# Forelesning 1: Innledning

## Læringsmål

Vårt mål med dette kurset, er at du skal forstå i store trekk hva det innebærer å utvikle et software-system. Når vi snakker om å utvikle et software-system, mener vi altså å lage et (muligens komplekst) system bygget opp av ulike software-moduler, software-komponenter og subsystemer. Her vil vi bruke ”software” og ”programvare” om hverandre, og vi vil også bruke ”systemutvikling”, ”softwareutvikling” og ”programvareutvikling” om hverandre, selv om det er mulig å argumentere for at det er forskjeller i disse begrepene på et mer detaljert nivå. Frasene ”software-system”, ”data-system” og ”IT-system” vil også bli brukt om hverandre.

Software-systemer finnes i så nær som all moderne teknologi: De styrer prosessene i bilmotorer, de utgjør styringsmekanismene i fly og kontrollsystemene i kjernekraftverk. Din mobiltelefon, iPod, TV, og vaskemaskin er fullpakket med innbakte software-systemer. Software-systemer bistår også i så nær som all saksbehandling, økonomi- og strategistyring i bedrifter, og de gjør at du i dag kan gjøre svært mye av dine ærender over internett .

Det å utvikle et software-system involverer flere aspekter. Sammenlikn gjerne med byggingen av andre komplekse ting: En oljeboringsplatform, operabygget i Bjørvika, utviklingen av en ny bilmodel. Det software-utvikling har til felles med bygging av andre komplekse ting, er at det kreves gode organisatoriske rutiner. Det kreves også økonomistyring, personalhåndtering og samarbeid. I tillegg kreves det sterk *domene*-faglig kunnskap: For bygging av en oljeboringsplatform kreves bl.a. bygg-ingeniør-kunnskap, og for software-utvikling kreves kunnskap om hvordan man strukturerer og lager programvare. *Bygging av komplekse ting er altså et tverrfaglig foretagende*. Utvikling av software-systemer er intet unntak! I tillegg, har software-utvikling helt spesielle teknologiske utfordringer, siden software fort blir meget uoversiktlig og er såpass lite konkret.

For å delta i utviklingen av et software-system, bør du derfor ha en oversiktskunnskap om alle de nødvendige aspektene. *Det er ikke nok å være et råskinn til å programmere*! Det er et uttalt ønske fra norsk IT-industri at opplæringen i så måte må bli bedre. Flere ledere i industrien erfarer jobbsøkere med svært manglende kunnskaper om software-utvikling, selv om de kan programmere.

Dette er den viktigste grunnen til at vi har med forelesere fra industrien og til at vi dekker såpass mange temaer i systemutviklingsspektret som vi gjør i dette kurset. Pensum og temaer i dette kurset er valgt i samråd med våre kontakter i norsk IT-industri og er ment, blant annet, å gi dere et målrettet grunnlag for å jobbe i IT-industrien, og som utgangspunkt for videre faglig spesialisering innen et gitt tema. Temaene er således høyrelevante.

## Strukturen på kurset

De tre første forelesningene (inkludert denne) tar for seg systemutvikling som helhet, før de senere forelesningene tar for seg mer spesialiserte temaer. Forelesning 1 (denne) motiverer for faget som


Copyright © 2009 Jo Hannay, Institutt for informatikk, Universitetet i Oslo

sådan, mens forelesning 2 og 3 introduserer noen teknikker for å styre systemutvikling fra begynnelse til slutt.

Deretter tar forelesning 4-7 for seg den delen av systemutvikling som kanskje er den vanskeligste og mest feilbarlige. Den involverer nemlig interaksjonen og kommunikasjonen mellom tre hovedaktører (stakeholders) i systemutvikling:

- Kunde --- som ønsker å anskaffe/utvikle et software-system for et gitt formål.
- Leverandør --- som utvikler systemet for kunden.
- Bruker---som skal bruke systemet når det har kommet i drift.

Forelesning 8-11 beveger seg over i de teknologiske særtrekkene for systemutvikling, og tar for seg måter å modellere, eller skjematisere mer presist, hva systemet skal gjøre og hvordan det skal gjøre det, samt hvilken struktur systemet skal ha. Dere har jo lært at objektorientering er en bra måte å dele inn programmene deres på. For store systemer er en inndeling i objekter ikke nok! Man må ha strukturer på en større skala. Det er dette vi kaller arkitektur.

Forelesning 12-14 forstetter med de teknologiske særtrekkene av systemutvikling, nemlig ting som koding (programmering) og testing (debugging) av kode. Imidlertid skal vi ikke skrive noe særlig kode, men heller bruke et kodegenereringsverktøy.

Planen ser ut som følger:

- **Systemutvikling som helhet**
    1. **Systemutvikling: motivasjon** ..... .....Jo Hannay, Simula & Ifi
    2. **Systemutviklingsprosessen** ..... Rune Steinberg, Visma Software AS
    3. **Prosjektledelse og prosjektarbeid** ... .....Rune Steinberg, Visma Software AS
- **Leverandør/kunde/bruker-forhold**
    4. **Kravhåndtering** .....Erik Arisholm, Simula & Ifi
    5. **Avtaler & kontrakter** ..... Jørgen Petersen, Promis AS
    6. **Estimering** ..... Stein Grimstad, Simula
    7. **Jus & etikk** ..... Dag W. Schartum, Senter for Rettsinformatikk
- **Systemets struktur og design**
    8. **Modellering av krav med use cases** .....Erik Arisholm, Simula & Ifi
    9. **Objektorientert analyse** (2 forel.) ..... Erik Arisholm, Simula & Ifi
    10. **Persistens og databaser** .....Erik Arisholm, Simula & Ifi
    11. **Arkitektur** ..... Dag Lorås, Visma Software AS
- **Koding, validering og vedlikehold**
    12. **Modellbasert utvikling med Genova** ... .....Esito AS
    13. **Validering og verifisering** (2 forel.) ..... Lionel Briand, Simula & Ifi
    14. **Konfigurasjonsstyring** ..... Hans Christian Benestad, Simula
- **Avslutning**
    15. **Oppsummering & eksamenstips** .....Erik Arisholm
    16. **Faglig sosial ettermiddag** ..... Foreleserne og dere!

Merk at forelesning 7 blir flyttet til slutt, og blir dermed egentlig forelesning 14. Se detaljert undervisingsplan: **uio.no/studier/emner/matnat/ifi/INF1050/v09/undervisningsplan.xml**


Copyright © 2009 Jo Hannay, Institutt for informatikk, Universitetet i Oslo

## Systemutvikling og software engineering

Systemutvikling involverer ikke bare en rekke forskjellige fagfelt. Systemutvikling er i tillegg svært komplekst, og dette gjør at det er ikke-trivielt å komme i mål på en god måte! Rett som det er, går store IT-utviklingsprosjekter langt over budsjett og tid, eller skrinlegges helt pga. manglende evne til sluttgjennomføring. Altfor ofte har ikke systemene den nødvendige kvaliteten heller. Dette er i beste fall irriterende, og i verste fall katastrofalt. Uansett, er ikke tingenes tilstand akseptabel!

Heldigvis er det mulig å forbedre tingenes tilstand! Det finnes teorier og utprøvde metodikker for hvordan man kan håndtere noe så kompleks som systemutvikling. I dette kurset skal vi gjennomgå hovedpunkter i fagfeltet *software engineering*.

Software engineering (industriell systemutvikling) omhandler teorier, metoder og verktøy for spesifikasjon, design, konstruksjon og vedlikehold av programvare. Software engineering tar i betraktning de menneskelige aspektene i samspill med de teknologiske aspektene!

Målet er å bidra til at vi lager bedre software-systemer, raskere, med færre ressurser og på en mer forutsigbar måte. Software engineering baserer seg på ingeniørprinsipper (systematiske metoder) med fokus på følgende fem hovedprinsipper:

- **Planlegging og forutsigbarhet**
- **Oppdeling og strukturering av problemer i mindre komplekse bestanddeler**
- **Modularitet og gjenbruk**
- **Abstraksjon og modellering**
- **Systematisk kvalitetssikring**

Disse hovedprinsippene er alle ment å gjøre den komplekse situasjonen mer håndterbar. *Planlegging og forutsigbarhet* er essensen for å få overordnet kontroll på et stort prosjekt. Forutsigbarhet er svært viktig, spesielt for kunden, og gjør at omverdenen i det hele tatt kan forholde seg til et systemutviklingsprosjekt. Planlegging og forutsigbarhet innebærer:

- Veldefinerte, repeterbare og planlagte aktiviteter.
  - Alle personer vet hva de skal gjøre, hvordan det gjøres, hva de skal levere og når det skal leveres. Dette fordrer at alle involverte har en felles forståelse, og standarder/metoder/verktøy er essensielle her.
- Prosjektplaner og -rapportering.
  - Ressursplaner: Kostnadsrammer, personal, utstyr
  - Tidsplaner: Estimering, milepæler, aktivitetsnettverk
- Kvalitetsplaner og -rapportering.
  - Sjekklister, inspeksjoner, testplaner, testresultater …
  - Rutiner for å håndtere endringsforespørsler, sporbarhet, …

Merk at graden av planlegging og formalitet i systemutvikling er et kontroversielt tema. Lettvektsprosesser (f.eks. eXtreme Programming, XP) forfekter en desentralisert ledelses- og


Copyright © 2009 Jo Hannay, Institutt for informatikk, Universitetet i Oslo

kontrollstil, i forhold til mer tradisjonelle sekvensielle prosesser (f.eks. Fossefall) der ledelse og kontroll er svært sentralisert.

Prinsippet om *Oppdeling og strukturering av problemer i mindre komplekse bestanddeler* er hovedstrategien for å få bedre oversikt og øke håndterbarheten. (Splitt og hersk.) Det er flere måter man kan gjøre fornuftige oppdelinger på, og man vil som regel benytte alle måtene samtidig! Man kan dele opp i for eksempel:

- Tid (faser):
  - PS2000-kontraktstandarden: *Behovsfase, Løsningsbeskrivelse, Iterativ konstruksjonsfase, Godkjenningsfase.*
  - Timeboxing/tidsavgrensning.
- Oppgaver (aktiviteter og tilhørende leveranser):
  - *Analyse, design, programmering, testing, …*
- Tekniske aspekter:
  - *Kvalitetsaspekter, Funksjonalitet, Moduler, Komponenter*.
- Modeller på forskjellige abstraksjonsnivåer:
  - *Kravspesifikasjoner* versus *Objektorientert analyse* versus *Detaljert design* versus *Kode* .

De to siste måtene her å dele opp på (”Tekniske aspekter” og ”Modeller på forskjellige abstraksjonsnivåer”), gir opphav til de to neste av våre fem prinsipper. Oppdeling i ”Tekniske aspekter” gir opphav til prinsippet om *Modularitet og gjenbruk* som gir en oppdeling ifølge logikken i systemet. Hensikten er en oppdeling som gjør det lettere å få et konseptuelt grep om systemet, samt tidsbesparing gjennom gjenbruk av deler av systemet som kan brukes flere steder i systemet. Modularitet og gjenbruk innebærer:

- Datasystemer deles opp i mindre delsystemer (komponenter, moduler, aspekter) slik at:
  - Hvert delsystem implementerer et veldefinert problem (høy kohesjon) og
  - Man forsøker å redusere avhengigheter på tvers av delsystemer (lav kobling).
  - For “nyvinninger” innen dette, se aspekt-orientert utvikling. Eksempler på aspekter: *sikkerhet, kontroll og sporbarhet* (se SKARP-prosjektet).
- Modularisering:
  - Muliggjør gjenbruk innen et prosjekt eller på tvers av prosjekter.
  - Letter arbeidsfordeling og samarbeid.
  - Muliggjør inkrementell utvikling.

Oppdeling i ”Modeller på forskjellige abstraksjonsnivåer” gir opphav til prinsippet om *Abstraksjon og modellering*, som er nok en kraftig teknikk for å lette belastningen på vår stakkars lille hjerne. Virkeligheten er svært kompleks, og uten evnen til å lage modeller som forenkler og abstraherer bort detaljer som er irrelevante for å løse et gitt problem, ville ikke menneskeheten vært der den er i dag kunnskapsmessig (på godt og ondt). *En modell av et system er en forenklet beskrivelse av systemet som er hensiktsmessig til et gitt bruksområde.* Diagrammene av T-banens rutenett er modeller av T-banenettet, som er hensiktsmessig forenklet og utelater detaljer slik at det er lett å finne fram i T-banenettet. Når vi skal utvikle et kompleks software-system i et kompleks systemutviklingsprosjekt, er vi helt avhengige av å kunne lage slike hensiktsmessige forenklete modeller. Så, for Abstraksjon og modellering har vi:


Copyright © 2009 Jo Hannay, Institutt for informatikk, Universitetet i Oslo

- Identifiser de viktigste momentene og ignorer detaljer som er irrelevante for å løse et gitt problem
- Modeller er en type abstraksjon: spesifikasjoner og designmodeller skjuler irrelevante programmeringsdetaljer
  - Selve programmet kan også ses på som en presis modell av hvilke oppgaver som skal gjøres og hvordan, som deretter oversettes til maskinkode slik at datamaskinen kan utføre dem. Denne modellen er også hensiktsmessig til sitt bruk, men ikke til f.eks. å diskutere med kunden!
- I Inf1050 vil dere lære hvordan Unified Modeling Language (UML) kan brukes til å
  - spesifisere kravene til et system og gjøre en analyse av hvordan disse kravene kan realiseres i et objektorientert programmeringsspråk
  - definere en database for lagring av dataene
  - generere prototyper av blant annet brukergrensesnittet (vha Genova) for å få tidlige tilbakemeldinger fra potensielle brukere

Det siste prinsippet er *Systematisk kvalitetssikring*. Dette er ekstremt viktig og bør gjennomsyre et systemutviklingsprosjekt helt fra begynnelsen av. Nyere systemutviklingsmetoder forfekter kortere delmål og aktiv kundedeltakelse gjennom hele prosjektet (ikke bare i starten og på slutten). Systematisk kvalitetssikring innebærer:

- Validering og verifisering
  - Validering: Har vi spesifisert systemet riktig?
  - Verifisering: Lager vi det spesifiserte systemet riktig?
- Endringshåndtering og konfigurasjonsstyring
- Kundeinvolvering
- Inkrementell og iterativ utvikling
  - Reduserer risiko ved at man kan levere og evaluere (validere og verifisere) enkelte delsystemer

Dette var et meget kort overblikk over hovedprinsippene i software engineering. Hvordan disse prinsippene kommer til uttrykk og blir anvendt, vil dere få se flere ganger i løpet av kurset og de kommende forelesningene.

Faget software engineering er som sagt en samling av metoder, teorier osv. Mange av metodene og teoriene er framsatt av ”guruer”, andre metoder og teorier er utviklet gjennom forskning. Noen metoder har vi et grunnlag for å si at virker, men mye er også foreløpig spekulasjon og ”hype”. Dette kurset vil så langt mulig, ta utgangspunkt i metoder som er *empirisk utprøvd* (evidensbasert software engineering), eller som det finnes utprøvde teoretiske begrunnelser for. Samtidig vil vi å orientere dere om gjeldende trender (mulig ”hype”, som også kanskje virker), slik at dere også vet hva disse er.

God reise!


Copyright © 2009 Jo Hannay, Institutt for informatikk, Universitetet i Oslo