# Prosessmodeller og smidig programvareutvikling

# Plan

- Begrepet ”prosessmodell”
- Prosessmodeller og prinsipper for utvikling Fossefallsmodellen
  - Inkrementell og iterativ utvikling
  - Prototyping
  - Spiralmodellen
  - Rational Unified Process (RUP)
  - Gjenbruksbasert utvikling
- Begreper og prinsipper innen smidig utvikling
- Programmeringsfokuserte smidige metoder
  - Ekstrem programmering (XP)
- Prosessfokuserte smidige metoder
  - Tidsboksbasert (Scrum)
  - Flytbasert (Kanban)
- Lean systemutvikling
- Oppsummering smidige metoder

# Reell prosess versus modell prosess

- Systemutviklingsprosess (= faktisk, reell prosess):
  - de aktivitetene som utføres i et utviklingsprosjekt
- Prosessmodell (=formell prosess) (kalles også gjennomføringsmodell)
  - En abstrakt, forenklet representasjon av en prosess
    - Deskriptiv beskriver en prosess slik vi mener vi utfører den
    - Normativ (preskriptiv) beskriver en prosess slik noen mener den *bør* være (vanligste betydning)

# Modell versus virkelighet

# Formell versus reell prosess

- **Prosessbeskrivelse**
  - **Det vi sier vi gjør eller det vi bør gjøre**

- **Prosessutførelse**
  - **Det vi gjør**

# Nivåer av prosessmodeller

- **Definerte prosess- modeller (formell prosess)**
  - **Generelle prosessmodeller (fossefall, spiral, RUP, Scrum etc.)**
  - **Firma-spesifikke prosessmodeller**
  - **Prosjekt/gruppe-spesifikke prosessmodeller**

Prosess-samsvar

- **Reell prosess**
  - **Systemutviklingsprosess**

# Hvordan tilpasse prosesser?

- Prosesser må tilpasses
  - ingen prosjekter er like
  - Mange faktorer påvirker prosessen
- Hva kan tilpasses?
  - Antall faser/aktiviteter, roller, ansvarsforhold, dokumentformater, formalitet/frekvens på rapporter og gjennomganger
- Hvordan tilpasse?
  1. Identifiser prosjektomgivelser – utviklingsstrategi, risiko, krav, applikasjonsområde, type kunde etc.
  2. Innhent synspunkter fra utviklere, brukere, kunder
  3. Definer prosesser, aktiviteter og roller
  4. Dokumenter og begrunn tilpasningene

# Felles prosjektmodell i offentlig virksomheter

- For å sikre kvalitet!
- Er det lurt?
- Ulempe
  - Sjelden at samme modell passer for alle type virksomheter
- Fordel
  - Læring på tvers av etater

- Se artikkel i Aftenposten: http://www.aftenposten.no/digital/nyheter/Haper-klare-rad-skal-fa-fart-pa-digitaliseringen-7073514.html

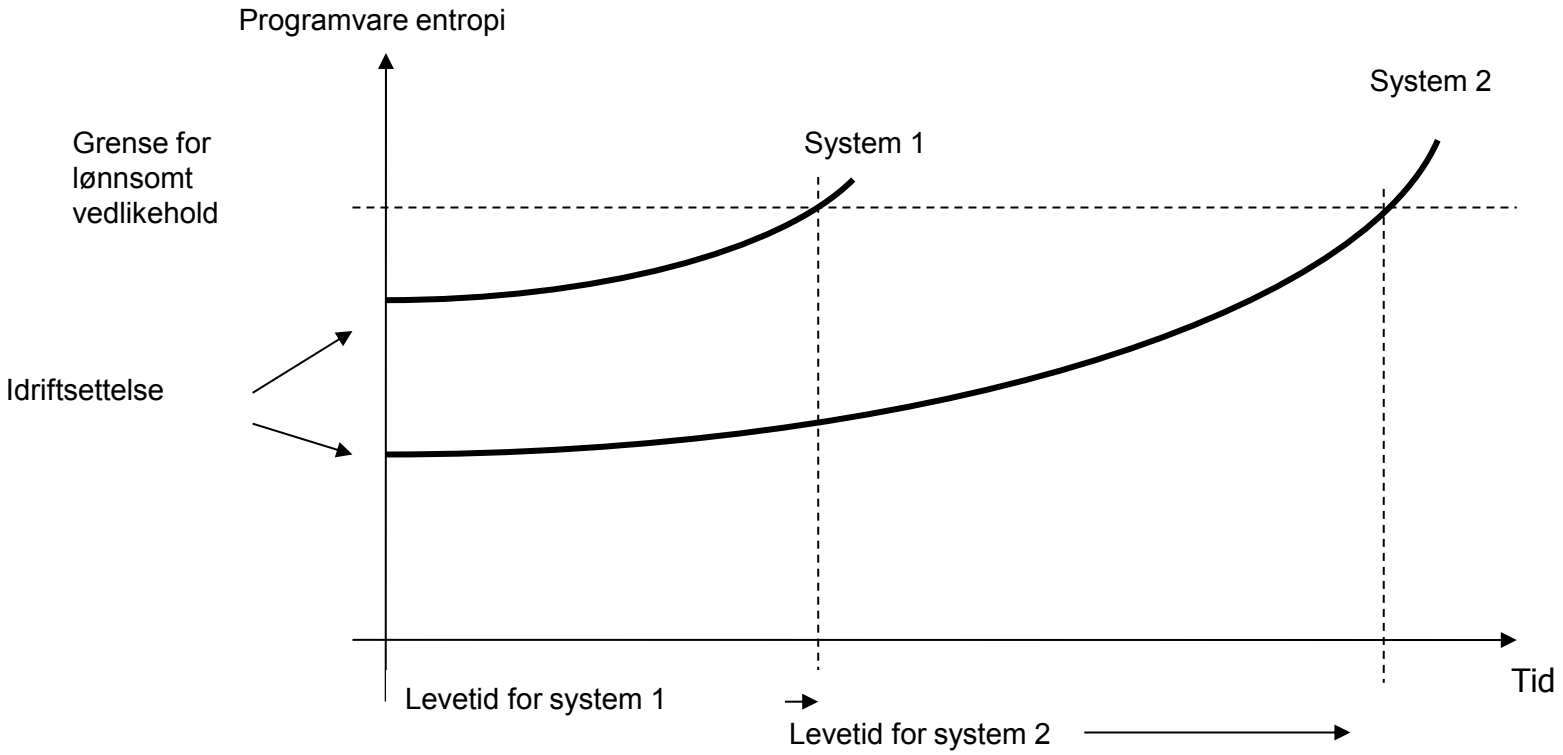
# Symptomer og hovedgrunner til programvare utviklings problem:

- Unøyaktig forståelse av brukernes behov
- Håndterer ikke endrede krav
- Moduler som ikke passer sammen
- Programvare som er vanskelig å vedlikeholde og utøke
- Oppdager sent viktige mangler i prosjektet
- Lav programvare-kvalitet
- Uakseptabel ytelse av programvaresystemet
- Prosjektmedlemmer arbeider ustrukturert
- Upålitelig programmering/testing og release av systemet

# Grunner til utviklingsproblemene:

- Dårlig og tilfeldig kravformulering
- Upresis kommunikasjon
- Svak arkitektur
- For høy kompleksitet
- Inkonsistens i krav, konstruksjon, implementering
- Ufullstendig testing
- Subjektiv vurdering av prosjektstatus
- Manglende risiko håndtering
- Ukontrollert endringshåndtering (konfigurasjonsh.)
- Utilstrekkelig bruk av verktøy

# Entropi-begrepet




Vi ser at et system som har høy entropi (kaos) ved idriftsettelse har kortere levetid enn et som har lav grad av entropi.

# Hva er en systemutviklingsprosess?

- Beskriver hvordan programvare skal produseres ved å angi mekanismer for å styre, kontrollere og organisere arbeidet.
- Kunnskap om systemutviklingsprosessen er sentral i arbeidet med å forbedre produktivitet og kvalitet i systemutvikling
- Merk at systemutviklingsprosess, utviklingsprosess og utviklingsprosessmodell betyr det samme

# Hvorfor er utviklingsprosesser viktig?

- Hvordan arbeidet utføres har stor påvirkning på produktivitet og kvalitet
- Noen prosesser er bedre egnet enn andre
- Prosesser innfører begreper og felles begrepsforståelse
- Felles begrepsforståelse er nødvendig for godt samarbeid
- Felles begrepsforståelse er nødvendig for standardisering
- Standardisering er nødvendig for forbedring av prosessen
- Feil prosess kan ha fatale følger for virksomheten!
- **Merk at standardisering her betyr innenfor en virksomhet. Det er ikke det samme som en offisiell standard som f. eks. ISO 9000**

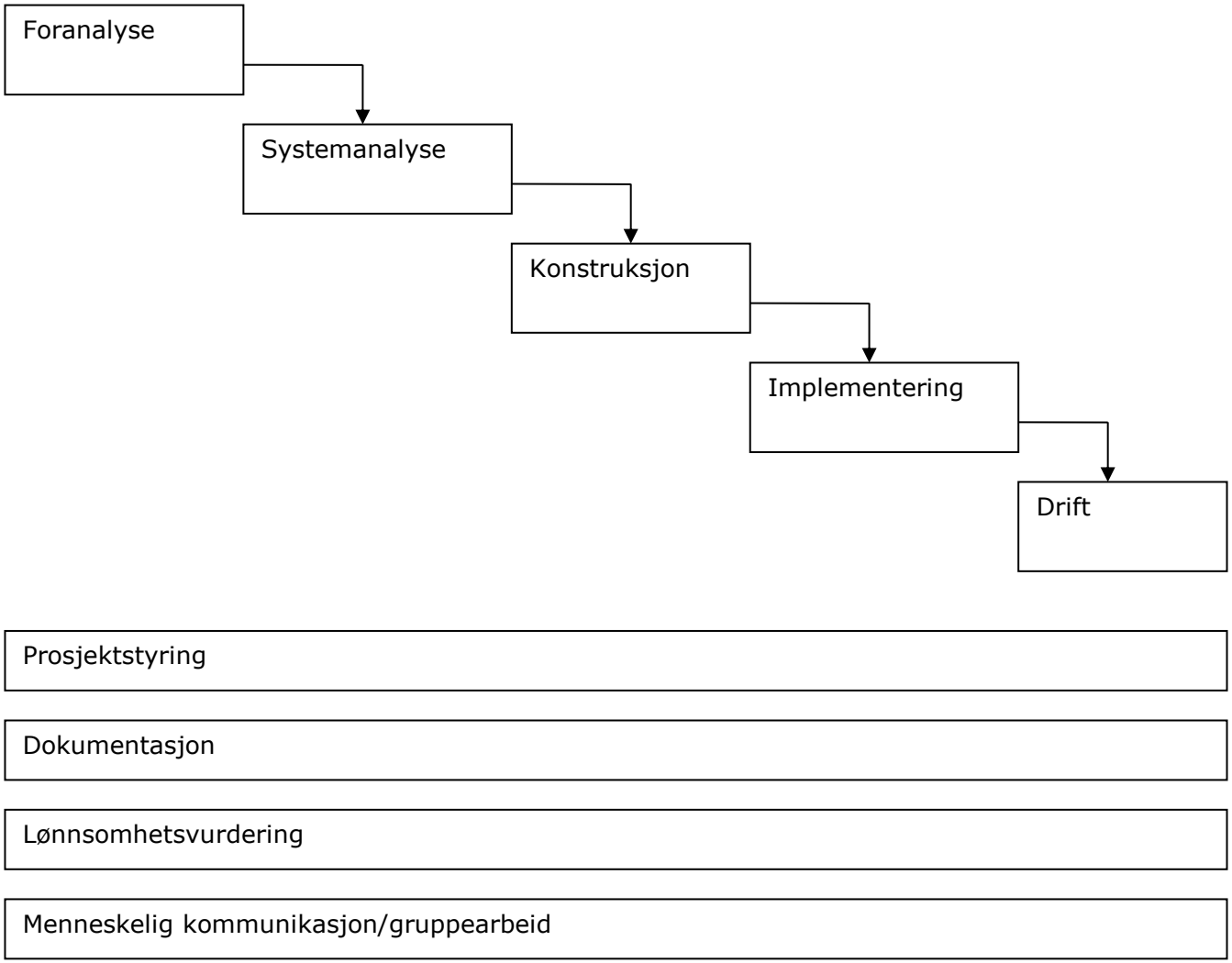
# Prosessbeskrivelse

- En prosess beskrives av *hvem* (utviklere) som gjør *hva* (artefakter), *hvordan* (aktiviteter) og *når* (arbeidsflyt).
- Andre prosess elementer:
  - Gi retningslinjer
  - Maler/mønstre
  - Verktøy hjelp
  - Konsepter for overvåkning/måling av fremdrift

# Plan

- Begrepet ”prosessmodell”
- Prosessmodeller og prinsipper for utvikling
  - Fossefallsmodellen
  - Prototyping
  - Inkrementell og iterativ utvikling
  - Spiralmodellen
  - Rational Unified Process (RUP)
  - Gjenbruksbasert utvikling
- Begreper og prinsipper innen smidig utvikling
- Programmeringsfokuserte smidige metoder
  - Ekstrem programmering (XP)
- Prosessfokuserte smidige metoder
  - Tidsboksbasert (Scrum)
  - Flytbasert (Kanban)
- Lean systemutvikling
- Oppsummering smidige metoder

# Fossefallsmodellen med gjennomgående aktiviteteter



# Hovedprinsippet i fossefallsmodellen

- Utvikling er en forutsigbar produksjonsprosess
- En pålitelig og detaljert plan kan etableres ved oppstart
- Kravene kvalitetssikres ved at de dokumenteres og gjennomgås før programmeringen starter
- Hver fase avsluttes før neste fase kan begynne
- Endringer i planen skal normalt ikke skje
- Programvaren antas å bli korrekt utviklet i første forsøk
- Systemet kan ikke utprøves før det er helt ferdig
- En repitisjon av prosessen vil levere samme resultat

# Fordeler med fossefallsmodellen

- En av de første forsøk på å standardisere systemutvikling (DoD Military Standard 2167)
  - (DoD = Department of Defense, USA)
- Påtvinger disiplin med tydelig start og stopp i hver fase
- Konseptuelt enkel, enkel å forstå og kontrollere for ledere, enkel å undervise
- Alle krav og design gjøres før programmering.
- Sparer mye kostnader hvis feil oppdages på dette stadiet

# Problemer med fossefallsmodellen

- Er det mulig å forstå hvordan et IT-system vil fungere ved å lese fra hundre til flere tusen sider med dokumenter?
- Er det først når vi sitter foran en datamaskin og prøver et system at vi oppdager feilene?
- Hvordan kan vi planlegge en testfase med en gitt slutt dato uten at vi vet noe om feilraten i systemet?

# Problemer med fossefallsmodellen

- Adekvate krav kan ofte ikke defineres på forhånd
  - Brukerne er ikke alltid sikre på hva de behøver
    - «Jeg vet hva jeg behøver når jeg ser det»
  - Endringer i eksterne forutsetninger er ikke forutsigbare
- Støtter ikke endring av krav underveis
  - Brukerne endrer oppfatning underveis i prosessen
  - Støtter ikke tilpassning til endrede eksterne forutsetninger
- Evaluering og test utføres til slutt
  - Feil og mangler oppdages for sent (dette gir høye kostnader)
- Vi har behov for bedre modeller som støtter evaluering og endring mye tidligere i utviklingen

# Utviklingsdiskusjon

- Fossefallsmodellen kan virke dårlig, fordi:
  - Feil forutsetning 1: ”Krav vil bli frosset!”
  - Feil forutsetning 2: ”Konstruksjonen kan gjøres fullstendig og korrekt før vi går videre.”
- Risikoplanlegging må gjøres
- Tidsperiodens størrelse er viktig
- Oppsamling av dokumenter for neste fase
- Planlegg ut i fra omfang for å få korrekt tid
- Iterer! Styr vha faser og mange milepeler

# Skifte av fokus i de ulike faser

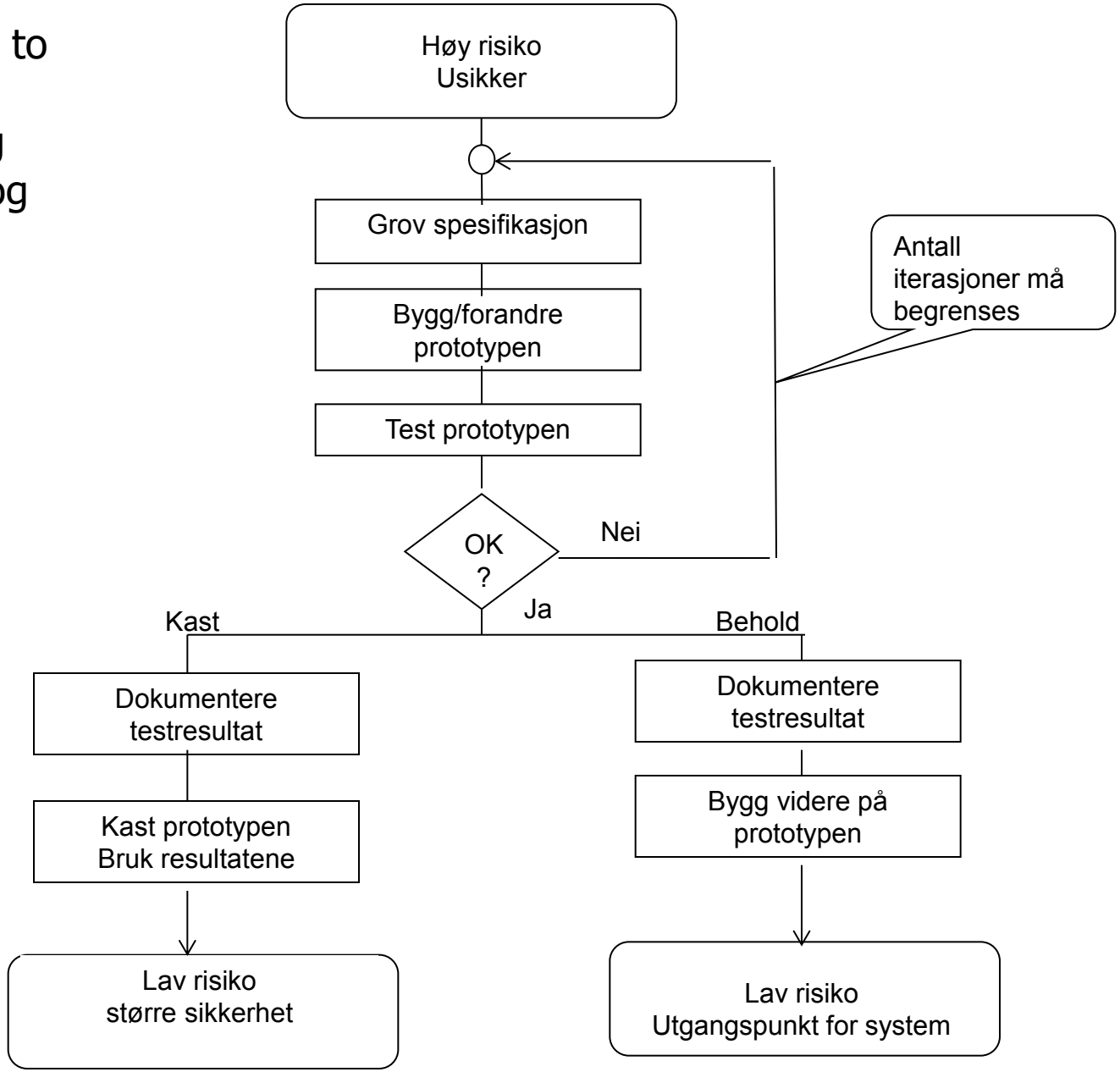
- I Foranalysefasen er fokus på forståelse av krav, omfang og lønnsomhet.
- I Systemanalysefasen er fokus på planlegging, krav, arkitektur, risiko, prototyping.
- I Konstruksjons- og implementerings-fasen er fokus på detaljkonstruksjon, implementering (programmering) og testing.
- I Overgangsfasen (drift) er fokus på kvalitetskontroll, opplæring av brukere, tilpassing av organisasjon

# Prototyping

- En prototype er en initiell versjon hvis formål er å demonstrere konsepter, utforske designvalg, og evaluere forståelsen av identifiserte krav. Formålet er å sikre at det riktig systemet utvikles.
- Introdusert for å avhjelpe problemene med fossefallsmodellen
- En mer strukturert utgave av prøv-og-feil
  - Tilbyr flere varianter:
- Bruk-og-kast prototyping
- Evolusjonær prototyping

**Prototyping**

Modellen viser to strategier for prototyping og tilstander før og etter :



# Fordeler med prototyping

- Gir en visuell og tidlig presentasjon av et tenkt slutt resultat
  - Husk: «Jeg vet hva jeg behøver når jeg ser det»
- Forbedrer forståelsen av behov og løsningskonsept

# Ulemper med prototyping

- Krav til hurtighet fører til kompromisser på kvalitet
- Lite fokus på arkitektur og andre tekniske kvaliteter
  - Lite fokus på vedlikehold
  - Interessenter betrakter en kjørende prototype som ferdig system
- **Prototyping benyttes idag mer som en teknikk for å studere krav og løsningsforslag enn som en fullstendig utviklingsmodell**

# Evolusjonære modeller

- Del prosjektet opp i mindre selvstendige mini-prosjekter som kalles *iterasjoner*. Hver iterasjon må levere en fungerende del av systemet. Dette kalles et *inkrement*. Formålet er å kontrollere at:
  - Mange små iterasjoner med leveranser og evaluering leder prosjektet i riktig retning. Dersom noe er galt oppdages dette tidlig
- Prosjektgruppen forstår interessentenes behov
- Interessentene bekrefter at prosjektgruppen forstår interessentenes behov
- Teknologi, verktøy, og metoder virker som forventet

# Iterativ og inkrementell utvikling

- En *iterativ* utviklingsprosess er en utviklingsprosess som består av flere mindre sekvensielt ordnede miniprosjekter som kalles iterasjoner
- Hver iterasjon er et selvstendig mini-prosjekt som består av kravinnsamling, design, implementering, og test
- Målsettingen med hver iterasjon er å levere en fungerende, stabil, integrert del av det totale systemet

# Iterativ og inkrementell utvikling

- Evolusjonær utvikling introduserer prinsippet om adaptiv endring i form av evaluering og justering av planen
- Prinsipper:
  - Ingen fullstendig kravspesifikasjon skrives ved oppstart
  - Regelmessige leveranser (inkrementer) til interessentene
  - Utviklingen foregår stegvis med nye inkrementer
  - Eksplisitte endring og bearbeiding av tidligere resultater er innebygget i modellen
  - Regelmessig endring av planene basert på evalueringer utført av interessentene

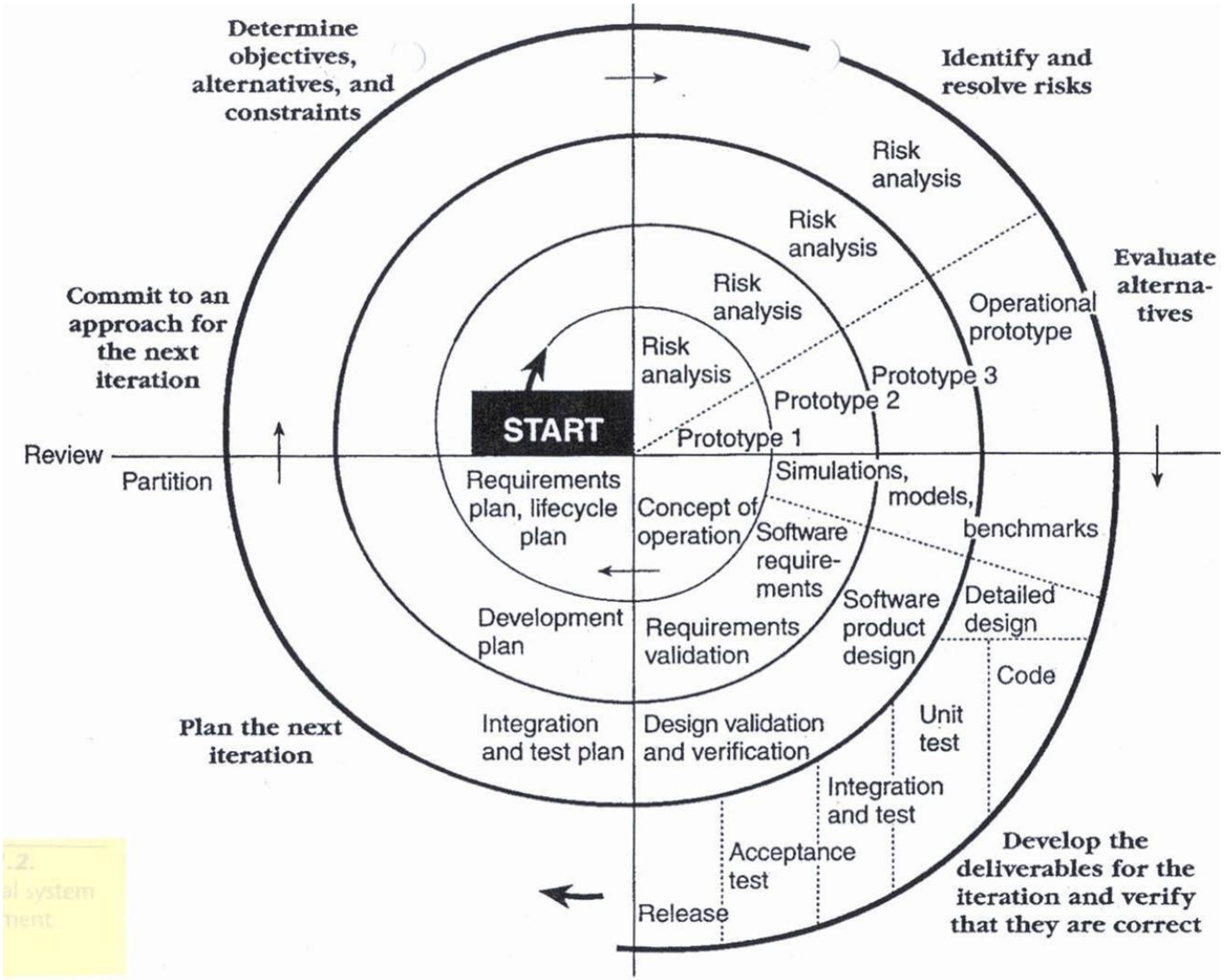
# Fordeler ved inkrementell utvikling og installering

- 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 og systemer
  - krever en relativt stabil arkitektur som inkrementene og teamene må forholde seg til
    - dvs. arkitekturen kan ikke utvikles i inkrementer
- Strukturen til systemet blir dårligere etter hvert som inkrementer legges til
  - vanskeligere å foreta endringer hvis ikke ressurser brukes på re-strukturering (re-faktorering)

# Spiralmodellen (referansemodell)




Source: Adapted from "A Spiral Model of Software Development and Enhancement" (Boehm 1988).

# Rational Unified Process (RUP)

- et rammeverk for å bygge arkitektur/UML-modeller
  - Ikke en konkret prosessmodell
- ta utgangspunkt i RUP
  - for å skreddersy en modell for sin utvikling
- beskrevet med fokus på faser, disipliner (aktiviteter) og anbefalt god praksis

# 4 iterative faser hvor resultater utvikles i inkrementer

- 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.

# 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

# Gjenbruksbasert utvikling

- 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 programvare-systemer som er utviklet for bruk i et spesielt miljø
- 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)

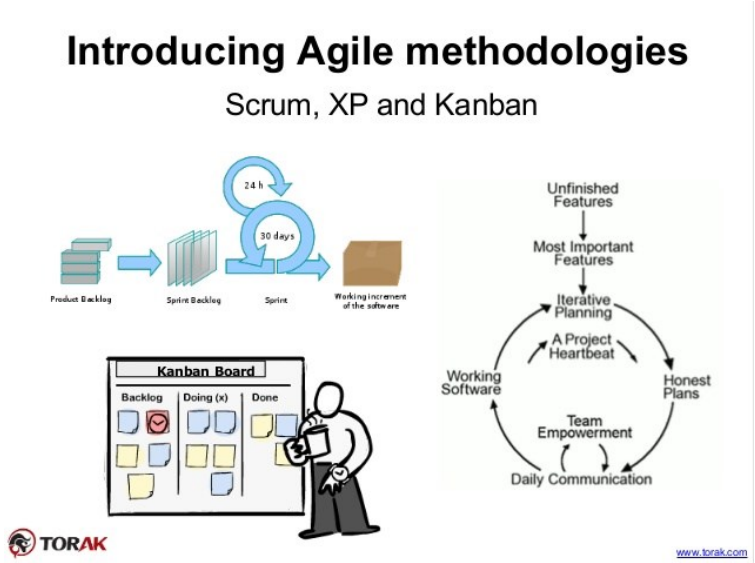
# Hvilken prosess er best?

- Sommerville:
  - "There are no right or wrong software processes"
- Ikke eksakt fagfelt
  - mangler sikker kunnskap om hvordan ulike prosesser fungerer i ulike situasjoner
- opplagt at noen prosesser er bedre enn andre
  - avhengig av hva slags system som skal utvikles
  - og i hvilken kontekst det skal foregå

# Plan

- Begrepet ”prosessmodell”
- Prosessmodeller og prinsipper
  - utvikling Fossefallsmodellen
  - Inkrementell og iterativ utvikling
  - Prototyping
  - Spiralmodellen
  - Rational Unified Process (RUP)
  - Gjenbruksbasert utvikling

- Begreper og prinsipper
  - smidig utvikling
- Programmeringsfokuserte
  - smidige metoder
  - Ekstrem programmering (XP)
- Prosessfokuserte smidige metoder
  - Tidsboksbasert (Scrum)
  - Flytbasert (Kanban)
- Lean systemutvikling
  -



# Behov for smidighet

- Den klassiske ingeniørtilnærmingen med fokus på plan-legging og dokumenter (jfr. fossefall) har vist seg ofte å ikke være egnet
- Derfor er “smidige metoder” blitt vanlige
  - Vektlegging av kode fremfor (omfattende) design og dokumentasjon
  - Vektlegging av samarbeid med kunde fremfor kontraktsforhandlinger
  - Raskere levering av kode og raske endringer tilpasset endrede brukerkrav
- **Agile Manifesto – 12 prinsipper**

# Plandrevet vs Smidig

- **Plandrevne (tunge) prosesser**
  - Prosessaktivitetene planlagt på forhånd. Progresjon måles i henhold til planen
  - En tung prosess inkluderer mange aktiviteter og ofte roller. Krever formelle, detaljerte og konsistente prosjektdokumenter
  - Ofte “for-tunge”, dvs. vektlegger aktiviteter som 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
  - Fokuserer mer på fundamentale prinsipper (f.eks. ”kontinuerlig testing”). Har færre formelle dokumenter og er ofte mer iterative

# “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*
  - Byggingen aksepteres bare hvis testene er vellykkede
- XP-prinsipper

# 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>”
- Kort beskrivelse, passer på et kort eller gul lapp

# Visualisering User stories

Noter en oppgave eller arbeidspakke på en gul lapp og sett den på en tavle

| Backlog (to do) | Analysis | | Development | | Test | | Release | |
|---|---|---|---|---|---|---|---|---|
| | In progress | Done | In progress | Done | In progress | Done | In progress | Done |
| | | | | | | | | |

# Refaktorering (omstrukturering)

- Se etter forbedringsmuligheter og implementer dem selv om ikke umiddelbart behov for dem
- Koden mer forståelig og enklere å endre
  - mindre behov for dokumentasjon
- Noen endringer krever at arkitekturen omstruktureres (kostbart)
- Eksempler på refaktorering
  - Reorganisering av klassehierarki for å fjerne duplisert kode
  - Forbedre navn på attributter og metoder
  - Erstatte kode med kall til metoder i et programbibliotek

# Parprogrammering - kan brukes uavhengig av smidig

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

# Prosessfokuserte metoder

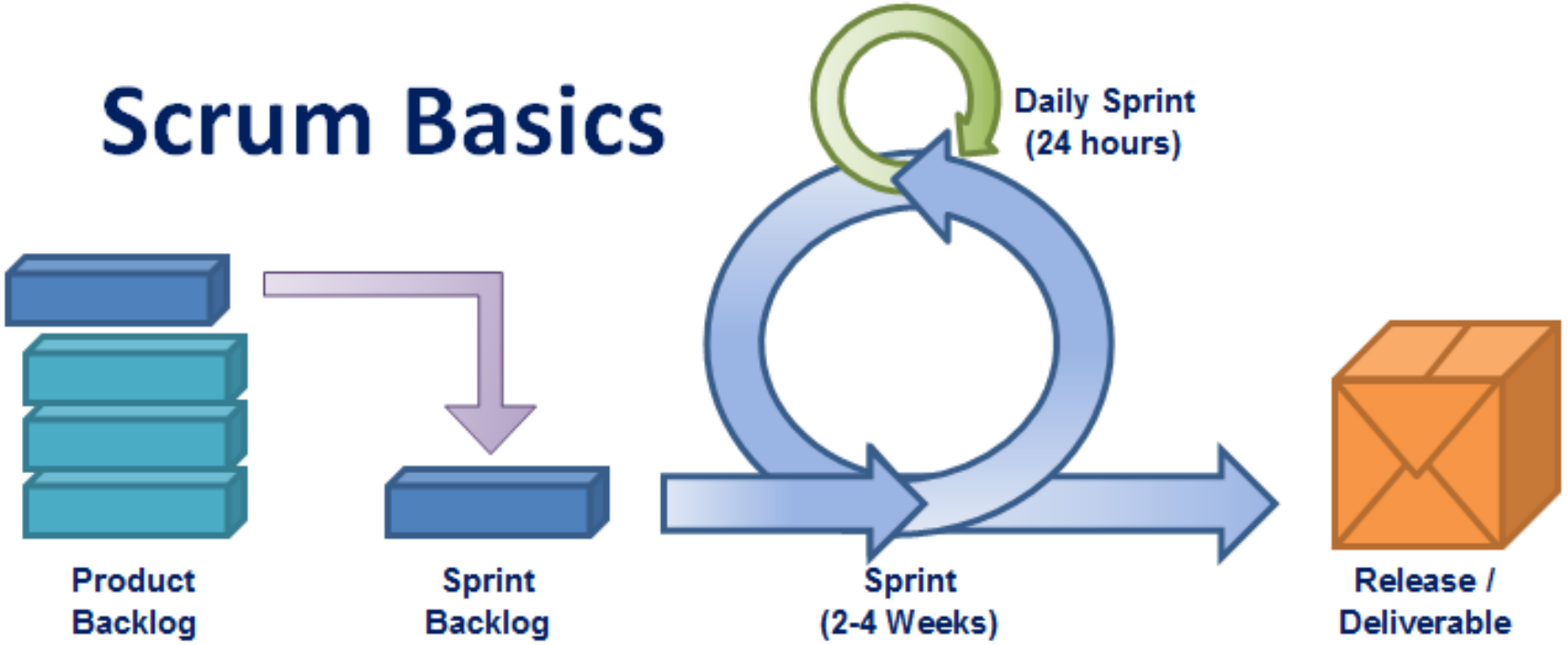
- Fokus på prosjektledelse av iterativ utvikling fremfor mer tekniske praksiser
- To hovedretninger
  - Velg noen prioriterte oppgaver og jobb med dem i faste tidsintervaller (tidsbokser*) med definerte oppstarts- og avslutningsaktiviteter (Scrum)
  - Fokuserer på at oppgaver skal ”flyte” uten avbrudd gjennom de nødvendige aktivitetene til de er ferdige (Kanban)

- *Tidsboks: en fast tidsperiode som et gitt arbeid skal være ferdig innenfor

# Scrum – tre faser

- **Planleggingsfasen**
  - overordnede mål for prosjektet etableres og programvarearkitekturen designes
- **Gjennomføringsfasen**
  - en serie med iterasjoner (kalt ”sprint”), der hver iterasjon leverer et inkrement av systemet
- **Avslutningsfasen**
  - nødvendig dokumentasjon som hjelp-funksjoner og brukermanualer fullføres, og man oppsummerer hva man har lært i prosjektet

# Scrum Basics




**Product Backlog**

Requirements are organized and prioritized in the product backlog.

**Sprint Backlog**

The highest priority requirements become the sprint backlog. The development work to satisfy the requirements become one sprint.

**Sprint (2-4 Weeks)**

Each sprint lasts between 2 and 4 weeks. The work done in a day is sometimes called the Daily Sprint. Each day there is a short Stand-Up Meeting to keep the Sprint on track.

**Release / Deliverable**

Each sprint results in a release or delivery of key functionality.

# (Antatte) fordeler ved Scrum

- Systemet blir delt opp i en mengde forståelige og håndterbare deler
- Ustabile krav hindrer ikke progresjon i prosjekt-gjennomføringen
- Hele teamet observerer hva som skjer i prosjektet, og kommunikasjon innen teamet blir god
- Kundene får inkrementer levert til avtalt tid og får fortløpende tilbakemelding på hvordan deler av systemet fungerer
- Tillit mellom kunder og utviklere etableres tidlig og en positiv kultur skapes
- Kryss-funksjonelle team (kompetansene på ulike områder finnes innen teamet) sikrer framdrift og reduserer risiko
- **Mer om Scrum i egen forelesningen**

## Prosessprinsipp
## Timeboxing versus “task-boxing” / “task-flyt”

- Scrum
  - Ikke alltid greit å dele inn oppgaver eller “features” av systemet tilpasset ”sprintene”
  - f.eks. vedlikehold, videreutvikling og support

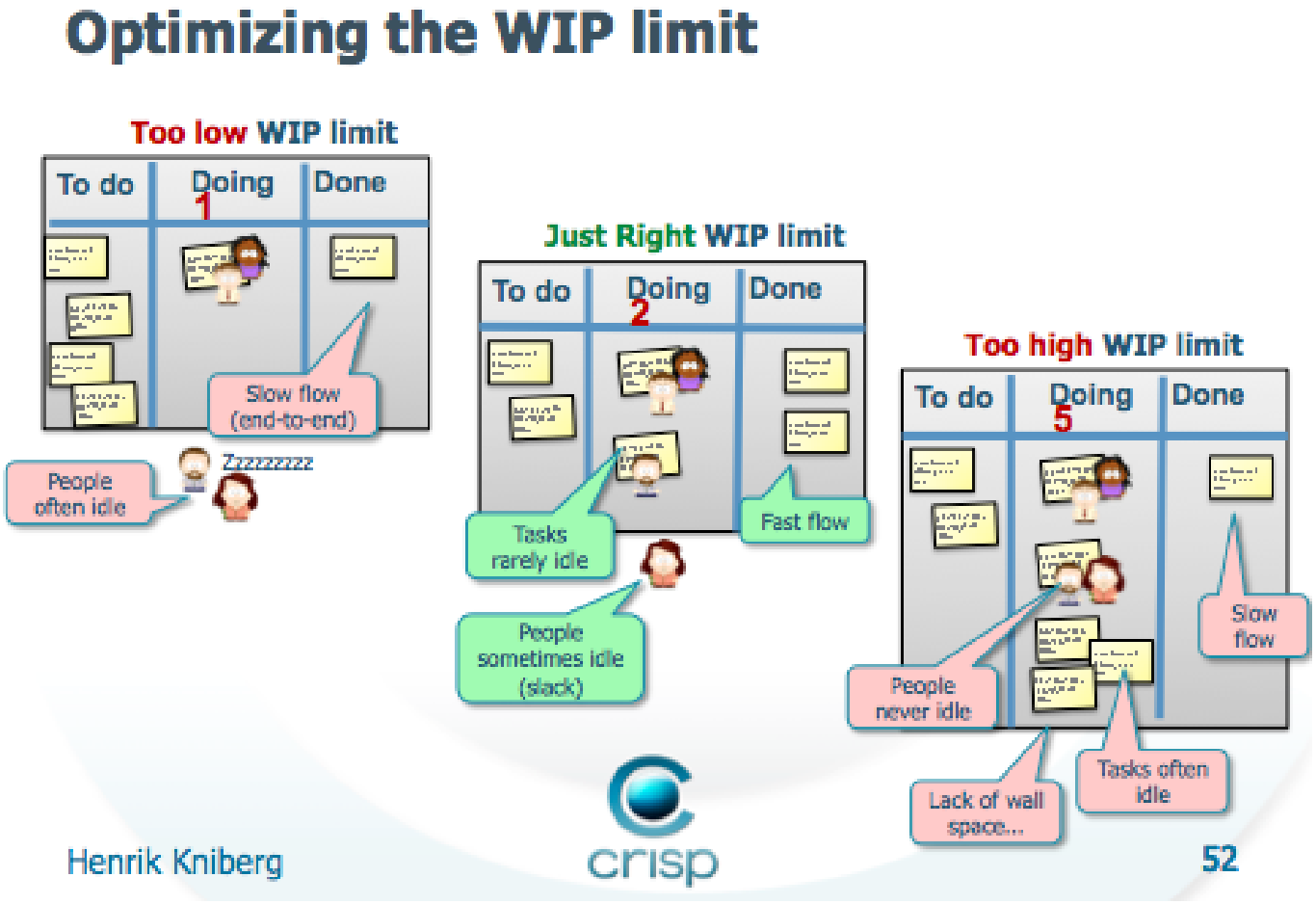
- Kanban
  - Definerer et sett med oppgaver eller “features” og lever så snart man er ferdig
  - Oppgaver skal ”flyte” uten avbrudd gjennom de nødvendige aktivitetene til de er ferdige (oppgaveboksing)

# Flytbasert utvikling - Kanban

- Fokus på gjennomstrømningshastighet på arbeidspakkene = antall brukerhistorier (features) implementert per tidsenhet
- Begrense antall arbeidspakker som det jobbes med i parallell (WIP = Work In Progress) for å hindre flaskehalser
- Antakelse: Jo høyere WIP, jo saktere flyter arbeidspakken gjennom arbeidsprosessene
- Når en pakke er ferdig, kan man etterspørre en ny som man begynner å jobbe med (pull)
- Slakk i tidsplanen er OK, dvs. en utvikler vil kunne vente hvis det optimaliserer overordnet flyt
- Mindre fokus på estimering

# Forskjell på Scrum-tavle og Kanban-tavle: Max WIP

- From: Kanban and Scrum - making the most of both by H. Kniberg/M. Skarin on Dec 21, 2009



# Fordeler ved Kanban

- Flaskehalser i prosessen blir synlige
  - Fokus på å bli ferdig med oppgaver som hindrer gjennomstrømning fremfor å begynne på flere oppgaver som vil hope seg opp
- Kan drive smidig utvikling uten å bruke “tidsbokser”
  - Spesielt for drifts- og supportoppgaver og vedlikeholdsoppgaver vil veldefinerte “sprinter” ofte ikke gi mye mening
- Gunstig der det er svært vanskelig å estimere oppgavene

# Lean – “den japanske skolen”, primært Toyota

- **Kanban** – en teknikk fra Lean (slank) production
- **LEAN**
  - Kontinuerlig læring og forbedring (Kaizen)
    - Forbedre helheten, dvs. ikke sub-optimalisere
    - Ved feil, stopp samlebåndet og finn årsaken til feilen fremfor å samle og rette opp alle feil i bolker
  - “Just-in-time”-prinsippet
    - Ikke lag noe før noen etterspør det
  - Komponentbasert produksjon (samme understell etc. på ulike biltyper)
  - Kundefokus
  - Unngå ”waste”
    - Fjerning av mellomlagring
- For interesserte, se «Lean Primer»(fronter) og «The Toyota Way»

# Waste

- Alt som krever ressurser
  - tid
  - arbeidsinnsats
  - rom
  - lager (unngå mellomlagring)
  - utstyr
  - penger

- som ikke gir verdi for kunden
- Verdi = det kunden ønsker og er villig til å betale for

# Den norske/nordiske modellen

- Selvstyrte (autonome) team
- Læring, redundans/jobbrotasjon
- Medvirkning og arbeidsmiljø
- Livskvalitet
- Samarbeid ledelse, arbeidstakerorganisasjoner og myndigheter
- Hydro, Volvo og mange flere


- B. Gustavsen, T.U. Quale, B.A. Sørensen, M. Midtbø og P.H. Engelstad. Innovasjonssamarbeid mellom bedrifter og forskning – den norske modellen. Gyldendal 2010

# Bilproduksjon vs. programvareutvikling

- Bilindustri er produksjon av fysiske produkter, mens programvareutvikling fokuserer på kode (tekst)
- Likevel, lean prinsippene kan anvendes i programvareutvikling
- Lean management er et hett tema i mange sektorer som ikke driver produktutvikling
  - Offentlig forvaltning (f.eks. helsevesen, universiteter)
  - Privat sektor

# Utfordringer ved smidige metoder

- Vanskelig å opprettholde kundens interesse i prosjektet hele tiden
- Utviklerne vil kunne mangle det intense engasjement som kreves
- Vanskelig å prioritere endringer hvis mange interessenter (stakeholders)
- Krever ekstra tid å stadig gjøre endringer og opprettholde enkelhet
- Kontrakter kan være et vanskelig tema (mer senere)