Kim Benfaddoul, Dmitry Filosofov, Magnus Klausen, Cindy Madeleine Svendsen Ceesay.
Obligatorisk innlevering 3

Oppgave 1A

Størrelsen på systemet kan måles i funksjonspoeng. Vi tar utgangspunkt i funksjonelle krav, identifiserer og kategoriserer hver funksjon, som så vurderes og tilordnes et antall funksjonspoeng. Vi må anta at kravspesifikasjonen er utferdiget på forhånd. Fordelen ved denne måten å måle på, er at det kan gi oss et størrelsesmål på et tidlig tidspunkt i prosjektet. Flere argumenterer også for at funksjonspoeng er et mer presist størresesmål enn for eksempel antall linjer kode. Det finnes også tabeller spesifikke for enkelte programmeringsspråk, som konverterer det ene til det andre.

Oppgave 1B

Kompleksiteten til prosjektet kan måles ved å ta utgangspunkt i fem forskjellige kriterier:

- *Precedentedness,* som avspeiler organisasjonens tidligere erfaring med liknenende prosjekter.
- Utviklingsfleksibilitet, som avspeiler graden av fleksibilitet i utviklingsprosessen.
- Arkitektur/risikoavklaring, som avspeiler hvor omfattende risikoanalyse som er blitt utført.
- Lagsamhold, som avspeiler hvor godt utviklingsteamene kjenner hverandre og jobber sammen.
- Prosessmodenhet, som avspeiler prosessmodenheten i organisasjonen (jf. for eksempel CMM Maturity Questionnaire).

Hvert av kriteriene tilordnes en tallverdi mellom null og fem, hvor null representerer svært høy, og fem representerer svært lav. Deretter beregnes en variabel *b* ved å legge sammen verdiene, dividere summen på 100 og legge kvotienten til 1,01. Variabelen *b* er relatert til kompleksiteten, og inngår som eksponent (til størrelsen) i forskjellige formler for å beregne arbeidsmengden.

Oppgave 2A

4+1 view-modellen ble designet av kanadieren Philippe Krutcher i 1995. Den beskriver arkitekturen ved programvaresystemer ved hjelp av fire såkalte views, som kan forstås som ulike interessenters synsvinkler på systemet.

- *Logisk view* viser hovedabstraksjonene i systemet som objekter eller objektklasser.

- *Prosessview* viser hvordan systemet under kjøring består av forskjellige prosesser som interagerer med hverandre.
- *Utviklingsview* viser hvordan systemet lar seg dele opp i atskilte komponenter som en enkel utvikler eller et utviklingsteam implementerer.
- *Fysisk view* viser systemets maskinvare og hvordan systemets komponenter er fordelt på prosessorer.

Oppgave 2B

4 + 1-modellen gjør at det er enklere å tilfredsstille alle interessenter i prosjektet. Det gjør modellering av systemet blir enklere fordi 4 + 1-modellen gjør det enklere å organisere.Forskjellige views i 4 + 1 gjør at det er lettere å organisere alle diagrammer etter formål.

Denne modellen lar arkitekten prioretere ting som må gjøres først. Siden i store prosjekter er det ofte slik at det ikke er nok tid for å tegne hver enkelt diagram, så ved å bruke 4 + 1 modellen er det da mulig for arkitekten å prioritere hvilke diagrammer som bør tegnes først og hvilke deler kan bli utsatt til senere tid eller ikke tegnes i det hele tatt.

Oppgave 2C

1. 3-lags logisk arkitektur

Grensesnitt/presentasjonslag
Logikk - prosessering av bruker interaksjon/business logikk
Datalag - lagring av systemdata

2. 4-lags fysisk arkitektur

Klient - mobilapp/webside/biletterminal
Webserver
Applikasjonsserver
Database

Oppgave 2D

Fordeler:

- Lagene former et abstrakt hierarki
- Hvert enkelt lag kan byttes ut uten å gjøre store endringer i resten av systemet.
- I tilfelle med lukket arkitektur, så kan endringer i et lag påvirke maks 2 andre lag, som minimerer avhengighet mellom lag.
- I tilfelle med åpen arkitektur er koden mer kompakt hvis funksjonalitet i lavere lag kan brukes direkte.

Oppgave 2E

Ulemper:

- Er vanligvis noe tregere siden all kommunikasjon må gå gjennom alle lag i stedet for å gå direkte til det nødvendige stedet (Lukket arkitektur).
    - Fordi i en 'lukket lag arkitektur' må lagene bare snakke med nærliggende lag. F.eks Klient til App-server etc
- I tilfelle med åpen arkitektur blir avhengighet mellom lag større og derfor brytes innkapsling av hvert enkelt lag.

Oppgave 2F

Klient-server

Systemets funksjonalitet er delt opp i tjenester. Hver tjeneste er levert fra sin egen server. Det at det er relativt lett å skalere systemet horisontalt gir det mulighet for tilgang fra flere steder samtiding. Ulempe med den type arkitektur er at det er lite redundant og derfor er sårbar til DDoS/DoS attakk.

MVC

MVC står for Model View Controller. Denne modelen separer presentasjonslag fra systemets lag hvor systemdata er håndtert.
Der hvor *modell* har ansvar for kommunikasjon med databasen, er *controller* ansvarlig for håndtering av data og businesslogikk og *view* har ansvar for UI som er vist til brukeren.

SOA

SOA kalles for Tjenesteorientert arkitektur på norsk. Det hjelper med å redusere kostnader ved å drive veldig store systemer. SOA benytter tjenester som er uavhengig av programvaren som bruker dem. Fokuserer veldig mye på "loose coupling" dvs. at det er veldig lite avhengighet mellom komponenter og forskjellige tjenester noe som gjør at det er veldig lett å bytte og videreutvikle. Alle tjenester som

regel utfører bare en funksjon. Tjenester bruker veldefinerte protokoller og standarder for å sende beskjeder mellom tjenester (f.eks. http/s er ofte brukt).

Pipe and Filter

Pipe and filter er veldig robust, men samtidig relativt enkel arkitektur. Denne arkitekturen består av filtere som transformerer eller filtrerer data og rør som sørger for å transportere data fra filter til filter. Filtere skal jobbe parallelt, de behøver ikke å vite noe om hva de er koblet til.

Oppgave 3A

Tilstandsdiagrammer er en del av UML 2.0 og er en grafiske representasjon av de ulike tilstandene i en tilstandsmaskin. Systemet beskriver hvilke eksplisitte hendelser som får systemet til å endre tilstand. Det gir dermed en lett oversikt over hva slags ulike hendelser som kan oppstå i systemet, og gjør det lettere for oss å oppdage dealocks. Diagrammet er også nyttig, da det viser hva slags tilstand systemet er i. Tilstandsdiagrammet er også nyttig da vi vet hva som skal til for at vi går fra en tilstand til en annen.

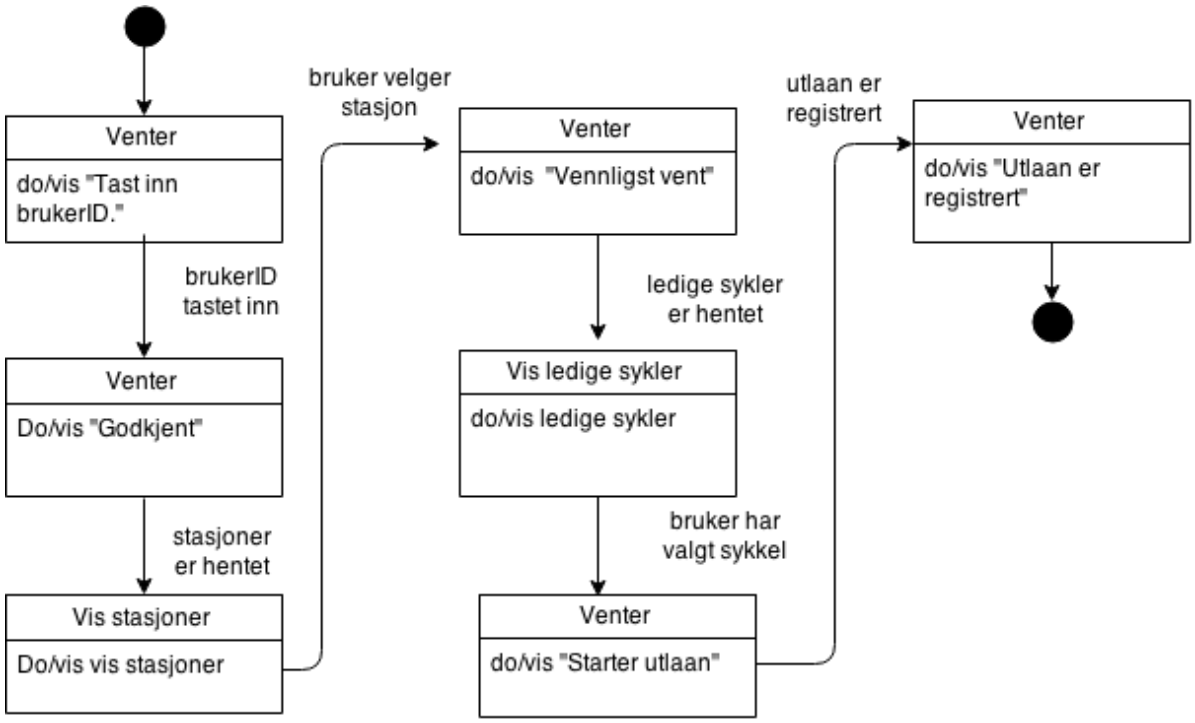
Tilstandsdiagramet består av to-delte bokser. Øverst i boksen står det hva slags tilstand maskinen er i. I den nederste delen beskriver hva slags handling som utføres. Om det står entry foran handlingen skal handlingen starte med en gang tilstanden til systemet endrer denne tilstanden. Om det står do, utføres handlingen mens dette er tilstanden i systemet. Om det derimot står exit utføres handlingen når systemet går ut av tilstanden. Mellom de ulike tilstandsboksene er det piler. Disse viser hva slags tilstand som er neste. Ellers starter diagrammet med en svarte rundinger, og avslutter med en svart runding.

Oppgave 3B

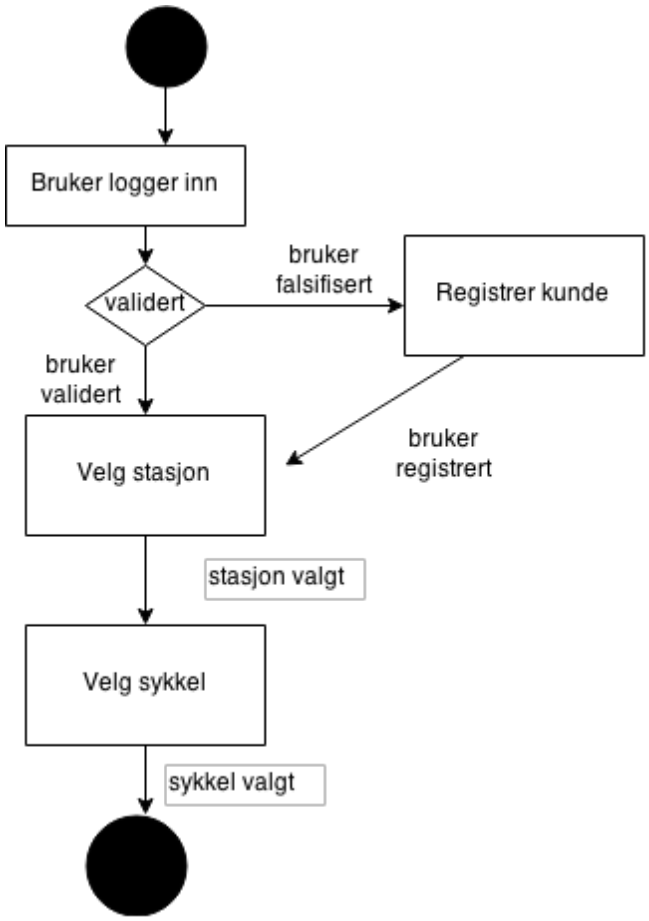
Aktivitetsdiagram er en grafisk representasjon av en arbeidsflyt, som ligger som en standard til UML 2.0. Den beskriver mulige utfall av hvordan en aktivitet påvirker flyten, og hva slags aktiviteter som kan utføres parallelt.

Diagrammet starter og avslutter med en svart runding. Den inneholder også bokser, som viser hva slags aktivitet som utføres. Mellom hver boks følger det piler, som viser sammenhengen mellom de ulike handlingene. Diagrammet inneholder også valgdiamanter, som beskriver hvordan handlinger påvirker systemet.

Oppgave 3C


Venter
do/vis "Tast inn brukerID."
brukerID tastet inn
Venter
Do/vis "Godkjent"
stasjoner er hentet
Vis stasjoner
Do/vis vis stasjoner
bruker velger stasjon
Venter
do/vis "Vennligst vent"
ledige sykler er hentet
Vis ledige sykler
do/vis ledige sykler
bruker har valgt sykkel
Venter
do/vis "Starter utlaan"
utlaan er registrert
Venter
do/vis "Utlaan er registrert"

Oppgave 3D



Oppgave 4A

Enhetstesting innebærer å teste programkomponenter, som metoder og objektklasser, enkeltvis. Testene omfatter både normal bruk for å se om komponentene oppfører seg som forventet – og unormal bruk for å se hvordan komponentene håndterer unormale inndata. Enhetstesting gjøres ofte ved hjelp av et automatiseringsrammeverk (for eksempel JUnit), som både skriver og kjører testene uten behov for manuell innblanding.

Integrasjonstesting eller *grensesnitttesting* innebærer å teste sammensatte programkomponenter, altså objekter som interagerer med hverandre. Hensikten er å undersøke om grensesnittene oppfører seg i henhold til spesifikasjonen. Grensesnittfeil er blant de vanligste typene feil i komplekse systemer. Integrasjonstesting fordrer at enhetstesting av objektene som utgjør komponenten allerede er fullført.

Systemtesting innebærer å integrere alle programkomponenter til en versjon av systemet og å teste det integrerte systemet. Hensikten er å undersøke om komponentene er kompatible og interagerer på riktig vis, samt om riktige data overføres gjennom grensesnittene. Systemtesting overlapper til dels med integrasjonstesting, men innebærer til forskjell at gjenbrukskomponenter og/eller tredjepartkomponenter integreres med komponentene som nylig er utviklet. I tillegg gjennomføres systemtesting typisk som et samarbeid mellom utviklerne, som hver for seg eller i team har skapt de forskjellige komponentene.

Akseptansetesting innebærer at kunden tester systemet og avgjør om produktet er godt nok – om det kan aksepteres. Testene bør omhandle både funksjonelle og ikke-funksjonelle egenskaper ved systemet, og dekke kravspesifikasjonen i størst mulig grad. I smidige utviklingsprosesser er akseptansetesting ingen egen aktivitet. Brukeren stiller krav til systemet gjennom brukerhistorier, og definerer tester som avgjør om systemet oppfyller historiene eller ikke.

Oppgave 4B

Delene av systemet som kan enhetstestes, er klassene MarkaSykler, Stasjon, Kunde og Sykkel hver for seg med tilhørende metoder. Siden systemet er såpass lite og oversiktlig, kan vi ta utgangspunkt i at alle klassene på forhånd er ferdigskrevne, slik at avhengigheter mellom objekter ikke forårsaker problemer.

Delene av systemet som kan integrasjonstestes, er for eksempel klassen MarkaSykler i samspill med klassene Stasjon og Kunde, klassen Stasjon i samspill med klassen Sykkel, klassen Sykkel i samspill med Klassene Stasjon og Kunde, samt klassen Kunde i samspill med klassen Sykkel.

Under systemtesting kan hele systemet, det vil si alle klassene integrert med hverandre og eventuelle tredjepartskomponenter (for eksempel fra Ruter), testes sammen. Det gjelder også under akseptansetesting, men da er det kunden (det vil si Oslo kommune) som forestår testingen heller enn utviklerne.