INF 3120

Utvikling av store programsystemer

”Software engineering”

Kursansvarlige:

Bente Anda, Hans Gallis, Magne Jørgensen, Dag Sjøberg,
Gruppe for ”Industriell Systemutvikling”

Plan første forelesning

- Motivasjon
- Kursopplegg
- Introduksjon til ”software engineering”/ industriell systemutvikling
- Kognitive prosesser
- Gruppedynamikk

Motivasjon

- **Det utvikles og vedlikeholdes trolig programvare for mellom 10 og 40 milliarder kroner i året i Norge.**
  - **Ca. 75% av IT-prosjektene har overskridelser (2003)**
  - **I gjennomsnitt er kostnadsoverskridelsene på 30-40%**
  - **En stor andel prosjektene fullføres aldri – særlig er de svært store prosjektene er utsatt for totalstopp eller langvarige forsinkelser.**
    - » **Eksempel på nyere mega-prosjekter med problemer:**
      - **SKARP (Skatte-etaten)**
      - **GOLF (Forsvaret)**
  - **Utviklingsmetode har stor påvirkning på kontrollen. Studier vi har gjennomført viste bla at (2003):**
    - » **Vannfallsmetode: 55% overskridelse**
    - » **Inkrementelle/iterative metoder: 24% overskridelse**

Kursopplegg

- Hovedforelesere
  - Bente Anda, Gruppe for industriell systemutvikling, Ifi, Uio
  - Hans Gallis, Gruppe for industriell systemutvikling, Ifi, Uio
  - Magne Jørgensen, Gruppe for industriell systemutvikling, Ifi, Uio
  - Dag Sjøberg, Gruppe for industriell systemutvikling, Ifi, Uio
- Gjesteforelesere med spesialkompetanse:
  - Ray Welland (kravhåndtering)
  - Jan Øyvind Aagedahl (arkitektur)
  - Kristoffer Kvam, Rodin Lie og Kjetil Jørgensen-Dahl (kvalitetsmålinger vha XRadar)
  - Stein Grimstad og Nils Christian Haugen (Testing)
  - Tom Gilb (målinger, prosessforbedring)
- Foreleserne har lang industrierfaring (de aller fleste), forskningserfaring innen feltet (alle hoveforeleserne og flere av gjesteforeleserne), og har blitt blitt valgt ut etter kriteriene: gode forberedelser, faglig kompetanse og presentasjonsevner.

Kursopplegg

- Pensum
  - Ian Sommervillle: *Software Engineering, 6th edition*, 2000. Addison-Wesley. NB: 7. utgave. Ikke kjøp 6th edition!.
  - Craig Larman: *Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design and the Unified Process, 3rd edition*, 2004. Prentice-Hall.
  - I tillegg er stoff som presenteres på lysark (foiler) og annet som er angitt under "Litteratur" på forelesningsplanen pensum.
  - **Tips:** Fokuser pensumlesingen på de samme deler som forelesernes presentasjoner. (Hand-outs av lysarkene vil stort sett bli delt ut på forelesningene.)
- Obligatorisk prosjektoppgave
- INF3120-festen (med informasjon om ”veien videre” mot mastergrad)
- **Eksamenstips**: Eksamen vil vektlegge stoff og framstillinger fra lysarkene, samt læring fra prosjektoppgaven.

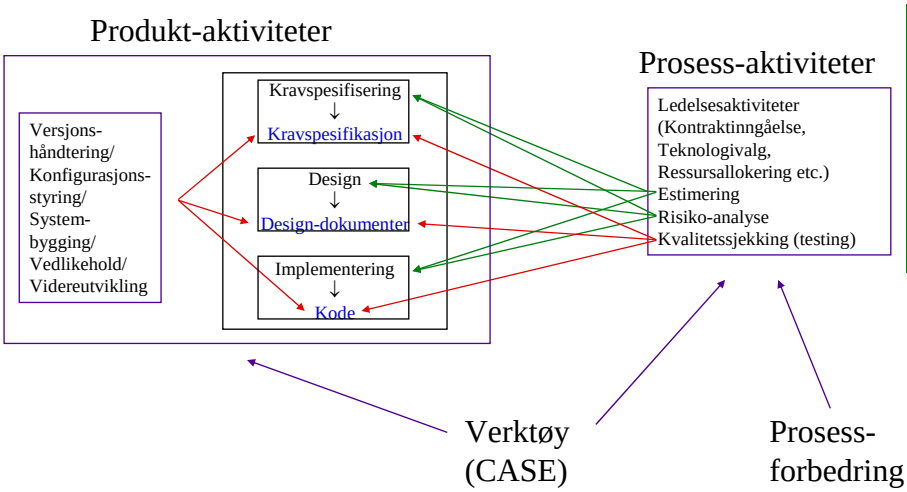
Om forelesningene

- **Pensum er pedagogisk fremstilt i lærebøkene.**
- **Det er lite effektivt å vektlegge gjentagelse av det som står i lærebøkene i forelesningene.**
- **Hvorfor da forelesninger?**
  - **Uten industrierfaring oppfattes mye av pensum enten som ”svada” eller er uforståelig. Gjennom forelesningene vil vi forsøke å sette pensum i bedre kontekst av reelle systemutviklingssituasjoner.**
  - **Forelesninger gir dere utfyllende stoff som gjør det mulig å score ekstrapoeng på eksamen.**
  - **Forelesninger vil gi mer kritisk holdning til pensum og ”guruers” påstander.**
  - **Forelesninger gir mulighet til diskusjoner og spørsmål.**

Lønner det seg å komme på forelesningene?

- **Avhenger av målet ditt og deg som person:**
  - Lære faget "software engineering" vs gjøre det OK på eksamen med lite innsats - ha mye fritid, tid til jobbing
  - Er strukturert som person vs trenger disiplinerende tiltak
  - Indre motivasjon (interesse for faget) vs ytre motivasjon (bli ferdig)
- **Vår erfaring:**
  - Det som sies på forelesningene er viktig for å forstå faget og gjøre det godt på eksamen.
  - Forelesninger kan virke motiverende for å lære mer på egenhånd.
  - Forelesninger kan være disiplinerende.
  - Dersom maksimal effektivitet er målet, interesse for faget er lav, man ikke sikter mot topp karakterer og man er strukturert som person, kan det være effektivt å bli hjemme ;-)

Pensum-oversikt



Temaer

- Utviklingsmodeller
- Prosjektplanlegging og styring
- Estimering
- Kravspesifikasjon
- Design
- Konfigurasjonsstyring og systembygging
- Testing
- Systemutviklingsverktøy
- Gjenbruk
- Arkitektur
- ”Extreme programming”
- Vedlikehold
- Målinger
- Kvalitetssikring og prosessforbedring
- Erfaringer fra industrien

Pedagogisk prinsipp i INF3120

LÆRE **BRUKE**

EVALUERE

Hva er industriell systemutvikling?

- Industriell systemutvikling (software engineering) baserer seg på ingeniørprinsipper ("systematiske metoder") med fokus på:
  - planlegging og forutsigbarhet (vs. "ta den tiden som trengs")
  - strukturering og normering (vs. "gjør som du vil bare produktet blir bra")
  - kvalitetssikring (vs. "produktegenskaper som varierer mye fra gang til gang")
  - rasjonelle valg (vs. "mageføelse")
  - systematisk læring (vs. læring skjer av seg selv)

Hva omhandler industriell systemutvikling?

- Software engineering omhandler teorier, metoder og verktøy for spesifikasjon, design, konstruksjon og vedlikehold av programvare.

På hvilke måter støtter industriell systemutvikling systemutviklingsprosjekter?

- Industriell systemutvikling skal gi støtte til utviklingsprosjekter bla ved å:
  - gi strukturer som "automatiserer" deler av prosessen (f eks oppsplitting i delproblemer)
  - gi strukturer som letter samarbeid (f eks faseinndeling og kodestandarder)
  - gi strukturer som motvirker typiske "svakheter" hos mennesker (f eks at vi har lett for å gyve løs på problemløsningen før problemet er forstått)
  - gi strukturer som forenkler gjenbruk av erfaringer (f eks gjennom at alle bruker samme utviklingsmodell, kodestandard og mal for prosjektplan)

Det utvidede syn på systemutvikling

- Systemutvikling er en endringsprosess
- Endringene omfatter både mennesker og IT
- Må sees som bedriftspolitikk, ikke uvedkommende teknologi
- Systemutviklingen er aldri "verdinøytral"

Systemutviklingens rammebetingelser

- Behov
- Lover og avtaler
- Teknologi
- Økonomi
- Kompetanse
- Etikk
- Kompatibilitet med fortid, nåtid og framtid

Profesjonelt ansvar

- Som systemutviklere skal dere ikke bare fokusere på tekniske spørsmål. Dere har også etisk, sosialt og profesjonelt ansvar
- Ikke alltid enkelt å avgjøre hva som er riktig og galt i mange spørsmål

Etiske forhold

- Hva skal systemene brukes/bidra til? (Overvåkning av de ansatte uten at de vet det)
- Kunnskap om systemet (Programvaren vil slutte å fungere ved årsskiftet, men det vet ikke kunden.)
- Konfidensialitet (Databasen inneholder sensitiv informasjon om naboer/bekjente/venner.)
- Opphavsrettigheter (Bruk av programvare og komponenter utviklet av andre – uten kompensasjon)
- Kompetanse (Deling av kunnskap – kan gjøre egen kompetanse mindre verdifull)
- Misbruk av datamaskiner (Bruk av arbeidsgivers utstyr til privat bruk.)

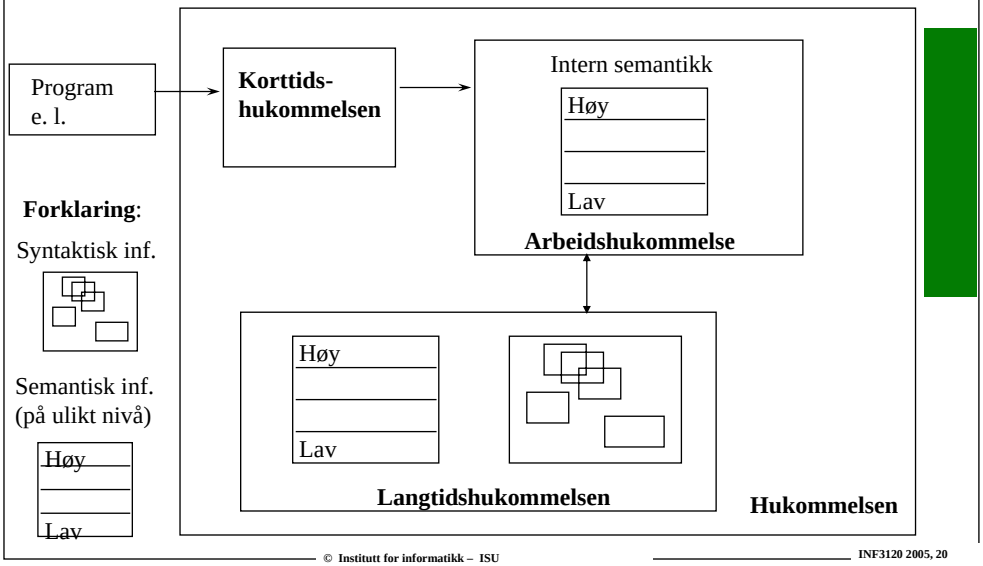
Kognitive prosesser

Litt oppvarming: Hva skiller “ekspert” fra “novise”?

Er en “Stormester” i sjakk bedre enn en amatør mhp:

- **antall mulige trekk de analyserer per minutt?**
- **antall trekk “forover” (dvs dybden i HVIS-SÅ-analysen) vurdert per minutt?**
- **bedre korttidshukommelse?**
- **søkestrategi (“search heuristic”)?**
- **utsiling av dårlige trekk**
- **huske “tilfeldig” oppsatte brikker på et brett**
- **bedre prosesseringsevne (antall bit/sekund i “bevistheten”/”arbeidshukommelsen”)?**
- **se større enheter av gangen, f eks ikke vurdere en og en brikkes posisjon men flere brikkers posisjoner på en gang.**

Modell for programforståelse (Schneiderman)



Konsekvenser...

- Modellen fjerner noe av mystikken rundt eksperters (f eks innen systemutvikling) overlegenhet.
  - Eksperter er ikke nødvendigvis mer intelligente, har bedre hukommelse, eller observerer mer ... de er grunnet lang erfaring bedre i å se (og å gruppere i) større sammenhenger (høyere semantisk nivå på “chunk”-ene).
  - Forståelse kan ses på som en prosess hvor informasjon siles og slås sammen til større håndterbare enheter, som den begrensede arbeidshukommelsen vår kan håndtere.
  - Jo større enheter (chunks) som håndteres jo mer data kan analyseres på likt.

Konsekvenser ...

- Kompleksitet (f eks til programvare) bestemmes ihht teorien av hvor lett høy-nivå semantisk informasjon kan settes sammen fra lav-nivås semantisk informasjon.
  - Eks, modularisering og objektorientert programmerings suksess kan forklares ut fra denne chunking-teorien. Teorien medfører bla at modularisering kun bidrar til forståbarheten dersom den letter sammensettingen av lavere-nivås semantisk informasjon, dvs det er ikke modulariseringen (eller objektifiseringen) i seg selv som hjelper.
- Gir perspektiv mhp (de ofte overflatiske) diskusjonene om kompleksitet, objektorientering vs. funksjonsorientert ....

Flere konsekvenser ...

- Alle systemutviklingsmodeller gir strukturer for rekkefølge på aktiviteter. Modeller er ment å sikre at en hensiktsmessig rekkefølge på systemutviklingen gjennomføres.
- Systemutviklingsprosessen blir mer oversiktlig gjennom oppdeling i delproblemer (faser, aktiviteter, prosesssteg), mao støtte av en nokså universell problemløsningsstrategi. Det kan være problemer dersom oppdelingen ikke er naturlig, f eks dersom det er unaturlig å håndtere "hva" og "hvordan" som to ulike delproblemer.
- Noen systemutviklingsmodeller (inkrementelle) gir strukturer for leveranseoppdeling. Dette er gunstig for problemløsningen dersom delproblemene har svært ulik kompleksitet eller at leveransetidspunktet bør være ulike. Mao, matcher en naturlig problemløsningsstrategi for visse type problemer.
- Noen systemutviklingsmodeller gir strukturer for problemforståelse gjennom utprøving (f eks evolusjonær systemutvikling). Dette er naturlig problemløsningsstrategi for problemer der forståelsen mhp når problemet er løst (behovet er dekket) må antas å være ufullstendig.

Gruppedynamikk

Maslows behovshierarki

Individbehov ihht Maslows behovshierarki:

- Fysiologisk**: Lønn, honorarer, fysisk trygt og tiltalende arbeidsmiljø**
- Trygghet**: Pensjoner og bedriftshelsetjenester, fast ansettelse, karrrieremuligheter innen organisasjonen**
- Sosialt**: Arbeidsoppgaver som tillater kontakt med kolleger, kulturelle/sosiale/sportslige aktiviteter, firmafester**
- Ego**: Utfoldelse/autonomi/ansvar/personlig kontroll, fremming av personlig identitet, respons og anerkjennelse for gode prestasjoner (f eks forfremmelse, bonus, etc.)**
- Selvrealisering**: Oppmuntring til å vise totalt engasjement (se helheten i virksomhetene m.m.), meningsfullhet (meningen med livet ....)**

Gruppens utvikling

Fem typiske stadier**:**

- **oppstart («forming»)**
  - **tilbakeholdenhet, unngå å «miste ansikt», i starten på å bli kjent med hverandre**
  - **fokus på ens egne følelser og behov**
- **konflikt («storming»)**
  - **oppdagelse av ulike forventninger, ulike arbeidsmåter, misforståelser**
  - **fokus på en selv i gruppen**
- **konvergering («norming»)**
  - **gruppefølelse, teamdannelse, «vi-holdning»**
  - **fokus på gruppen (vi-følelsen)**

Gruppens utvikling

- produksjon («performing»)
  - fokus på poduksjon og måloppnåelse(mange grupper når aldri hit)
- oppløsning («adjourning»)
  - kan være full av frustrasjoner (spesielt dersom «høy-kohesive» grupper oppløses)

Kommentarer til «Storming»

- lavt konfliktnivå i gruppen ofte et signal på lite involvering
- konflikter ofte nyttige for å få samhold (dersom konflikter håndteres fornuftig og ikke eskalerer voldsomt)
- «The threshold theory of conflict»

Hva så …. (eksempler)

- Del kunnskap under “forming”. Dette er bla viktig for å skape tillit ved selv å vise tillit.
- Fokuser på struktur under “storming”, ikke så mye på problemløsning.
- Ta hensyn til ”kultur” under normeringsfasen.

Hvorfor påvirker andres tilstedeværelse oss?

1) “Mere presence”: Publikum eller paralelloppgaver gir økt aktivering/stimulering. Dette øker de dominante responsene, på bekostning av de ikke-dominante responsene. (PS: Dette har man funnet gjelder også for dyr (Zajonc, Heingartner & Herman, 1969))
2) “Evaluation apprehension”: Når andre er tilstede blir vi evaluert (eller tror at vi blir det) - dvs vi kan vente oss “belønning” eller “straff” (miste ansikt) alt etter som hvordan vi opptrer. F eks, feil er en kilde til å miste ansikt og vi vil derfor arbeide langsommere enn om vi hadde vært isolert for å unngå å gjøre feil - dersom det er lett å gjøre feil (kompleks oppgave) (Sanders, 1984)
3) “Distraction”: Overfor et publikum blir vi mer “selvbevisste” (fokusert på hvordan vi opptrer etc) og fokuserer på hvordan publikum reagerer. For komplekse oppgaver vil økningen i “drive” pga publikum være mindre enn handicappet ved å bli distrahert mhp oppgavefokusering (Baron 1986).

Noen observasjoner om arbeid i grupper

1) Publikum medfører at dominante responser vil øke, og ikke-dominante responser minke. (Cohen 1980) (Dvs, mindre refleksjoner og mer instinkt!!)

2) Arbeid sammen med andre gir distraksjoner, f eks ved at viktige sider ved oppgaven blir glemt (sammenlignet med isolert oppgaveløsning) - (Sanders, Baron & Moore, 1978). Men, ved svært enkle oppgaver øker ytelsen under distraksjoner. (Baron 1986)

3) Publikum med svært høye forventninger kan (selv ved enkle oppgaver) være et hinder mer enn en hjelp!! (Baumeister, Hamilton & Tice, 1985)

4) Zajonc (1965) gir følgende råd: Vanskelig stoff læres best alene (og ikke i grupper), øvelse i presentasjon og repetisjon er derimot godt egnet til grupper.

Andre observasjoner om arbeid i grupper

5) Kvinner gjør det gjennomgående best på oppgaver der et høyt soialt aktivitetsnivå var nødvendig, og menn på oppgaver der fokus på oppgaveløsningen er nødvendig (Wood 1987).

6) Gruppe-diskusjoner som foregår elektronisk medfører en “likestilling” i forhold til “in personae”-møter, hvor høy-status deltakere får mer tid/oppmerksomhet og bruker mer av tale-tiden.

Virkemidler for å unngå ytelsesreduksjoner i grupper

1. Identifiserbarhet og evaluering av den enkeltes bidrag
2. Gi de enkelte interesante, engasjerende og utfordrende oppgaver
3. Skap en tillitt til at andre gir sin maks-ytelse
4. Klargjør personlig ansvar (og myndighet) - om mulig relatert til påvirkning av sluttproduktet
5. Vurder om type oppgave som skal løses passer for arbeid i gruppe (eller om den f eks bør tildeles en person alene). F eks kan statusmøter, der det er lite interaksjon mellom deltakerne på statusmøte, ofte gjøres mer effektivt enn at alle sitter og hører på alles problemer.

Grupper som avgjørelsestakere

1. Gode indikasjoner på grupper som bruker tid på diskusjon mhp avgjørelsesprosessen gjør bedre avgjørelser. (De fleste grupper (møter) bruker svært liten eller ingen tid på å avklare/diskutere avgjørelsesprosessen.)
3. Stort sett skjer en polarisering av meninger i grupper!! (Myers & Lamm 1976) (og da oftest mot det risikobetonte, pga at grupper pulveriserer ansvar, høy-risko mennesker har større inflytelse på gruppas avgjørelse, høy-risiko blir ufarliggjort ved å diskutere det og (kanskje mest) risiko-taking er forbundet med positive verdier)
4. Grupper kan føre til “gruppetenkning”, dvs en mangel på reell diskusjon av alternativer og en sterk motivasjon mhp å være enige og lojale.

Obligatorisk oppgave

- Generelt om prosjektarbeidet:
  - 4-5 personer på hver gruppe (inndeling basert på svar gitt i spørreskjema)
  - Gruppelærerne er kunde (oppdragsgiver) – prosjektgruppene er leverandører
  - Fire leveranser (godkjent/ikke godkjent)
    - » Forholdsvis store leveranser
      - Må planlegges godt!
      - Fordeling av arbeidet er viktig
    - » Hvorfor ikke karakterer?
  - 25% av oppgavene på eksamen vil være relatert til erfaringer i det obligatoriske prosjektarbeidet
  - Systemet man skal jobbe med er det samme som i 2003 og 2004 (Project Hospital), men funksjonaliteten man skal utvikle/endre er ikke den samme (derfor ingen vits i å kopiere tidligere prosjektgruppers oblig!).
  - Det skal være lærerikt, utfordrende, krevende og gøy ☺
    - » NB! 10 studiepoeng ≈ 12,5 timer arbeid pr person pr uke!

Obligatorisk oppgave

- Læringsmål
  - Ferdigheter og evalueringskompetanse innen:
    1. Prosjektplanlegging og –gjennomføring
    2. Beskrivelse av krav v.h.a. UML Use Cases
    3. Utarbeidelse og gjennomgang av objektorientert design
    4. Generelle prinsipper for ”Software Engineering” (se temaer på foil 9)
    5. Bruk av et CASE verktøy (Computer Aided Software Engineering): Telelogic Tau UML
    6. Kodegenerering og ”reverse engineering”

Obligatorisk oppgave

- Retningslinjene og innleveringsfristene legges ut på kursets hjemmeside.
- Gruppelærerne har fått instrukser om at innleveringer skal oppfylle **"SKAL-kravene"** i retningslinjene for å bli godkjent. **Les retningslinjene nøye!**
- Gruppelærerne vil gi mer detaljert informasjon angående prosjektarbeidet.
- ***Tips:*** *Sett dere inn i prosjektoppgaven og skaff dere oversikt over hva som forventes og skal gjøres selv om ikke prosjektgruppene er klare.*

Tilbakemeldinger fra tidligere års studenter

- "Interessant. Knyttet mot reelle jobbsituasjoner"
- "Kjedelig. Alt for mye fokus på prosjektet"
- "… mye luft kanskje, og som det ble sagt fikk man sikkert mer ut av kurset om man hadde litt arbeidserfaring"
- "Virker kanskje litt unødvendig å lære å bli prosjektleder før man har deltatt noe særlig i prosjekt selv, men har jo lært litt i hvertfall da. …"
- "Altfor mye pensum og tørr svada"
- "Det er et veldig nyttig kurs, hvor man får et godt perspektiv på systemutvikling."
- "Kjempekurs :) Det er gjort en kjempejobb med spennende gjesteforelesere. Flinke og kunnskapsrike forelesere"