Systemutviklingsprosessen

Plan for forelesningen

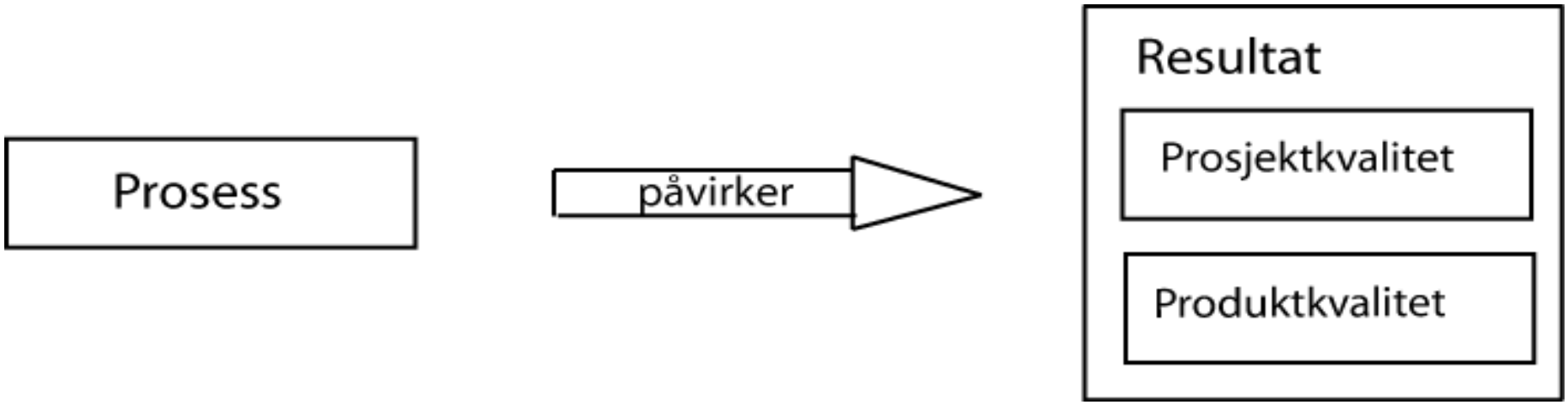
- Prosessbegreper
- Prosessmodeller
  - Fossefallsmodellen
  - Inkrementell og iterativ utvikling
  - Spiralmodellen
  - Rational Unified Process (RUP)
- Gjenbruksbasert utvikling
- Evolusjon (endring) av programvare
- Smidig metodikk

Overordnet mål:

Hvordan utvikle, videreutvikle og vedlikeholde IT-systemer av bedre kvalitet på kortere tid og med lavere kostnader (enn i dag)?

Prosessen påvirker resultatet

- Systemutviklingsprosessen, dvs. måten man jobber på, i et utviklingsprosjekt vil påvirke kvaliteten både på prosjektet selv og systemet som utvikles
- Måten man jobber på påvirker også arbeidsmiljøet (trivsel, motivasjon, kompetanseutvikling etc.) som igjen påvirker prosjekt- og produktkvalitet generelt



Systemutviklingsprosess

- Systemutviklingsprosess (= programvare-prosess) er de aktivitetene som utføres for å utvikle et datasystem
- Aktivitetene varierer, men vil alltid ha elementer av
  - spesifisering av kravene, dvs. hva systemet skal gjøre
  - design av systemet (for eksempel lage en datamodell)
  - implementering av koden (programmering)
  - validering av at systemet gjør det kunden ønsker
  - endringer av systemet i forhold til nye og endrede krav hos kunden

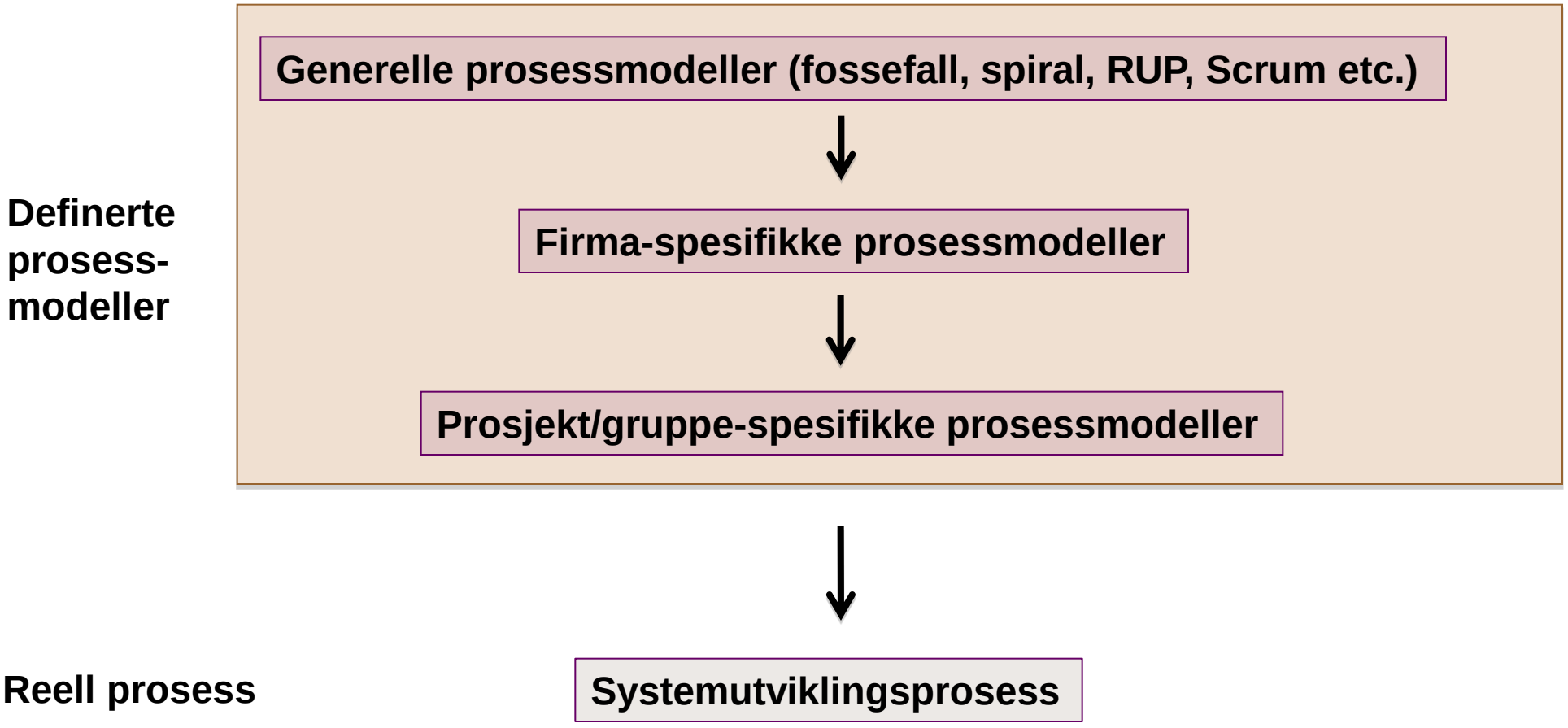
Prosess-egenskaper

- Hvilke aktiviteter inngår i prosessen?
- Hvor mye av hver aktivitet (absolutt og relativt i forhold til hverandre)?
- Når i utviklingsfasen gjøres (og hvor mye av) hver aktivitet?
- Prosessbeskrivelser vil også kunne inneholde
  - delprodukter/resultater (modeller, figurer, tekst, kode etc.) av en aktivitet
  - før- og etterbetingelser (pre- and post-conditions), dvs. betingelser som er sanne før og etter en fase eller et delprodukt er levert
  - rollene til dem som er involvert i prosessen
  - metoder, verktøy og teknikker som brukes

Reell prosess versus modell av prosess

- Systemutviklingsprosess (= faktisk, reell prosess):
  - de aktivitetene som utføres i et utviklingsprosjekt
- Prosessmodell
  - En abstrakt representasjon av en prosess. Modellen er gjerne normativ (preskriptiv), dvs. beskriver en prosess slik noen mener den *bør* være

Nivåer av prosessmodeller



Tunge versus lette prosesser

- En tung prosess inkluderer mange aktiviteter og roller, og krever formelle, detaljerte og konsistente prosjektdokumenter
- Tunge prosesser er ofte “for-tunge”, dvs. vektlegger aktiviteter som vanligvis gjøres tidlig i prosessen (planlegging, analyse & design)
- Lette prosesser fokuserer mer på fundamentale prosesser (“f.eks. kontinuerlig testing”) og har færre formelle dokumenter

Smidige prosesser (metoder)

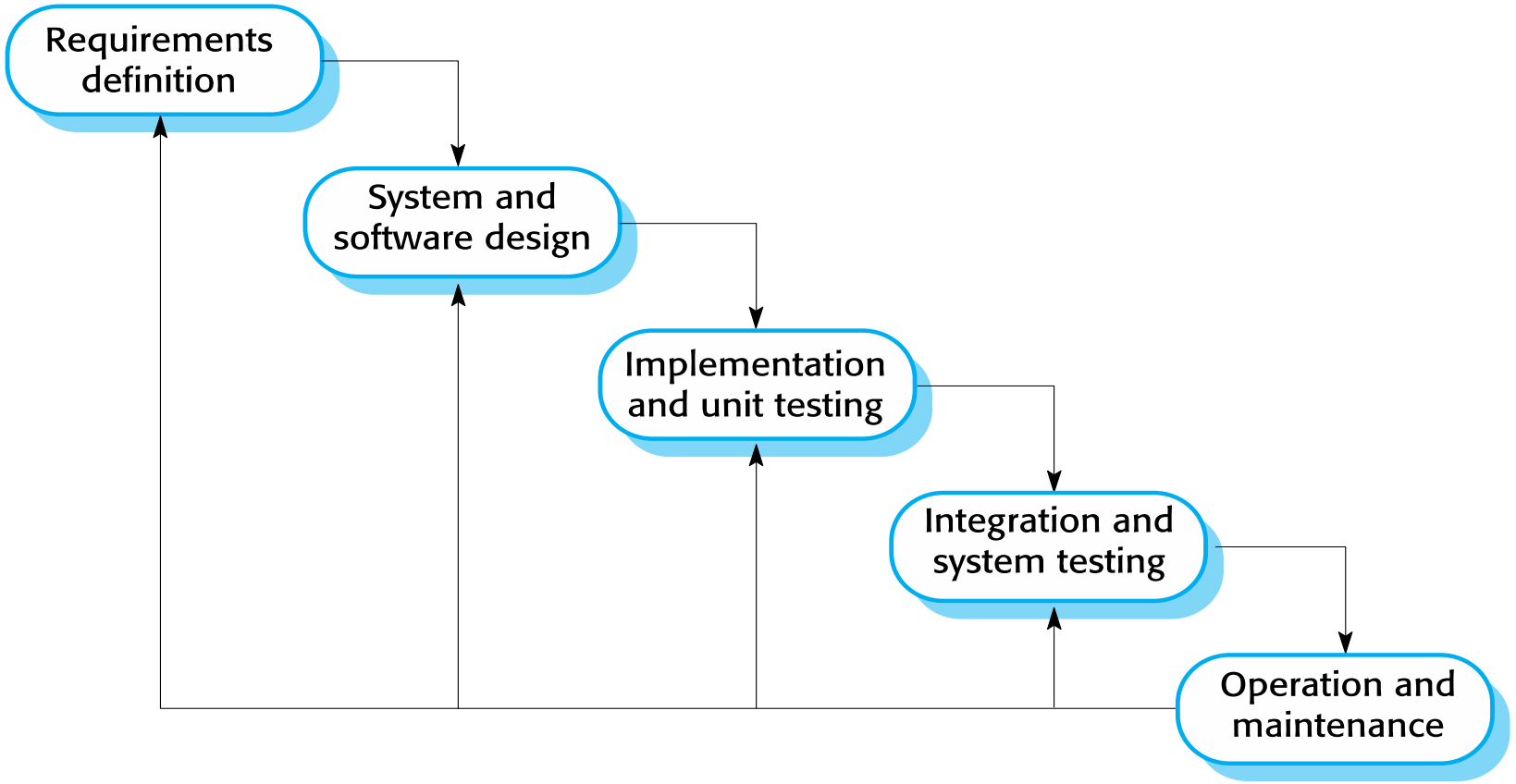
- Elementer fra inkrementell og iterativ utvikling, eksemplifisert ved RUP, danner utgangspunkt for “smidige metoder”
- I plandrevne prosesser er alle prosessaktivitetene planlagt på forhånd og progresjon måles i henhold til denne planen. Plandrevne prosesser er ofte tunge.
- I smidige prosesser gjøres planleggingen litt etter litt (inkrementelt) og det er enklere å endre prosessen for å reflektere endrede krav fra kunden

Eksempler på prosessmodeller

- Fossefallsmodellen
  - Plandrevet modell. Utviklingen foregår i veldefinerte faser
- Inkrementell utvikling
  - Utviklingen foregår gradvis gjennom ulike aktiviteter som man veksler mellom. Kan være plandrevet eller smidig, f. eks. Scrum
- Gjenbruks-orientert systemutvikling
  - Systemet er satt sammen av eksisterende komponenter. Kan være plandrevet eller smidig

I praksis er de fleste store systemer utviklet ved bruk av en prosess som inkluderer elementer fra alle disse modellene

Fossefallsmodellen



I prinsippet går man ikke tilbake til tidligere hovedaktiviteter før systemet er satt i drift, men i praksis likevel stort overlapp i aktiviteter

Egenskaper ved fossefallsmodellen

- Vanskelig å tilpasse endringer i krav underveis i prosjektet
  - Egner seg best når kravene er godt forståtte og lite sannsynlig med mye endringer underveis
  - Men få systemer har stabile krav …
- Fossefallsmodellen brukes mest i store prosjekter som gjerne utvikles på ulike steder. Den plandrevne utviklingen gjør det enklere å koordinere arbeidet
- Men brukes også i små, godt forståtte prosjekter

Inkrementer og iterasjoner i systemutvikling

- Et *inkrement* er et tillegg i funksjonaliteten som er implementert i et system. Et inkrement er således et aspekt ved *systemet*
- En *iterasjon* er en syklus i utviklingen og er således et aspekt ved *prosessen*
  - Et nytt inkrement utvikles gjennom en ny iterasjon
  - En ny iterasjon kan også forbedre kvaliteten på samme funksjonalitet, dvs. man lager ikke noe nytt inkrement, men bare forbedrer det eksisterende systemet

Inkrementell utvikling

- Systemet utvikles gradvis i form av nye inkrementer som blir lagt til. Hvert inkrement evalueres før utviklingen av neste inkrement starter
- Vanlig tilnærming i smidige metoder
- Evalueringen gjøres av en bruker eller kunderepresentant (“product owner”)

Inkrementell installering

- Istedenfor at hele systemet leveres til kunden på en gang, leveres ett inkrement av gangen som tilsvarer deler av den totale funksjonaliteten
- Brukerkravene prioriteres slik at de viktigste kravene er implementert i de første inkrementene
- Når utviklingen av et inkrement er startet, så fryses kravene til det inkrementet, men kravene til senere inkrementer kan fortsatt endres

Fordeler ved inkrementell utvikling og installering

- Kostnadene ved endrede brukerkrav reduseres sammenlignet med fossefallsmodellen da delene som må endres, er mindre
- Enklere å få tilbakemeldinger fra kunden på det som har blitt utviklet
- Lettere å se hvor mye som er utviklet så langt
- Raskere levering av deler av systemet gir verdi for kunden raskere enn ved fossefallsmodellen
- Den prioriterte funksjonaliteten blir testet mest
- Lavere risiko for total prosjektfiasko

Utfordringer ved inkrementell utvikling og installering

- Store prosjekter/systemer krever en relativt stabil arkitektur som inkrementene og teamene må forholde seg til, dvs. arkitekturen kan ikke utvikles i inkrementer
- Strukturen til systemet har en tendens til å bli stadig verre etter hvert som inkrement legges til
- Derfor stadig vanskeligere å foreta endringer, med mindre ressurser brukes på re-faktorering (re-strukturering)

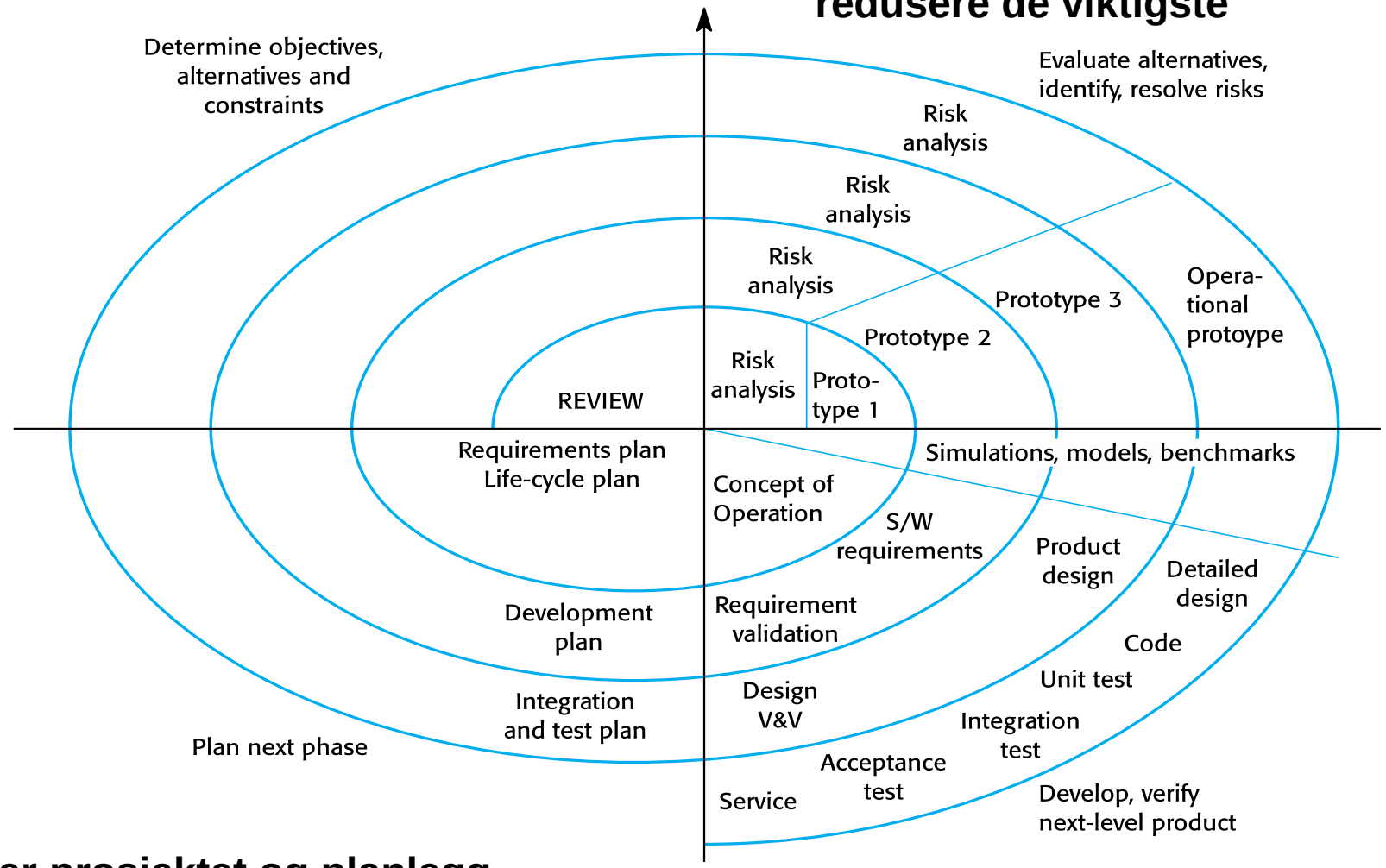
Spiralmodellen

- Utviklingsprosessen er representert som en spiral istedenfor en sekvens med aktiviteter der man evt. går tilbake til tidligere aktiviteter
- Hver runde i spiralen representerer en fase i prosessen, f.eks. kravspesifisering eller design
- Løkkene i spiralen velges etter behov
- Risikoanalyse: hva som kan gå galt, og med hvilken sannsynlighet og konsekvens, er vurdert og håndtert eksplisitt gjennom prosessen

Boehm's spiral model of the software process

Identifiser spesifikke mål for fasen

Analyser risiko og utfør aktiviteter for å redusere de viktigste



Evaluer prosjektet og planlegg neste fase i spiralen

Design, koding (programmering) etc.

Bruk av spiralmodellen

- Modellen er en av de mest kjente, klassiske modeller og har hatt stor betydning i utviklingen av tankegangen rundt iterasjoner og risikovurderinger i systemutviklingsprosessen
- Men brukes sjelden i konkret systemutvikling

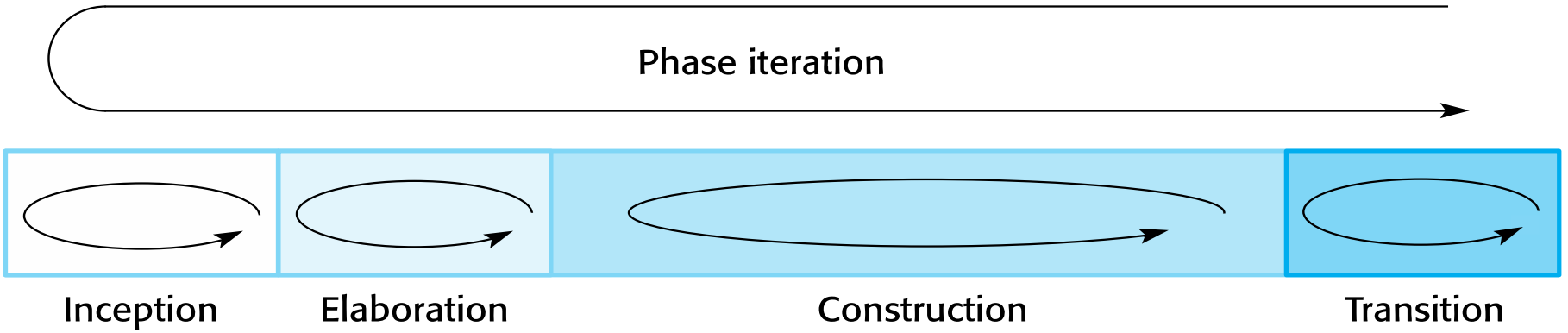
Rational Unified Process (RUP)

- RUP – prosessmodell som legger vekt på å bygge arkitektur/UML-modeller
- RUP er ikke en konkret prosessmodell, men mer et rammeverk som programvarebedrifter eller team kan ta utgangspunkt i for å skreddersy en modell for sin utvikling
- Benytter seg av prinsipper fra prosessmodellene beskrevet tidligere i forelesningen
- Vanligvis beskrevet med fokus på faser, disipliner (aktiviteter) og anbefalt god praksis

Fire faser i RUP

- Innledning/idé (lag business case) (*inception*)
  - Lag overordnet målsetting, behovsanalyse, budsjett, prosjektplan
  - Identifisere funksjonelle krav og modellere use cases (brukstilfeller)
- Utdypning (*elaboration*)
  - Fortsett med å forstå problemområdet, lag use cases
  - Start design av arkitektur, lag arkitektur prototype
  - Ferdigstill prosjektplanen
- Konstruksjon (*construction*)
  - Design-programmer-test, typisk i flere iterasjoner
- Installering/driftssetting (*transition*)
  - Overfør systemet til sitt produksjonsmiljø og sett det i drift, gi nødvendig opplæring til sluttbrukerne og vedlikeholdere, valider systemet i forhold til kvalitetskrav spesifisert i innledningen etc.

Iterasjoner i RUP-fasene



- Hver fase er iterativ med resultater som utvikles i inkrementer
- De fire fasene sett under ett vil også kunne inngå i iterasjoner

Anbefalte praksiser i RUP

- Utvikle systemet i iterasjoner
  - I hver iterasjon, legg til et nytt inkrement. Først lag de inkrementene som kunden har prioritert høyest
- Sørg for god håndtering av krav
  - Dokumenter kundekrav nøye og sørg for dokumentasjon av endringer i kravene
- Bruk komponent-basert arkitektur
  - Organiser systemets arkitektur som en mengde gjenbrukbare komponenter
- Lag visuelle modeller av programvaren
  - Bruk grafiske UML-modeller for å presentere statiske og dynamiske sider ved systemet
- Verifiser kvaliteten
  - Sjekk at programvaren tilfredsstiller organisasjonens kvalitetsstandarder
- Kontroller endringer i programvaren
  - Bruk endringshåndteringsverktøy og konfigurasjonsstyringsverktøy

Systemutvikling med gjenbruk

- Eksisterende programvare gjenbrukes i større eller mindre grad utviklingen av nye systemer
- Komponentbasert utvikling
  - Samling av komponenter i en pakke som del av komponentrammeverk som .NET eller J2EE eller andre typer komponent-biblioteker
  - Selvstendige software-systemer som er utviklet for bruk i et spesielt miljø (COTS) (Commercial-off-the-shelf)
- Service-orientert (tjenesteorientert) utvikling
  - Web-services som er utviklet i henhold til en standard og som kan kalles fra andre steder

Service-orientert arkitektur (SOA)

- Brukes for å utvikle distribuerte systemer der komponentene er selvstendige tjenester
- Tjenestene vil kunne utføres på ulike maskiner fra ulike tjenesteleverandører
- Standard protokoller har blitt utviklet for å støtte kommunikasjon og utveksling av informasjon

Endring (evolusjon) av programvare

- Programvare er fleksibel og derfor tilsynelatende enkel å endre
- Etter hvert som virksomheten som datasystemet skal støtte, endrer seg, så må kravene til systemet endre seg tilsvarende for å opprettholde verdi hos kunden/brukere.
- Tidligere var det et tydelig skille mellom nyutvikling og vedlikehold/videreutvikling, men nå som som stadig færre systemer er helt nye, blir dette skillet stadig mindre relevant

Ulike prosesser i ulike kontekster

- Sommerville skriver:
  “There are no right or wrong software processes”
- Selv om systemutvikling ikke et eksakt fagfelt, er det opplagt at noen prosesser er bedre enn andre avhengig av hva slags system som skal utvikles og i hvilken kontekst det skal foregå
- Men fagfeltet mangler fortsatt sikker kunnskap om hvordan ulike prosesser fungerer i ulike situasjoner

Behov for smidighet

- Den klassiske ingeniørtilnærmingen med fokus på plan-legging og dokumenter har vist seg ofte å ikke være egnet
- Alternativ: “smidig systemutvikling”
  - **Personer og samspill** fremfor prosesser og verktøy
  - **Programvare som virker** fremfor omfattende dokumentasjon
  - **Samarbeid med kunden** fremfor kontraktsforhandlinger
  - **Å reagere på endringer** fremfor å følge en plan
    - Selv om punktene som står til høyre har verdi,så verdsettes punktene til venstre enda høyere.

Plandrevne (tunge) prosesser

- Prosessaktivitetene planlagt på forhånd og progresjon måles i henhold til denne planen
- En tung prosess inkluderer mange aktiviteter og roller, og krever formelle, detaljerte og konsistente prosjektdokumenter
- Tunge prosesser er ofte “for-tunge”, dvs. vektlegger aktiviteter som vanligvis gjøres tidlig i prosessen (planlegging, analyse & design)

Smidige (lette) prosesser

- Planleggingen gjøres litt etter litt (inkrementelt)
- Enklere å endre prosessen for å tilpasse endrede krav fra kunden
- Lette prosesser fokuserer mer på fundamentale prosesser (f.eks. ”kontinuerlig testing”), har færre formelle dokumenter og er ofte mer iterative

En samling software-guruer i 2001: Agile Manifesto – 12 prinsipper*

1	Kundefokus	Prioriter å tilfredsstille kunden gjennom tidlige og kontinuerlige leveranser av programvare med verdi
2	Positiv til endringer	Ønsk endringer i krav velkommen, selv sent i utviklingen. Bruk endringer til å skape konkurransefortrinn for kunden
3	Lever inkrementer hyppig	Lever fungerende programvare med et par ukers til et par måneders mellomrom. Jo oftere, jo bedre
4	Kunde-involvering	Forretningssiden og utviklerne må arbeide sammen daglig gjennom hele prosjektet
5	Stol på den enkelte	Bygg prosjektet rundt motiverte personer. Gi dem og miljøet støtten de trenger. Stol på at de får jobben gjort
6	Ansikt-til-ansikt kommunikasjon	Mest effektivt å formidle informasjon til og innad i et utviklingsteam ved å snakke ansikt til ansikt

*http://agilemanifesto.org og http://agilemanifesto.org/iso/no/principles.html

Agile Manifesto 2001 (forts.)

7	Kode er hovedproduktet	Fungerende programvare er det primære målet på fremdrift
8	Stabile omgivelser	Smidighet fremmer bærekraftig programvare-utvikling. Sponsorene, utviklerne og brukerne bør kunne opprettholde et jevnt tempo hele tiden
9	Teknisk kvalitet	Kontinuerlig fokus på fremragende teknisk kvalitet og godt design fremmer smidighet
10	Enkelhet	Enkelhet – kunsten å maksimere mengden arbeid som ikke blir gjort – er essensielt
11	Selvstyrte team	De beste arkitekturer, krav og design vokser frem fra selvstyrte team
12	Kontinuerlig prosessforbedring	Med jevne mellomrom reflekterer teamet over hvordan det kan bli mer effektivt og så justerer det adferden sin deretter

Smidig - 4 overgripende utsagn

1. **Personer og samspill** fremfor prosesser og verktøy
2. **Programvare som virker** fremfor omfattende dokumentasjon
3. **Samarbeid med kunden** fremfor kontraktsforhandlinger
4. **Å reagere på endringer** fremfor å følge en plan

“Ekstrem programmering” (XP)

- Ekstrem ved at:
  - Hele systemet kan bygges (rekompileres) opp til flere ganger daglig
  - Inkrementer av systemet leveres til kunden annenhver uke
  - Alle tester må kjøres før hver bygging*, og byggingen aksepteres bare hvis testene er vellykkede

***bygging: Alle komponentene til systemet kompileres og linkes til hverandre og til data og biblioteker som er nødvendige for å lage et kjørbart system**

Praksiser i XP

Praksis	Beskrivelse
Inkrementell planlegging	Kravene skrives på ”historiekort” og hvilke **brukerhistorier** som skal inkluderes i en release blir bestemt ut fra prioritet og tilgjengelig tid. En brukerhistorie vil typisk tilsvare en utviklingsoppgave
Små releaser	Lag først det minste settet av funksjonalitet som gir verdi for kunden. Lever hyppig nye releaser med nye inkrementer med funksjonalitet
Enkelt design	Bare design så mye som strengt tatt nødvendig
Test-først utvikling	Bruk et automatisk testrammeverk til å skrive tester for ny funksjonalitet FØR funksjonaliteten selv implementeres
Refaktorering	Forbedre koden kontinuerlig når muligheter for forbedring oppdages
Parprogrammering	Programmer i par, sjekk hverandres koder og kom med innspill
Kollektivt eierskap	Alle par skal jobbe på alle deler av koden – ikke ha lokale eksperter på deler av koden. Alle tar ansvar for all kode og alle kan endre overalt
Kontinuerlig integrasjon	Umiddelbart etter at en oppgave er ferdig, må den tilhørende koden integreres i hele systemet. Alle enhetstester må kjøres
Holdbart tempo	Ikke jobb mye overtid fordi konsekvensen er dårligere kode og lavere produktivitet i lengden
Kunde på stedet	En representant for sluttbrukeren eller kunden bør være tilgjengelig for utviklingstemaet hele tiden. Representanten er ansvarlig for at brukerkravene blir overlevert utviklerne for implementering

Brukerhistorie (user story)

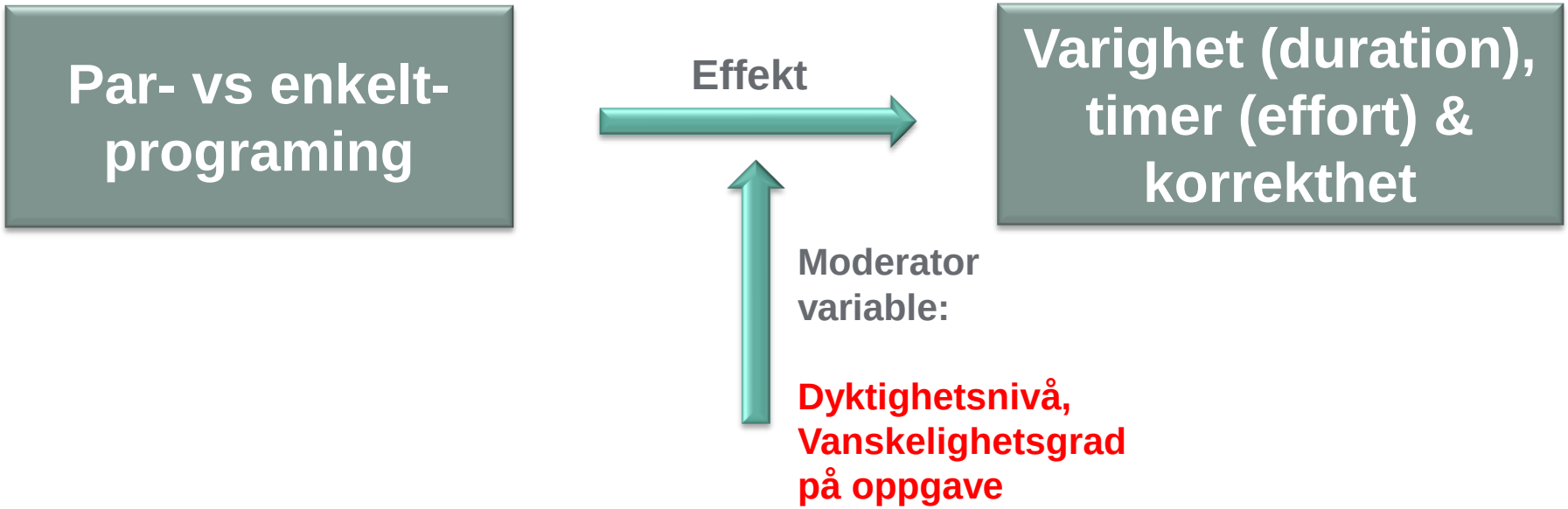
- Én eller flere setninger som beskriver hva brukeren av et system ønsker å få ut av systemet på formen:
  - ”Som en <rolle> ønsker jeg <funksjon> for å oppnå <verdi>”
    - Som bruker i nettbanken ønsker jeg å legge inn faste betalingsmottakere for at det skal bli enklere å betale regninger
- Beskrivelsen skal være kort slik at den passer på et kort eller gul lapp

Parprogramming
- kjent praksis fra XP
- kan brukes uavhengig av smidig

To programmerere utvikler kode sammen:

- Fører
  - skriver på tastaturet
- Navigatør
  - observerer arbeidet til føreren og ser etter feil og svakheter
  - ser etter alternativer
  - noterer ting som må gjøres
  - slår opp referanser

Eksperiment



Deltakere og oppgaver

- 295 junior, mellomnivå og senior Java konsulenter fra 29 firmaer ble betalt en arbeidsdag for å delta
- 99 jobbet enkeltvis
- 98 i par (Norge: 41, Sverige: 28, Storbritannia: 29)
- Enkeltindividene og parene utførte de samme endringsoppgavene i Java på
  - Et “enkelt” system med “sentralisert kontrollstil” og
  - Et “komplekst” system med “delegert kontrollstil”

Effekten av å bruke parprogrammering "kommer an på"

Programmerings-erfaring	Vanskelighetsgrad	Bruke PP?	Kommentarer
Junior	Enkel	Ja	Forutsatt at økt kvalitet er hovedmålet
	Kompleks	Ja	Forutsatt at økt kvalitet er hovedmålet
Middels	Enkel	Nei	
	Kompleks	Ja	Forutsatt at økt kvalitet er hovedmålet
Senior	Enkell	Nei	
	Kompleks	Nei*	

*** Med mindre man vet at oppgaven er for vanskelig til å bli løst av en senior alene**

Prosessfokuserte ”smidige” metoder

- To hovedretninger
  1. Velg noen prioriterte oppgaver og jobb med dem i faste tidsintervaller med definerte oppstart- og avslutningsaktiviteter (metoden Scrum)
  2. En ”lean”-basert retning fokuserer på at oppgaver skal ”flyte” uten avbrudd gjennom de nødvendige aktivitetene til de er ferdige. Ingen aktiviteter som er ”waste” skal utføres (metoden Kanban)