HiST/AITeL - RAPPORT 2006: 2
Avdeling for informatikk og e-læring
Høgskolen i Sør-Trondheim

HØGSKOLEN
I SØR-TRØNDELAG

UTVIKLINGSMODELLER

ISBN 82-7877-137-5
ISSN 1504-5587

HiST/AITeL - RAPPORT 2

Trondheim 2006

av
Tore Berg Hansen og Greta Hjertø

tore.b.hansen@hist.no
greta.hjerto@hist.no

Høgskolen i Sør-Trøndelag
Avdeling for informatikk og e-læring
2006

Innhold

Innledning

I denne rapporten ser vi på det som karakteriserer en del kjente, og kanskje mindre kjente, utviklingsmodeller. Bakgrunnen for vurderingen av disse modellene var et IP-prosjekt som ble gjennomført ved Hogeschool van Amsterdam i perioden 4. – 15. april 2005. Det er laget en egen rapport som tar for seg mål, opplegg samt en vurdering av resultatene fra prosjektet. (Berg Hansen og Hjertø 2005).

Hovedmålet for IP-prosjektet kan sammenfattes i dette spørsmålet:

Hvilke metoder og teknikker i systemutviklingsprosessen er best egnet til å sikre kvalitet i sluttproduktet?

Det er altså i denne sammenhengen vi har sett litt nærmere på de utviklingsmodeller som ble evaluert i prosjektet. Vi har funnet det hensiktsmessig å oppsummere det vi fant i en egen rapport, fordi resultatene kan ha generell interesse ut over IP-prosjektet.

Modellene

I rapporten tar vi for oss disse modellene:

- Métrica 3 som har sin opprinnelse og største utbredelse i Spania.
- Integrated Development (ID) som er en svensk modell som på svensk heter Integrerad Utvecklingsmetod (IU).
- Extreme programming (XP) som er en av de mest omdiskuterte smidige modeller.
- Unified Process (UP) som også er kjent som Rational Unified Process (RUP) fordi modellen markedsføres som et kommersielt rammeverk av firmaet Rational (nå overtatt av IBM).
- Dynamic Systems Development Method (DSDM) som har sin opprinnelse og største utbredelse i Storbritannia, men også mye brukt i Nederland.
- Prince2 som er en generell prosjektstyringsmodell opprinnelig laget for planlegging og styring av prosjekter for utvikling av informasjonssystemer.

DSDM hadde de kombinert med Prince II som er en prosjektstyringsmodell, og var således den eneste gruppen som hadde benyttet en spesifikk prosjektstyringsmodell.

Systemutviklingsmodellene

Først en liten begrepsavklaring – generelt bruker vi begrepet modell om de utviklingsprosesser som er brukt i dette prosjektet. Men fordi noen modeller i egen dokumentasjon blir beskrevet som en metode, kan vi komme til å bruke begrepene *metode* og *modell* om hverandre i omtalen av en bestemt modell. En modell beskrives også som en metodikk. Så det er mange begreper som brukes.

Men først skal vi se hvordan prosesser generelt kan klassifiseres.

Klassifisering av prosessmodeller

Det er flere måter å klassifisere prosessmodeller på. Grovt kan man skille mellom lineære modeller og ikke-lineære modeller. Man kan på en måte si at det er én lineær modell, fossefallsmodellen, og så alle modeller som senere er lansert som svar på svakhetene ved fossefallsmodellen. Dette kommer til uttrykk i *Technical Report ISO/IEC TR 15271 Information technology – Guide for ISO/IEC 12207* hvor man diskuterer begrepene inkrementell livsløpsmodell og evolusjonær livsløpsmodell. I en inkrementell livsløpsmodell starter man med å bestemme alle kravene og arkitekturen til systemet. Deretter bygges systemet i inkrementer. Et inkrement består av et bygg. Første inkrement realiserer en del av kravene. Hvert etterfølgende inkrement realiserer et ytterligere sett krav.

I en evolusjonær livsløpsmodell utvikles også systemet i inkrementer. Mer funksjonalitet legges til i hvert inkrement. Forskjellen på denne modellen og den inkrementelle er at man ikke finner alle krav på forhånd. Man innser at dette ikke alltid er mulig. Derfor starter man med å definere noen krav og fortsetter med et bygg. Dette gjentas inntil man har et komplett system.

Prosessmodeller kan videre klassifiseres som:

- *Tungvektsprosess* – som har mange artefakter og det kreves mye formalisme og byråkrati. Det er lite rom for fleksibilitet og mye kontroll. Mye detaljplanlegging gjøres i starten av prosjektet. En slik prosess er predikativ. Se nedenfor.
- *Lettvektsprosess* – hvor det er viktig å få frem kjørbare (del)systemer tidlig. Man nedtoner betydningen av formelle dokumenter. Det utarbeides ikke detaljplaner for mer enn neste fase eller iterasjon. Kunder/brukere deltar aktivt i utviklingsgruppen som er relativt liten (ikke mer enn ti personer).
- *Predikativ prosess* – hvor man prøver å forutsi og planlegge aktiviteter og ressurser i detalj tidlig i prosjektet. Tungvektsprosesser er gjerne predikative. Fossefallsmodellen er et eksempel på en predikativ tungvektsprosess.
- *Adaptiv prosess* – er karakterisert ved at man aksepterer at endringer er det normale. Endringer driver på en måte prosessen. Adaptive prosesser er lettvekts og evolusjonære.
- *Fortung prosess* – tids- og ressursbruken er relativ stor i de tidlige fasene frem til koding. Mindre tid og ressurser brukes på koding og testing.
- *Baktung prosess* – tids- og ressursbruken er liten i tidlige faser, men større i de senere faser, det vil si det motsatte av en fortung prosess.
- *Balansert prosess* – har en mer jevn fordeling av tids- og ressursbruken mellom fasene i prosessen.

- *Smidig prosess* – kan stå som et samlebegrep for adaptive lettvekstprosesser. I engelsk terminologi brukes begrepet agile process. Dette begrepet er lansert av The Agile Alliance og konkretisert i The Agile Manifesto.

The Agile Alliance og deres manifest

Bakgrunnen for dannelsen av denne alliansen er de problemer og utfordringer som utvikling av programvaresystemer har. De er en slags reaksjon på at løsningene synes å være mer formalisme og byråkratiske prosesser (lineær, fortung og predikativ). Alliansen ble dannet i februar 2001 av 17 personer med foreskjellige bakgrunner fra utviklingsarbeid. Men de klarte å bli enige om et manifest som de mener skal stimulere til bedre måter å utvikle programvare på. Manifestet ble detaljert i en samling prinsipper.

Manifestet

Enkeltmennesker og samspill	*fremfor prosesser og verktøy*
Fungerende programvare	*fremfor omfattende dokumentasjon*
Kundesamarbeid	*fremfor kontraktsforhandlinger*
Reaksjon på endring	*fremfor å følge en plan*

Manifestet er satt opp slik at det viktigste er det som står til venstre. Men det betyr ikke at man dermed ikke skal legge vekt på det som står til høyre. Altså, man viser preferanser, ikke alternativer.

Prinsippene

For å bedre forståelsen av hva smidig utvikling betyr er filosofien fra manifestet utdypet i disse prinsippene:

1. *Kundetilfredsstillelse gjennom tidlig og kontinuerlig levering av programvare som gir verdi, har førsteprioritet.*
2. *Vi sier velkommen til endrede krav selv sent i utviklingen. Smidige prosesser utnytter endringer slik at kunden oppnår konkurransemessige fordeler.*
3. *Levér fungerende programvare hyppig, fra et par uker til et par måneder, med preferanse for de korte tidsskalaer.*
4. *Forretningsfolkene og utviklerne må arbeide sammen daglig gjennom prosjektet.*
5. *Etabler prosjektet rundt motiverte enkeltmennesker. Gi dem de omgivelser og støtte de trenger og ha tillit til at de får gjort jobben.*
6. *Den mest effektive måten å utveksle informasjon på i et arbeidslag er samtaler ansikt til ansikt.*
7. *Fungerende programvare er den primære indikasjon på fremgang.*
8. *Smidige prosesser fremmer vedvarende utvikling.*
9. *Sponsorer, utviklere og brukere skal bli i stand til å holde en konstant fart i det uendelige.*
10. *Vedvarende oppmerksomhet på teknisk fortreffelighet og god design fremmer smidighet.*
11. *Enkelhet – kunsten å maksimere arbeid som ikke blir gjort – er essensielt.*
12. *Den beste arkitektur, krav og design kommer fra selvstyrte arbeidslag.*
13. *Med jevne mellomrom reflekterer arbeidslaget på om man kan bli mer effektiv og gjør nødvendige endringer deretter.*

DSDM

Bakgrunn

DSDM står for Dynamic Systems Development Method. Det er altså en utviklingsmetode, men begrepet assosieres også med et konsortium, DSDM-Consortium. Konsortiet ble etablert i 1996 i England. Stifterne kom fra både store og små bedrifter i IT-industrien. Senere har konsortiet fått medlemmer i flere land både i Europa og i Nord-Amerika, men er ennå ikke etablert i Norge.

Motivasjonen for etableringen var de problemer vi tidligere har omtalt som typiske for programvareindustrien med forsinkede prosjekter og overskridelser av budsjetter. Konsortiet skal ikke tjene penger, men sørge for å dyktiggjøre medlemmene. Inntektene kommer fra medlemskontingenter. Det konkrete resultatet fra arbeidet i konsortiet er DSDM som utviklingsmodell eller det er kanskje riktigere å si at det er et rammeverk for en lettvekts og smidig (agil) utvikling av programvaresystemer. Man kan godt si at konsortiet har som mål å bidra til å fjerne oppfatningen om at smidig utvikling er noe som bare hackere holder på med. Samtidig mente man at det var helt nødvendig med alternativer til fossefallsmodellen med de åpenbare svakheter den har.

Prinsippene

DSDM-rammeverket beskriver en prosessmodell basert på prinsippet om iterativ og inkrementell utvikling. Det inneholder også en del artefakter som skal lages i løpet av en utviklingsprosess. Men bortsett fra disse overordnede prinsipper gir ikke modellen detaljerte anvisninger for hvordan utviklingsarbeid skal drives. Det vil si at rammeverket skal tilpasses det aktuelle utviklingsprosjektet og at dette kan gjennomføres etter mange forskjellige paradigmer. Man kan altså følge et tradisjonelt funksjonsorientert paradigme så vel som objektorientering og sågar extreme programming. Det siste er naturlig ettersom både XP og DSDM er påvirket av den smidige (agile) bevegelsen. En introduksjon til DSDM finner vi i (Stapleton 2002).

Rammeverket er basert på disse ni prinsippene:

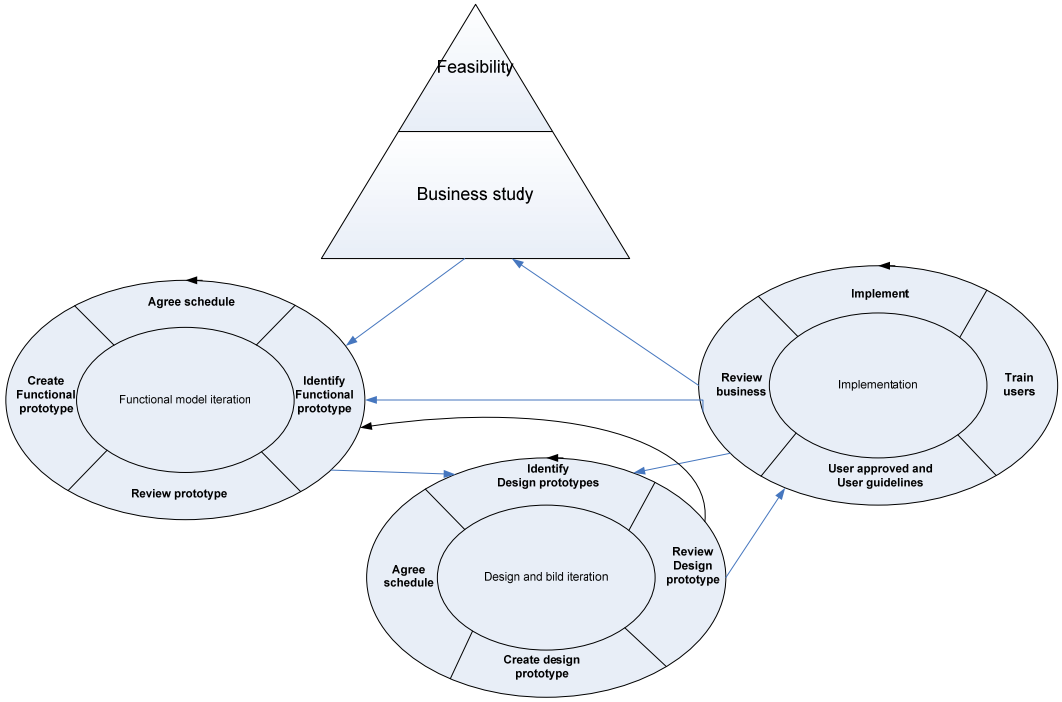
1. Aktiv brukermedvirkning er uomtvistelig.
2. Grupper som praktiserer DSDM må ha stor grad av selvstyre. Det vil si at gruppen må ha fullmakt til å fatte vidtrekkende beslutninger.
3. Fokus er på hyppig levering av produkter.
4. For at det som leveres skal aksepteres må det vise sin nytte for virksomheten som skal ha programvaren.
5. Iterativ og inkrementell utvikling er nødvendig for at man skal oppnå en riktig løsning.
6. Alle endringer som gjøres er reversible.
7. Krav fastsettes (is base lined) på et høyt nivå.
8. Testing er integrert gjennom hele livsløpet.
9. Det er essensielt at det er et tett samarbeid og samspill mellom alle interesseparter.

Det er konsortiets oppfatning at alle disse prinsippene må følges hvis resultatet skal bli et kvalitetssystem.

Modellen

DSDM-rammeverket deler et utviklingsprosjekt inn i syv faser. Eller hvis man skal være mer presis så er det to faser som ikke er knyttet direkte til utvikling og fem utviklingsfaser. Det er en f*orprosjektfase* (pre-project phase) hvor man sikrer at prosjektet får en sunn forankring og

en *etterprosjektfase* (post-project phase) hvor man summerer opp erfaringer og sikrer at den leverte løsning fortsatt er operativ. Figur 1 viser utviklingsfasene i DSDM.



Figur 1 Utviklingsfasene i DSDM

Modellen kalles populært for tre pizzaer og en ost. Osten viser to lineære faser. De iterative og inkrementelle faser utgjøres av de tre pizzaer. Den første delen av osten er forstudien hvor man gjør de tradisjonelle tingene for å få en overordnet vurdering av ønsket funksjonalitet og løsningsmuligheter samt grove kostnads- og ressursestimater. I business study analyseres virksomheten og grunnlaget for videreføring av prosjektet legges. Så følger tre iterative faser som man kan gå frem og tilbake mellom. I *Functional model* iterasjonen detaljeres det arbeidet som ble startet i business modelling. Man starter arbeidet med en evolusjonær prototyp. Høynivåarkitekturen fastsettes. Man itererer gjennom denne fasen inntil man har skaffet nok informasjon om funksjonaliteten som ønskes slik at man kan gå over i design and build, hvor systemet videreutvikles slik at det kan overføres til virksomheten i implementasjonsfasen. DSDM tillater, i god smidig stil, endringer hvis ny kunnskap tilsier det. Derfor piler frem og tilbake mellom pizzaene.

Et viktig element i DSDM er MoSCoW-reglene. MoSCoW er et akronym for prioritering av krav. Som konsortiet selv skriver er de to o-ene satt inn for moro skyld. De andre bokstavene står for:

- *Must have* – de krav som er helt nødvendige for at systemet skal kunne brukes i det hele tatt.
- *Should have* – er krav som ville blitt satt som nødvendige hvis tidsrammene ikke er for stramme, men som man i første omgang kan klare seg uten.
- *Could have* – er krav man kan klare seg uten i det inkrementet man er inne i.
- *Want to have but will not have this time round* – er krav som kan vente til eventuell videreutvikling av systemet.

Det viktige med MoSCoW-reglene er de danner basisen for beslutninger som skal gjøres i en bestemt timebox. Og timeboxing er et viktig redskap for alltid å levere i tide og innenfor budsjett. Det betyr at i en timebox er man ikke opptatt av aktiviteter, men at noe skal leveres etter hver timebox. I en timebox, som kan være mellom to og seks uker lang, skal man ha både Must have og Should have krav. Slik kan man ha krav som kan kastes ut hvis ting av uforutsette årsaker ikke skulle gå som opprinnelig planlagt i en timebox.

Av andre ting som vektlegges i DSDM er at lange arbeidsdager ikke skal være normen. Målet er å arbeide normale arbeidsdager og bruke kvelder og helger til rekreasjon og avkobling.

Unified process (UP)

Bakgrunn

Unified Process (UP) er kanskje like godt kjent under begrepet Rational Unified Process (RUP). Det skyldes at de som lanserte prosessmodellen, kommersialiserte den i firmaet Rational som senere er overtatt av IBM. Man kan vel si at UP vokste frem fra det objektorienterte paradigme og problemene med fossefallsmodellen.

UP er et resultat av samarbeidet mellom de tre (amigos), Ivar Jacobson, Grady Booch og James Rumbaugh (Jacobson mfl. 1999). Disse tre er alle pionerer innenfor objektorientert systemutvikling. De hadde tidligere gitt forskjellige bidrag i form av metoder, notasjoner og prosesser.

Den overordnede filosofi for UP er inkrementell og iterativ utvikling.

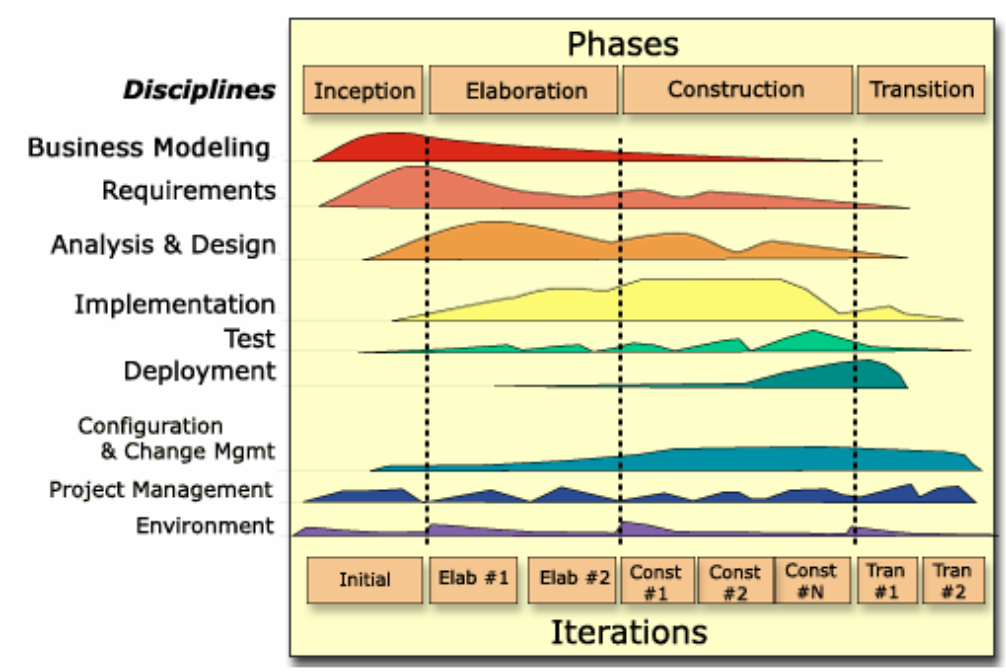
Prinsippene

UP er en modell for utvikling av objektorienterte systemer. Det avspeiles i det som er de beste praksiser som modellen bygger på. Disse er:

- Inkrementell og iterativ utvikling
- Use case drevet
- Risikohåndtering
- Brukermedvirkning
- Visuelle modeller fremfor dokumenter
- Tidlig fokus på arkitektur
- Kontinuerlig verifisering av kvalitet
- Kontroller endringer

Modellen

Figur 2 viser prosessmodellen. Den har to dimensjoner. Den horisontale dimensjonen viser tidsaspektet. Den vertikale dimensjon har disiplinene. De kan kjennes igjen som



Figur 2 Unified process

faser i en fossefallsmodell. Men et av poengene er at til forskjell fra fasene i fossefallsmodellen så skal ikke disiplinene nødvendigvis gjennomløpes i en gitt sekvens og de har ikke samme omfang i alle deler av et utviklingsprosjekt. Som modellen viser itererer man seg gjennom de fire fasene – *oppstart* (inception), *bearbeiding* (elaboration), *konstruksjon* (construction) og *overføring* (transition). I en gitt iterasjon vil man bruke et sett av disiplinene avhengig av hva målet til iterasjonen er. For eksempel så ser man at i den første fasen ligger hovedvekten på forretningsmodellering og krav. For å avklare kravene kan man lage prototyper og det kan medføre litt analyse, design, implementasjon og test. Testing er for øvrig noe man gjør kontinuerlig og ikke helt til slutt som i fossefallsmodellen. Dette er et element i den kontinuerlige kvalitetskontrollen.

Hver fase har sine konkrete mål og ender opp i et beslutningspunkt. Modellen er et rammeverk som må tilpasses et gitt utviklingsprosjekt og organisasjonskulturen. I rammeverket defineres:

- roller
- artefakter
- aktiviteter
- retningslinjer
- konsepter
- mentorer

Målet i de enkelte faser er

- *Oppstart.* Etablere visjon, omfang og en grov overordnet plan for hele prosjektet. Få visshet for at prosjektet kan gjennomføres. En kjørbar prototyp kan lages for å skaffe bevis for at prosjektet kan gjennomføres.

- *Bearbeiding.* Designe, implementere og teste en grunnleggende og robust arkitektur. Ved slutten av denne fasen har man et kjørbart system som gir visshet om at en robust arkitektur er på plass. De viktigste use case er realisert i dette kjørbare systemet.
- *Konstruksjon.* Bygg den første operasjonelle versjon av systemet. Gjennom flere iterasjoner legges mer ”kjøtt” på det ”skjelettet” som ble etablert i bearbeidingsfasen. Ved utgangen av konstruksjonsfasen kan man beslutte om systemet kan overleveres til sitt driftsmiljø.
- *Overføring.* Overlever systemet til sluttbrukeren. Under dette arbeidet kan det tenkes at det dukker opp ting som gjør det nødvendig med endringer og mer utvikling. Ved slutten av denne fasen vurderes det om målene med prosjektet er nådd og om et nytt utviklingsløp eventuelt skal startes.

Integrerad utvekling (IU)

Bakgrunn

Integrerad utvekling kommer fra Sverige. Den er utviklet av det svenske konsulentselskapet CAG Diator. Dette selskapet ser nå ut til å være en del av det norske Itera ASA med hovedkontor i Oslo.

IU dreier seg ikke først og fremst om utvikling av programvare. Ambisjonen er, som det heter, å beskrive et sett med aktiviteter fra analyse av forretningsprosesser til konstruksjon av informasjonssystemer fra et objektorientert perspektiv. Vi kan si at IU er et rammeverk som beskriver en rekke faser for forandringsarbeid i en virksomhet. Etter en innledende fase med aktiviteter for virksomhetsanalyse, kan arbeidet følge tre parallelle løp – nye arbeidsrutiner, en ny organisasjon og et nytt informasjonssystem. I et gitt forandringsprosjekt kan man følge et eller flere av løpene avhengig av situasjonen som avdekkes i den innledende fasen.

Prinsippene

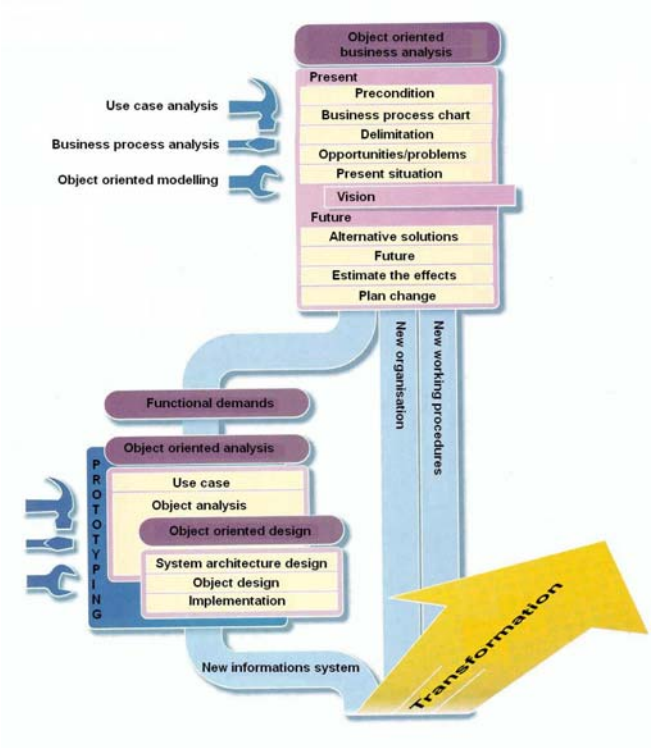
Vi har ikke i litteraturen funnet noen erklæring om en grunnleggende filosofi eller prinsipper bak rammeverket. I (Fagerstrøm mfl. 1998), finner vi IU-metoden beskrevet som en modellkjede helt igjennom basert på et objektorientert angrepssett, objektorienterte teknikker og UML. Videre fremheves det at *virksomhetsanalys* og IT-utvikling ikke kan beskrives ved hjelp av en kokebok, men at all utvikling bygger på hardt arbeid, kreativitet, sunn fornuft og tommelfingerregler for hva man skal gjøre. Det må bety at man også må kunne arbeide iterativt og inkrementelt ettersom det også heter at metoden (IU) er **nøytral når det gjelder hvilken utviklingsprosess som skal følges!** (vår uthevning). En sentral plass har dessuten det de kaller intensive arbeidsseminarer. Det er den samme type møter som finner i forskjellige RAD-metoder og som for så vidt går igjen i alle smidige modeller.

Modellen

En skjematisk beskrivelse av modellen finne vi i Figur 3. Man starter med den objektorienterte virksomhetsanalysen. Den har tre deler:

- Nåtid – forstå det eksisterende systemet for å kunne forandre.
- Visjon – en kreativ aktivitet for å finne alternative måter å arbeide på som oppfyller virksomhetens mål.
- Fremtid – vurdere alternativer og planlegge.

Det brukes tre verktøy – use case analyse, forretningsprosessanalyse og objektorientert modellering.



Figur 3 IU-modellen

Den konkrete arbeidsformen er ikke detaljert beskrevet. Den må velges avhengig av prosjektets egenart. I analysen av virksomhetsprosesser brukes sekvensdiagrammer.

Etter analysen av virksomhetens prosesser splittes aktivitetene i tre løp. Utviklingsløpet skal resultere i et nytt eller endret informasjonssystem. Løpet starter med å finne de funksjonelle kravene. Deretter følger objektorientert analyse. Den skal resultere i:

- Klassediagrammer som gir en oversikt over klasser og deres relasjoner.
- En beskrivelse av hver klasses attributter og metoder.
- Forskjellige diagrammer som beskriver hvordan objekter samspiller.
- Moduldiagrammer for å dele opp beskrivelsen.

Analysen består av syv trinn
Forberedende fase

1. Finn og beskriv use case for systemet.

Utforskende fase

2. Finn objekter i problemdomenet.
3. Klassifiser objektene.
4. For hver klasse dokumenteres kortfattet deres hensikt, tjenester og kunnskap som forvaltes.

Analyserende fase

5. Beskriv relasjoner mellom klasser.
6. Lag og kjør scenarier for å validere systemet.
7. Grupper klasser.

Noen tanker så langt - det står at man må iterere, men ingenting om hvordan man gjennomfører iterasjoner og mål for hver iterasjon eller om de skal timeboxes. Hvordan man skal finne objekter er det heller ingen oppskrift på. Det synes som man i fasen hvor man skal analysere virksomheten går veldig langt i å legge føringer på det fremtidige design. Det er etter vår mening noe uheldig og det er litt rart når man skriver at i analysen skal man finne ut hva systemet skal gjøre samtidig som mesteparten av arbeidet ser ut til å bestå i å finne samarbeidende klasser, noe som egentlig tilhører design. Riktignok heter det at skillet mellom objektorientert analyse og objektorientert design ikke er helt klart. Noe av forvirringen kan ha sin rot i at det ikke alltid er helt klart hvor man befinner seg. Er man i analysen av virksomheten som går forut for splittingen i tre parallelle løp, eller er man i det ene løpet med objektorientert analyse og design?

Design foregår i grenen for utvikling. Her er man mindre preskriptiv ved at man ikke definerer et detaljert antall trinn som man gjør i analysen. Man nøyer seg med å dele inn i to hovedaktiviteter:

- Systemarkitektur eller overordnet design.
- Objektdesign hvor man detaljert beskriver klasser og deres relasjoner.

Selv om man skriver at IU er nøytral når det gjelder utviklingsprosesser, så er det tydelig at tankegangen til Jacobson slik den beskrives i hans bok (Jacobson mfl. 1992) om *Object-oriented software engineering* går igjen. Problemet med Jacobson er at det er uklart hvordan krav til systemet skal finnes. Han introduserte use case som i dag brukes som en måte å representere de funksjonelle kravene på. Men i boken hans er use casemodellering noe som kommer etter kravanalysen og ikke som en del av denne.

Omtrent det eneste som beskrives i noen detalj i forbindelse med IU er seminarteknikker.

IU kan virke noe preskriptiv fordi den krever at det før design kan begynne så skal det foreligge en ”exakt og fullständig dokumentasjon av vad systemet skal klara av”. Med andre ord så ser ikke IU ut til å være særlig påvirket av den smidige (agile) bevegelse – i alle fall ikke i utgangspunktet. Men man kan ikke se bort ifra at man i praktiske anvendelser kan trekke inn smidige ideer.

Extreme programming

Bakgrunn

Extreme Programming (XP) er en prosessmodell som har fått stor oppmerksomhet i den senere tid. Hovedpoenget for de som står bak er at for å utvikle god programvare må man føre ut i det ekstreme det som er nedfelt som de beste praksiser. Kent Beck regnes som hovedpersonen bak modellen. Han må sies å være en av den smidige bevegelses fremste advokater.

Prinsippene

Kent Beck (Beck 1999) skriver følgende om hva som er det ekstreme:

- Hvis koderevisjoner er bra, må vi revidere kode hele tiden. To programmerere må arbeide sammen hele tiden. (Pair programming)
- Hvis testing er bra, skal alle teste hele tiden, også kundene.
- Hvis det å designe er bra, må det være noe enhver holder på med hver dag.

- Hvis enkelhet er bra, skal systemet være i den enkleste form som støtter ønsket funksjonalitet.
- Hvis arkitektur er viktig, skal alle arbeide med å definere og raffinere arkitekturen hele tiden.
- Hvis integrasjonstesting er viktig, så skal man integrere og teste mange ganger hver dag.
- Hvis korte iterasjoner er bra, gjør man iterasjonene virkelig korte – sekunder, minutter og timer, ikke uker, måneder og år.

Videre skriver han at XP, i tillegg til en del andre ting, er en morsom måte å utvikle programvare på. Den skiller seg fra andre prosesser ved

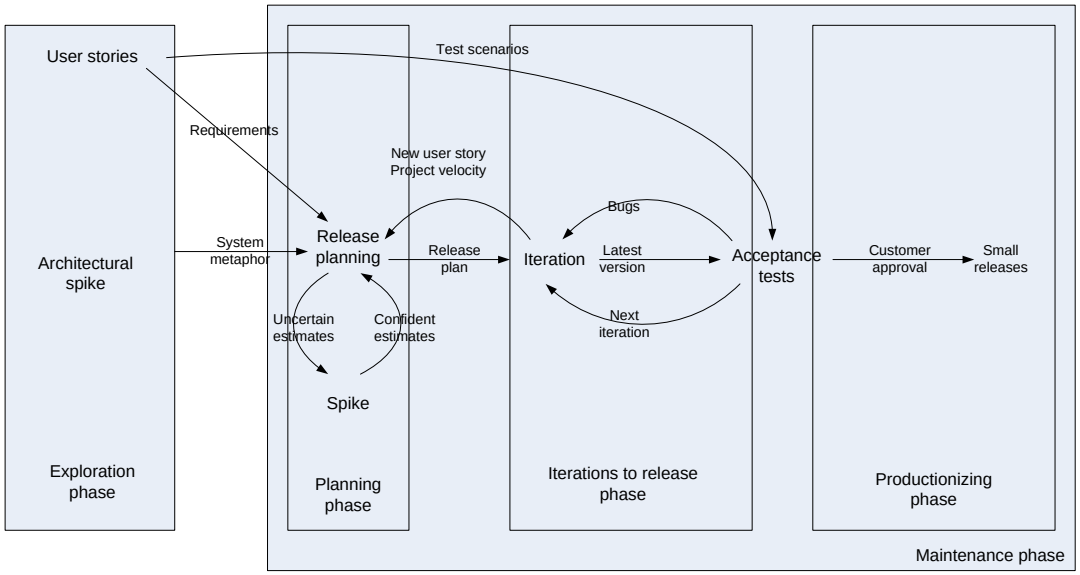
- tidlig, konkret og kontinuerlig tilbakemelding fra korte sykluser
- inkrementell planlegging som tidlig frembringer en grov overordnet plan som skal utvikles videre gjennom livsløpet
- fleksibel implementasjon av funksjonalitet i takt med endrede behov
- bruk av automatiske tester skrevet av programmerere og kunder for tidlig å fange opp defekter
- muntlig kommunikasjon, tester og kildekode for å kommunisere strukturen for og hensikten med systemet
- basering på en evolusjonær designprosess som varer så lenge systemet varer
- tett samarbeid mellom programmerere med gjennomsnittlige ferdigheter
- praksiser som for programmerere instinktivt føles riktige i det daglige, og som tjener prosjektene på lang sikt.

Det som er innovativt med XP er

- Alle gjennomprøvde praksiser settes under en paraply.
- At man sikrer at praksisene følges så nøye som mulig.
- At man sikrer at alle som er involvert støtter hverandre på best mulig måte.

Modellen

XP's fasemodell ser ut som på Figur 4.



Figur 4 Livsløpet i XP

Egentlig er ikke XP-ere så opptatt av faser fordi det kan gi assosiasjoner til en fossefallsmodell. XP-ere gjør en jobb og i denne følger de visse trinn som de kan gå frem og tilbake mellom og gjenta. Første trinn er et *utforskningstrinn* (exploration phase). Utforskning kommer i stedet for kravspesifikasjonen. Kunde og programmerer møtes for å komme til enighet om hva som skal lages. Kunden skriver korte fortellinger (stories) på små kort (kartotekkort). Programmerer sjekker kortene for tvetydigheter og om fortellingene er testbare. Man kan godt sammenligne en fortelling med en høynivå use case. Det vil si at teksten bare består av noen få linjer. Programmereren estimerer hvor lang tid det vil ta å realisere fortellingen. Måleenheten kan være dager målt i ideell tid. På engelsk brukes begrepet *ideal engineering time*. Det vil si den tiden man ville brukt hvis man kunne arbeide alene og uforstyrret. En fortelling skal være så kort at den kan realiseres innen en iterasjon (1 til 3 uker). Kunden spesifiserer en akseptansetest for hver fortelling. Kunden prioriterer fortellingene. De kan for eksempel legges i tre bunker – en med de fortellinger som umiddelbart må realiseres, en bunke med de som kan vente en kort tid og resten i en bunke for de som kan vente noe lengre[1].

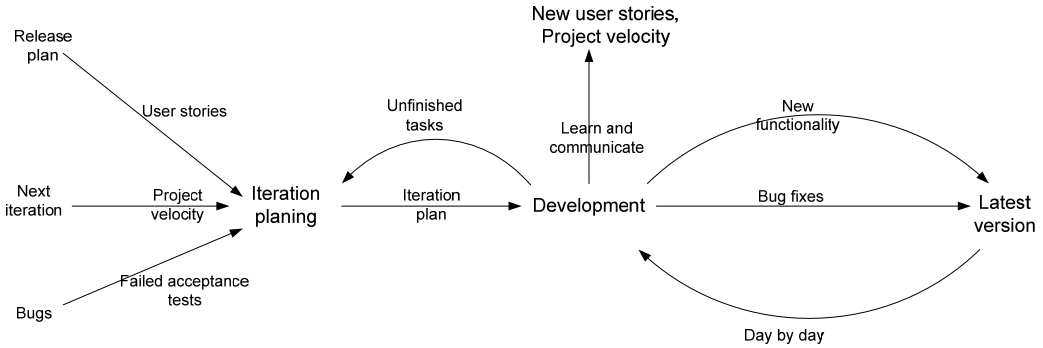
Arbeidet i dette trinnet kan være kaotisk. Det er ikke slik at fortellingene spretter frem en etter en nesten av seg selv. Fortellingene kommer ganske tilfeldig som resultat av diskusjoner mellom kunde og programmerere. Ny innsikt kan føre til at fortellinger endres og at nye blir skrevet. Man kan også eksperimentere under utforskningen. Hvis man er usikker på om ting lar seg realisere eller det er knyttet risiko til ting, kan man skrive ”bruk-og-kast-kode” for å få større visshet. I XP-terminologi sier man at man gjør en *spike*. Det er også viktig tidlig å fastsette en arkitektur for systemet. Forskjellige arkitekturer kan prøves ut ved å gjøre arkitektoniske spike.

Etter utforskningen kommer *planleggingstrinnet* (planning phase). Et XP-prosjekt brytes ned i en rekke frislipp (release). Hvert frislipp skal gi noe av verdi for kunden. Kunden bestemmer hvilke fortellinger som skal med i et frislipp. Det tar typisk en til tre måneder å realisere et frislipp. Måleenheten er arbeidsdager. Hvis man sier at tiden til frislipp er en måned, så betyr

[1] Ikke ulikt MoSCoW-reglene i DSDM.

det at resultatet skal foreligge etter fire arbeidsuker. I Norge er som kjent en arbeidsuke 5 arbeidsdager. Hvor mange fortellinger som kan realiseres i et frislipp, bestemmes av programmerernes ”hastighet”. Den er definert som antall ideelle dager som kan gjennomføres i løpet av en arbeidsuke. Hvis for eksempel en programmerer kan gjennomføre 2,5 ideelle dager i en uke så er hastigheten 2,5. Eller sagt på en annen måte – programmereren gjør 2,5 mengder arbeid i en uke. Skal man fylle opp en hel uke med arbeid for prosjektet, må man ha flere programmerere som arbeider samtidig.

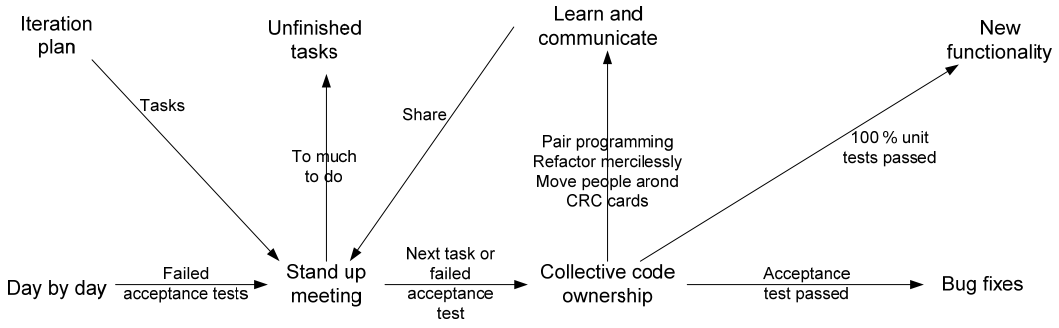
Et frislipp deles opp i iterasjoner. Figur 5 viser livsløpet til en iterasjon.



Figur 5 Livsløpet til en iterasjon

En iterasjon tar fra en til tre uker. Også her er måleenheten arbeidsdager. Altså, hvis iterasjonslengden er en uke, betyr det at iterasjonen varer fem arbeidsdager. Iterasjonslengden bestemmes i starten av prosjektet og skal holdes fast i hele prosjektet. Hvor mye som kan gjøres i en iterasjon er altså bestemt av programmerernes hastighet. Til hjelp for iterasjonsplanleggingen gis kunden et budsjett. Det beskriver den mengde arbeid som utviklingsgruppen kan gjennomføre i en iterasjon. Det brukes samme måleenhet som ved estimeringen av fortellingene. Kunden bestemmer så hvilke fortellinger som skal realiseres, og forplikter seg til ikke å komme med endringer eller tillegg så lenge iterasjonen løper. Kunden spesifiserer en akseptansetest for iterasjonen.

Programmereren bryter fortellingene ned i *oppgaver* (tasks). Størrelsen på en oppgave er slik at den kan utføres i løpet av en til to dager. En oppgave er altså en bit av en fortelling, som for eksempel å få på plass en database, lage en algoritme for å utføre en beregning, osv. En oppgave kan også berøre flere fortellinger. Figur 6 viser utviklingsaktivitetene.



Figur 6 Utviklingsaktivitetene

Programmererne i utviklingsgruppen melder seg nå på de oppgaver de har lyst til å utføre. Ingen blir altså pådyttet oppgaver, men alle oppgaver må ha en ansvarlig og må løses innenfor

iterasjonen. Programmereren lager et estimat for oppgaven, gjerne i timer. Estimatene for alle oppgaver summeres og sammenholdes med lengden av iterasjonen. Hvis det blir for mye arbeid, må kunden velge hva som skal realiseres innenfor denne iterasjonen og hva som eventuelt skal overføres til neste eller senere iterasjoner. Hvis det er ledig tid, sørger kunden for å få frem flere fortellinger. Man praktiserer her strengt prinsippet om fordeling av ansvar – kunden bestemmer hva som skal lages og når. Rammene settes av de estimater som programmererne lager. Programmererne har ansvaret for å velge implementasjonsstrategi.

Det siste trinnet er *produksjon* (productionizing phase). Fokus er på å gjøre programvaren klar til å gå i produksjon. Hovedaktiviteten er på systemtester av forskjellige slag. Det kan være akseptansetest, lasttest, installasjonstest osv.

I XP er vedlikeholdsfasen den normale tilstand. Dette fordi programvaren vil videreutvikles over tid. Det betyr at i et vedlikehold gjentas de tre trinnene med planlegging, frislipp og produksjon.

Métrica 3

Bakgrunn

Métrica 3 (Metodología de Planificción, Desarrollo y Manteniendo de Sistema de Información) er en spansk modell. Den har sin opprinnelse i behov fra spanske offentlige myndigheter for en utviklingsmetodikk. Legg merke til at man bruker begrepet metodikk, ikke metode som mange andre gjør. Igjen er det ønsket om å få levert systemer med kvalitet, i tide og som er i samsvar med brukernes ønsker og krav som er drivkraften. I egen terminologi heter det at det er en metodikk som definerer et sett prosesser, med tilhørende teknikker og definerte produkter.
Metodikken er inspirert bl.a. av det franske Merise, SSADM fra UK og SUMMIT-D fra det som en gang het Coopers & Lybrand. I tillegg bygges det på standardene ISO 12207, ISO/IEC 15504 SPICE og ISO 9000 serien. Métrica har vært gjennom en utvikling med flere versjoner. Dagens versjon er altså Métrica 3.

Prinsippene

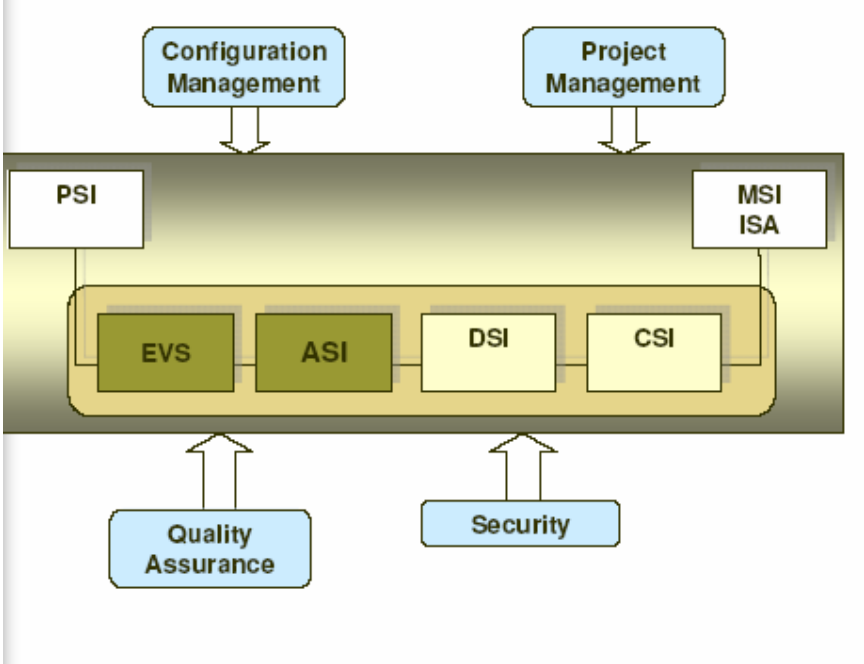
Det er ikke så lett å finne den grunnleggende filosofi bak Métrica 3. Ved søk på nettet støter vi bare borti spanske tekster. Forelesningen som ble holdt i Amsterdam var dessverre ikke klargjørende. Vi baserer oss derfor på den informasjonen vi kan trekke ut av dokumentasjonen fra forelesningen. Det kan virke som Métrica 3 er en tungvektsprosess om ikke nødvendigvis preskriptiv som fossefallsmodellen. Métrica 3 skal kunne skaleres til prosjekter av alle typer og den er fleksibel og kan ta opp i seg andre prosesser og teknikker.

Modellen

Figur 7 viser modellen av Métrica 3 slik den ble presentert i Amsterdam. Den inneholder en rekke prosesser, teknikker og interesseparter. Prosesstankegangen er noe vi finner i ISO standardene som ligger i bunnen. I andre modeller brukes gjerne begrepet faser om disse prosessene og delprosessene. Det er tre hovedprosesser (forkortelsene i parentes er sannsynligvis spanske)

- Information system planning (PSI)
- Information system development
  - Feasibility study (EVS)

– IS analysis (ASI)
– IS design (DSI)
– IS implementation (CSI)
– Deployment and acceptance (ISA)



Figur 7

Det er fire grensesnitt:

- Project management
- Security
- Configuration management
- Quality assurance

Interesseparter defineres gjennom såkalte profiler:

- Executive profile som representerer ledelsen og ekspertbrukere.
- Project manager profile som representerer prosjektledelsen, men også områdeledere (vedlikehold, kvalitet, …)
- Consultant profile
- Analyst profile
- Programmer profile

Hver hovedprosess er delt opp i delprosesser med input, hovedoppgave, utputt og teknikker/praksiser. For eksempel så består System feasibility study (EVS) av disse delprosesser:

- Definition of system scope
- Study of current situation
- System requirements definition

- Study of alternative solutions
- Alternative solutions evaluation
- Solution choice

EVS teknikker og praksiser er bl.a.:

- Kost/nytteanalyse
- Use case
- Klassediagrammer
- Dataflytdiagrammer
- Interaksjonsdiagrammer
- ER-modellering
- Møter

Som vi ser kan man bruke både objektorienterte og strukturerte teknikker. Det går igjen i alle prosesser og delprosesser. Likedan har intensive arbeidsmøter en plass.

Prosjektstyringsmodeller

PRINCE2

Bakgrunn

PRINCE er et akronym for Projects in Controlled Environments. Metoden, som er det begrepet de selv bruker, ble opprinnelig utviklet av det britiske Central Computer and Telecommunication Agency (CCTA). Den er nå lagt inn under Office of Government Commerce (OGC) og er de britiske myndigheters offisielle standard til bruk i IT-prosjekter. Det har ført til at den brukes i mange private bedrifter og offentlig institusjoner som en de facto standard for prosjektstyring, og ikke bare i IT-prosjekter.

Videreutviklingen håndteres nå av et konsortium under OGC. Nåværende versjon er PRINCE2. Om den heter det at den er en prosessbasert måte å administrere og styre prosjekter på og som skal kunne skreddersys til en hvilken som helst type prosjekt.

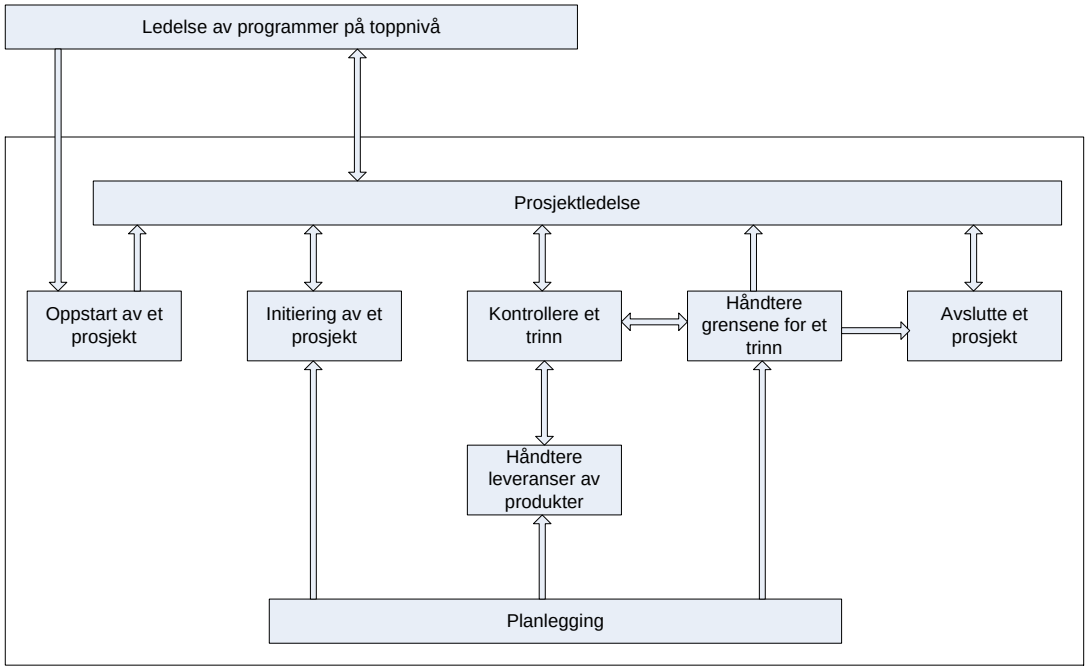
Prinsippene

Bruk av PRINCE2 skal føre til at prosjekter har

- En kontrollert og organisert start, gjennomføring og slutt
- Jevne gjennomgåelser av prosjektforløpet mot planer og forretningsmål
- Fleksible beslutningspunkter
- Automatisk kontroll med avvik fra planer
- Involvering av ledelsen og interesseparter til rett tid og sted gjennom prosjektløpet
- Gode kommunikasjonskanaler mellom prosjektet, prosjektledelsen og resten av organisasjonen

Modellen

PRINCE2 har tre elementer – *prosesser*, *komponenter* og *teknikker*. Prosesser er definert med innputt, utputt, mål og aktiviteter som skal utføres. Fra den offisielle hjemmesiden til PRINCE2 har vi hentet prosessmodellen. Se Figur 8.



Figur 8

Det er åtte høynivåprosesser

- Prosjektledelse (Directing a project)
- Planlegging (Planning)
- Oppstart av et prosjekt (Starting up a project)
- Initiering av et prosjekt (Initiating a project)
- Kontrollere et trinn (Controlling a stage)
- Håndtere leveranser av produkter (Managing product delivery)
- Håndtere grensene for et trinn (Managing stage boundaries)
- Avslutte et prosjekt (Closing a project)

I Figur 8 er relasjonene mellom disse åtte prosessene vist inne i en ramme. Utenfor rammen er en prosess som ikke er en del av PRINCE2, men som ligger på overordnet bedriftsnivå. Der bestemmes hvilke prosjekter som skal gjennomføres i bedriften.

Metoden deler et prosjekt inn i håndterbare trinn som skal gjøre det mulig å ha effektiv kontroll med ressursene og forløpet av prosjektet. Trinn (stage) er begrepet som brukes om det vi vanligvis kaller prosjektfase.

Hensikten med de forskjellige prosessene

- *Prosjektledelse*. Definerer oppgavene til styringsgruppen (Project board) som er ansvarlig for prosjektet. Prosjektleder skal rapportere jevnlig til styringsgruppen. Prosjektlederen har den daglige ledelsen av prosjektet. Styringsgruppen involveres bare ved avslutningen av et trinn (stage boundary) når den skal godkjenne det som er gjort og avgjøre videreføring. Et grunnleggende prinsipp i PRINCE2 er *unntaksledelse* (management by exception). Det vil si at styringsgruppen bare skal trekkes inn når ting ser ut til å gå galt.
- *Planlegging*. Planlegging foregår gjennom hele prosjektets livsløp. Planlegging innebærer alle de ting vi forbinder med prosjektplanlegging.

- *Oppstart av et prosjekt*. Dette er en prosess som kjører før det egentlige prosjektet starter. Hensikten er å få klarhet i om prosjektet bør starte og i så fall gi prosjektet et solid fundament å starte på. Innputt er et prosjektmandat. Styringsgruppen oppnevnes og den utpeker en prosjektleder.
- *Initiering av et prosjekt*. Hovedresultatet fra denne prosessen er ”Project Initiation Document (PID)”. Dette må godkjennes av styringsgruppen. Dette dokumentet inneholder prosjektplanen, kvalitetsplaner, en raffinert business case, risikoplan osv., eller med andre ord det man alltid gjør i starten av et prosjekt.
- *Kontrollere et trinn*. PRINCE2 prosjekter deles opp i trinn (faser). Men det foreskrives ikke et bestemt antall trinn. Det må bestemmes ut fra prosjektets egenart. Et prosjekt vil som regel ha i alle fall fire trinn – oppstart, initiering, gjennomføring og avslutning. Det er gjennomføringstrinnet som eventuelt kan deles videre opp. Prosessen med kontroll av trinn inneholder det som gjøres fra dag til dag i et prosjekt som fordeling av arbeid, oppfølging, rapportering, og korrigering av kurs.
- *Håndtere leveranser av produkter*. En av hovedideene i PRINCE2 er at man skal fokusere på produkter ikke aktiviteter. Med andre ord, det er resultatene fra prosjektet som det hele dreier seg om. Et produkt kan være noe fysisk eller en tjeneste, eller egentlig alt som lages inkludert dokumenter. Det kan se ut som produkt er det som i andre modeller (for eksempel UP) kalles artefakt. Prosessen lager produktene og det er i denne prosessen det fleste ressurser brukes.
- *Håndtere grensene for et trinn*. Prinsippene i PRINCE2 krever at hvert trinn må godkjennes av styringsgruppen før neste trinn kan startes. Elementene i denne prosessen er planlegging av neste trinn, oppdatering av overordnet prosjektplan, oppdatering av ”business case”, oppdatering av risikologg og rapportering av status.
- *Avslutte et prosjekt*. Denne prosessen er også en konsekvens av et grunnlegende prinsipp i PRINCE2. Et prosjekt må avsluttes på en kontrollert måte. Prosjektet må evalueres og erfaringene må dokumenteres og tas vare på.

Komponentene

PRINCE2 definerer åtte prosjektstyringskomponenter som en del av modellen

- *Business Case*. En beskrivelse av begrunnelsen for prosjektet.
- *Organisasjonen*. Her defineres roller og ansvar for de som er involvert i prosjektet. Det er både ledelsesroller og utførende roller.
- *Planer* som er på tre nivåer – prosjektplaner, trinnplaner, gruppeplaner.
- *Kontroller* (controls) som skal sikre at riktig prosjekt kjøres til riktig tid og at prosjektet hele tiden tilfredsstiller ”business case”.
- *Risikostyring*. Det dreier seg om å ha risikofaktorene under kontroll.
- *Kvalitet*. Kort og godt kvalitetssikring i prosjektet slik at krav og forventninger blir oppfylt.
- *Konfigurasjonsstyring*.
- *Endringskontroll*.

Teknikker

Det er få konkrete teknikker som inngår i PRINCE2. Det betyr at teknikker må velges avhengig av prosjektets egenart. Tre teknikker er spesifisert:

- *Produktbasert planlegging*. Planlegging og oppfølging skal gjøres mot oppfølging av objektive målinger på produkter.

- *Endringskontroll*. Det er tre typer endringssaker som skal håndteres. Den ene er endringer av krav eller av produkt. Den andre er endringer som følge av at et produkt viser seg ikke å tilfredsstille kravene som er spesifisert. Den siste er generelle spørsmål. Forespørsler om endringer må godkjennes av styringsgruppen.
- *Kvalitetsgjennomgåelser*. PRINCE2 krever at produkter må underkastes produktgjennomgåelser. Gjennomgåelsene er formelle.

Diskusjon av modeller

Generelt

Diskusjonen er i hovedsak knyttet opp mot resultatene i IP-prosjektet, men har også en mer generell side. På den annen side er det vel ikke mulig å si på generelt grunnlag at en modell er bedre enn en annen, i alle fall ikke ut fra resultatene fra dette prosjektet og den gitte casen. Det er flere grunner til det. For det første ble det ikke av noen gruppe levert et ferdig kjørbart produkt som kunne evalueres. Gruppen som benyttet XP hadde riktignok laget ”spikes” for sine fortellinger (stories), men ikke et fullgodt system. Man har heller ikke kunnet vurdere andre smidige modeller som Crystal (Cockburn 2004) og Scrum (Schwaber mfl. 2001).

For det andre hadde ikke studentene erfaring nok til å kunne plukke ut fra de større rammeverk som UP, IU, DSDM og Métrica 3, det som er nødvendig og tilstrekkelig for en så liten problemstilling som dette prosjektet dreide seg om. Det fører gjerne til at de produserer dokumenter som kan lages, uten å vurdere behovet for og nytten av dokumentene. Denne nytten ser man gjerne ikke før programvaren har vært i drift og gjennomgått flere endringer. Hvordan sikrer for eksempel et rammeverk som IU at man får effektiv, vedlikeholdbar og flyttbar kode? Dette er i stor grad avhengig av designmetode og implementasjonsspråk, noe IU ikke er spesifikk på. Den krever riktignok objektorientering. Men det er ikke gitt at det er det riktige for alle typer systemer. Det samme kan sies om Métrica 3 hvor man kan velge å utvikle objektorientert eller funksjonsorientert. Og det gjelder ikke minst PRINCE2 som ikke er en utviklingsmodell, men en prosjektmodell. UP og XP spesifiserer heller ikke en bestemt måte å designe på, men de vektlegger betydningen av arkitektur og krever objektorientering. Disse modellene legger derfor til rette for vedlikeholdbar og flyttbar kode. Det er jo nettopp noen av de påståtte goder ved objektorientering, men slett ingen garanti. På den annen side vil et attributt som effektivitet være vanskelig å oppnå uten å redusere på spesielt vedlikeholdbarhet og flyttbarhet. Det kan bety at hvis effektivitet er viktig, må man gjøre andre ting enn bare å følge en utviklingsmodell.

En slags konklusjon som vi og studentene kan trekke, er at XP, UP og DSDM generelt ikke kan brukes alene i større prosjekter fordi de er svake på prosjektstyring. På den annen side synes IU og Métrica 3 å være for omfattende for mindre prosjekter og må i tillegg ta opp i seg hele eller deler av ting fra andre modeller som XP, UP og DSDM. PRINCE2 er et interessant prosjektstyringsrammeverk som kan gi en manglende dimensjon til XP, UP og DSDM. Men det finnes alternativer til PRINCE2, som vi altså ikke har kunnet vurdere her.

En annen ting er at man neppe får vist styrken eller svakheten til smidige, iterative og inkrementelle modeller fordi ingen grupper leverte et ferdig produkt. Slike modeller skal jo ha et fortinn når det gjelder å sikre rett funksjonalitet og god brukbarhet blant annet gjennom aktivt og kontinuerlig brukermedvirkning. Men uten noe kjørbart og uten en reell deltakende kunde eller bruker, er det vanskelig å trekke noen endelige konklusjoner om disse modellers fortrinn, selv om vi nok tror at de har det. Suksess med smidige modeller avhenger av et effektivt lagarbeid. Et smidig lag er selvstyrt (autonomt). Medlemmene har tillit til hverandre og utfyller hverandre slik at direkte, uformell muntlig kommunikasjon kan erstatte mange formelle dokumenter. Det krever er tett samarbeid i selvstyrte grupper med medlemmer som har tillitt till hverandre, samt andre avtaleformer mellom oppdragsgiver og oppdragstaker.

Det vi i alle fall kan slå fast, er at det finnes mange utviklingsmodeller. Noen modeller er spesifikke for bestemte land. IU og Métrica 3 er eksempler på det. Men det er antagelig lite

trolig at disse vil finne noen bred anvendelse utenfor Sverige og Spania. Selv om DSDM og PRINCE2 opprinnelig er britiske, ser de ut til å ha en viss utbredelse, i alle fall i Europa, men vil nok ha konkurranse fra UP for større prosjekter. DSDM med elementer fra PRINCE2 kan være et alternativ til UP. På den annen side er UP spesifikk på utvikling av objektorienterte systemer og betraktes nok som en de facto standard for utvikling av slike systemer.

En sammenligning av modellene

I Tabell 1 har vi laget et sammendrag av egenskapene til de forskjellige prosessmodellene. Det må understrekes at for en del modeller har det vært vanskelig å finne nok stoff. Det gjelder spesielt Métrica 3 hvor det meste som er tilgjengelig er på spansk. Vi har heller ikke hatt tilgang på detaljert dokumentasjon av PRINCE2. Omfattende dokumentasjon finnes, men har en kostnad. For selv om PRINCE2 er gratis å bruke, er den opphavsrettsbeskyttet. Det betyr at dokumentasjonen, bortsett fra generelle oversikter, ikke er fritt tilgjengelig og må kjøpes. Så det kan hende vi er litt urettferdig overfor noen ettersom vi har basert våre vurderinger på de forelesningene vi overvar og informasjon på nettsider. Noen av forelesningene hadde et visst preg av reklamefremstøt for visse modeller.

Prosessmodell	**Type modell**	**Styrker**	**Svakheter**	**Andre kommentarer**
UP (RUP)	Lett tungvekt. Adaptiv. Kan være smidig.	Tar opp i seg det spesielle ved programvareintensive prosjekter. Setter inn i et rammeverk de beste praksiser for objektorientert utvikling. Fokuserer på tidlig risikohåndtering. Løser de vanskelig og kritiske tingene først.	Laget for det objektorienterte paradigme. Kan virke tung. Har vært uklar på innsamling av krav.	UP og spesielt RUP oppfattes av mange som en tungvektsprosess. Det er ikke intensjonen. UP er et rammeverk som skal tilpasses et aktuelt behov. XP kan godt innpasses. Det kan være at oppfatningen av UP som tungvekts kan ha sammenheng med de mange retningslinjer, verktøy og dokumentmaler som leveres med RUP.

Prosessmodell	Type modell	Styrker	Svakheter	Andre kommentarer
XP	Lettvekt. Adaptiv. Smidig.	Tett kommunikasjon mot brukere. Tidlig tilbakemelding fra kjørbart system. Fokuserer på ting som er nyttig. Gjør det morsomt å utvikle programvare. Kort vei fra ide til realisert system.	Kan virke uegnet for store prosjekter og prosjekter hvor det ikke kan defineres en bestemt bruker. Dårlig egnet for kritisk programvare? Kun det objektorienterte paradigme.	Er av kritikere oppfattet som en modell uten struktur egnet for hackere. Det er ikke tilfelle. XP er ekstremt strukturert uten å vektlegge ting som ikke har direkte betydning for sluttresultatet - et kjørbart system som tilfredsstiller brukernes krav og forventninger
DSDM	Lett tungvekt. Adaptiv. Smidig.	Ikke bare for det objektorienterte paradigme.	Omfavner det agile manifest, men kan virke omfattende på grensen til tungvekt.	
IU	Tungvekt. Adaptiv.	Tar for seg utvikling av både prosedyrer, organisasjon og programvare.	Er fokusert på virksomhetsforbedring og innføring av informasjonssystemer. Har derfor liten verdi hvor dette ikke er hovedpoenget. Bruker det objektorienterte paradigme i alle faser. Uklar på skillet mellom analyse av problemområdet og design av programvaren. Virker unødvendig å skille mellom funksjonelle krav og use case.	Fordi modellen fremhever som en fordel at man arbeider med objekter helt fra virksomhetsanalyse til implementasjon, kommer man i skade for tidlig å legge føringer på design av programvaren. Det er både en fordel og svakhet at man ikke er konkret på en utviklingsmodell for selve programvaren. Det gir fleksibilitet. Men krever likevel use case og bruk av UML. Opphengt i tidlig Jacobsontenkning?
Métrica 3	Tungvekt. Predikativ	Setter systemutvikling inn i et rammeverk.	Virker meget dokumentdrevet. Kan lett gli over i et fossefallsløp.	Virker nærmest super tungvekt, selv om den skal kunne skaleres.

Prosessmodell	Type modell	Styrker	Svakheter	Andre kommentarer
PRINCE2	Tungvekt. Virker predikativ.	Samler beste praksiser i et rammeverk. Introduserer standard fremgangsmåter, begreper og dokumenter. Etter hvert mange brukere. Maler for dokumenter. Kan skaleres og tilpasses.	Ikke lenger spesielt for utvikling av programvare. Derfor ikke fokusert på problemer og egenheter ved programvareintensive prosjekter som håndtering av krav i endring, korte leveringstider. Testing er ikke berørt. Må kombineres med modeller og praksiser for utvikling av programvare.	
Fossefall	Tungvekt. Predikativ.	Første forsøk på å strukturere utviklingsarbeidet.	Forutsetter at alle krav kan finnes tidlig og at enhver analyse og modellering er stabil over tid. Involverer brukere og andre interesseparter dårlig. Tilbakemelding for sent. Testing som egen aktivitet helt i slutten av prosjektet. Løser de enkleste problemer først.	

Tabell 1

PRINCE2 og Métrica 3 bærer preg av å være utviklet for bruk i offentlig sektor. Det er mye dokumentasjon og formalisme for å oppnå kontroll. Man kan derfor ikke fri seg fra tanken om at disse modellene drar med seg mange av svakhetene med fossefallsmodellen. PRINCE2 var opprinnelig laget for IT-prosjekter, men er etter hvert blitt en generell prosessmodell. I Storbritannia er det den modellen som skal følges i prosjekter hvor det offentlige er involvert.

I hvilken grad kan kvalitetskriterier oppfylles?

I ettertid skulle man vært i stand til å stille opp en tabell som den som følger her. Tabell 2 viser hovedkategoriene av kvalitetsattributter etter ISO 9126 og i hvilken grad disse blir sikret tilfredsstillelse ved bruk av prinsipper og beste praksiser fra de forskjellige utviklingsmodellene. Vi har utelatt fossefallsmodellen fordi den neppe er aktuell, i alle fall ikke i sin rene sekvensielle form.

Hovedkategori	**UP**	XP	**IU**	Métrica 3	**DSDM**	PRINCE2
Funksjonalitet	Bruker-medvirkning Prioritering av use case Tidlig kjørbart system	Bruker-medvirkning Prioritering av user stories Tidlig kjørbart system	Grundig virksomhets-analyse		Bruker-medvirkning MoSCoW-reglene	
Pålitelighet	Risiko-håndtering Stadig testing Prototyping	Stadig testing som må passeres Skriv tester før koding Refakak-torering			Risiko-håndtering Stadig testing Prototyping	
Brukbarhet		Enkelhet Parprogram-mering				
Effektivitet		Enkelhet Refak-torering				
Vedlikeholdbarhet	Visuell modellering Objekt-orientering	Objekt-orientering Parprogram-mering Enkelhet				
Flyttbarhet	Objekt-orientering Fokus på arkitektur	Objekt-orientering Fokus på arkitektur	Objekt-orientering			
Innefor budsjett	Timeboxing Prioritering av use case Risiko-fokusering	Timeboxing Prioritering av stories			Timeboxing MoSCoW-reglene	
Innenfor tidsfrister	Timeboxing Prioritering av use case Risiko-fokusering	Timeboxing Prioritering av stories			Timeboxing MoSCoW-reglene	

Tabell 2 Prinsipper og praksisers bidrag til kvalitetsoppnåelse

Blanke celler betyr at vi ikke har funnet noe konkret i modellen som sikrer at kvalitetsattributtet blir oppfylt. Men det betyr ikke at kvalitetsattributtet ikke kan oppnås. Man må bare selv velge metoder og teknikker. For eksempel så kan man utvikle objektorientert i alle modeller som ikke krever et bestemt paradigme, og dermed oppnå fordelene med objektorientering.

Ord og uttrykk brukt i rapporten

Ord/uttrykk	*Forklaring*
Agile	Smidig. Brukes om lettvekts utvikligsprosesser.
DSDM	Dynamic Systems Development Method. En utviklingsmetode, men begrepet assosieres også med et konsortium, DSDM-Consortium.
HvA	Hogeschool van Amsterdam
ID	Integrated Development method. Engelsk oversettelse av Integrerad Utvecklingsmetod. Se IU.
IKT	Informasjons- og kommunikasjonsteknologi.
IP	Intensivprogram. Et område som det gis støtte til i EUs Sokratesprogram.
ISO	International Organization for Standardization. Et nettverk av nasjonale standardiseringsinstitusjoner. ISO er ikke et akronym, men hentet fra gresk isos som betyr lik. Gjort for at man skal kunne bruke samme begrep i alle medlemsland.
IU	Integrerad Utvecklingsmetod. Svensk rammeverk for utvikling av totale informasjonssystemer basert på det objektorienterte paradigme.
Métrica 3	Metodología de Planificción, Desarrollo y Manteniendo de Sistema de Información. Versjon 3 av en spansk utviklingsmodellmodell.
PRINCE2	Projects in Controlled Environments. Vesjon 2 av et britisk prosjektstyringsrammeverk.
RUP	Rational Unified Process. Et kommersielt rammeverk for utvikling av objektorienterte systemer. Se også UP.
UP	Unified Process. Et rammeverk for utvikling av objektorienterte systemer. Også ofte omtalt som RUP, Rational Unified Process.
XP	Extreme Programming. En smidig utviklingsmodell lansert av Kent Beck.

Referanser

Litteratur

Beck, Kent 1999. *Extreme programming explained.* Reading: Addison Wesley

Tore Berg Hansen og Greta Hjertø 2005. *IP project – Improving the success of ICT projects quality assurance Amsterdam 2005.* Trondheim: AITeL

Cockburn, Allister 2004. *Crystal clear: A human powered methodology for small teams.* Reading: Addison Wesley

Fagerström, Johan, Pierre Bjurhager, Anders Wallström og Thomas Jönsson 1998. *Objektorientering i hela företaget,* Lund: Studentlitteratur

Jacobson, Ivar, Grady Booch og James Rumbaugh 1999. *The unified software development process.* Reading: Addison Wesley

Jacobson, Ivar, Magnus Christerson, Patrik Jonsson og Gunnar Övergaard 1992. *Object-oriented software engineering. A use case driven approach.* Wokingham: Addison Wesley

Schwaber, Ken og Mike Beedle 2001. *Agile software development with Scrum.* : Prentice Hall

Stapleton, Jennifer 2002. *DSDM Business focused development.* :Pearson Education

Websteder

Senter for internasjonalisering av høyere utdanning (SIU). http://www.siu.no/

PRINCE2. http://www.prince2.com

DSDM Consortium. http://www.dsdm.org/

Métrica 3. http://www.csi.map.es/csi/metrica3/

Avdeling for informatikk og e-læring
Høgskolen i Sør-Trøndelag
Besøksadresse:
Brygghuset ved Leutenhaven
3 etg., E. C Dahls gt. 2.

Postadresse:
7004 Trondheim

Tlf. 73 55 95 40
Fax 73 55 95 41

Web-adresse: aitel.hist.no
E-post: postmottak@aitel.hist.no

ISBN 82-7877-137-5
ISSN 1504-5587