NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informatikk, matematikk og elektroteknikk

Institutt for datateknikk og informasjonsvitenskap

BOKMÅL

Sensurfrist: 14. juni

Løsning til Eksamen i fag TDT4140 Systemutvikling

Tirsdag 24. mai 2004
kl 0900 - 1300

Hjelpemidler A:
Kalkulator tillatt
Alle trykte og håndskrevne hjelpemidler tillatt

Faglig kontakt under eksamen:
Professor Guttorm Sindre, tlf. 94479

I parentes bak hver deloppgave vises hvor mange poeng denne gir. Innen en oppgave teller eventuelle deloppgaver likt, med mindre annet er angitt.

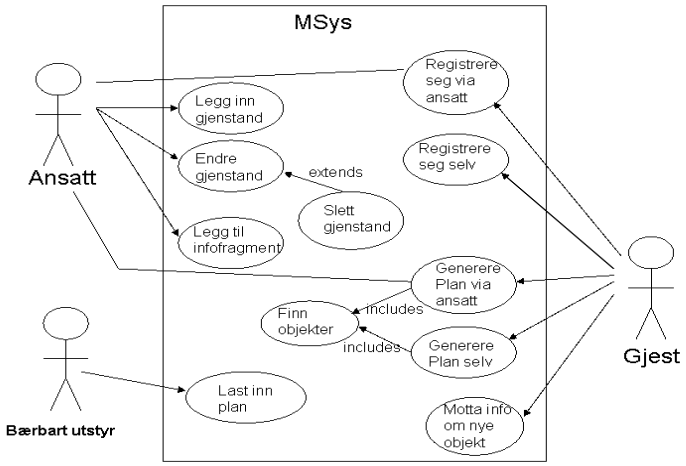
Dersom du trenger informasjon som ikke står i oppgaveteksten må du:
- Kort forklare hvorfor du trenger denne informasjonen
- Gjøre de nødvendige antagelsene. Disse antagelsene må beskrives i besvarelsen.

Lykke til!

.

Oppgave 1 – Use case, 30 poeng

Ut fra kravene gitt i vedlegget har systemutvikleren "Gudmund" laget nedenstående use case diagram og tekstlige use case for systemet. Evaluer både diagram og tekst i forhold til (a) hvorvidt de stemmer med kravene i vedlegget, (b) hvorvidt diagram og tekst stemmer overens, og (c) hvorvidt diagram og tekst er i tråd med vanlige retningslinjer for use case diagrammer og tekstlige use case. Hvilke endringer burde vært gjort? (ikke nødvendig å tegne nytt diagram så lenge du klarer å få forslagene klart fram ved hjelp av tekst)



Generere Plan selv
1. Gjest velger funksjonen "Generere Plan" ved å peke på skjermen 2. Systemet ber gjest enten identifisere seg (hvis dette er en allerede registrert gjest), eller legge inn opplysninger om seg selv (funksjonshemninger, preferanser) 3. Gjest identifiserer seg, eller gir inn opplysninger 4. Systemet finner fram alle de objekter som kan være av interesse gitt gjestens profil og genererer et Plan-objekt som inneholder en liste av disse 5. Systemet fyller denne listen i prioritert rekkefølge, slik at de objekter som passer best for gjesten kommer først, men dog med noen modifikasjoner for å redusere gangavstand under besøket (for eksempel hindre at man blir gående fram og tilbake mellom to ulike rom) 6. Gjest kan velge mellom utskrift av plan (på papir) eller å få lastet planen i elektronisk form over på bærbart utstyr 7. Systemet skriver ut / laster over planen
1.1 Gjest velger ved en feiltagelse en annen funksjon enn han hadde tenkt 3.1 Oppgitt gjesteidentitet fins ikke i systemet – gi feilmelding 4.1 Finner ingen objekter som passer til funksjonshemning og preferanser – gi feilmelding

<u>Svar:</u> I forhold til krav:
- mangler use case for å endre / slette infofragmenter
- mangler use case for å legge til nye <u>typer</u> infofragmenter
- mangler use case for å legge til nye typer funksjonshemninger

De to siste punktene forutsetter at man tolker "fleksibelt mhp å legge til..." som at dette bør kunne gjøres runtime heller enn ved å skrive om programmet, og runtime er jo definitivt mer fleksibelt. Naturlige endringer: legge til slike use case i diagrammet.

Forholdet mellom tekst og diagram:

- **teksten antyder at det er gjesten som er initiativtager (primær aktør) til å laste planen over på bærbart utstyr, og dette står som en del av use case "Generere planen selv". I diagrammet er det derimot vist som en frittstående use case, og det bærbare utstyret er primær aktør. Naturlig endring: fjerne use case "Laste inn plan", eller alternativt (hvis det virkelig er meningen at det bærbare utstyret skal kunne styre dette automatisk) endre teksten slik at det passer.**

I forhold til retningslinjer for use cases:

- noen use cases i diagrammet tuller med systemgrensen. Mens de fleste (korrekt) dreier seg om ting som gjøres direkte mot MSys, er "Registrere seg via ansatt" og "Generere Plan via ansatt" i utgangspunktet interaksjoner mellom museumsgjesten og den ansatte, dvs. en samtale som skjer utenfor systemet. Det er her kun den ansatte som faktisk bruker systemet, basert på (for eksempel muntlig) informasjon fra gjesten. Derfor burde disse use cases egentlig ha hett "Registrer gjest" og hatt den ansatte som primær aktør, eller alternativt (med navngiving og aktører som nå) ha vært plassert utenfor MSys-boksen, hvis man av en eller annen grunn faktisk også ønsker å modellere manuelle interaksjoner mellom gjester og ansatte. Mest naturlige endring: førstnevnte.

- relasjonen mellom "Slett gjenstand" og "Endre gjenstand" er gal bruk av "extends". Modellereren har her kanskje tenkt at sletting er et spesialtilfelle av endring. Extends skal imidlertid brukes når det oppstår en unntakssituasjon i den opprinnelige use case og man derfor blir nødt til å gjøre noe mer enn vanlig. Det er ikke tilfelle her. Naturlig endring: Sette sletting som en frittstående use case, heller enn relatert til endre.

- "Finn objekter" burde ikke vært med i diagrammet, da den ikke har noen aktør, dette ser ut til å være noe som foregår internt i systemet. Mye bruk av "includes" er frarådt i en tidlig fase.

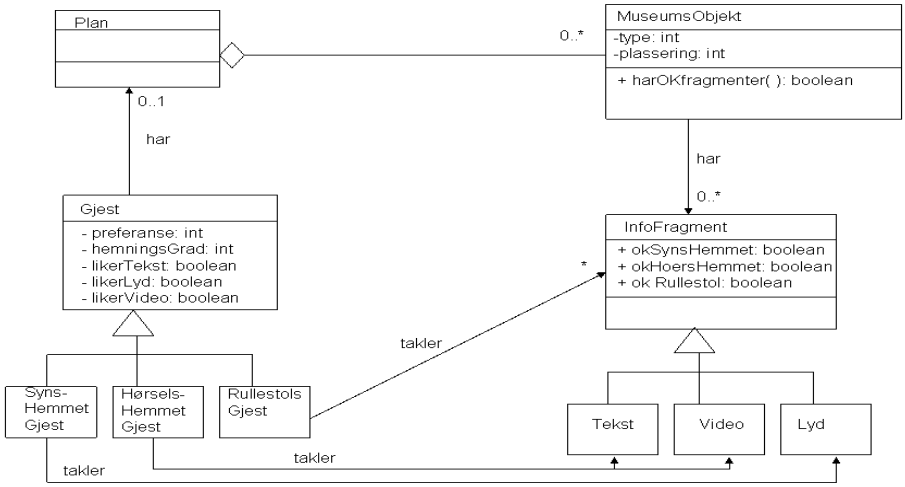
- tekstlig use case foregriper designet. For eksempel sies "peke på skjermen" som uten dekning i kravene antar ett bestemt valg av brukergrensesnitt (og ikke nødvendigvis det beste eller eneste; hva med de som pga funksjonshemning ikke er i stand til å peke?) I steg 4-5 ser "genererer et Plan-objekt", "fyller denne listen i prioritert rekkefølge" ut til å dreie seg mer om systemets indre virkemåte enn om å spesifisere hva som utad skal leveres til brukeren. Det at systemet gjør to steg på rad er også i seg selv mistenkelig i en use case.

- valg og alternativer mikses inn i handlingsstien, først i steg 2 (hvor gjest enten kan være registrert fra før eller ikke) og dette kjøres videre i to alternative handlinger i steg 3. Det gis også alternative handlingsmåter i steg 6-7. Dette gir en rotete beskrivelse som blir vanskeligere å forstå. Anbefalt endring: skrive hovedstien som en handlingsstreng og skille ut alternativer for seg.

- Unntak 1.1 har intet her å gjøre: hvis gjesten ved et uhell velger en annen funksjon enn "Generere Plan selv" kommer han overhodet ikke inn i den aktuelle handlingsstien.

Oppgave 2 – Klassediagram, 30 poeng

Systemutvikleren ”Gudmund” har også kommet i gang med å skissere et klassediagram for det detaljerte designet av systemet. Dette diagrammet dreier seg foreløpig kun om systemets logikklag (dvs., se bort fra at det også kunne trenges klasser spesielt for brukergrensesnittet eller for å sørge for persistent datalagring). For enkelhets skyld er også kolleksjonsklasser droppet (ville f.eks trenge et objekt som ga aksess til alle registrerte museumsobjekter eller alle registrerte gjester).



”Gudmund” har også skrevet følgende kommentarer til diagrammet:
Ved å subklasse Gjest etter ulike typer funksjonshemninger blir det lett å utvide programmet med nye; dette er bare å legge til flere subklasser. Likeledes med typer InfoFragment. Relasjonene kalt ”takler” viser hvilke typer InfoFragmenter som passer til ulike grupper, rullestolbrukere har i utgangspunktet ikke problemer med noen fragmenter (deres problem går snarere på objektenes plassering). Antar at type museumsobjekt er gitt som int (f.eks 1=våpen, 2=klesdrakt, osv.), samme int kan da brukes til å angi preferanse for Gjest (f.eks 1=preferanse for våpen). Plassering er også gitt som int (antar en nummerering av ulike rom i museet).

a) Evaluer diagrammet: Hvorvidt ville et design etter denne skissen tilfredsstille kravene gitt i appendiks? Og er det i tråd med retningslinjer for god OO design? (NB! Av plasshensyn er kun et fåtall av attributter og metoder vist. Dvs., ikke bruk tid på å kritisere evt. manglende attributter og metoder). Hvilke endringer bør gjøres? Du trenger ikke tegne nytt klassediagram, men kan skissere deler av det dersom du føler at du ikke får argumentene dine klart frem med ren tekstlig diskusjon.

Svar: **Feil / svakheter i forhold til krav:**

- **i motsetning til det ”Gudmund” påstår, er subklassingen av funksjonshemninger ingen god ide. For det første innebærer den at hver gjest kan ha kun én funksjonshemning, mens krav i vedlegg (samt sunn fornuft) indikerer at en person godt kan ha flere hemninger. For det andre gjør den programmet lite**

fleksibelt mhp å legge til funksjonshemninger, da dette krever endring av programkoden. Ved å bruke delegering i stedet for subklassing (dvs. at hvert Gjest-objekt kunne hatt assosiasjoner til ett eller flere Hemning-objekter, kunne man bare lagt til nye Hemning-objekter for å registrere en ny type funksjonshemning), en Gjest kunne eventuelt hatt tilknytning til et Profil-objekt som samlet info både om hemninger og preferanser for gjenstander.

- **En lignende kritikk kan reises mot subklassingen av infofragmenter, som umuliggjør multimediafragmenter (som f.eks kombinerer video, lyd og tekst), men her går det ikke like klart fram av kravene at dette er dumt.**

- **preferanse: int i klassen Gjest ser ut til å anta at hver enkelt gjest kun kan ha en preferanse, som kan være diskutabelt i forhold til kravene**

Feil / svakheter i forhold til god OO design:

- **det er svært sterk kobling mellom Gjest og InfoFragment, ved at de ulike typer fragmenter er hardkodet inn i Gjest som attributter (likerTekst, likerLyd, likerVideo), og tilsvarende at ulike typer Gjester er hardkodet inn i InfoFragment (okSynsHemmet osv.) Dette gjør at en endring på den ene siden (f.eks tillegg av en ny type funksjonshemning) gjør at man også må skrive om koden på den andre siden. Assosiasjonene "takler" går litt i samme retning, et forsøk på å hardkode inn hvilke typer fragmenter som passer for hvilke typer hemninger. Her bør man fjerne disse attributtene og assosiasjonene og finne en annen fremgangsmåte for å finne ut hvilke infofragmenter som passer for hvilke gjester.**

- **Attributten type: int i MuseumsObjekt og preferanse: int i Gjest er også eksempel på en kobling som kan være uheldig**

b) Fins det noen designmønstre (patterns) som kunne være naturlig å bruke for dette problemet? Foreslå ett eller to slike mønstre og begrunn hva det skulle brukes til og hvorfor det kunne være gunstig i dette designet. Igjen er det ikke krav om diagrammatisk illustrasjon, men lov å bruke det hvis du føler det trengs for klarhetens skyld.

Svar: **Det sentrale problemet i det skisserte designet var den sterke koblingen mellom Gjest og InfoFragment, så en mulighet kunne være å vurdere mønsteret "Mediator", hvor man overlater til et mellomliggende formidlerobjekt å undersøke hvorvidt et fragment (eller museumsobjekt) passer til en viss gjest. Dermed trenger ikke Gjest vite hva slags fragmenter som fins og vice versa. Mønsteret "Adapter" har en noe lignende hensikt og kan derfor også diskuteres. Siden kravene nevner at gjester skal kunne motta notifikasjon når det dukker opp nye museumsobjekter som passer for dem, kan også mønsteret "Messenger" være verdt å vurdere. En annen utfordring som nevnes er å kunne legge til nye typer funksjonshemninger og museumsgjenstander på en fleksibel måte, for dette kan et "Abstract Factory" pattern være aktuelt. Kan heller ikke fullstendig utelukke andre mønstre, men uansett hvilke mønstre studentene foreslår (også ovennevnte) er det viktig at det kommer en god begrunnelse for hvorfor de kan passe. Siden navn og beskrivelse på mønstrene fins i læreboka (og det var åpen bok eksamen) er det i og for seg ingen prestasjon bare å nevne noen mønster og forklare hva de brukes til uten å relatere det til det konkrete problemet i oppgaven.**

Oppgave 3 – Kode og enhetstesting, 20 poeng

"Gudmund" har også begynt å skissere noe av koden for programmet.

```
class Plan {
      private MuseumsObjekt mineObjekter;
      // samt mer som ikke er vist her
      public Plan(Gjest g) {
            //konstruktor, mottar gjesten som parameter
      }
}

class Museumsobjekt {
      private int type;
      private int plassering;
      private int antallFragmenter;
      public boolean harOKfragmenter(Gjest g) {
            // skal finne ut om objektet har ett eller flere
            // infofragmenter som passer for Gjest g, hvis ja
            // returneres true, ellers false
      }
}
```

a) Er det noe i koden over som ikke stemmer med designdiagrammet? (Det er kun de to klassene som det er vist kode for, som skal vurderes, og du trenger ikke kritisere fravær av attributter eller metoder som heller ikke fremgår av klassediagrammet. Se også bort fra eventuelle syntaksfeil).

Svar: To betydelige feil:

- **klassediagrammet antyder at en Plan skal kunne inneha flere MuseumsObjekt-objekter, men attributten `private MuseumsObjekt mineObjekter` kan kun inneholde en enkelt objektreferanse**
- **likeledes skal et MuseumsObjekt kunne være tilknyttet flere InformasjonsFragment-objekter. Attributten `private int antallFragmenter` antyder riktignok dette, men gir kun antallet, ikke noen knytning til selve objektene**.

Det fins også noen andre småfeil / misforhold mellom kode og design; påpekning av disse bør gi noe uttelling, men det bør trekkes klart hvis de ovennevnte ikke er med (og hvis man har med de ovennevnte er andre feil ikke så viktige å ha nevnt, da dette tross alt bare er en kodeskisse, ikke ment som ferdig kode)

b) Hvordan ville du gjennomføre en enhetstest av metoden *harOKfragmenter( )* i klassen MuseumsObjekt? Du behøver ikke diskutere hvordan testen rent teknisk skulle gjennomføres (f.eks ved hjelp av drivere, stubs eller annet), det holder å forklare hvilke ulike tester du ville ha gjennomført (dvs. med hvilke forhåndstilstander og inndata til metoden). Siden metodens indre kode ikke er vist, kan du anta at testen betrakter metoden som en svart boks.

Svar: (Trenger ikke å sette opp en formell testplan / tabell, som lett kan bli temmelig omfattende, men hvis noen studenter har gjort dette, bør dette selvsagt ikke gi trekk. Men det holder å diskutere mer uformelt tekstlig hva som bør testes, som vist i det følgende).
Mhp forhåndstilstander er det viktig å få prøvd ut ulike sammenhenger av Museumsobjekt vs InformasjonsFragment, dvs.

- **teste mot et museumsobjekt som har null infofragmenter tilknyttet seg**
- **teste mot et museumsobjekt som har ett infofragment tilknyttet seg, og hvor dette**

    - **passer for den aktuelle gjesten**
    - **ikke passer for den aktuelle gjesten**
- **teste mot et museumsobjekt som har <u>flere infofragmenter</u> tilknyttet seg, og hvor**
    - **ingen passer for gjesten**
    - **ett passer for gjesten**
    - **flere passer for gjesten**
    - **alle passer for gjesten**

Metodens eneste innparameter er `Gjest g`. Dette tilsier at man også må prøve med ulike typer gjester, for eksempel

- **gjester med ulike typer funksjonshemninger**
- **gjester med ulike preferanser, for eksempel**
    - **ingen spesielle preferanser mhp gjenstander**
    - **kun en preferanse**
    - **flere preferanser (dog under forutsetning av at dette overhodet skal være mulig i programmet)**

For kompletthets skyld kan det også være hensiktsmessig å teste hva som skjer hvis metoden får inn en nullreferanse som parameter (dvs. tom gjest). Det kan godt hende metoden ikke er ment å fungere i denne situasjonen, men uansett kan det være interessant å se om den i så fall feiler på en akseptabel måte heller enn med katastrofale sideeffekter.

For en dekkende test må man ortogonalt kombinere ulike fragmentmuligheter med ulike gjestemuligheter, men det kan stilles spørsmål ved om det er kostnadseffektivt å prøve å dekke hele matrisen av kombinasjonsmuligheter.

Oppgave 4 – Utviklingsprosess og krav, 20 poeng

a) Beskriv kort (max 200 ord) essensen i Extreme Programming (XP), særlig med hensyn på forholdet til kravspesifikasjon.

<u>Svar:</u> XP er en iterativ og inkrementell utviklingsmetode. Med hensyn på krav tar den avstand fra metoder som bruker mye tid på detaljert spesifisering av krav tidlig i prosjektet. I stedet formuleres funksjonelle krav kortfattet som "user stories" som er en kortversjon av use cases. Mer detaljert forståelse for kravene innhentes ved at man har en kunderepresentant til stede mens man programmerer, slik at man kan innhente direkte tilbakemelding på om funksjonalitet er i tråd med kundens behov. Kravene kunden gir i slike interaktive sesjoner dokumenteres gjerne ikke på noen annen måte enn i koden som sådan. Andre kjente virkemidler i XP er par-programmering og et sterkt fokus på testing (at hvert eneste inkrement, dvs. hver implementerte user story, testes og leveres for seg). XP kan passe særlig godt for programvareutvikling i domener som er i rask endring og der det er usikkerhet om kravene. Det er også blitt hevdet at det primært passer i mindre prosjekter (helst inntil 10, max 20, utviklere).

b) Diskuter hvorvidt det kunne passe å bruke XP i utviklingen av MSys, i forhold til følgende mulige prosjektsituasjoner og hvilke behov disse ville ha mhp representasjon av krav: (i) hvis MSys var et skreddersømprosjekt for et enkelt museum, (ii) hvis MSys var et skreddersømprosjekt for et enkelt museum men deler av kodingen skulle settes ut til et lavkostland, (iii) hvis MSys var et skreddersømprosjekt for å lage felles programvare for mange norske museer, (iv) samme som (iii), pluss at prosjektets størrelse og offentlig eierskap gjør at EU-regler tvinger en til å sette MSys ut på anbud, (v) hvis det var et programvarehus som tok initiativet til MSys, f.eks i et forsøk på å lage hyllevare som man ville håpe å få solgt til en rekke museer, (vi) hvis det var et prosjekt hvor man håper å kunne kjøpe hyllevare, og (vii) hvis MSys utvikles som

open source av gratisarbeidende idealister rundt omkring i verden. Prøv å føre en mest mulig systematisk diskusjon, heller enn bare å ramse opp betraktninger om situasjonene i-vii.

Svar: Mhp kravinnhentingssituasjonen forutsetter XP i utgangspunktet at både kunde og utvikler er gitt, samt at det skal utvikles programvare i prosjektet. Dette tilsier at det vil være mest passende i et skreddersømprosjekt for en enkelt kunde (situasjon i). For situasjon ii kunne det nok også brukes for mye av systemet, men akkurat den delen som skulle settes ut til et lavkostland burde sannsynligvis spesifiseres mer detaljert på forhånd, på grunn av vanskeligere kommunikasjon. Ikke uaktuell for iii heller, men et potensielt problem å finne en eller et lite antall kunderepresentanter som ville være representative for det samlende behovet. At mange museer er involvert gjør også at størrelsen på prosjektet må øke. XP er mindre passende for
(iv): utvikler ukjent (pga anbudssituasjonen; dette gjelder kravspek som man utarbeider som grunnlag for anbudsprosessen – etter at man har valgt leverandør)
(v): kunden ukjent, vet ikke hvem som vil kjøpe hyllevareproduktet (kan selvsagt prøve med en pilotkunde, men et stort risikomoment om denne da vil være representativ for markedet generelt, dvs. ville definitivt måtte komplettere med andre teknikker)
(vi): fordi det kanskje slett ikke skal utvikles programvare (men i stedet kjøpes)
(vii): fins ingen klar kunde. Mhp open source har imidlertid ofte utviklerne selv en klar produktvisjon (og er gjerne selv potensielle brukere av programvaren), har derfor uansett mindre behov for en omfattende kravspek.

Vedlegg A: MSys – System for tilpassede museumsbesøk.

Det skal lages et programvareprodukt (MSys) for å bedre støtte funksjonshemmedes museumsbesøk. Det generelle problemet er at en del typer funksjonshemninger vil ha sterk innvirkning på hvilke gjenstander i museets samling som overhodet er interessante. For eksempel vil en blind museumsgjest ikke ha noe utbytte av en gjenstand som kun kan ses, eller en videosnutt som illustrerer den historiske bruken av en gjenstand, men vil til gjengjeld kunne ha spesiell glede av en gjenstand det er lov å ta på (eller hvor det fins en kopi som det er lov å ta på), akkompagnert av en lydinnspilt forklaring av denne gjenstanden. For museumsgjester som derimot er døve, eller ikke har hender, vil det stille seg annerledes mhp hvilke gjenstander og illustrasjoner / forklaringer som er aktuelle. Kravene til systemet er grovt skissert som følger:

1. Museumsansatte skal kunne registrere museumsgjenstander i Msys, med info som katalognummer, navn, plassering (rom i museet hvor objektet er utplassert), datering, opphavssted, og hva slags type objekt det dreier seg om (f.eks. maleri, skulptur, musikkinstrument, klesplagg, redskap, våpen, smykke, mynt, dokument, ...)
2. Museumsansatte skal også kunne endre informasjonen om allerede registrerte museumsgjenstander, eller slette gjenstander.
3. Museumsansatte skal kunne knytte informasjonsfragmenter til museumsgjenstander. Et informasjonsfragment kan typisk være tekst, bilder, video eller lydspor som forteller noe om gjenstanden, og informasjonen som må ivaretas om fragmentet vil være en beskrivelse av fragmentet og hvilken type det er, samt eventuelt en lenke til selve filen som inneholder informasjonen (f.eks teksten, videoen). MSys trenger ikke funksjonalitet for å utarbeide selve disse fragmentfilene; dette antas gjort i andre programmer.
4. Noen informasjonsfragmenter kan være fysisk utplassert i museumslokalet (f.eks. tekstlige oppslag på vegg, lyd- og videobånd som kjøres på utstyr utplassert ved de aktuelle gjenstandene) – i så fall er det også interessant å registrere dets plassering, mens andre fins på fil.
5. Informasjon om fragmentene må også kunne endres eller slettes.
6. Museumsgjester skal kunne
    i. Registrere seg i Msys med hvilke funksjonshemninger de har (kan enten gjøre dette selv, eller via dialog med en museumsansatt), og eventuelt andre preferanser (f.eks kun interessert i våpen, eller i gjenstander fra middelalderen). Denne profilen må også kunne endres.
    ii. Be om å få autogenerert en plan som gir en rekkefølge av gjenstander som det passer å besøke i forhold til vedkommendes hemninger og interesseprofil. En gjenstand anses som passende hvis: (a) selve gjenstanden er tilgjengelig med de hemningene som personen har, og (b) det fins minst ett informasjonsfragment tilknyttet gjenstanden som passer for en person med disse hemningene. Den funksjonshemmede bør enten kunne generere planen selv (f.eks hjemmefra over internett før en drar til museet eller ved automater utplassert i museet) eller få gjort det ved henvendelse til museets personell.
    iii. Få lastet planen, inkludert elektroniske informasjonsfragmenter, over på bærbart utstyr som den funksjonshemmede kan ha med seg rundt i lokalene. Programvare for det bærbare utstyret er utenfor skopet for Msys-prosjektet; det eneste som trengs i så måte er å kunne eksportere

genererte planer med et passende subsett av informasjonsfragmenter til et format som passer for dette utstyret.

iv. Motta informasjon (f.eks via epost) hvis museet har fått nye gjenstander som kan være av interesse for den funksjonshemmede – dette kan enten ha skjedd fordi en ny gjenstand er anskaffet, eller fordi en gammel gjenstand er blitt utstyrt med nye informasjonsfragmenter eller flyttet il et annet rom som gjør den mer tilgjengelig for den funksjonshemmede.

7. Programvaren bør være fleksibel mhp å legge til nye typer funksjonshemninger og nye typer informasjonsfragmenter.
8. I fremtiden ser man for seg et ønske om å kunne utvide programvaren til å være relevant også for andre enn funksjonshemmede (f.eks at museumgjesters alder også kan spille inn på hvilke gjenstander og infofragmenter som passer, eller for den del at vanlige voksne besøkende kan ha ulike preferanser og få generert besøksplaner på bakgrunn av dette. (Dette skal imidlertid ikke inn i nåværende versjon av systemet)