Uke 13

IN1030 – Gruppe 5 og 6

Dagens plan:

- Repetisjon
  - DevOps
- Ukesoppgaver
- Obligjobbing

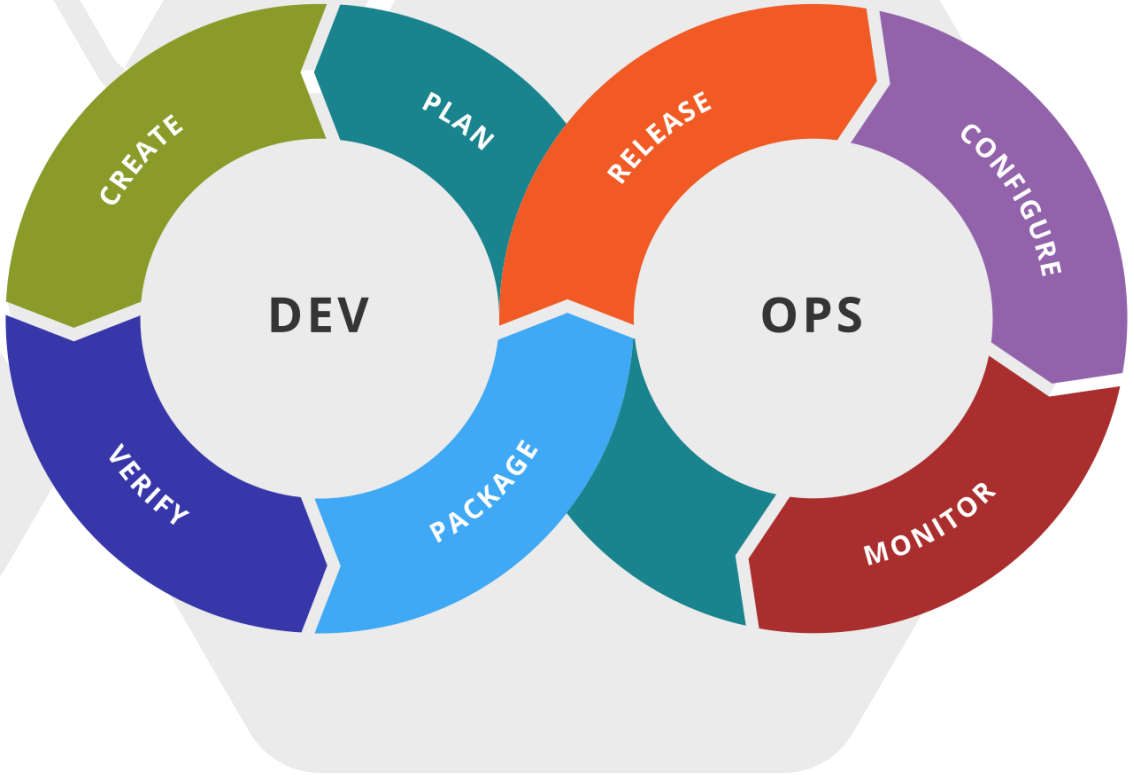
Mailadressene våre er
tara.soderholm@jus.uio.no og **mysc@uio.no**

Våre navn (til kontakt gjennom teams etc.) er
Tara Soderholm og My Schultheiss

DevOps (Development & operations)

Hvorfor benyttes DevOps?

- Vi trenger ny programvare hurtig.
  - Én person alene bruker mye lengre tid på å lage noe enn et team som jobber sammen
    - Dersom alle jobber med samme mål (utgivelse) går prosessen mot målet (utgivelsen) fortere



Hva er DevOps?

Produktutvikling og vedlikehold har vanligvis bestått av flere team:

- Utvikling
- Utgivelse
- Support
- Vedlikehold

Dette kan skape problemer:

- Forsinkelser i kommunikasjon mellom teamene og løsning av problemer
- Bruk av ulike verktøy, ulike ferdigheter og manglende forståelse av andres problemer

Devops er en alternativ modell som integrerer alle teamene ovenfor i ett

- Fordi alle organisasjoner implementerer modellen ulikt finnes det ingen klar definisjon på hva den inneholder/er

Hva er DevOps?

«DevOps is a set of practices that combines software development (Dev) and IT operations (Ops). It aims to shorten the systems development life cycle and provide continuous delivery with high software quality»

Altså:

- Et sett «praksiser»/handlinger hvor en kombinerer programvareutvikling og IT-drift
  - For å korte ned systemutviklingsprosessen og sørge for kontinuerlig levering med høy kvalitet i programvaren

DevOps prinsipp	Forklaring
Alle ansvarlige for alt	Alle i teamet har delt ansvar for utvikling, utgivelse og vedlikehold/support av programvaren.
Alt som kan bli automatisert burde bli det	All testing-, utgivelse- og supportsaktiviteter bør bli automatisert når det er mulig. Legg opp til minst mulig manuelt arbeid med utgivelsen av programvaren.
Mål først, endre etterpå	DevOps burde bli drevet av målingsprogram hvor du samler data om systemet og operasjonene. Avgjørelser som angår endring i DevOps prosessen og verktøy, bør tas med denne dataen som grunnlag.

Kodeadministrasjon

- Kodeoverføring
  - Utviklere tar kode inn i sitt personlige fillager for å jobbe med det, for så å returnere det nye arbeidet til kodeadministrasjonssystemet
- Versjonslagring og henting
  - Filer kan lagres i ulike versjoner, og spesifikke versjoner kan hentes ut
- Merging/branching
  - Parallellutviklingsgrener kan opprettes slik at flere kan arbeide med det samme samtidig. Endringer gjøres i ulike branches som deretter merges hvis de blir godkjent
- Versjonsinformasjon
  - Informasjon om de ulike versjonene i systemet kan lagres og hentes ut igjen

Automatisering

- Kontinuerlig integrasjon
  - Kodeendring og testing
- Kontinuerlig testing
  - Brukssimulering og testing
- Kontinuerlig utgivelse
  - Oppdateringer når masterbrancher endres
- Infrastruktur som kode
  - Maskinprosessbar kode, ingen manuelle oppdateringer av software

Diskuter:

Hvilke fordeler får vi ved å gjøre integrering, levering og utgivelse automatisk?

Fordeler ved automatisering

- Redusert kostnad
  - kontinuerlig utgivelse er mer hurtig og effektivt enn manuell – som ofte også feiler
- Hurtigere problemløsning
  - Problemer påvirker kun en liten del, fordi ting oppdateres kontinuerlig – noe som forenkler prosessen om å finne årsaker til feil
- Hurtigere kundetilbakemeldinger
  - En kan identifisere forbedringer gjennom å stadig gi ut nye funksjonaliteter
- A/B testing
  - Noen (brukere) bruker gammel- og andre ny versjon, slik at vi kan sjekke hva som fungerer best

Mål

- **Prosessmetrikker:**
- Gjennomsnittstid på gjenopprettelse
- Prosentandel feilutgivelser
- Utgivelsefrekvens
- Endringsvolum
- Forsinkelse fra utvikling til utgivelse

Mål

- **Prosessmetrikker:**
- Gjennomsnittstid på gjenopprettelse
- Prosentandel feilutgivelser
- Utgivelsefrekvens
- Endringsvolum
- Forsinkelse fra utvikling til utgivelse

Diskuter:

Hvilke tilpasninger kan man gjøre for å redusere klager?

Eller for å øke prosentandel nye sluttbrukere?

Mål

Tjenestemetrikker:

- Prosent økning i sluttbrukere
- Antall klager
- Tilgjengelighet
- Ytelse

Tjenesteendringer:

- Markedsføring
- Universell utforming
- Brukerundersøkelser
- Kode for robusthet
- Endre systemarkitektur

Diskuter:

Hvilke ulemper finnes med kontinuerlig utgivelse/lansering av ny programvare? Gi gjerne eksempler

Kontinuerlig utgivelse/lansering - obsobs

Ukomplette funksjonalitet - lekkasje av geniale ideer.

- Irriterte kunder som ikke oppdaterer fordi de må gjøre det så ofte.
  - F.eks.: Mac-oppdatering, Chrome-oppdatering.
- Synkronisering av bedriftenes (kundens) ønsker/behov og dine utgivelser - husk på det.
  - F.eks.: lønnssystem til UiO

PAUSE

Ukesoppgaver

1. Forklar hvorfor å ta i bruk DevOps gir et grunnlag for mer effektiv programvaredistribusjon og drift
2. Forklar hvorfor det er viktig å bruke et kodestyringssystem når flere utviklere er involvert i å lage et programvaresystem. Hva er fordelene med å bruke et kodestyringssystem hvis kun én enkelt utvikler er involvert?
3. Hva er forskjellene mellom kontinuerlig integrasjon, kontinuerlig levering og kontinuerlig distribusjon

Ukesoppgaver

1. Forklar hvorfor å ta i bruk DevOps gir et grunnlag for mer effektiv programvaredistribusjon og drift

DevOps gir hurtigere og bedre utgivelser fordi – når alle er med på alt – reduserer det tiden man bruker på å kommunisere, man kan sette alle til å fikse bugs, alle i teamet har like stort eierskap og forståelse av hva som er lurt og ikke, og små endringer i hver utgivelse gjør fatale feil mindre sannsynlig. Automatiseringen gjør også at man kan jobbe effektivt med de viktigste arbeidsoppgavene.

Ukesoppgaver

2. *Forklar hvorfor det er viktig å bruke et kodestyringssystem når flere utviklere er involvert i å lage et programvaresystem. Hva er fordelene med å bruke et kodestyringssystem hvis kun én enkelt utvikler er involvert?*

En kan kontrollere koden som styrer programvaren. Med et håndteringssystem kan man jobbe parallelt, unngå å skriv over viktig kode, se endringshistorikk, legge til funksjonalitet uten å endre master-koden, håndtere ulike versjoner av koden, dobbeltsjekke og teste før man legge til endringer i masterbranch og håndtere lagring.

Det samme gjelder fordelene når en jobber alene. Det er lurt å kunne se tidligere versjoner av egen kode slik at en kan backtracke, rette opp i- og unngå feil.

Ukesoppgaver

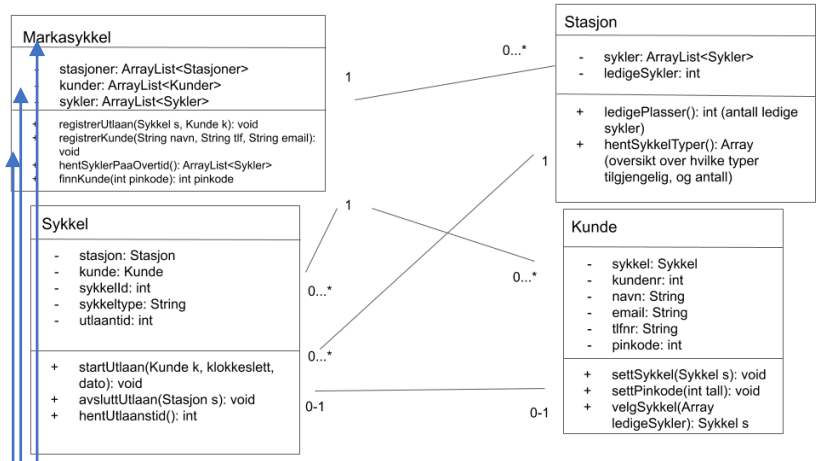
3. Hva er forskjellene mellom kontinuerlig integrasjon, kontinuerlig levering og kontinuerlig distribusjon?

Kontinuerlig integrasjon: Kodehåndtering. Integrerer ny, ferdig testet kode i programmet etter hvert som utviklere ferdigstiller koden. Koden skal være så oppdatert som mulig hele tiden.

Kontinuerlig levering: Å sørge for at programmet alltid er klart for å slippes (lansering). En tester både selve programmet og funksjoner samt kodesnutter kontinuerlig. Masterbranch skal alltid være ferdig testet, klar for levering

Kontinuerlig distribusjon: En skal alltid i prinsippet kunne rulle ut programmet til brukerne. Derfor tester en programvaren automatisk på de maskinene og plattformene den skal kjøres i.

Klassediagram

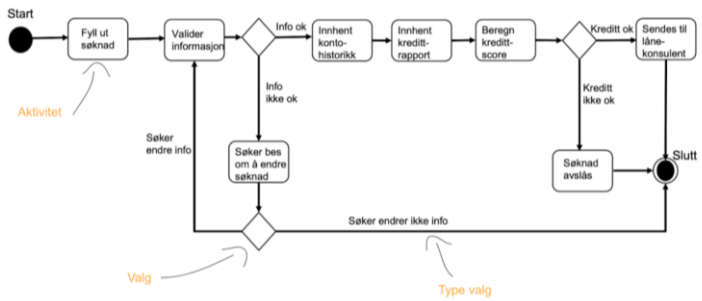


Et klassediagram beskriver strukturen til et system ved å vise systemets

- klasser, deres attributter
- operasjoner
- forholdet mellom objekter

- Det øverste «rommet» inneholder navnet på klassen.
  - Den er som regel skrevet med fet skrift og sentrert, og den første bokstaven er stor.
- Midtrommet inneholder klassens attributter.
  - De er venstrejusterte og den første bokstaven er liten.
- Det nederste rommet inneholder operasjonene klassen kan utføre.
  - De er også venstrejustert og den første bokstaven er liten.

Aktivitetsdiagram

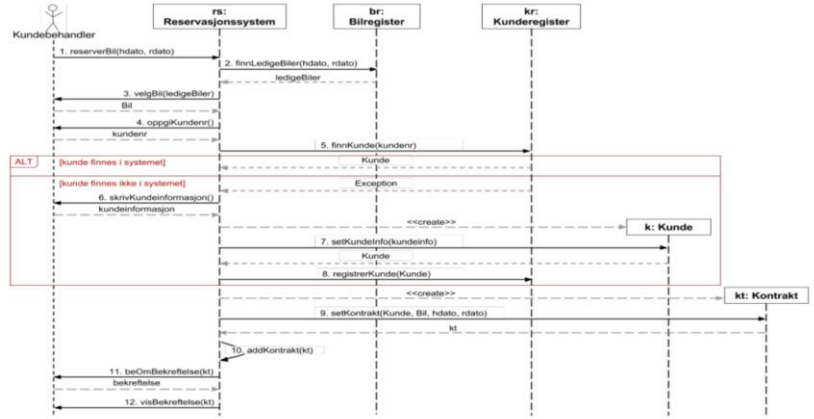


- Et aktivitetsdiagram kan grafisk representere hendelsesflyten i et use case
- Stegene i use case vises som aktiviteter (rektangel)
- Beslutninger undervises som (diamant)
- Aktivitetsdiagrammer og sekvensdiagrammer brukes noe overlappende, men sekvensdiagrammer er typisk mer kodenært mens aktivitetsdiagrammer er mer forretnings nært

Aktivitetsdiagram = flytskjema(flowchart)

- Grafisk representasjon av arbeidsflyt
- Viser aktiviteter og tilhørende handlinger
- Viser overordnet kontrollflyt
- Beskriver hvordan, ulike utfall av en aktivitet påvirker flyten o Viser aktiviteter som kan utføres parallelt

Sekvensdiagram



- Flyt i et use case kan modelleres med sekvensdiagrammer
- For hvert use case lages typisk sekvensdiagrammer for hovedflyt og for hyppig forekommende alternativ flyt
- Stegene (sekvensene) i et use case vises som meldinger som sendes mellom objektene ved kall på objektenes metoder

Et sekvensdiagram er et interaksjonsdiagram

- Modellen viser en interaksjonssekvens mellom objektene som finner sted under et bestemt use case
  - Hvilke objekter som inngår i et bruksmønster
  - Interaksjonen mellom objektene/deres rekkefølge o Data/informasjon som sendes mellom objektene

Desto mer detaljert den tekstlige beskrivelsen er, desto enklere blir det å modellere det tilhørende sekvensdiagrammet. Tekstlig beskrivelse viser interaksjon mellom bruker og systemet

- Gir oss informasjon om hvem(bruker) og hva (objekter/metodekall/data)
- Følg rekkefølgen som gis fra beskrivelsen
- **Tips!** Lag en tekstlig beskrivelse (for bruksmønsteret) om denne ikke er gitt
  - Beskriv hendelsesforløpet i detalj → vis interaksjonen
- Fra beskrivelsenåkartlegg aktører og objekter
  - Følg oppsett gitt av rekkefølgen i beskrivelse
  - Modeller flyten (piler frem og tilbake) med dette som utgangspunkt

OBLIGJOBBING