Amna Dastgir

SYSTEMUTVIKLING

Eksamen 20 november

2.år Bachelorstudium IT

Sommerville sier: "there are no right or wrong in software process"

Hva er systemutvikling? starter når noen interessenter (stakeholders) ønsker et datasystem for å håndtere utfordringer eller sutnytte potensialet i en virksomhet. handler om hvordan man utvikler, oppdaterer, tilpasser og utvider IT-systemer av tilstrekkelig kvalitet innenfor akseptable tids- og kostnadsrammer

Hvorfor lære systemutvikling? **Programmering er viktig, men også...** analyse av problemområdet (kunde og utvikler må bli enige om hva som ønskes å oppnå), design av systemet, testing (validering og verifisering) av systemet, installering hos kunder eller bruker

Datasystemer = Programvaresystemer = software systemer = IT systemer = danner bærebjelken i de økonomiske, politiske, sosiale, kulturelle og vitenskapelige sfærene av det moderne informasjonssamfunnet

Masseprodukter vs. skreddersydde systemer

Masseprodukter: selvstendig, markedsførte «hyllevare»-systemer → tekstbehandlingsprogrammer, prosjektstyringsverktøy og spill ect.

Skreddersydde- eller spesialtilpassede: spesifiseres og bestilles av bestemt kunde eller oppdragsgiver → offentlige systemer for skatt, flykontroll og miljøovervåking ect.

Porteføljer: systemer av systemer → Er ofte «ultralarge-systems» dvs: ekstremt komplekse, endres ofte, mange interessentgrupper → eks: world wide web, internasjonal flykontroll-trafikk, og som Telenor med mange hovedsystemer i sin portefølje.

Ulike systemer har ulike egenskaper og konsekvenser med ulike krav.

Variasjon i størrelse	
Programvare	Få til tusen millioner
Utviklingsteamene	Enkeltpersoner til over tusen stk
Kostnad utvikling og vedlikehold	Noen

Årsak til forskjell i resultater kommer av din kompetanse og måten du og ditt team jobber på, → ulikt sluttprodukt

En ingeniørdisiplin

Bidrar til at vi lager bedre systemer med færre ressurser på raskere og mer uforutsigbare måter. Betrakter menneskelige og teknologiske aspekter. Baseres på ingeniørprinsipper (evidensbaserte/empiriske metoder) med fokus på:

1. **planlegging og uforutsigbarhet (i motsetning til «ta tiden som trengs»)** → Graden av planlegging og formalitet i systemutvikling er et kontroversielt tema – da snakker man om planbaserte versus smidige metoder.
2. **oppdeling og strukturering av problemer i mindre komplekse deler (i motsetning til «prøv og feil»)** → analyse, design, programmering, testing, funksjonalitet, kravspesifikasjoner versus objekt orienterte designmodeller.
3. **modularitet og gjenbruk (i motsetning til «å lage alt fra bunn»)** → innebærer at datasystemer deles i mindre delsystemer (komponenter, moduler).
    a. **HØY KOHESJON:** hvert delsystem implementerer et veldefinert problem
    Mål på hva slags ansvar et objekt har og hvor fokusert ansvaret er. Et objekt som har moderat ansvar og utfører et begrenset antall oppgaver innenfor ett funksjonelt område har høy kohesjon. Objekter med lav kohesjon har ansvar for mange oppgaver innen ulike finksjonelle områder.

    b. **LAV KOBLING:** reduserer avhengigheter på tvers av delsystemer.
    Et mål på hvor sterkt et objekt er knyttet til andre objekter. Et objekt med sterk tilkobling er avhengig av mange andre objekter, noe som kan gjøre endring vanskelig.

    c. muliggjør gjenbruk innen et prosjekt eller på tvers av prosjekter, og det er lettere arbeidsfordeling og samarbeid.
4. **abstraksjon og modellering (i motsetning til «bare koden er systemet»)** → identifiser de viktigste momentene og ignorer irrelevante detaljer. Selve programmet kan også sees på som en modell av hvilke oppgaver som skal gjøres og hvordan. **UML**
5. **systematisk kvalitetssikring (i motsetning til «gjør som du selv syntes er best»)** →
    a) kundeinvolvering
    b) validering: har vi spesifisert det riktige systemet?

c) verifisering: er systemet viktig dvs har vi minimert antall feil?
d) Smidig utvikling: reduserer risiko ved at man leverer og evaluerer (validerer og verifiserer) delsystemer fortløpende, tett kundekontakt

Etikk innen systemutvikling

Ansvar utover anvendelsen av teknisk kunnskap og følge loven – handle moralsk riktig. Ved å utvikle datasystemer kan utviklere bidra til goder eller skade. Det er et ansvar for positiv profesjonsomtale.

Kompetanseområder innenfor systemutvikling

Domenekunnskap, utviklingsprosesser, estimering, arkitektur, design, modellering, programmering, utviklingsverktøy, virksomhetsforståelse, kravhåndtering, avtaler og kontrakter, juss og etikk, kvalitetssikring/testing, endringshåndtering, konfigurasjonstesting og databaser

UKESOPPGAVER 1

Hvilke fundamentale aktiviteter utføres i systemutvikling utover programmering?

Sentrale aktiviteter: problemanalyse/løsning, kravarbeid, utforming/design, testing, validering, innføring og vedlikehold.

Hvilke aspekter ved systemutvikling tilsier at det er en ingeniørdisiplin?

1. evidensbasert: baserer seg på observasjon
2. empirisk: baseres seg på erfaring
3. planlegging og forutsigbarhet: ta tiden som trengs
4. oppdeling og strukturering av problemer: prøving og feiling
5. modularitet og gjenbruk: lag alt fra bunnen hver gang
6. abstraksjon og modellering: det er kun koden som utgjør systemet
7. gjør det du selv syntes er best

Hva er en systemutviklingsprosess?

Et rammeverk for hvordan man skal komme fra A til Å i utviklingen av et system.

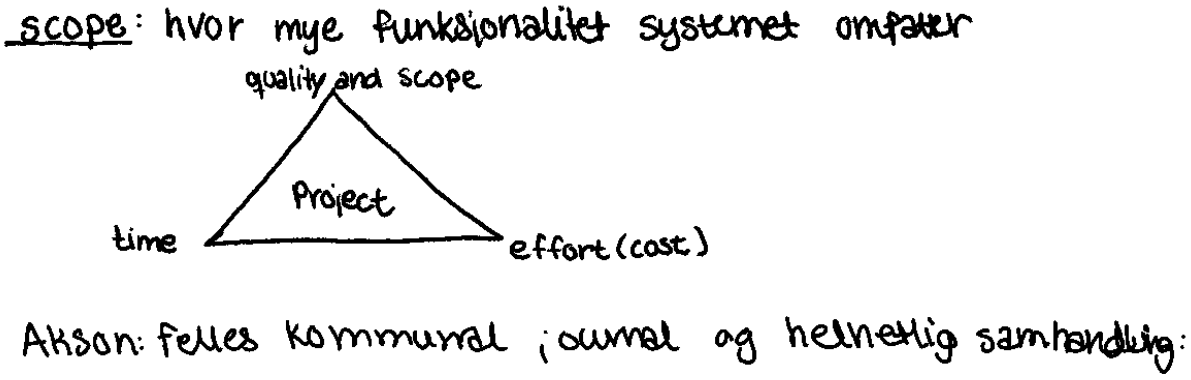
Hvorfor er det viktig å ha en god systemutviklingsprosess?

Påvirker resultatet: tid, kostnader, prosjektstyring, arbeidsmiljø, og kommunikasjon

Det er ofte stor prisvariasjon i anbud på IT-prosjekter sammenlignet med andre bransjer, slik som bygg-- og anleggsbransjen. Hva kan denne prisvariasjonen skyldes?

1. utvikling av it systemer → Manglende statistikk for estimering av ressursbruk
2. usikkerhet→kunden må vite hva de vil ha og utviklere må forstå hva de skal lage
3. ulike utviklingsprosesser/måter å jobbe på→variasjon i kostnader og ressurser

Grenser for anvendelser av IT-systemer som bør kunne reservere seg mot å bidra til å utvikle? Våpensystemer, overvåking og design av såkalt supermenneske



Prosess er ett sett av aktiviteter for å få utføre en jobb: planlegge prosjekt, forberede og gjennomføre møter, kravspesifikasjoner og programmere. Prosessen påvirker prosjektkvalitet og produktkvalitet.

Prosessbeskrivelser vil inneholde: delprodukter/resultater (modeller, figurer og kode) av en aktivitet, før og etterbetingelser dvs betingelser som er sanne før og etter en fase eller et delprodukt er levert.

Eksempler på roller:

Utvikler, arkitekt/system, designer, grafisk designer, test, prosjektleder, fasilitator (scrum master), bruker/kunde, representant, produkteier.

Eksempler på verktøy

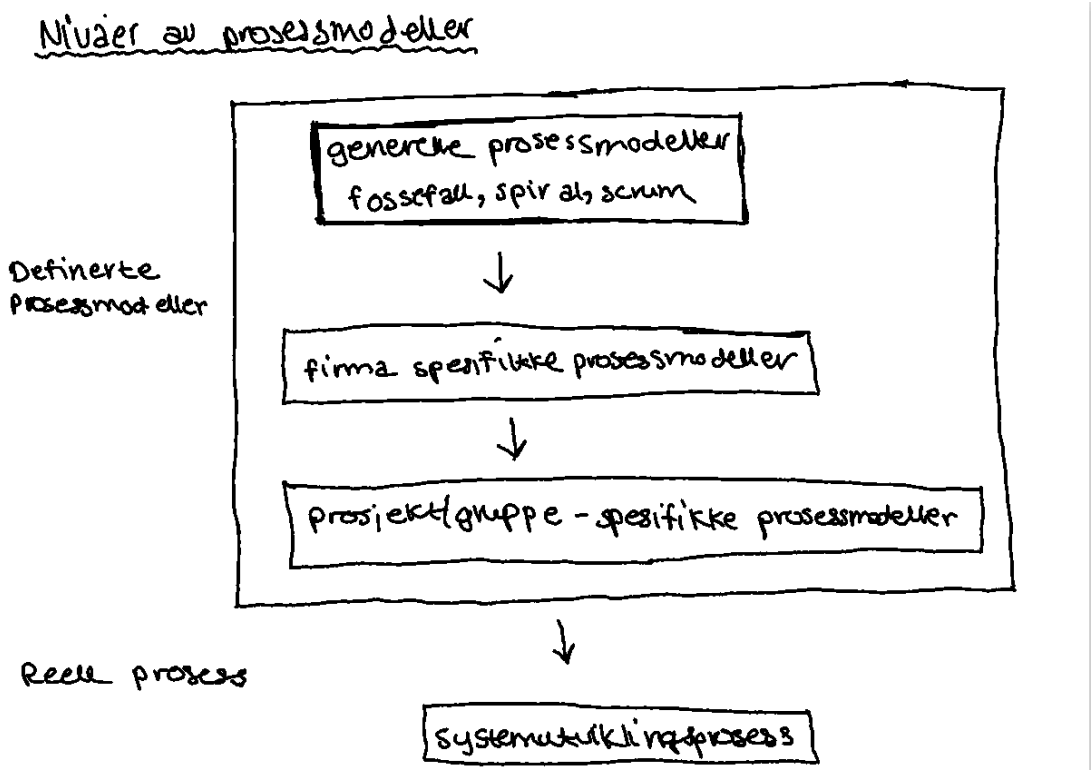
- Utviklingsverktøy: IDE f eks Android studio
- Modellering/diagram konstruksjon (f eks tegne UML diagrammer – use case diagram, sekvens, klasse osv.)

- Versjonskontroll (Git)
- Testing
- Bug og issue tracking (f eks TRELLO)
- Prosjektstyring
- Samarbeidsverktøy (discord, messenger, slack, notion, github, trello, zoom, google)

Reell prosess versus modell av prosess

abstrakt (forenklet modell) representasjon av en prosess. Definerer hvordan arbeidsprosessen struktureres i et prosjekt. Normativ - dvs. at modellen beskriver en prosess slik den bør være.

Nivåer av prosessmodeller:



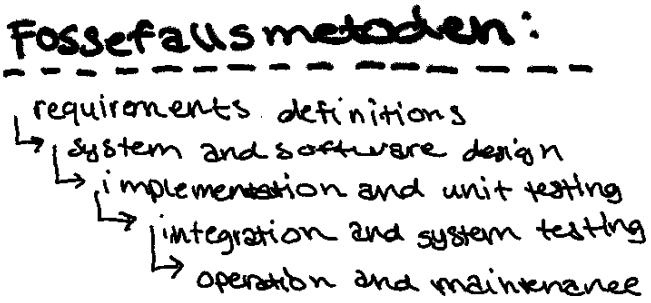
Smidige prosessmodeller

Elementer fra inkrementell og iterativ utvikling, danner utgangspunkt for «smidige metoder».
I plandrevne prosesser er alle prosessaktivitetene plamlagt, på forhånt 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. I praksis er systemer bygd av BEGGE.

PROSESSMODELLER

Fossefallsmodellen:

Plandreven modell →utviklingen foregår i veldefinerte faser.



Egenskaper: vanskelig å tilpasse endringer i krav underveis i prosjektet. Egner seg godt når kravene er forstått og lite sannsynlig med mye endringer underveis, men det er veldig få systemer som har stabile krav.

Denne metoden brukes mest i store prosjekter som gjerne utvikler på ulike steder. Den plandrevne utviklingen gjør det enklere å koordinere arbeidet. Sivilingeniørprosjekter (bygg og anlegg) er plandrevet, dvs. relativt mye tid på planlegging og dokumenter som styrer prosjektet.

Hva skjer når en fase er ferdig? Arbeidsflyt ved endt fase – avvik mellom prinsipp og praksis:
Prinsipp: beveg deg nedover modellen mot neste aktivitet, ikke tilbake.
Praksis: stor overlapp i aktiviteter.

Inkrementer og iterasjoner i systemutvikling

Et ***inkrement*** er et tillegg i funksjonaliteten som er implementert i et system. Et inkrement er et aspekt ved systemet. En ***iterasjon*** er en syklus i utviklingen og er 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 utvikler gradvis i form av nye inkrementer som legges til. Hvert inkrement evalueres før utviklingen av neste inkrement starter. Det er i smidige metoder. Selve evalueringen gjøres av en bruker eller kunderepresentant.

Inkrementell installering:

I stedet for at hele systemet leveres til en kunde på engang, leveres ett inkrement av gangen som tilsvarer deler av total funksjonalitet. Brukerkravene prioriteres slik at de viktige kravene er implementert i de første inkrementene. Når utviklingen av et inkrement startes, så fryses alle kravene til det inkrementet, men kravene til senere inkrementer kan fremdeles endres.

Fordeler og ulemper ved inkrementell utvikling og installering

+Fordeler	-Ulemper
Kostandene ved endrede brukerkrav reduseres sammenliknet med fossefallmetoden da delene som må endres er mindre	Store prosjekter/systemer krever en relativt stabil arkitektur som inkrementene og teamene må forholde seg til, dvs arkitekturen kan ikke utvikles i inkrementer
Enklere å få tilbakemeldinger fra kunden	Strukturen til systemet har en tendens til å bli stadig verre etter hvert som inkrement legges til
Lettere å følge prosess og utvikling	Derfor stadig vanskeligere å foreta endringer, med mindre ressurser bruker på refaktorering/eller strukturering
Raskere levering av deler enn ved fossefallmetoden	
Funksjonaliteten som er prioritert testes mest	
Lavere risiko for total prosjektfiasko	

Spiralmodellen

Utviklingsprosessen er representert som en spiral i stedet for 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.

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

1. Identifisere spesifikke mål for fasen
2. Analyser risiko og utfør aktiviteter for å redusere de viktigste
3. design, koding (programmering) etc.
4. evaluer prosjektet og planlegg neste fase i spiralen

Spiralmodellen – klassisk og velkjent

Stor betydning i utviklingen av tankegang rundt iterasjoner og risikovurdering i systemutviklingsprosessen. Men brukes sjelden i konkret systemutvikling.

Endring (evolusjon) av programvare

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

Smidig utvikling

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 til høyre har verdi verdsettes de til venstre enda mer.

Plandrevne (tunge) prosesser	Smidige (lette) prosesser
Prosessaktivitetene er planlagt på forhånd og progresjon måles i henhold til denne planen	Planlegging gjøres litt etter litt (inkrementelt)
En tung prosess inkluderer mange aktiviteter og roller, og krever formelle, detaljerte og konsistente prosjektdokumenter	Enklere å endre prosessen for å tilpasse endrede krav fra kunden
Tunge prosesser er ofte «for-tunge» dvs vektlegger aktiviteter som vanligvis gjøres tidlig i prosessen (planlegging, analyse og design)	Lette prosesser fokuserer mer på fundamentale prosesser (f.eks «kontinuerlig testing»), har færre formelle dokumenter og er fote mer iterative

Ekstrem programmering (XP)

Ekstrem ved at:

- hele systemet kan bygges (kompileres) flere ganger daglig
- inkrementer av systemer leveres til kunden annenhver uke

- alle tester må kjøres for hver bygning* og bygningen aksepteres bare hvis testene er vellykkede

*bygging: alle komponentene til systemet kompileres og linkes til hverandre, data og biblioteker som er nødvending for å lage et kjørbart system.

Scrum

Daglig standup – 3 spørsmål:

Hva har du gjort siden i går, hva skal du i morgen? Hvilke eventuelle hindringer har du?

Scrum master sørger for at scrum prosessen blir fulgt

Leder «Daglige standup», hindrer støy slik at utviklerne kan fokusere på oppgavene, teamets «coach og bestevenn». Koordinerer SPRINT planlegging, og estimering av ***sprint backlog***.

Sprint backlog: en slags «to-do»-liste. Prioritert liste med arbeidsoppgaver, funksjoner, bugs, tekniske ting (koding). Dette vedlikeholdes av produkteieren.

Scrum team

Typisk 5-9 personer → utviklere, testere og arkitekter → utvikler systemet

Pressfokuserte smidige metoder

TIME-BOXING VERSUS TASK-BOXING

Timeboxing – tidsbokser (SCRUM):

Velg noen prioriterte oppgaver → jobb med dem i faste tidsintervaller med definert oppstart og avslutningsaktiviteter. En sprint: 2-4 uker.

Ved scrum blir systemet delt opp i mengde forståelige og håndterbare deler. Ustabile krav hindrer ikke progresjon. Hele teamet observerer kunder kontinuerlig som får inkrementene avtalt til tid.

Kryssfunksjonelle team: kompetanse i ulike områder → reduserer risikoer.

Taskboxing - Flyt av oppgaver (KANBAN):

Definerer sett med oppgaver eller features som skal lages →lever så snart man er ferdig. Oppgaver skal «flyte» uten avbrudd gjennom de nødvendige aktivitetene til de er ferdige («oppgaveboksing»).

Fokus på gjennomsnittshastighet på arbeidspakkene = antall brukerhistorier (features) implementert per tidsenhet. Man begrenser antall arbeidspakker som det jobbes med i parallell

(WIP = work in progress). Jo høyere ***WIP*** = jo saktere flyter arbeidsoppgavene. Når en pakke er ferdig, kan man etterspørre en ny (pull).

FORDELER: gunstig der det er vanskelig å estimere oppgavene, fokus på å løse oppgaver som hindrer kaos, kan bruke smidig utvikling uten å bruke «tidsbokser».

Funksjonelle krav:

- Hva skal systemet gjøre?
    - Hvilke tjenester skal systemet tilby?
    - Hvordan skal systemet reagere på ulik input?

Ikke funksjonelle krav:

- Hvordan skal systemet implementere de ulike kravene?

Alle krav bør være:

→ forståelige – interessenter må kunne forstå kravspesifikasjoner

→ testbare – vi må kunne avgjøre om det ferdige systemet gjør det som kreves

→ sporbare – vi må vite hvilken del av koden som skal endres når det kommer nye krav

Utfordringer i kravhåndtering

Fra fagsiden (business) og hvis dem dominerer kan funksjonaliteten bli besluttet uten tilstrekkelig innsikt i hva som kan realiseres i systemet.

Hvis IT dominerer, kan den tekniske sjargongen bli dominerende ,og dermed blir det vanskelig å få kravene riktige.

UKESOPPGAVER 2

2) Nevn noen praksiser som blir benyttet i Extreme Programming (XP).

Praksis:	Beskrivelse: (side 39)
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 svare en utviklingsoppgave.
små releaser	hyppige releaser
refaktorering	med engang sjansen dukker opp
Parprogrammering	programmering i par, inspill og samarbeid
kollektivt eierskap	alle jobber på ulike deler av koden, men endrer fritt
holdbart tempo	mye overtid = dårlig kode

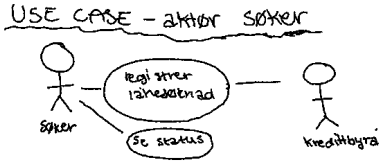
XP er den vanligste programmeringsfokuserte smidige metoden.

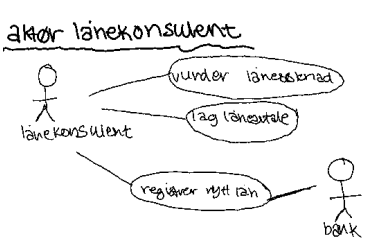
BRUKERHISTORIER – USE CASE – BESKRIVE KRAV

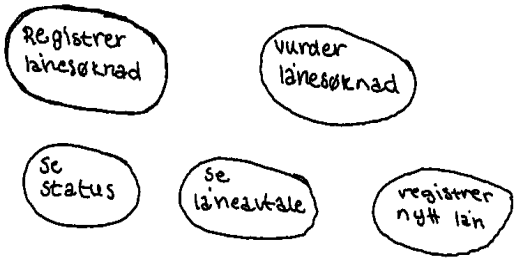
«som en <rolle> ønsker jeg <funksjon> for å oppnå <verdi>»

→ Use story (brukerhistorie)

→ Use case (brukstilfelle) – beskriver hvordan systemet oppnår et mål av verdi for en aktør. Et komplett use case består av flere ulike hendelsesforløp (flyt). Ett case beskriver en komplett funksjonell enhet og er testbart. Eks: use case for lånesystem:

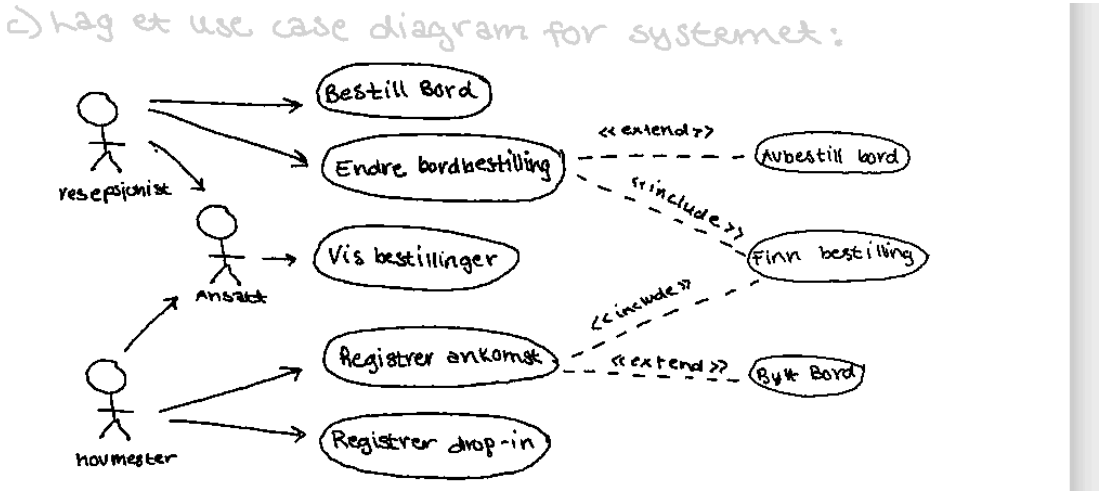






Funksjonelle krav: input (lånesøker må oppgi navn, adresse, telefon, fødselsdato, arbeidsgiver, årslønn og samlet gjeld). Foretningsgjeld eksempel: lån kan bare gis til personer med fast jobb.
Ikke funksjonelle krav: brukervennlighet → en ny bruker skal kunne registrere søknad på mindre enn 10 min. Ytelse: systemet skal inntil 100 samtidige brukere.

- Modeller
    - UML – unified modeling language
    - BPMN – business process model notation



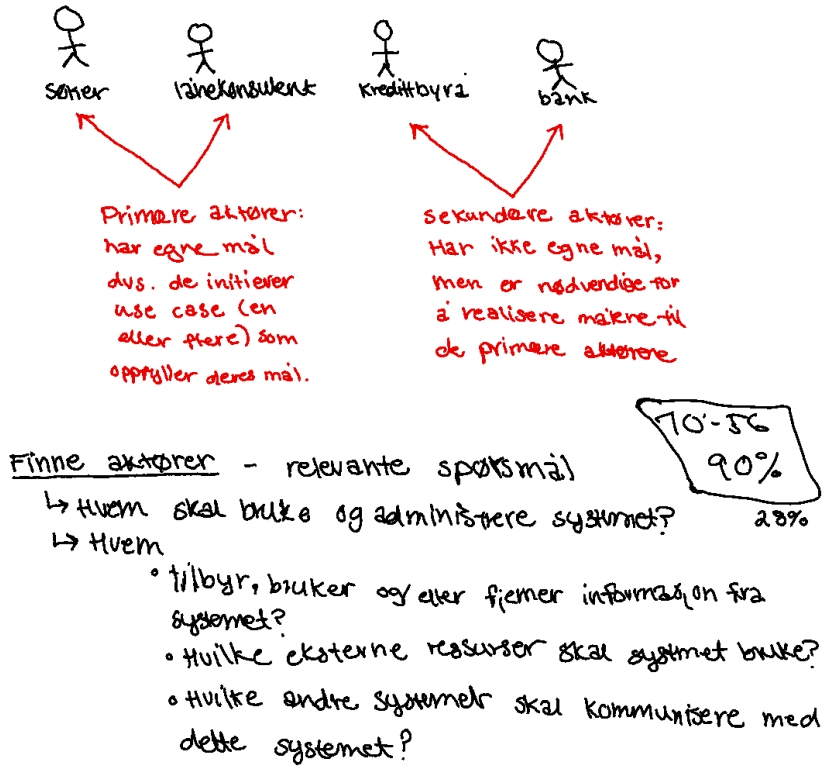
DIAGRAMMER I UML

- Aktivitetsdiagrammer:
    - Viser forretningsprosesser og arbeidsprosesser
- Use case diagrammer:
    - Viser systemets funksjonalitet og samspillet mellom systemet og omgivelsene (brukere, andre systemer og komponenter)
- Sekvensdiagrammer:
    - Viser samspill mellom system, omgivelser og mellom de forskjellige delene av systemet (mer detaljert enn use case)
- Klassediagrammer:
    - Viser objektklassene i systemet og assosiasjonene mellom disse klassene
- Tilstandsdiagrammer:
    - Viser hvordan systemet reagerer på eksterne og interne hendelser

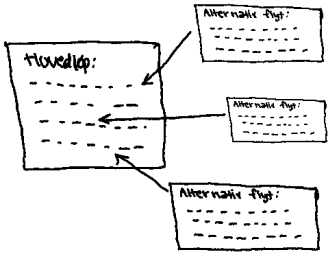
Interessenter: samlebetegnelse for alle personer, grupper eller organer, som påvirkes av eller påvirker systemets utvikling bruk direkte eller indirekte. Eks: arbeidsgivere (kunder), brukere, utviklere og vedlikeholder, systemeiere og forvaltere, fagforeninger, lovgivere og myndigheter.

En ***aktør*** for et system representerer en rolle som et menneske, eller et eller annet system som har et mål med systemet. Det er ofte flere interessenter enn aktører, og en aktør er som oftest også en interessant (stakeholder), og kommuniserer med systemet via ett eller flere use case.

Skille mellom disse 2 omhandler rolle/tilknytning til brukeren og utviklingen av systemet.
Primære aktører: har et eget mål.
Sekundære aktører: trengs for å oppfylle de primære aktørenes mål.
Prebetingelse: må være på plass før et use case kan utføres
Postbetingelse: en endring i systemet som skal være på plass etter at et use case er utført.



Alternativ flyt: finnes fordi use case kan lykkes og feiles på flere måter, så detaljerte use case har også alternative flyt. Alternativ flyt (blant annet feilsituasjoner) er viktige da det ofte er mer «uenighet» blant prosjektets interessenter om hva som skjer i de

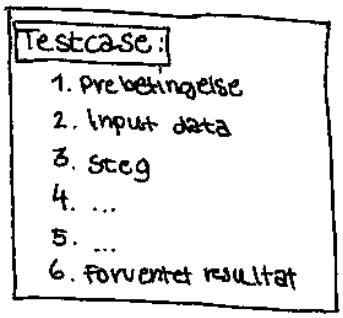


tilfellene enn før hovedløpet. Et hvert steg i hovedløpet kan være utgangspunkt for en alternativ flyt.

Extend use case: beskriver hvordan oppnå tilleggsresultater, mens alternativ flyt beskriver hva som skjer ved avvik i normal flyt.
Include relasjonen: Use case kan være en del av ett eller flere andre use case
Exclude relasjonen: Use case som beskriver tilleggsoppførsel som utføres under gitte omstendigheter.

LIKHETER	FORSKJELLER
Hvem som skal bruke systemet	Omfang, kompletthet, livslengde og hensikt
Hva de skal gjøre med det	User stories er godt egnet for å finne krav og bruke disse i smidig utvikling i samarbeid med produkteier/kunde
Hvorfor de skal gjøre det	Use case er mer detaljert, har flere bruksområder videre i prosjektet og er mer egnet som dokumentasjon

Hendelsesflyt i use casene detaljeres ut i sekvensdiagram.



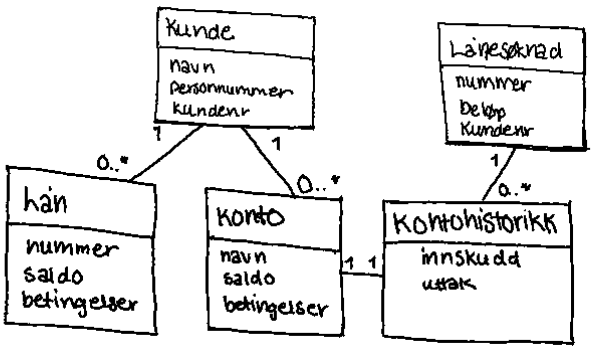
DOMENEMODELL

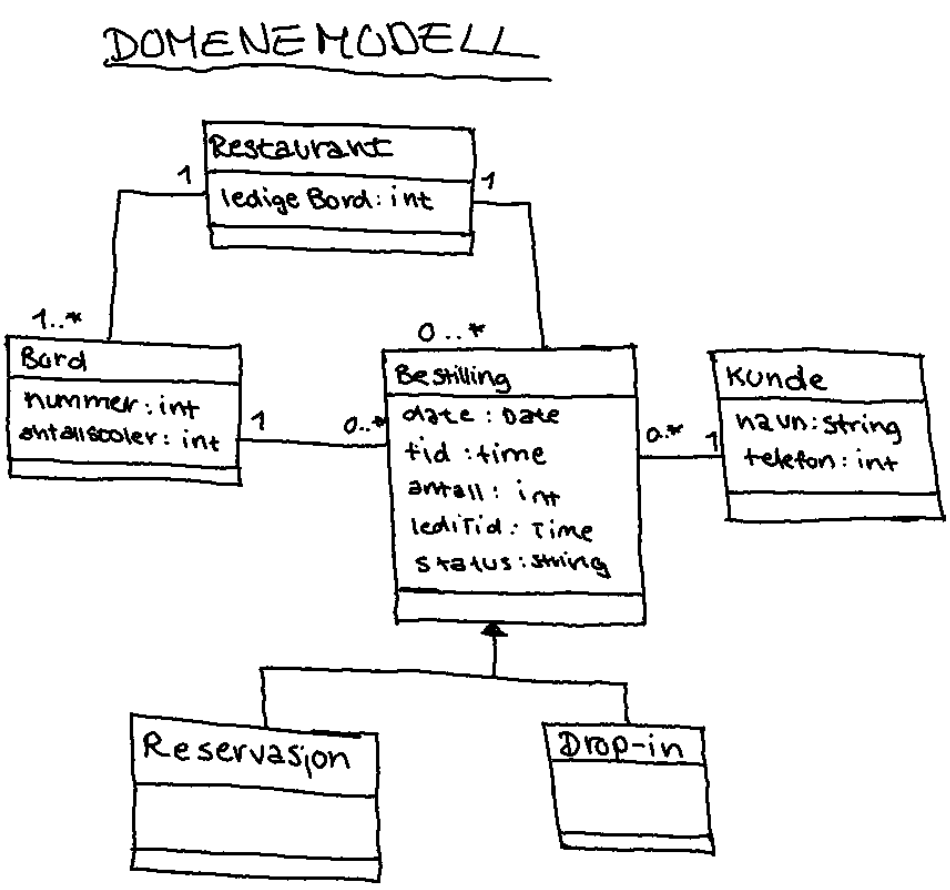
Domenemodellen utvides til klassediagram med systemklasser.
UML klassediagram uten metoder, viser objekter i problemdomenet.
Hensikten med denne modellen er å forstå objektene og få en hensikt over terminologi.

Nyttig fordi

→ viser informasjon om objekter i use casene

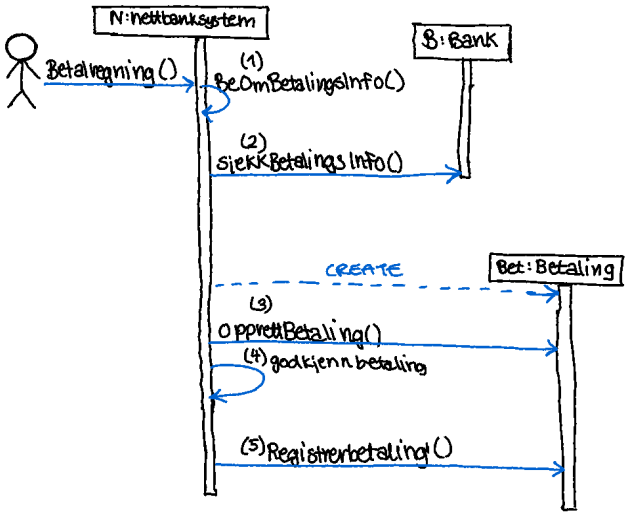
→ den er et viktig verktøy for å sjekke at use casene er beskrevet med riktig detaljeringsnivå





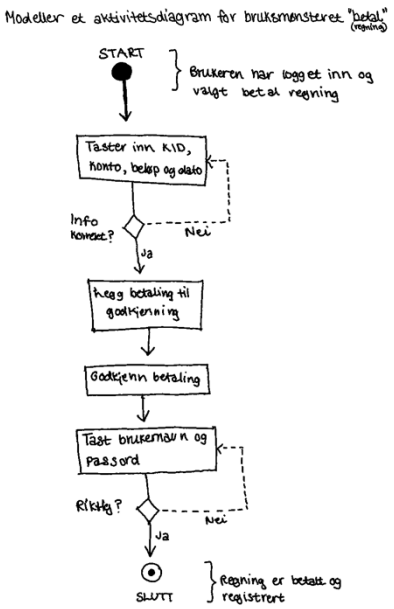
SEKVENSDIAGRAMMER

Modellerer en flyt i et use case. For hvert use case lages typisk sekvensdiagrammer for hovedflyt og for hyppig forekommende alternativ flyt. Stegene i et use case vises som meldinger som sendes mellom objektene ved kall på objektenes metoder.



AKTIVITETSDIAGRAMMER

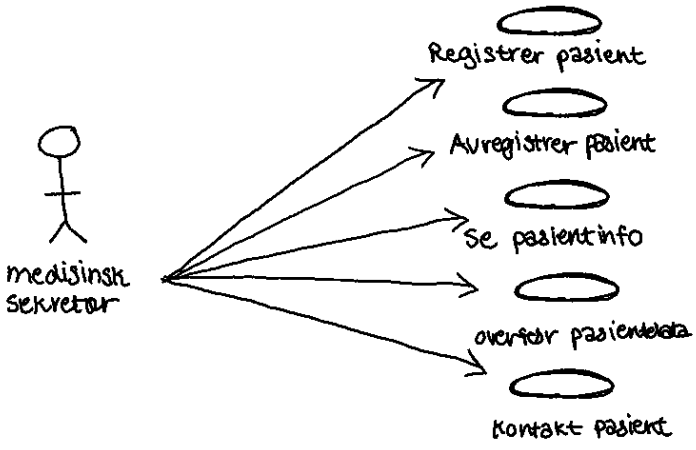
Kan grafisk representere hendelsesflyten i et use case. Stegene vises som akitiviteter, beslutninger underveis vises som diamanter. Aktivitets- og sekvensdiagrammer brukes noe overlappende, men sekvensdiagrammer er typisk mer kodenært, mens aktivitetsdiagrammer er mer forretningsnært.



Systemmodellering: utvikle abstrakte modeller av et system, der hver modell representerer ulike perspektiv av et system. Viktige for å forstå funksjonaliteten i et system og modellene brukes til å kommunisere med kundene, og til dokumentasjon.

Systemperspektiv

1. eksternt perspektiv– der du modellerer konteksten til systemet
2. interaksjonsperspektiv – der du modellerer interaksjonen mellom et system og omgivelsene, eller mellom komponentene i et system
3. strukturelt perspektiv– der du modellerer organisasjonen systemet inngår i eller datastrukturen som brukes av systemet
4. adferds perspektiv - du modellerer systemets adferd og hvordan det reagerer på hendelser



KLASSEDIAGRAMMER

En klasse kan bli sett på som en generell definisjon av objekter som er instanser av klassen. En assosiasjon mellom to klasser angir at det er en forbindelse mellom disse klassene.

GENERALISERING

Teknikk for å håndtere kompleksitet, og unngår detaljer. Gir muligheten til å se at ulike objekter har felles karakteristikker, for eks lege og pasient er personer. I objektorientert språk, som Java, er generalisering en del av språket – gjennom såkalt arvemekanisme («inheritence»). Attributter og metoder er assosiert med superklasser og subklasser. Subklassen er lege og pasient.

ATFERDSMODELLER

Modell av dynamikken i et system ved kjøring som viser hva som skjer ved respons på stimuli. To typer stimuli:

- Data: systemet fanger opp data som må prosesseres
- Hendelser: en hendelse som trigger systemet og fører til prosessering. En hendelse har ofte assosierte data i tillegg, men ikke nødvendigvis

Datadrevet modellering

Mange forretningssystemer er primært drevet av data. De kontrolleres av input data, med få eksterne hendelser. Datadrevet modeller viser sekvensen av handlinger som er involvert i prosesseringen av «inputdata» og generering av «outputdata». Modellene er spesielt nyttige under kravspesifikasjon og brukes til å vise «end-to-end» prosessering av systemet.

Handelsdrevet («event-driven) modellering

Mange systemer er drevet av «hendelser» med minimal data prosessering. Modellene viser hvordan et system reagerer på eksterne og interne hendelser. Baseres på antakelsen av at systemet har et endelig antall tilstander og at hendelser (stimuli) fører til at systemet går fra en tilstand til en annen.

Modelldrevet systemutvikling

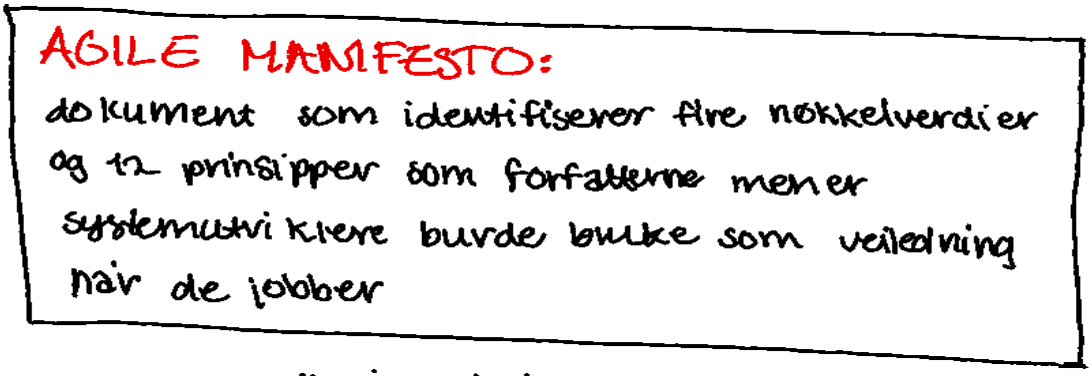
Modellene i seg selv er målet. Kode og programmer blir generert automatisk fra modellene.

+Fordeler: høyere abstraksjonsnivå, automatisk kodegenerering gjør det enklere og rimeligere å tilpasse systemer til nye plattformer.

-Ulemper: abstraksjonsmodeller er ikke alltid det riktige for implementering, mange mener koden er lite effektiv og endinger i koden er nødvendig. Det er vanskelig å holde modellen oppdater ved endringer og feilretting.

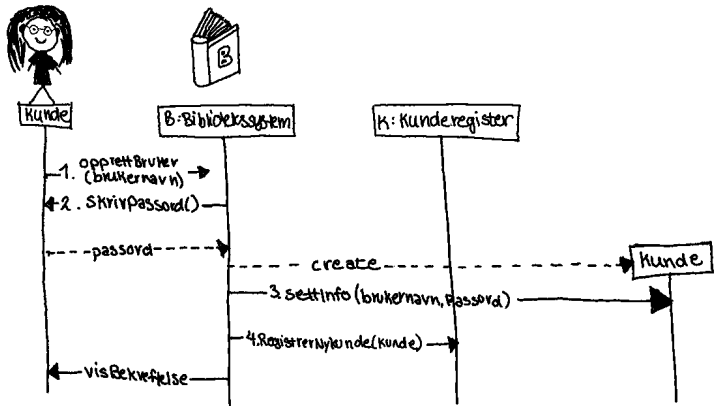
Den fundamentale ideen bak modelldrevet systemutvikling er at det er mulig med en komplett automatisk transformasjon fra modell til kode. Dette er mulig innenfor UML 2, og kalles executable UML eller XUML. En domenemodell identifiserer de overordnede prinsippene, den bruker UML klassediagram og inkluderer objekter, attributter og assosiasjoner.

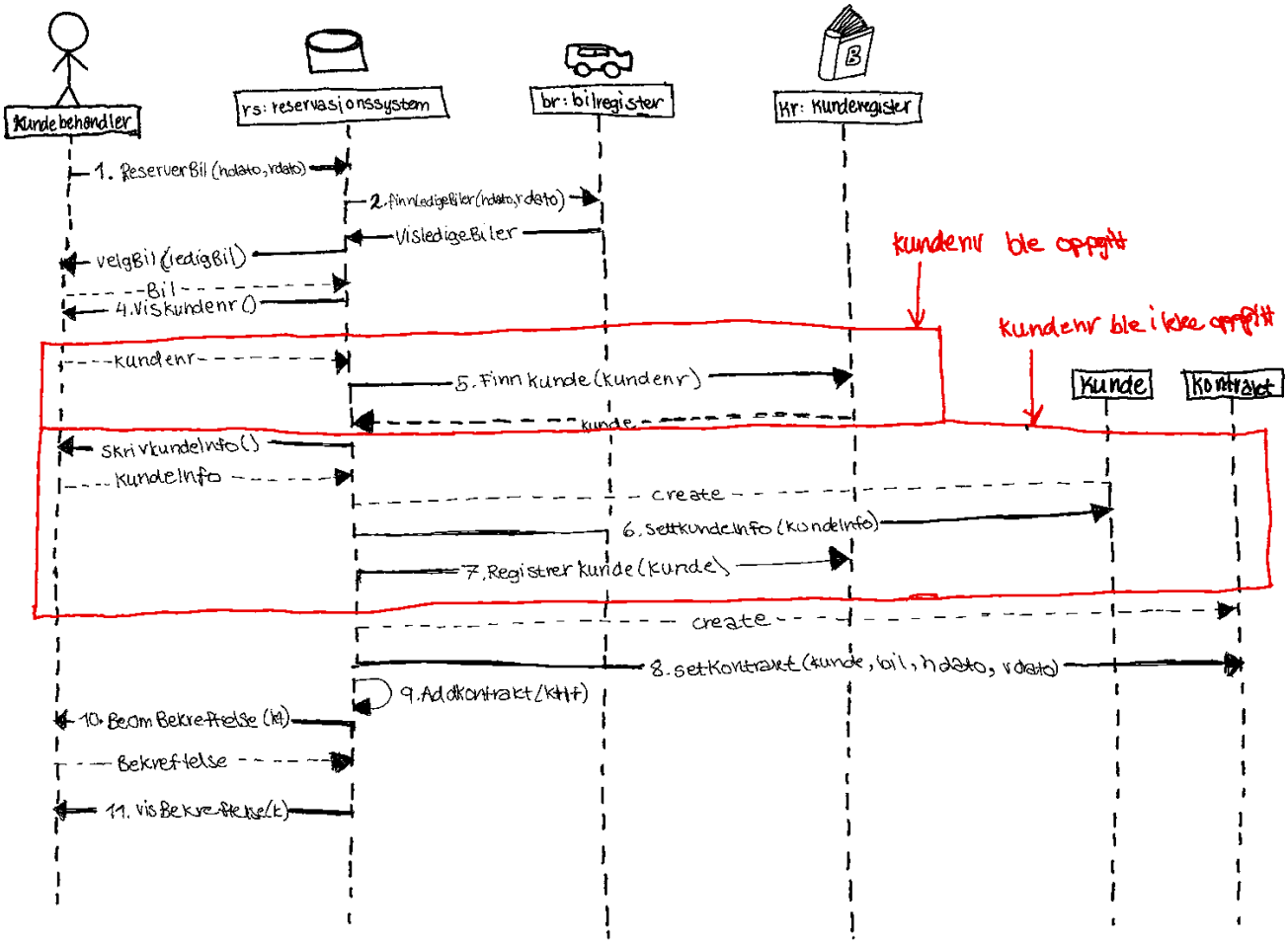
Mange brukere av modelldrevet utvikling hevder at metoden understøtter en iterativ tilnærming og passer dermed veldig godt innenfor smidig metodikk.



EKS: SEKVENSDIAGRAM

Vi har et enkelt bibliotekssystem der kunder selv kan velge å registrere seg over nett. Dette gjøres ved at kunden lager seg et brukernavn, passord og oppgir evt annen relevant info (mail/navn/alder). Bibliotekssystemet har også et system over alle kundene sine.





Oppsummering: SCRUM

- smidig utvikling: kjennetegn
  - Planlegging gjøres inkrementelt
  - Enklere å endre prosessen ved endringer i krav fra kunden
- planleggingsfasen:
  - overordnede mål etableres/arkitektur designes
- Gjennomføringsfasen:
  - serie med iterasjoner → sprinter (2-4 uker)
  - Hver iterasjon leverer et inkrement av systemet
- Avslutningsfasen:
  - Dokumentasjon og manualer ferdigstilles

Oppsummering: FOSSEFALL

- plandrevet prosessmodell
  - utviklingen styres av forhåndsspesifiserte planer
- separate, veldefinerte faser
  - kravspesifisering
  - Design (av system og programvare)
  - Implementasjon (og enhetstesting)
  - Integrasjon (og systemtesting)
  - Drift (installasjon og vedlikehold)

INTERESSENT OG AKTØR

Skillet mellom aktør og interessent

- Omhandler: Rolle / tilknytning til brukern og utviklingen av systemet.

Aktører må være: Brukere av systemet, andre systemer som brukes av/bruker systemet.

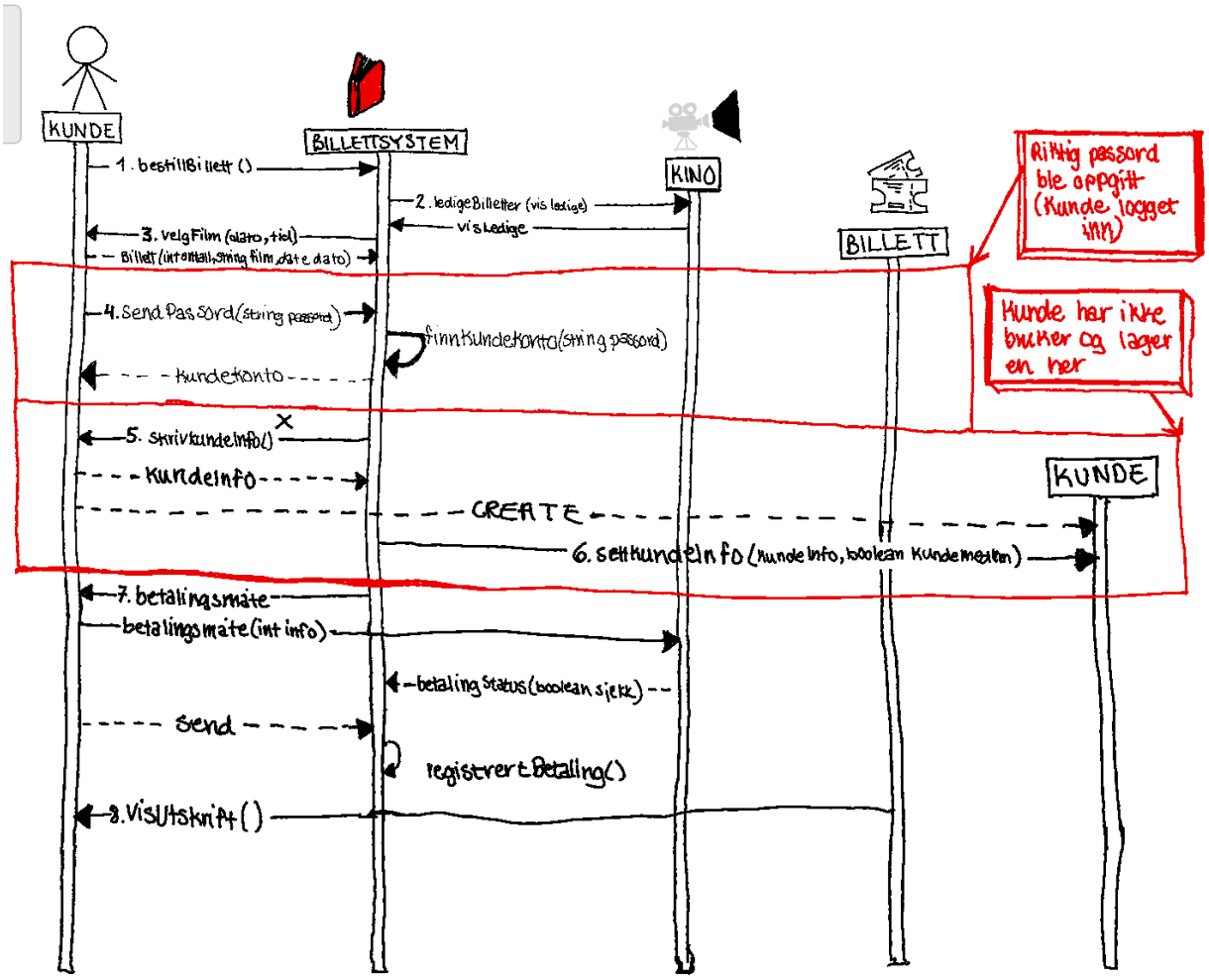
- Interessenter kan...
  - være brukere av systemet
  - inkluderer alle som påvirker/påvirkes av systemets kravspesifikasjon og utvikling.

Objekt design – ansvarstilordning

Ansvar er knyttet til objekt i forma av dets oppførsel:

- Handling: opprette objekt, beregne, initiere handler i andre objekter og kontrollere
- Kunnskap: vite om private data, relaterte objekter, ting som det kan utlede og beregne

Kategorier av ansvar: sette og hente verdier av attributter, opprette nye instanser (objekter), hente fra og lagre til persistent minne (databaser), slette instanser, legge til og slette linker til assosiasjoner, kopiere, konvertere og endre, navigere og søke.



Metoder for evaluering av systemutviklingsmetoder

Organisasjoner, team og individer må stadig velge mellom teknologi som støtter systemutvikling. Systemutvikling – menneskelig aktivitet i organisasjoner kan bruke matematikk/logg for på å utlede hvordan utviklingen skal utføres. Lytt til folk med erfaring.

Empiriske forskningsstudier: - står om disse under

1. eksperiment
2. case studie
3. etnografi
4. spørreskjema undersøkelse
5. aksjonsforsikring
6. systematisk litteraturstudie

Hypotesetesting

- nullhypotese = påstand man ønsker å forkaste
- hvis et eksperiment viser svært lite sannsynlighet <5% (signifikansnivå) forkaster vi den

Statisk styrke

Eksperimenter svarer på «hva» case studier mer enn «hvordan» og «hvorfor». Teoriene bør styre datainnsamling og analyse (generalisering).

> ***Statisk styrke:*** Sannsynligheten for at en statisk test vil forkaste nullhypotesen hvis den faktisk er feil. En test uten nok styrke vil ikke forkaste nullhypotesten selv om den er feil. Vi må ofte gjøre et forsøk flere ganger eller ha mange deltakere i et eksperiment for å få nok styrke.

Enkel case studie og multiple case studie: flere studier studerer, og resultater sammenliknes.
Etnografi - Forskeren er mer involvert i gruppen som studerer. Bruker gjerne ground theory.

Case-studier bruker teorier for å bestemme hva slags data som samles. Teorien styrkes, forkastes, eller justeres, avhengig av det man finner.

Spørreskjema undersøkelser (Surveys)

Lager statistikk og tester hypoteser over egenskaper ved gruppen som studeres. Man svarer på hva folk mener og inn på påstander. Det demografiske må tilrettelegges og tas hensyn med tanke på hva som undersøkes. Eks: identifisere problematiske områder i systemutvikling.
Man bruker spørreundersøkelser når det meste er bestemt for undersøkelsen. Er det faste spørsmål, har vi godt utvalg, er det sikkert at vi får pålitelige svar?

Ulike type skalaer brukt:

- ordinal skala = holdninger og preferanser
- likert skala = enig/uenig/delvis
- frekvensskala = aldri/sjelden/av og til
- evalueringsskalaer = dårlig/bra

Aksjonsforskning

Praksis og teori tett, integrert slik at begge lærer av hverandre. Forskere og ansatte i organisasjoner erfarer problemer og utfordringer sammen, og lærer hvordan man takler dem. Sammen utfører de «aksjoner» (tiltak for endring) for å forbedre situasjonen.

Aksjonsforsking prøver å oppnå praktisk nytte for organisasjonen samtidig med å fremskaffe ny teoretisk kunnskap.

Systematisk litteraturstudie

Finner, evaluerer og setter sammen enkeltstudier innen et spesielt tema til en helhet (syntese).
Systematisk prosess: planlegg litteraturstudie: lag protokoll for gjennomføring. Gjennomfør primærstudiet, samle data osv. også rapporter studien.

Tilsynelatende motstridende resultater: i fag og studier kan være BIAS-ed. F eks i medisin: effekten av kaffe.

Triangulering

Triangulering: unngå subjektive vurderinger.
For å unngå subjektive vurderinger bør man ha:

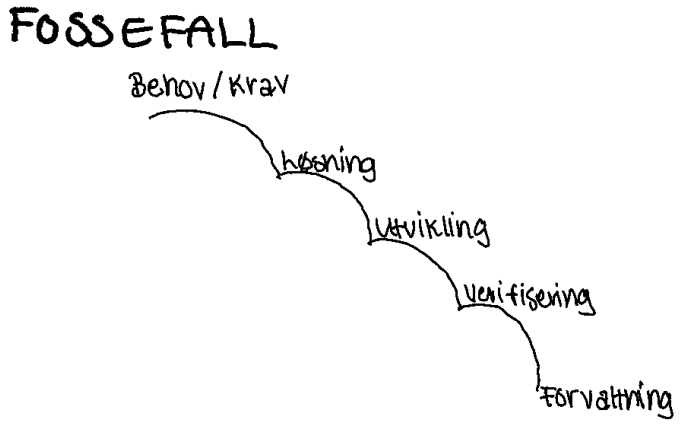
Datatriangulering: data fra flere kilder bør beskrive samme fenomener.

Forsker-triangulering: ulike personer bør undersøke det samme.

Kontraktregulering av programvareutvikling

«En kontrakt er en avtale som mellom partnere etablerer en bindende forpliktelse til å gjøre eller å unnlate å gjøre noe».

TILBUD + AKSEPTER = AVTALE



Hva er «smidig»?

⇒ Forretningsverdi som viktigste kvalitetsmål

⇒ Kontinuerlig prioritering av funksjonalitet ut fra kost/nytte

⇒ Tett dialog mellom fagpersoner og utviklere

⇒ Autonomi = selvorganiserte tverrfaglige team

⇒ Korte iterasjoner

⇒ Hyppige leveranser til produksjoner

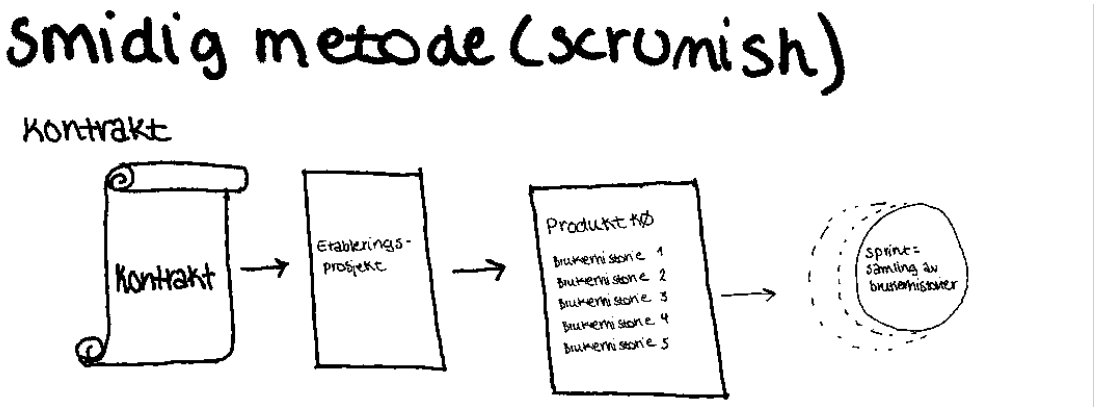
⇒ Beslutninger tas så seint som mulig

⇒ Evaluering, læring og forbedring underveis

FORDELER OG ULEMPER SMIDIG

+Rask igangsetting, lite ressursbruk på kravspesifikasjon og endringshåndtering, fleksibilitet, brukerinvolvering, læring underveis, kompetanseoverføring og effektivitet.

-Begrenset ansvarliggjøring av leverandør for resultat og budsjett.



PERFEKT VALG

Den perfekte kontrakten for programvareutvikling avhenger av bruksområde, men vil kunne være

◊ «Fossefall»: når resultatet er klart spesifisert, og omfang og kompleksitet er begrenset

◊ «Seriefossefall»: når resultatet kan defineres på overordnet nivå og partene er enige om å følge en avtalt gjennomføringsmodell hvor resultatet i hver delleveranse spesifiseres underveis

◊ «Ressurskjøp»: når det skal utvikles etter smidig metode etter resultater av andre grunner ikke er klart definert

Testing

Utforske et system/systemkomponenter for å avdekke feil + det som mangler

P1 Testing viser feil = testing kan bare vise at feil eksisterer, men ikke bevise at et program er feilfritt.

P2 Fullstendig testing er umulig = det er umulig å teste alt. Testing burde prioriteres ut ifra hva man skal teste og hvor stor risiko det er. Da måler man risikoen mellom hvor stor sannsynligheten er for at en feil oppstår (faktor 1) med konsekvensen feilen får (faktor 2).

P3 Tidlig testing = still kritiske spørsmål allerede i begynnelsen: skal vi virkelig utvikle det her? Er det noe poeng? Kravspesifikasjonen bør være godt forstått. Ved gransking av kravspesifikasjon, så er det spesialt mangler og inkonsistens man må være bevist på.

Det er mye som skjer før man begynner å implementere et program (alt fra planlegging, kravspesifikasjon, funksjonelle krav og ikke funksjonelle krav).

- Allerede i planleggingsfasen kan man kartlegge testteknikker som skal benyttes for å kvalitetssikre og fjerne mest mulig feil i startfasen.
- Feil utvikles raskt og i hverandre, og må derfor oppdages tidlig.
- Økonomisk gevinst: avdekke feil under utvikling er ofte rimeligere å håndtere enn når systemet er tatt i bruk.

P4 Feilene i et program er ikke spredt jevnt utover, men har en tendens til å samle seg i enkelte moduler = Fordi deler av koden har fellestrekk.
Hvordan håndtere dette?

- Statisk analyse verktøy (kan avdekke kvalitet i koden):
  - Identifisere steder som det kan være fare for mye feil
  - Avdekke steder med høy kompleksitet i koden

P5 Planteverdmiddelparadoks = om vi tester og bruker de samme testmetodene, fjerner vi bare de type feilene vi har konsentrert oss om.

- Kan finnes andre feil som vi har oversett
- For å overvinne dette paradokset må vi stadig fornye testene våre og finne nye områder som vi tester på. Testene burde varieres og forbedres.

P6 testing er kontekst avhengig = noen programmer må være mer eller mindre feilfri i den grad det er mulig. F.eks.: vil ikke sikkerhetskritiske systemer virke på samme måte som handelssystemer.

P7 «Fravær av feil» - fellen = et feilfritt system er ikke nødvendigvis et brukbart system. Det må valideres og verifiseres.

- Validering: bygger vi det riktige produktet?
- Verifisering: bygger vi produktet riktig?

V-MODELLEN OG ALLE «NODER» - benyttes enten i en iterasjon eller for hele systemet.

BRUKERKRAV: her setter man opp brukerkravene så fullstendig som mulig – hva skal systemet gjøre og hvordan?

SYSTEMKRAV: settes etter brukerkrav. Hvordan skal brukerkravene settes i bruk og implementeres. Krav til systemet og hva systemet skal gjøres settes. Fra brukerens og systemets synspunkt.

GLOBAL DESIGN: sette opp selve arkitekturen til programmet. Hvilke moduler: frontend, database, programlogikk osv. Hvordan er grensesnittet mellom de ulike modulene og hvordan er de satt sammen?

DETALJERT DESIGN: beskriver mer hvordan koden skal implementeres (ift klasser, klassehierarkier, datastrukturer osv.)

IMPLEMENTASJON: vanligvis benytter man enhetstesting for å teste implementasjonen hele veien ned venstresiden av V-modellen. Man benytter statiske testteknikker for å sikre at alt det som ligger i grunn er i orden før man begynner implementering. Testbasisen for systemet blir all den informasjonen man har og generer. Deretter så implementerer man, men man tester de med komponenttester/enhetstester.

ENHETSTESTING: tester alle delmodulene separat i programmet. Dette baserer seg på detaljert design og beskrivelse av kode.

INTEGRASJONSTESTING: mellom hver modul vil det være et grensesnitt som må testes (eks: hvordan API henger sammen med frontend). Integrasjonstesing baserer seg på global design og man tester grensesnitt mellom ulike moduler.

SYSTEMTESTING: verifiserer hva systemet gjør ift beskrivelse i systemkrav. Tester funksjonaliteten til programmet og hvordan hele systemet oppfører seg.

AKSEPTANSETESTING: utføres av sluttbruker eller representanter for det kan hente at bruker har noen krav til brukervennlighet, responstid osv. Målet et å få tillit til systemet, spesielt når det kommer til de ikke-funksjonelle karasterikkene ved det.

Pilene betyr at når vi har:

1. laget brukerkrav kan vi forbedre akseptansetesting
2. satt opp systemkravene kan vi forbedre systemutesting
3. global design kan vi forbedre integrasjonstestingen
4. detaljert design på plass kan vi forbedre enhetstesting

Statisk og dynamisk testing – hva er forskjellen?

Statisk	Dynamisk
Testing uten at vi kjører programmet under test. Nedgående venstresiden av V-modellen	Krever at vi kjører programmet
Eks: - gransking av testbasis eller statisk analyse som input til et analyseprogram og sjekker kvaliteten	Over i en annen type testing som foregår etter at vi har begynt å implementere, altså på høyresiden av V-modellen.

Testtyper – 4 forskjellige

1. Funksjonell testing: tester hva systemet gjør/skal gjøre, slik at vi har beskrevet det i systemkravene. Verifiserer koden og passer på der. Bekrefter forventet input.

2. Ikke-funksjonell testing: går mer på hvordan systemet gjør det. Har mer med kvalitet og utformingen å gjøre.
3. Strukturell testing: handler om å bruke strukturer i koden eller i testbasis til å beregne hvor mye man har testet.
    a. Man understreker hvor stor testdekning man har (test coverage).
    b. Testing av hvor mange krav i kravspesifikasjonen som er testet
    c. Code coverage – hvor mye kode har vi dekket?
        i. Vi må ha noe tellbart – teller opp hvor mye vi kan teste med henblikk på et eller annet også ser vi hvor stor andel vi virkelig har testet
        ii. Har vi bare testet halvparten har vi bare 50% dekning. Selv om vi tester alt betyr det ikke at programmet er feilfritt.
4. Endringsbaserte: disse skjer mer med forandringer i programmet.
    a. Confirmation testing: sjekker om det oppstår feil og fikse
    b. Regression testing: går ut på å teste hele eller deler av systemet

Selv om en liten feil oppdages på hele systemet testes!

Testnivåer – 4 forskjellige

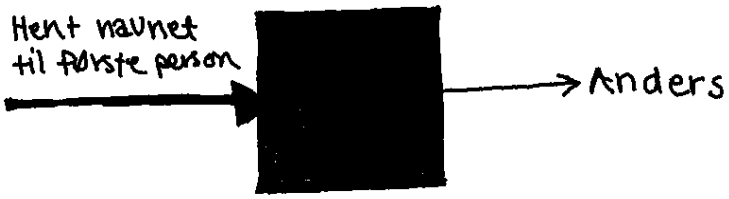
1. komponenttesting/enhetstesting: når vi har testet de ulike modulene separat så
2. integrasjonstesting: tester vi grensesnitt mellom de ulike moduler/interaksjonen med andre systemer. Når alt fungerer så..
3. systemtesting: tester og verifiserer at programmet gjør det som er systemkravet
4. akseptansetesting: vi tester og bruker fra bruker perspektiv. Sjekker om mer må tillegges. Oppfyller systemet det systemkravet etterspør.

Testteknikker – 2 hovedgrupper

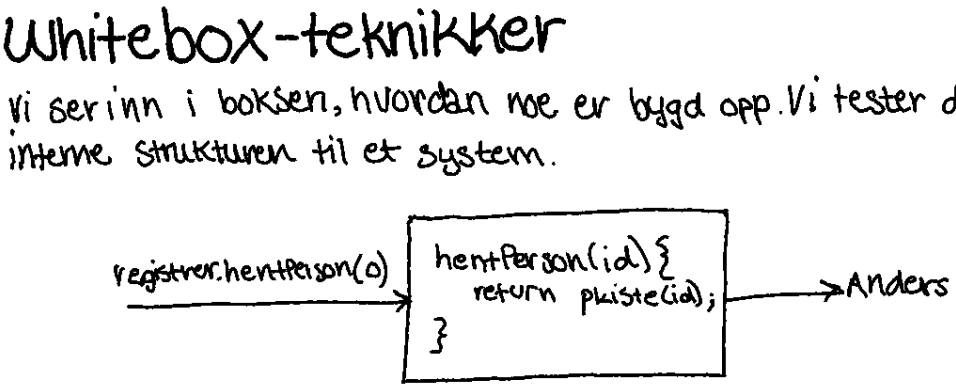
Teknikker vi kan bruke for å lage testene.

1. statiske: tester kode uten å kompilere den
    a. reviews: granskinger
    b. statisk analyse (av kilde og generert output)
        i. ubrukte variabler, referanser til variabler med udefinerte verdi, brudd med kodestandarder, deadlocks (vranglås). Sårbarhet mtp sikkerhet.
2. Dynamiske: krever at vi kjører programmet (kompilerer koden)

a. Blackbox: spesifikasjonsbaserte
    i. Tester funksjonaliteten til en applikasjon. Vi ser på hva vi sender inn til en modul også måler vi det som kommer ut. Vi ser ikke på koden som ligger bak.



b. Whitebox: strukturbaserte
    i. Vi ser inn i boksen, hvordan noe er bygd opp. Vi tester den interne strukturen til et system.



Brukervennlighet

Går på hvor brukervennlig programmet er: hvor lett det er å lære seg å kjenne seg igjen, hvor tilfreds brukeren er under bruk.

Aksessbarhet

Går mer på at programmet er tilgjengelig for flest mulig folk. Det går mer på i hvilken grad sluttbruker har mulighet til å bruke programmet. Er applikasjonen ment for alle?

Brukervennlighet forutsetter aksessbarhet! Det er bare når vi har tilgang til programmet vi kan snakke om brukervennlighet.

UKE 8 Forskningsmetoder

Oppgave ①
Tenk deg at du er i et firma som utvikler programvare der utviklingssjefen har foreslått å innføre bruk av UML fordi han har lest at "forskning viser at det er nyttig". Samtidig innvender en av sjefsarkitektene at hun har hørt at "forskning viser at det er bortkastet å bruke tid på å lage UML-diagrammer".

② Hvordan har det kommet seg at forskning har kommet frem til tilsynelatende motstridende resultater?

① Ulike avhengige variabler:
   a) noen kan ha målt forståelse av kode, andre antall feil.
   b) Tid er tatt med som parameter noen steder, ikke alle.

② Studiene som ligger bak kan ha fokusert på ulike utviklingsfaser, for eks utvikling eller vedlikehold.

③ studiene kan ha vært utført i svært ulike omgivelser, for eksempel ulik størrelse på bedrifter og prosjekter, og kan ha hatt forskjellige applikasjonsområder (systemer)

④ Det ene systemet på UML kan ha basert seg på en enkeltstudie, men det andre kan ha basert seg på en sekundærstudie, for eks meta-analyse av resultater fra mange enkeltstudier. Det siste synet vil sådeles være mer pålitelig.

b) Lag et forslag til design av et eksperiment for å evaluere effekten av UML i dette firmaet.

For eksempel:

- Uavhengige variabel / Treatment: MED UML vs UTEN UML
  - Gi utviklere en oppgave som er å endre en del av et system (kode) som ikke er så stor, men gjerne litt kompleks. Del utviklerne i to: den ene gruppen har tilgang til en UML-modell som beskriver hva koden gjør, den andre har det ikke.
- Avhengige variabel (resultat):
  - Mål tiden de bruker og sjekk kvaliteten på arbeidet. Definer på forhånd hvordan kvalitet skal måles.

c) Lag et design av en case studie for å evaluere effekten av UML i dette firmaet

FOR EKSEMPEL:

Man kan velge ett (mindre) prosjekt som pilot og la folk bruke UML der. Også intervjuer man prosjektmedlemmene om erfaringene. Bør også prøve å få noen andre stakeholder til å vurdere hvordan prosjektet gikk.

Når man i dette tilfellet skal ta en beslutning angående bruk av UML er det viktig å vektlegge de studiene som virker mest relevante i forhold til dette firmaet.

Slakk: aktiviteter som ikke trenger å starte med engang uten at det går utover total prosjekttid. Hvordan finne slakk? = større sett med oppgaver med avhengigheter til hverandre. Hvordan vet du hvilke aktiviteter du burde starte med?

PERT-DIAGRAM

- Hvilke oppgaver som må startes umiddelbart for ungå forsinkelser.
- Hvilke oppgaver som kan utsettes uten å skape forsinkelser.
- Hvor lenge oppgavene kan utsettes

PERT = Program evaluation and review technique

1. Sett opp aktivitene fra start til slutt og avhengigheter med piler
2. angi varighet for hver aktivitet
3. beregn tidsbruk:
    a. nåværende aktivitet + samlet tid for stien som ledet hit
4. gjenta steg 3 til alle stier er dekket
5. finn kritisk sti → aktiviteter som ikke inngår i kritisk sti kan utsettes, de har slakk!

Utvikler: ansvar for å løse sine oppgaver

Prosjektleder: ansvar for å få andre til å løse oppgavene de har fått tildelt

Utvikler:	Prosjektleder:
• Implementasjon • Testing • Integrasjonsarbeid • Kildekodehåndtering • Installasjon og drift	• Holde oversikt over fremgang • Delegere arbeidsoppgaver • Kommunikasjon på tross av fagfelt • Tilrettelegging for systemutviklere Nøkkelpunkter: - Konsekvent = ansatte bør behandles på en sammenliknbar måte - Respekt = ulike mennesker har ulike forutsetninger - Inkludering = mennesker stimuleres på ulik måte - Ærlighet = konstruktiv kritikk og ærlig tilbakemelding

Læreboken og forelesningen beskriver tre personligheter som er knyttet til motivasjon:

+oppgaveorientert: motiveres av arbeidet de gjør, setter pris på utfordring

+sosialt orienterte: motiveres av sosial stimulans og medmenneskelige relasjoner

+målorientert: motiveres av personlig suksess og langsiktig karriereprogresjon

Gruppedynamikk = sosialpsykologisk studium av hvordan grupper fungerer.

Smidige prinsipper som understøttes av scrum master

⇒ Individer og interaksjon → fremfor prosesser og verktøy (5) (6)

⇒ Samarbeid med kunden, fremfor kontraktsforhandlinger (1) (2)

⇒ Kvalitetssikring ved å gi gruppen stor frihet (11)

⇒ Refleksjon av arbeidsprosess og progresjon (12)

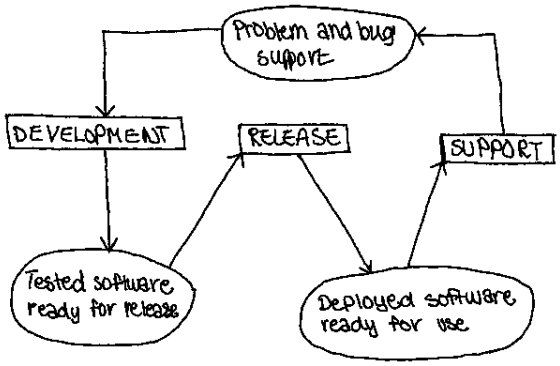
Retrospektiv

Scrum teamet. Samles for å diskutere siste sprint (12)

Formålet: reflektere over hvordan utviklingen går/identifisere forbedringsmåter

1. hva bør man slutte med?
2. hva bør man fortsette med?
3. hva bør man starte med?

DEVOPS – deployment, release and support



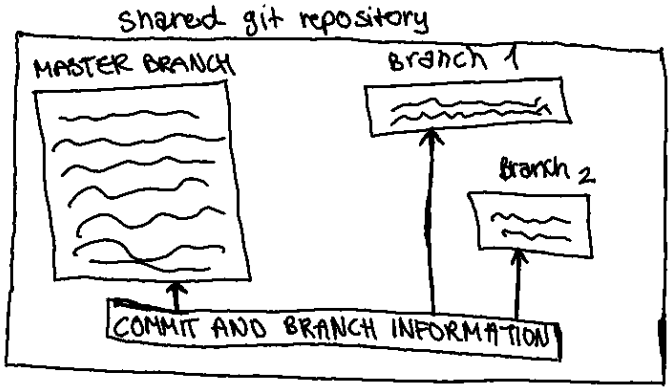
3 prinsipper

Alle er ansvarlig for alt	Alle lagmedlemmer har delt ansvar for utviklingen, levering og støtting av systemet
Alt (som kan) bør være automatisert	Det burde være minimalt med manuelt arbeid i deploying software
Mål først, endre senere	Should be driven by a measurement program where you collect data about the system and its operations. You then use the collected data to inform decisions about changing DevOps process and tools.

FORDELER: raskere deployment, reduced risk, faster repair og more produkctive teams.

Github

Et fundamentalt konsept i GIT er «master branch», som er den nåværende master versjonen av systemet som temaet jobber på. Du lager nye versjoner av å lage en ny branch.



Når brukerne forspør en reposiroty klone får de en kopi av master branch som de kan jobbe på individuelt. Git og andre distribuerte kodehåndteringssystemer har mange fordeler over sentraliserte systemer.

1. resilince: alle som jobber på et prosjekt har egne kopier av repoet
2. speed: committe endringer går fort
3. flexibility: du kan lett gjøre mange endringer og tester

Github er et code managment som DevOps kan bruke

Devops tolkes forskjellige avhengig av selskap – kommer ann på kulturen og programvare de utvikler. Men de 3 hovedprinsippene er «basen» for å følge effektiv devops.

Alle i teamet har et felles ansvar når det kommer til utvikling, leveranse og support.

Siden alle er ansvarlige for alt så er det essensielt å bruke et kodehåndteringssystem (code managment) når man er flere utviklere for å kunne kontrollere koden som styrer programvaren. Med et håndteringssystem kan man jobbe parallelt, unngå å skrive over viktig kode, se endringshistorikk, legge til funksjonalitet uten å endre master koden, dobbeltsjekke og teste før man legger til endringer i masterkoden og håndtere lagring.