SU Noter

---

1. Unified Process

Giv en beskrivelse af Unified Process (UP) som systemudviklingsmetode. Hvad kendetegner metoden, hvad er de grundlæggende antagelser?

Gør rede for de forskellige faser og discipliner i metoden.

Skitser et systemudviklingsforløb under anvendelse af UP.

Beskriv evt. dine erfaringer med UP fra projektet.

Besvarelse 1: Unified Process

grundlæggende principper

- iterativt arbejde
- modsat vandfaldsmodellen
- vi skal altid igennem business modelling, krav, analyse/design, test osv
- i stedet for at gøre krav helt færdig før vi går videre til analyse arbejder vi med alle ting på en mindre del af systemet (på usecase niveau) i hver iteration

struktur

- 6 discipliner som der arbejdes med i hver af de 4 faser
- disciplinerne fylder forskelligt alt efter fase

faser -

inception

- fastlægge rammer for projektet
- beskrive arbejdsproces
- fastlægge krav
- finde essentielle usecases
- lav evt prototype
- cost benefit, risikoanalyse

elaboration

- design og udvikl grundarkitektur og -system (herunder klassediagrammer)
- forfine krav og udbygge usecase diagram og beskrivelser
- definere ikke funktionelle krav
- forfine risikoanalyse
- udarbejde plan for construction

construction

- færdiggøre analyse og design modeller
- kode systemet færdig

transition

- installation
- uddanne bruger
- betatests
- fejlretning
- sætte systemet i drift
- gennem alle faserne udføres et antal iterationer bla afhængig af antal usecases

discipliner

- business modelling
- forundersøgelse/-analyse
- afdække problemer, visioner for projektet
- forretningsmodeller/-principper
- sikre fælles forståelse mellem udviklere og brugere af systemet - evt protptyper
- requirements
- indsamle information om problemområdet
- definere krav(use cases)
- udvikle dialog og brugergrænseflade
- usecase modeller og beskrivelser
- analysis and design
- analyse af systemets funktionalitet og indhold
- design arkitektur
- design realisering af use cases med klasser og sammenhænge
- design database, gui, sikkerhed og kontroller
- klassediagrammer, adfærdsmønstre, sekvensdiagrammer - prototyper af gui
- implementation
- programmer klasser og sammenhænge fundet i analysis+design
- inkorporer klasser fra klassebiblioteker
- integrer klasser og delsystemer til kørende systemer
- test
- unittest
- integrationstests
- brugervenlighedstests
- brugeraccepttests
- deployment
- uddanne brugere
- installere komponenter
- konvertere og initialisere data

alle faser arbejder med alle discipliner men de første discipliner er mest fremtrædende i de første faser, de sidste discipliner er mest fremtrædende i de sidste faser

eksempler

2sem projekt - FIT delen svarer meget til inception / busines modelling hvor vi lave analyserede virksomheden og markedet for at finde frem til en forståelse af problemet og hvordan vi bedst kunne løse det

iteration 2 var fokus på at udarbejde en kravliste, identificeret de fleste usecases, udarbejdet usecase diagram (requirements) og besluttet en overordnet arkitektur for systemet (3lags modellen) (analysis+design) --> godt eksempel på en iteration i inception fasen!

---

2. Krav og use cases

Gør rede for de forskellige typer krav. Hvordan dokumenteres krav og hvad bruges kravene til? Gør rede for definitionen af en use case samt formålet med at finde dem.

Gør rede for de forskellige niveauer use cases kan dokumenteres på. Vælg to af niveauerne og beskriv dem mere i dybden. Hvad er sammenhængen mellem krav og use cases og hvordan kan det dokumenteres?

Du er velkommen til at inddrage erfaringer med arbejdet med use cases i jeres projekt.

Besvarelse 2:

Funktionelle krav	**Ikke-funktionelle krav**
Hvilke funktionaliteter skal systemet leve op til?	Kvalitetskrav for systemet, dvs. sikkerhed brugbarhed, performance, etc

Find krav ved interview: "Hvad er det du gør, når...?", "Hvordan gøres det...?", "Hvilken information er der behov for, når du gør...?", osv

Alternativt, udvilke workshop(s) hvor bruger/udvikler diskuterer krav.
Krav dokumenteres i en kravliste, funktionelle / ikke-funktionelle, nummering, beskrivelse, prioritet.

Prioritet kan være efter MoSCow-princippet – **M** ust have, **S** hould have, **C** ould have, **W** ould have – det kan også bare være "1,2,3..." eller "lav, middel, høj..."

Krav skal være konkrete og enkle, så det er let at afgøre om de er gennemført i det endelige system. Dette giver let overblik.

Use case definition: "En tidsmæssig afgrænset sekvens af interaktioner med systemet der giver et resultat af værdi for aktøren"

Aktør definition: "En rolle omfattende brugere eller andre systemer, der integrerer direkte med systemet"

Use cases bruges til at illustrerer kravene – den supplerer kravlisten, og fortæller en handling om aktørens interaktion med systemet

Streger mellem use case og aktør viser at aktøren interagerer med systemet.

Use case navn opstilles verb - navneord

CRUD: **C** reate, **R** emove, **U** pdate, **D** elete – kombineres i én use case, fx. CRUD lærer

Include: Use case B er indeholdt i Use case A, med pil fra A til B, sker hver gang der interageres med A - B kan dog godt interageres med uden A

Extend: Udvider en use case - beskriver en speciel handling der sker af og til, når en betingelse er opfyldt - fra B til A, angives i A, hvad der skal opfyldes for at B udføres (kaldes extension point)

Use case beskrivelsen detaljerer interaktionen mellem aktør og system - der kan laves en kort, mellemlang og fuld use case beskrivelse

Kort: en kort tekst som er en udvidelse af overskriften

Mellem: beskriver mere formelt de trin, interaktionen består af – måske med exceptions

Lang: giver et fuldt overblik over hver trin samt andre informationer så som Relaterede Use Cases, Preconditions, Postconditions, Beskrivelse osv.

---

3. Analyse

Gør rede for hvad analyse og specielt den objektorienterede analyse i systemudvikling handler om. Hvilke forskellige UML-modeller skabes under analysen? Vælg et par af modellerne og beskriv dem nærmere mht. formål og hvordan de tegnes.

Du er velkommen til at inddrage erfaringer med arbejdet med de forskellige modeller i jeres projekt.

Besvarelse 3:

Spørgsmål 3:
Analyse
Gør rede for hvad analyse og specielt den objektorienterede analyse.

- Hvad handler analyse om

Analysen: Undersøger den del af virkeligheden som visionen er baseret på.
Problemområdet / domænet er den del af virkeligheden hvor problemerne er konstateret.
Formålet med analysen er at finde ud af hvad systemet skal indeholde, dvs data registrering og hvad anvendelset af systemet er. Derudover en detaljeret beskrivelse af kravene til systemet.

- Analyse modeller
    - System sekvensdiagrammer
        - Systemsekvensdiagrammerne bruges også som use case beskrivelser til at
        - Modellere interaktionen mellem system og aktøren. Deres formål er at
        - Have specielt fokus på at vise hvilke beskeder der er sendt til systemet, hvad og ■ Hvornår der er input og output til og fra systemet.
    - Use case diagram
        - Diagram over systemets use cases og deres relationer til aktøren.
    - Adfærdsmønster
        - Hvordan hændelserne påvirker objekter og hvilke eventuelle tilstande de ender i.
        - Beskrives i et adfærdsmønster.
    - Analyseklassediagram
        - Funde klasser forbundet med strukturene associering, aggregering osv.

---

4. Design

Gør rede for hvad design og specielt det objektorienterede design i systemudvikling handler om. Hvilke forskellige UML-modeller skabes under design.
Vælg et par af modellerne og beskriv dem nærmere mht. formål og hvordan de tegnes. Du kan evt. komme ind på, hvad de 5 GRASP patterns går ud på, og hvordan man arbejder med dem under design.
Du er velkommen til at inddrage erfaringer med arbejdet med de forskellige modeller i jeres projekt.

Besvarelse 4:

Design er en udvidelse af analyse – det giver en opskrift på hvordan et system skal programmeres

Nogle mulige kriterier, der vægtes forskelligt alt efter hvad kunden vil have:

Brugbart: Tilpasningen af systemet til de organisatoriske, arbejdsmæssige og tekniske rammer
Sikkert: Sikringen mod uønsket adgang til systemets data og faciliteter

Effektivt: Udnyttelsen af faciliteterne i den tekniske platform

Korrekt: Opfyldelsen af de opstillede krav

Pålideligt: Opfyldelsen af den krævede funktionalitet med den ønskede præcision

Vedligeholdbart: Omkostningerne ved lokalisering og retning af fejl i det kørende system

Testbart: Omkostningen ved test af systemet i forhold til de opstillede krav

Fleksibelt: Omkostningen ved at ændre i det kørende system

Forståeligt: Besværet for udvikleren ved at skaffe sig overblik over og forstå systemet

Genbrugbart: Anvendeligheden af dele af systemet i andre beslægtede systemer

Flytbart: Omkostningen ved at flytte systemet til andre tekniske platforme

Integrerbart: Problemerne ved at sammenkoble systemet med andre systemer

Hvad handler design om?

At få udvidet analysen, få en "opskrift" på hvordan systemet skal programmeres. Overvejelser om hvordan klasserne skal hænge sammen.

Design kriterier	Mål for
Brugbart	Tilpasningen af systemet til de organisatoriske, arbejdsmæssige og tekniske rammer
Sikkert	Sikringen mod uønsket adgang til systemets data og facilitete
Effektivt	Udnyttelsen af faciliteterne i den tekniske platform
Korrekt	Opfyldelsen af de opstillede krav
Pålideligt	Opfyldelsen af den krævede funktionalitet med den ønskede præcision
Vedligeholdbart	Omkostningerne ved lokalisering og retning af fejl i det kørende system
Testbart	Omkostningen ved test af systemet i forhold til de opstillede krav
Fleksibelt	Omkostningen ved at ændre i det kørende system
Forståeligt	Besværet for udvikleren ved at skaffe sig overblik over og forstå systemet
Genbrugbart	Anvendeligheden af dele af systemet i andre beslægtede systemer
Flytbart	Omkostningen ved at flytte systemet til andre tekniske platforme
Integrerbart	Problemerne ved at sammenkoble systemet med andre systemer

Sammenhæng og forskel mellem analyse og design, indhold af design

Analysen laver overblik i form af analyseklassediagrammer – i designklasssediagram får man typer på attributter (String, int, osv), metoder, og sammenhænge mellem klasse.

Designklassemodellen (tilføjelser og justering i forhold til analyseklassemodellen)

- Metoder, sammenhænge mellem klasser, typer på attributter

GRASP patterns (hvilke patterns og hvad kan de bidrage med)

- *Information Expert:* Identificér hvilken klasse af objekter der har den nødvendige information til at udføre det nødvendige kode.

- *Creator*: Kan der være en klasse der opretter en anden klasse – hvis forbindelsen mellem 2 klasser er hvor en klasse indeholder mange af en anden klasse, og den oprettede klasse kun indeholder en enkelt af den anden

- *High Cohesion*: Hvor meget ansvar har en klasse? Jo mere ansvar en enkelt klasse har, jo højere samhørighed (cohesion) – low cohesion (lav samhørighed) er ønsket

- *Low Coupling*: Hvor stærk et element er forbundet, har information omkring, eller har behov for andre elementer. Afhænger er systemet hvad der er "for højt". For lav kobling gør systemet svær at genbruge, og hvis der sker ændringer er der flere ting der skal ændres

- *Controller*: At anvende en controller – holder styr på metoder og objekter; bruges til at kommunikere mellem GUI og application (sikrer lav kobling)

---

5. Designsekvensdiagrammer og GRASP

Gør rede for hvad et designsekvensdiagram er. Gør i den forbindelse rede for sammenhængen til klasser og use cases. Du kan evt. komme ind på, hvad de 5 GRASP patterns går ud på, og hvordan man arbejder med dem i forbindelse med udarbejdelsen af designsekvensdiagrammerne.

Du er velkommen til at inddrage erfaringer med arbejdet med designsekvensdiagrammer i jeres projekt.

Besvarelse 5:

I designsekvensdiagrammerne modelleres samarbejde og interaktion mellem objekter, der realiserer en use case. Vi kan fokusere på en hel use case, dele af en use case eller enkelt metodekald i en use case osv.

GRASP patterns (hvilke patterns og hvad kan de bidrage med)

- *Information Expert:* Identificér hvilken klasse af objekter der har den nødvendige information til at udføre det nødvendige kode.

- *Creator*: Kan der være en klasse der opretter en anden klasse – hvis forbindelsen mellem 2 klasser er hvor en klasse indeholder mange af en anden klasse, og den oprettede klasse kun indeholder en enkelt af den anden

- *High Cohesion*: Hvor meget ansvar har en klasse? Jo mere ansvar en enkelt klasse har, jo højere samhørighed (cohesion) – low cohesion (lav samhørighed) er ønsket

- *Low Coupling*: Hvor stærk et element er forbundet, har information omkring, eller har behov for andre elementer. Afhænger er systemet hvad der er "for højt". For lav kobling gør systemet svær at genbruge, og hvis der sker ændringer er der flere ting der skal ændres

- *Controller*: At anvende en controller – holder styr på metoder og objekter; bruges til at kommunikere mellem GUI og application (sikrer lav kobling)

---

6. Test

Gør rede for formålet med test.
Gør rede for de forskellige testniveauer og hvordan der arbejdes på de forskellige niveauer.
Gør rede for de forskellige testteknikker du kender.
Gør rede for, hvordan man i praksis planlægger og gennemfører black-box tests.
Du er velkommen til at inddrage erfaringer med arbejdet med test i jeres projekt.

Besvarelse 6:

Formålet med test:

- Validering: laves det rigtige system – laves det system brugerne gerne vil have
- Verifikation: laves systemet rigtigt – opfører systemet sig som vi forventer

Generelt handler test om at

- **Planlægge** de nødvendige tests for hver fase/iteration, planlægge **hvilket system** eller hvilke dele af systemet der skal testes, planlægge **hvad** der skal testes, samt planlægge **hvornår** der skal testes
- **Designe og implementere** test ved at finde et passende og dækkende antal testdata og udarbejde testcases på baggrund af disse
- **Udføre** de forskellige test, og systematisk håndtere resultatet af hver test, dvs. rapportere fejl på en systematisk måde, så rette personer kan følge op

Strategisk niveau: mål og strategi

Taktisk niveau: planlægning

Operationelle niveau: udførelse

Test niveauer:

- Unittest: test af metoder, klasser (som helhed, dvs attributter, metoder, tilstande, osv), komponenter, osv, før de integreres med anden software
- Integrationstest: test af sammenhænge/samarbejde mellem komponenter og klasser
- Systemtest: test om systemet er komplet og korrekt, f.ex at use cases kan gennemføres og virker korrekt
- Brugertest: test af forskellige scenarier af use cases, om de fungerer tilfredsstillende og tilstrækkeligt brugervenligt for brugeren, afhængig af kravspecifikation

Test teknikker (kan bruges mere eller mindre uafhængig af situation, behov, og test niveau):

- Black-box: koden er ukendt, tager i stedet udgangspunkt i beskrivelser og modeller af systemet, komponent(er), metode(r), use case(s), eller andet
    - Identificer gyldig/ugyldig værdiområde for input
    - Opdel de to værdiområder i *ækvivalensmængder*
    - Beskriv *grænseværdierne*
    - Udarbejd testcases
- White-box: koden er kendt, strukturbaseret, dvs tager udgangspunkt i programmets interne struktur
    - *Statementdækning*: hvis 100% statementdækning designes testcases så alle statements gennemløbes min én gang
    - *Forgreningsdækning*: hvis 100% forgreningsdækning designes testcases så alle if/else, for- og while-statements identificeres og rammes
    - *Stidækning*: hvis 100% stidækning designes testcases så alle mulige stier rundt i programmet identificeres og gennemløbes min én gang
    - *Betingelsesdækning*: hvis 100% betingelsesdækning designes testcases så alle sammensatte betingelser afprøves, så alle mulige kombinationer af true/false er dækket ind
- Grey-box: en black-box test, hvor koden er kendt, hvilket nødvendigvis betyder at nogle ting fra white-box bliver blandet lidt i

---

7. Design og arkitektur

Gør rede for hvad design og specielt det objektorienterede design i systemudvikling handler om. Hvilke forskellige UML-modeller skabes under design.

Gør rede for et antal kriterier, der kan have indflydelse på valg af arkitektur.

Gør rede for den lagdelte arkitektur I har lært.

Redegør for fordele og ulemper ved en sådan arkitektur, kom evt. her ind på hvad begreberne kobling og samhørighed betyder, samt hvilket mål vi typisk har med dem.

Du er velkommen til at inddrage erfaringer med arbejdet med den lagdelte arkitektur i jeres projekt.

Besvarelse 7:
Design og arkitektur

- **Hvad handler design om**

Design handler om at få udvidet analysen, så man får en "opskrift" på hvordan et system skal programmeres. Når vi går ind i designfasen, får vi mere fokus på, hvordan vi kan realisere modellen i forhold til designprincipper. Her skal der gøres nogle overvejelser om hvordan klasserne skal hænge sammen. For at kunne træffe nogle gode valg om logikken i systemet skal man have nogle kriterier at designe efter. Disse kriterier nedenfor vurderes i et skema med henholdsvis "Meget vigtig", "Vigtig", "Mindre vigtig" eller "Irrelevant".

Design kriterier	**Mål for**
Brugbart	Tilpasningen af systemet til de organisatoriske, arbejdsmæssige og tekniske rammer
Sikkert	Sikringen mod uønsket adgang til systemets data og facilitete
Effektivt	Udnyttelsen af faciliteterne i den tekniske platform
Korrekt	Opfyldelsen af de opstillede krav
Pålideligt	Opfyldelsen af den krævede funktionalitet med den ønskede præcision
Vedligeholdbart	Omkostningerne ved lokalisering og retning af fejl i det kørende system
Testbart	Omkostningen ved test af systemet i forhold til de opstillede krav
Fleksibelt	Omkostningen ved at ændre i det kørende system
Forståeligt	Besværet for udvikleren ved at skaffe sig overblik over og forstå systemet
Genbrugbart	Anvendeligheden af dele af systemet i andre beslægtede systemer
Flytbart	Omkostningen ved at flytte systemet til andre tekniske platforme
Integrerbart	Problemerne ved at sammenkoble systemet med andre systemer

- **Hvilken betydning har arkitekturen**

Arkitekturen har en betydning på, hvordan systemet skal opbygges. Her skal besluttes hvor metoderne skal ligge, retninger på associeringer og aggregeringer, om nogle aggregeringer skal ophøjes til kompositioner. Der skal vælges arkitektur efter de valgte kriterier og prioriteringer.
fordele ved 3 lags modellen er at,

1. Det minimerer afhængighederne mellem lagene 2.
   Et lag kan bruges af flere overliggende lag

ulemper

1. Ekstra lag kan forringe performance
2. Ekstra lag kan give unødvendig doubletarbejde

- **Eksempel på en arkitektur I kender, og hvordan I har brugt den**

En eksempel på arkitektur er 3 lags modellen, som er den eneste arkitektur vi har brugt. 3 lags modellen går ud på at, man ikke kan bruge begreber, klasser eller metoder i lag over sig, men kun dem i samme lag eller under.
3 lags modellen består at 3 lag gui, application og storage.

Vi har brugt det i f.eks. Aarhus bryghus. Gui er vores brugergrænseflade, som kan bruge metoder fra application og gemme data i storage.

Application laget består af en controller som typisk har metoder som typisk involverer flere klasser, og opretter eller ændre en del af modellens objekter.
Model pakken indholder model klasserne med associeringer og andre forbindelser.

Storage laget indeholder metoder til at gemme objekter af klasser. Laget indeholder metoder til at hente, opdatere og slette objekter.

- **Sammenhæng og forskel mellem analyse og design, indhold af design**

I analysefasen laver man et overblik i form af analyseklassediagrammer. Når man skal begynde og lave sit system er det nødvendigt at lave et designklassediagram. Her får man typer på attributter (String, int osv.), metoder og sammenhængene mellem klasser sat på.

- **Designklassemodellen**
    - Tilføjelser og justering i forhold til analyseklassemodellen

  Man tilføjer typer på attributter, sammenhænge og metoder

---

8. Analyse og design

Gør rede for hvad forskellen på analyse og design er. Med fokus på klassediagrammet hvad er så forskellen på analyseklassediagrammet og designklassediagrammet. Hvad er formålet med hver af dem. Hvilke strukturer anvendes på de to diagrammer.

Kom evt. også ind på hvilke analysemønstre, der kan komme i spil i et klassediagram.

Du er velkommen til at inddrage erfaringer med arbejdet med de to klassediagrammer i jeres projekt.

Besvarelse 8:

I analysefasen laver man et overblik i form af analyseklassediagrammer – når man går videre til designfasen, laver man et designklassediagram, hvor man får typer på attributter
(String, int osv.), metoder og sammenhængene mellem klasser sat på
Analyse undersøger den del af virkeligheden som visionen er baseret på – formålet er at finde ud af hvad systemet skal indeholde, dvs data registrering, hvad anvendelset af
systemet er og en detaljeret beskrivelse af kravene til systemet Analyse
modeller:
Systemsekvensdiagrammer: bruges også som use case beskrivelser til at modellere interaktion mellem system og aktøren - formål at have fokus på at vise hvilke beskeder der
er sendt til systemet, hvad og hvornår der er input og output til og fra systemet Aktør:
En aktør er en brugere eller andre systemer, som interagerer med systemet. Objekt med
livline:

i SSD er det kun én boks som udtrykker et objekt. Hver boks er én klasse. Livlinen viser hvornår objektet modtager og returnerer beskeder/svar.
Besked:
En besked sendes af aktøren eller et objekt.
Returpil:
Svar pilen er en stiplet pil, som er modsat besked pilen. Svar pilen returnerer svaret på den foregående besked.
Loop-kasse / *:
I et SSD bruges en loop-kasse til at illustrere at en interaktion gentages.
If-kasse:
I et SSD bruges en if-kasse til at illustrere når der under en betingelse skal ske en interaktion.
X:
Et kryds nederst i en livline betyder at objektet dør.

SSD illustrerer hvad man skal kunne med systemet, og viser udelukkende interaktionen mellem aktøren og systemet. I DSD udarbejdes et detaljeret programmeringsgrundlag. DSD udvides derfor med de forskellige klasser, som får tilsendt beskeder og afgiver svar

Use case diagram: diagram over systemets use cases og deres relationer til aktøren
Adfærdsmønster: hvordan hændelser påvirker objekter og hvilke tilstande de ender i
Analyseklassediagram: klasser forbindes med associeringer, aggregeringer, osv

Klasse: en model af et begreb fra den virkelige verden – en beskrivelse af en mængde af objekter med samme struktur, adfærd, attributter
Objekt: en model af et konkret fænomen fra den virkelige verden – helhed med identitet, tilstand, adfærd – et objekt er en instans af en klasse

Find klasser: kandidatliste til klassenavne og attributnavne

Vurderingskriterier: Er det en unik ting, der er behov for at registrere noget omkring? Er det indenfor scope af systemet? Er der mere end en forekomst af den? Er det blot et synonym for en anden ting, der allerede er identificeret? Er det blot noget output produceret af anden information, der allerede er Identificeret? Er det blot en attribut ved en anden ting, der allerede er identificeret?

Associering: har på analyse niveau multipliciteter – retningspile kommer først på design niveau

En associeringsklasse er til at gemme information der er fælles mellem to klasser, hvor det ikke giver mening at gemme det i en af de to enkelte klasser – tegnes med stiplet linje Aggregering: stærkere sammenhæng mellem to klasser end associering, kan have en opret metode af den understående klasse – f.ex. enrollment->course – på design niveau kan aggregering blive til en komposition
Generalisering/Specialisering: når en klasse er specialiseret af en superklasse, arver subklassen superklassens attributter og metoder

Mønstre bruges til at konkretisere en bestemt måde at sætte klasser op på i et klassediagram og bruges oftest til løsning af et generelt defineret problem – bruges når/hvis man støder på et problem som er gentagende
Analyse mønstre bruges i klassemodelleringen; design mønstre bruges i forbindelse af valg af arkitektur – f.ex. 3-lags-modellen – programmeringsmønstre (design patterns) bruges i selve kodningen – f.ex. singleton

Rollemønster: bruges når et overordnet individ kan være af flere forskellige roller, f.ex. en person som kan være student eller teacher – lav her en Person klasse; gør det lettere i fremtiden hvis der f.ex. kommer nye roller (vikar, praktikant, osv)

Genstand-beskrivelsesmønster: velegnet til, når man har et antal objekter, der har nogle generelle egenskaber til fælles, f.ex. attributværdier - modellers ved at lave et objekt af den overordnede klasse, som så holder på fælles værdierne for de underordnende objekter

Hierarkimønster: et system hvor der potentielt kan være uendelig mange niveauer – en skole består af afdelinger, som består af uddannelse, osv

Samlingsmønster: et hierarki hvor struktur og dybde ikke kendes på modelleringstidspunkt – f.ex. en indholdsfortegnelse til en opgave

For at et system er brugbart, fleksibelt og forståeligt, kræves det at systemet har en god kobling og samhørighed. En høj samhørighed er når en klasse har relativt få metoder der er let at vedligeholde, forstå og genbruge.

For at have en god kobling, skal der tildeles ansvar som derved giver høj samhørighed. Man skal derfor sørge for at GUI har lav kobling, og det er Controller klassen som har alt ansvaret og derfor høj kobling. Målet er at have høj samhørighed og lav kobling.

Adfærdsmønster/Tilstandsdiagram: Beskrivelse af mulige (lovlige) hændelsesforløb for alle objekter af en klasse

- Elementer i diagrammet
- Tilstand (Boks med tekst)
- Starttilstand (Prik)
- Sluttilstand (Prik med cirkel om)
- Hændelse (tekst over transition pil)
- Transition (Pil + "hændelse" + (betingelse) + /hvad hændelsen trigger/)

- Anvendelse / betydning
- Få et overblik over systemet
- Fælles forløb for alle objekter i en klasse
- Bliver kun brugt hvis objekternes adfærd er tilstrækkelig kompliceret

- Hvordan realiseres tilstande • Med udgangspunkt i banksystemet:
- Hvis saldoen kommer under 0 vil hændelsen blive ført over i boksen overtrukket da de parametre nu er overholdt. Den vil herefter befinde sig i den hændelse indtil denne parameter ikke længere er overholdt.
- Med udgangspunkt i enrollment systemet:
- Hvis en student går igennem en examination, har studenten enten bestået eller fejlet, hvis bestået -> studenten går videre til næste semester, hvis fejlet -> studenten går tilbage til starten af semesteret
- Kode: Kan være attribut, enum, metode.. alt efter behov

---

9. Brugervenlighed og brugergrænsefladedesign

Gør rede for hvordan man definerer brugervenlighed. Hvordan arbejder man med brugervenlighed, hvilke områder skal analyseres osv.? Hvordan kan brugervenlighed afprøves?

Gør rede for forskellige retningslinjer og gode råd for design af brugervenlige systemer.

Besvarelse 9:

Hvordan definerer man brugervenlighed.

- Molich: "brugervenlighed = anvenlighed + nemhed:

brugerV: Let, effektivt, forståeligt og tilfredsstillendene for brugeren
Handler om at målgruppen kan forvente et it system skal reagere og fungere på en bestemt måde, er det altså hensigtsmæssig at betjene og bliver kravene opfyldt Hvordan arbejder man med brugervenlighed(de to er meget ens) Gregersen:

- Brugersituationen: Persona (erhverv, kønsroller, udviklingsstadium)

- Man dykker ned i brugerens behov samt arbejdsgange og færdigheder man lytter til brugeren
- Brugervenlighedskriterier: omsættes til målbare krav (ikke edb kyndig skal kunne..

Molich:
Kend brugeren, indrag, test og ret, lær og koordiner

Brugergrænsefladedesign
- Vigtigt: hvorfor man vælger det design
- Hvordan kan altid løses
- I nyere tid er brugergrænsefladedesign ændret sig meget

Indbydene, behageligt, minimalistisk, budskab
farver, kontraster og harmoni
Linjer og det gylne snit
Navigationsdesign, direkte manipulation
The power of 3: aspekter, farver, font

Afprøvning:
Interaktiv evaluering (tænke højt testen)
Heuristisk evaluering (vurderingsmænd)
Walkthrough evaluering(præsentation)
Formel brugervenligheds test(laboratorium)

Jacob Nielsens 10 heuristikker (retningslinjer/tommelfingerregler)

---

10. Eksperimenter

Gør rede for hvad et eksperiment er og skitser forløbet af et eksperiment.

Gør rede for forskellige former for eksperimenter.

Gør rede for hvornår eksperimenter med fordel kan anvendes.

Gør rede for hvordan eksperimenter kan supplere et traditionelt systemudviklingsforløb.

Du er også velkommen til at komme ind på begrebet prototype og forskellige former for prototyper.

Besvarelse 10:

Et eksperiment er en læringsprocess hvor man afprøver og vurderer en prototype

En prototype er en ufærdig, men kørende/arbejdende model af det ønskede system

Eksperimenter og prototyper laves for at få samme billede af / konkretisere produktet, så der ikke sker kommunikationsfejl

Bruges når graden af usikkerhed er høj, hjælper med at afdække krav, finder fejl og mangler i systemet, lærer systemet at kende

Udforskende eksperimenter: flere prototyper, mål er at finde frem til hvad der ønskes

Afprøvende eksperimenter: ofte kun en prototype af et afgrænset område, mere detaljeret, behandler fx. effektivitet, fejlbehandling

Horisontale prototyper: Dækker hele systemet, men går ikke i dybden – overblik, god til kunder, let at ændre; for overfladisk til fuld forståelse

Vertikale prototyper: Dækker udvalgte dele af systemet, men går i dybden – giver fuld forståelse, god til udviklere; mindre overblik, dækker kun lille del af systemet, tung at lave/ændre

Diagonale prototyper: Dem som hverken er horisontale eller vertikale, dvs. kan være et hvilket som helst udsnit

Throwaway: Omprogrammering, Hurtigt, Smider koden væk efter, Ikke kvalitet

Keep-it: Videreudvikling, Et skridt på vejen mod det færdige system, Kvalitetskode
Scraps: Småprogrammering, Afprøv værktøj/sprog, Kun programmøren der får gavn af det Prototype
færdighedsgrader:
Høj: Giver bedst indblik, Kan genbruges, hvis tilfreds, Mindst risiko for misforståelse; Tager lang tid, Kunden tøver med input, Svær at lave

Lav: Nem at ændre, Hurtig, Nem at lave, Nem at skitsere for brugeren, Kunde tøver ikke med ændring; Ikke så godt indblik, Kan skabe urealistiske forventninger, Højere risiko for misforståelser

Test af eksperimenter

Tænke højt test: brugerne afprøver og tænker højt i mens

Heuristisk evaluering: vurderes ud fra heuristikkerne (anerkendte principper indenfor interface design)

Walkthrough evaluering: Systemet bliver præsenteret for brugerne der giver feedback

Formel brugervenlighedstest: labratorie, kamera osv. Overvåger brugeren imens systemet testes

Et eksperiment er en ***del*** af en process (unified process, vandfald, etc), men ***ikke*** en process i sig selv

Test vs. Eksperiment

Test: Man kender resultatet / hvad man skal ende ud med

Eksperiment: Man finder frem til resultatet/ hvad man skal ende ud med via eksperimenter

---