

Verifikasjon og validering

17. oktober 2005 - INF3120

Nils Christian Haugen & Stein Grimstad

Hvem er vi?

- Nils Christian Haugen
 - Seniorutvikler 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

[**simula** . research laboratory]

Agenda

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

4/5/2001

[**simula** . research laboratory]

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)

4/5/2001

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 eget
- I tillegg anbefales det å lese:
 - Myers : "The art of software testing" (\$\$\$\$)
 - Beck: "Test-driven Development: By Example"
 - Mugridge & Cunningham: "Fit for Developing Software"
 - <http://www.junit.org/>
 - <http://fit.c2.com/>

4/5/2001

Agenda: Validering og verifikasjon

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

4/5/2001

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

4/5/2001

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)

4/5/2001

V & V: Eksempel på feil



4/5/2001

V & V: Hvordan oppstår feil?



Magne Jørgensen
18.8.2000

4/5/2001

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".

4/5/2001

V & V: Tre viktige teknikker 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 tradisjonelt sent i utviklingsprosessen
 - Resultatet av kjøring sammenlignes med forhåndsdefinert fasit

4/5/2001

Agenda: Prototyping

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

4/5/2001

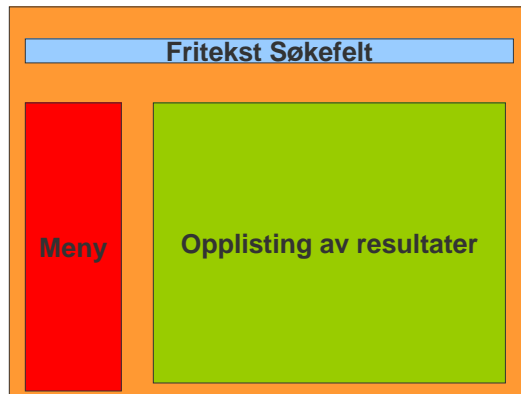
Prototyping: Formål

- 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

4/5/2001

[**simula** . research laboratory]

Prototyping: Eksempler (2)



4/5/2001

[**simula** . research laboratory]

Agenda: Inspeksjon

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

4/5/2001

Inspeksjon: Dokumentgjennomgang

- Gjennomgang av dokumenter med formål å finne feil og mangler
- Forskjellige teknikker kan benyttes
 - Skrive dokumentet sammen i par (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
 - Ekspertene (rollen / forretningsområdet)
 - Personer du av ulike årsaker ønsker en godkjennelse fra

4/5/2001

Inspeksjon: Kodegjennomgang

- Gjennomgang av kildekoden for å finne feil og mangler
 - Funksjonelle feil (avvik fra design og kravspesifikasjon)
 - Avvik fra kodenstandard 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

4/5/2001

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-parametere 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, Eclipse, jDepend, jMetric

4/5/2001

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)

4/5/2001

Agenda: Testing

- Introduksjon
- Validering og verifikasjon
- Prototyping
- Inspeksjon
- Testing
 - Innledning
 - Testing vs. debugging
 - Typer av test
 - Testplan
 - Testdata
 - Ekvivalensklasser
 - Gjenbruk
 - Introduksjon til Fit

4/5/2001

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

4/5/2001

Testing: Typer av test (1)

- Enhetstest
 - Tester at objektet virker isolert
 - Simulerer omgivelsene til objektet
 - Bruker konstruerte testdata
 - Utføres av utviklere
 - Skrives ofte som automatiske tester
- Modultest
 - Tester at objektet virker sammen med de andre objektene i modulen
 - Simulerer som regel omgivelsene til modulen
 - Bruker konstruerte testdata
 - Utføres ofte av utviklere
 - Gjøres ofte manuelt, men kan med fordel automatiseres

4/5/2001

Testing: Typer av test (2)

- Systemtest
 - Tester at hele systemet oppfører seg korrekt
 - 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)
- Akseptansetest
 - Tester at systemet lar brukerne gjøre det de trenger
 - Tester med reelle data
 - Utføres gjerne i samarbeid mellom kunder og testere
- Betatest
 - Utvalgte kunder tar i bruk systemet før offisiell lansering

4/5/2001

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 prosedyrer for
 - enhetstest
 - modultest
 - systemtest
 - akseptansetest
- Stor sannsynlighet for at man må re-planlegge som en følge av forsinkelser og endring av "scope" under utvikling

4/5/2001

Testing: Eksempel på innhold i testplan

- Generelt:
 - Tidsplan (hvilke tester når)
 - Ressursbruk (mennesker/systemer/maskiner/data)
 - Testleveranser (testscript, testrapporter, dokumentasjon, etc.)
- For hvert testdesign
 - Gjennomføring av test for en eller flere krav
 - Krav som skal testes/ikke testes
 - Hvilke test scenarier inngår
- For hver test scenario
 - Situasjoner som tester krav til programvaren
 - Inndata og forventet utdata (f.eks. hvilke kundedata som skal ligge i databasen)
 - Betingelser (f.eks. at alle moduler er oppe og kjører normalt)
 - Testprosedyre (detaljert steg-for-steg beskrivelse for gjennomføring av testen)

4/5/2001

Testing: Testdata

- Tilstanden til testdata må resettes mellom hver test kjøring
 - Eksempel:
 - Skal teste at en kunde som flytter fra Oslo til Svalbard slutter å betale moms
 - Bruker Stein som testdata
 - Ok første gang
 - Andre gang feiler testen siden Stein ble flyttet i test nr 1
- utfordringer:
 - 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

4/5/2001

Testing: Ekvivalensklasser og grensesnittsverdier

- Hver klasse av inndata 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?

4/5/2001

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

4/5/2001



ThoughtWorks®
The art of heavy lifting.™

Fit: Framework for Integrated Test

- Created by Ward Cunningham in 2002
- Tool for enhancing team collaboration and communication
 - Based on specification by example
 - Developed in collaboration between customer, tester and programmer
- Executable HTML documents
 - Readable to all team members
 - Self-testable

Fit Workflow: Describing Feature

SAVE
Dec 1st SDUs

Hourly Compensation

regular wage - 40 hrs @ \$20/hr = \$800/wk
 overtime wage - 45 hrs @ \$20/hr = \$900 + 150 = 950
 Sunday wage - 8 hrs @ \$20/hr = \$320
 Holiday wage - ← SAME →

What about >40 hrs holiday?? → Meaningless?? Ask Bob S.

typical example

M	T	W	Th	F	S	Su
8	8	8	8	8	4	4

Only fresh types of hours, not days

44 regular hours @ \$20/hr
 4 holiday hours
 \$1080 wage

700 + 150 + 150 = 1000

Source <http://fit.c2.com/>

Fit Workflow: Refining Examples

compensation.htm - Microsoft Word

45	0	20	\$950
48	8	20	\$1360

What about >40 hours holiday?

standard hours	holiday hours	wage	pay
0	48	20	??
0	0	20	0
40	0	0	0
-20	0	0	error
0	-4	0	error
0	0	-8	error

Page 1 Sec 1 1/1 At 3.9" Ln 16 Col 1 REC TRK EXT OVR E

Source <http://fit.c2.com/>

Fit Workflow: Fityfing Tables

```
public class WeeklyCompensation : ColumnFixture
{
    public int StandardHours;
    public int HolidayHours;
    public Currency Wage;

    public Currency Pay()
    {
        WeeklyTimesheet timesheet = new WeeklyTimesheet(StandardHours, HolidayHours);
        return timesheet.CalculatePay(Wage);
    }
}
```

What about ≥40 hours holiday?

Payroll.Fixtures.WeeklyCompensation	HolidayHours	Wage	Pay()
0	48	20	??
0	0	20	0

Source <http://fit.c2.com/>

Fit Workflow: Incremental Progress

compensation: first 40 hours x1, overtime x1.5, holiday x2

Payroll.Fixtures.WeeklyCompensation	StandardHours	HolidayHours	Wage	Pay()
	40	0	20	\$800
	45	0	20	\$950
	48	8	20	\$1360
				<i>expected</i>
				\$1040 <i>actual</i>

What about >40 hours holiday?

Payroll.Fixtures.WeeklyCompensation	StandardHours	HolidayHours	Wage	Pay()
	0	48	20	\$1920
				<i>expected</i>
	0	0	20	0
	40	0	0	0
	40	0	0	0

Source <http://fit.c2.com/>

Fit Workflow: Finished Fit Document

fit: [exec] tools\fit\runFile storytests\compensation.htm build\result.htm
60 right, 0 wrong, 0 ignored, 0 exceptions
all:
BUILD SUCCEEDED

Payroll Fixture	Weekly Compensation	Holiday Hours	Standard Hours	Total Hours	Total Pay
\$20	0	20	20	40	\$1920
\$20	50	50	50	100	\$3100

The system will never allow negative pay. A \$0 wage or zero hours worked is acceptable.

Payroll Fixture	Weekly Compensation	Standard Hours	Holiday Hours	Total Hours	Total Pay
\$0	0	0	0	0	\$0
\$20	0	0	0	0	\$0
\$0	40	0	0	40	\$0
\$20	0	0	0	0	\$0
\$20	40	0	0	40	\$0
-\$20	40	0	0	40	error

Source <http://fit.c2.com/>

Fit Fixtures

- Fit supports multiple types of fixtures
- Standard fixtures
 - ColumnFixture
 - ActionFixture
 - RowFixture
- Many more added by Fit extensions
 - Fit Library
 - Fitness

ColumnFixture

- For specifying business logic
- Maps columns in the table to fields and methods in the fixture

eg.Division		
numerator	denominator	quotient()
1000	10	100.0000
-1000	10	-100.0000
1000	7	142.85715
1000	.00001	100000000
4195835	3145729	1.3338196

Example from <http://fit.c2.com/>

ActionFixture

- For defining workflow
- Interprets a row as a sequence of commands

<u>fit.ActionFixture</u>		
enter	select	1
check	title	Akila
check	artist	Toure Kunda
enter	select	2
check	title	American Tango
check	artist	Weather Report
check	album	Mysterious Traveller
check	year	1974
check	time	3.70
check	track	2 of 7

Example from <http://fit.c2.com/>

RowFixture

- For specifying expected data
- Compares row in the table with objects in the system under test

eg. music.Display					
title	artist	album	year	time()	track()
Handy Man	James Taylor	JT	1977	3.30	7 of 12
Sailing To Philadelphia	James Taylor	October Rose	2001	5.47	3 of 3
Ananas	James Taylor	Hourglass	1997	5.73	5 of 13
Another Grey Morning	James Taylor	JT	1977	2.73	4 of 12
Copperline	James Taylor	New Moon Shine	1991	4.37	1 of 12

Example from <http://fit.c2.com/>

Q&A

