på alle data
  - Alle ekvivalensklasser bør være representert
  - Grenseverdier, ekstremverdier og feilverdier bør være representert
- Eksempel:
  - Ved innskudd på sparekonto skal kunder få forskjellig rente
    - De med < 100 000 på konto får 0,5% rente
    - De med >= 100 000 på konto får 1% rente
    - Overtrekk er ikke tillatt
  - Hvilke data bør man teste med?

[simula . research laboratory]

Testing: Enhetstesting i praksis

- Målet er å skrive automatiserte enhetstester
- Dette fordrer at koden er tilrettelagt for det

```
class Calculator {
   KeyPad keypad;

   Calculator() {
      this.keypad = new KeyPad();
   }
}
```

```
class Calculator {
   KeyPad keypad;

   Calculator(KeyPad keypad) {
      this.keypad = keypad;
   }
}
```

- xUnit er defacto standard for enhetstesting

Testing: Kode som skal enhetstestes

```
class Money {
    private int amount;
    private String currency;

    public Money(int amount, String currency) {
        this.amount= amount;
        this.currency= currency;
    }
    public int amount() {
        return amount;
    }
    public Money add(Money m) {
        return new Money(amount()+m.amount(), currency());
    }
}
```

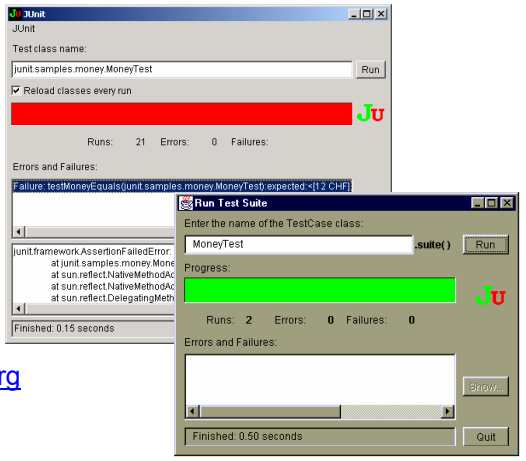
E.g. NOK, USD, EUR

[simula . research laboratory]

Testing: Eksempel på en jUnit-test

```
public class MoneyTest extends TestCase {
    //…
    public void testSimpleAdd() {
        Money m12EUR= new Money(12, "EUR");
        Money m14EUR= new Money(14, "EUR");
        Money expected= new Money(26, "EUR");
        Money result= m12EUR.add(m14EUR);
        Assert.assertTrue(expected.equals(result));
    }
}
```

Testing: Eksempel på kjøring av jUnit



http://www.junit.org

[simula . research laboratory]

Testing: Gjenbruk

- Kode som gjenbrukes er allerede testet (forhåpentligvis!)
  - Mange penger å spare
- Men ikke nødvendigvis testet for ditt bruk
  - ikke testet i din anvendelse
  - ikke testet med din konfigurasjon

Testing: Testingens dillemma

Men hvem tester testene?