Semesteropgave

1. Semester
Dat18b

Gruppe 2:

10. December 2018

Fag: ITO, design og construction

Indholdsfortegnelse

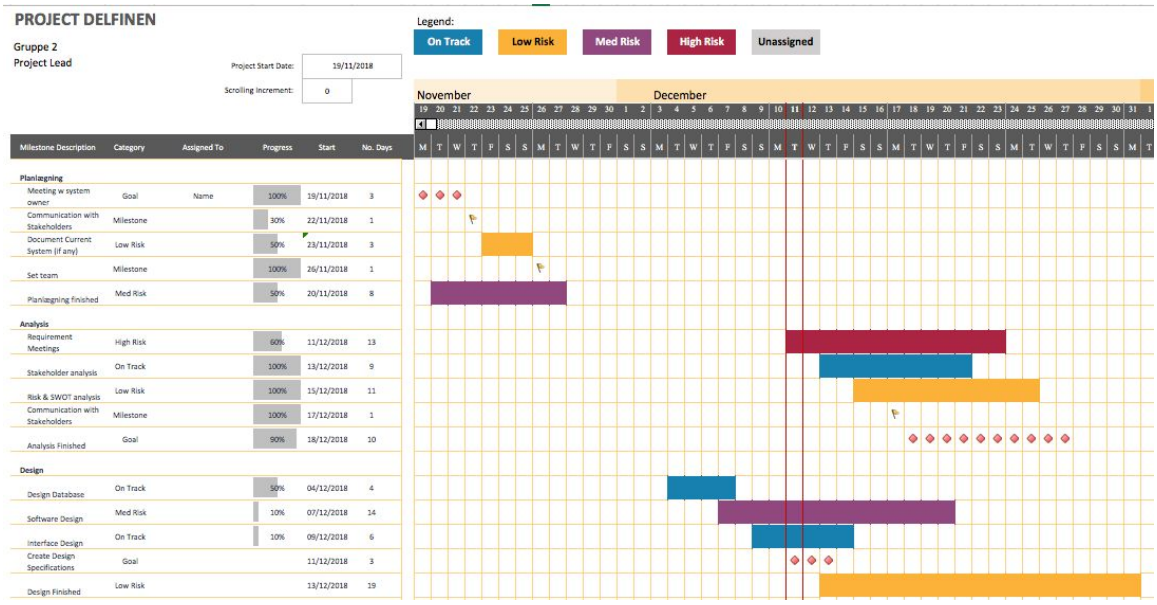
Rapport for Semesteropgave 1:

Svømmeklubben Delfinen

Introduktion

Vi har fået til opgave på Datamatiker-studiet på 1. semester at udarbejde et system for en svømmeklub. Svømmeklubben Delfinen ønsker et administrationsystem, der kan styre medlemsoplysninger, kontingenter og svømmeresultater.
Systemet skal kunne registrere diverse stamoplysninger inkl. alder, samt medlemmers ønskede aktivitetsform, såsom aktivt eller passivt medlemskab, junior eller senior svømmer, motionist eller konkurrencesvømmer.
Derudover skal systemet kunne administrere kontingentbetaling. Kontingentets størrelse er betinget af flere forhold. For aktive medlemmer er kontingentet for ungdomssvømmere (under 18 år) 1000 årligt, for seniorsvømmere (18 år og over) 1600 kr. årligt. For medlemmer over 60 år gives der 25 % rabat af seniortaksten. For passivt medlemskab er taksten 500 kr. Årligt (kilde: opgaveformulering).

Vores opgave er således disponeret som følgende:
Vi har startet med at udarbejdet et gantt chart, som ville udmønte det ideelle scenarie for et projekt som dette. Overordnet set er formålet med et Gantt Chart at illustrere, hvordan et projektforløb ser ud i et såkaldt diagramform, i dette tilfælde søjlediagram. Nedenstående diagram viser de opgaver, der skal udføres på den lodrette akse og tidsintervaller på den vandrette akse. Bredden af de vandrette stænger i grafen viser varigheden af hver aktivitet, som vi har valgt at medtage i dette projekt.



Figur 1: Gantt Chart

Vi har i denne opgave derfor fulgt Gantt chartet til at udføre vores opgave. Det skal dog bemærkes, at vi ikke har haft mulighed for at gennemgå/udføre hele processen, som der ses på chartet, da opgavens omfang ikke rækker til testning og debugging samt implementering hos Svømmeklubben Delfin.

Scope og afgrænsning

Til denne opgave, har vi valgt at afgrænse scopet af opgaven til to parametre rent udviklingsmæssigt: 1) Oprettelse af medlemmer, og 2) Oversigt over medlemmer i restance/ikke betalt. Dette skyldes, at vi gerne ville sikre, at vores opgave lever op til de kodemæssige egenskaber, vi har lært, men samtidig at sikre, at koden ikke fungerer mangelfuldt.

Ift. den organisatoriske del, har vi valgt at anskue modellerne ud fra vores perspektiv som leverandører/udvikler af systemet. Dette skyldes, at vi ikke mener, der er nok baggrundsinformation om Svømmeklub Delfinen, som er afgørende for at udarbejde en fyldestgørende SWOT analyse for Svømmeklubben.

Afslutningsvis er det vigtigt at understrege, at vi ikke har lavet et fuldt ud færdigt program til Svømmeklubben Delfinen, men som nævnt ovenfor fokuseret vores arbejde på væsentlige udpluk og sørget for at disse fungerer og complier rent kodemæssigt.

Analyse
Software Design

I det kommende afsnit vil vi beskrive arbejdet med software design. Vi lægger vægt på en beskrivende forklaring af, hvad, og hvorfor vi har gjort, som vi har gjort. Design delen fungerer som en understøttende del af vores udarbejdet kode.

Use cases

En Use Case er en teknik, der bruges til at afgrænse ens krav til et system, før det bliver udviklet. En Use Case går ud på at man har et success scenarie, for hvordan noget skal foregå i den idéele verden. Man finder derefter ud af hvem ens primære aktører. Dette er personen som skal interagere med ens system.
Herefter skriver man sine trin ned, for hvordan man skal bevæge sig igennem sit success scenarier.
Afhængigt af hvor dybdegående ens undersøgelse skal være, er der 3 forskellige typer af Use Cases:

Brief: Dette er den korteste Use Case model, og består kun af en titel, en primær aktør og et hoved scenarie. Denne bruges kun i starten af udviklingen, for at skabe en hurtigt overblik.
Casual: Her udvides der med nogle alternative scenarier. Det kan fx. Være hvis ens program/system kommer med en fejl, hvis der i stedet for at tilføje en bruger, skal der redigeres eller slettes.
Fully Dressed: Er den helt store Use Case analyse. Her udvider du med stakeholders og interessenter, hvilket er dem som har en interesse i ens projekt, af flere årsager. Det kan både være pga. økonomisk gevinst, eller fordi du giver en service mm.
Vigtigt også at skrive hvorfor de har en interesse i use casen.
Derudover har du nogle forudsætninger med for at dette kan lade sig gøre, samt efterfølger, hvis altså man kommer succesfuldt igennem.
Igen medtages alternativer. Dertil kommer også hvis der er nogle specielle krav for at kunne udføre use casen.
Til sidst skriver man også hvor ofte dette skal ske/hvor ofte man kan komme ud for det pågældende scenarie.
En Fully Dressed Use Case bruges efter man allerede har lavet en masse brief use cases. Man udvælger herefter nogle få (ca. 10%) som har høj værdi/vigtighed, og de skrives så meget detaljeret.

For at lave en god Use Case, skal der ikke være for mange processer i ens scenarie, gerne mellem 5-10, samt være meget specifikke.
Når man skriver sin Use Case, er det vigtigt at skrive det i dagligdags sprog, så alle forstår det, da ens kunde ikke vil forstå div. Software termer, man ellers kunne komme til at bruge.
Til begge vores Use Cases, har vi valgt at benytte Casual modellen.

Use Case #1

UC #1: Create User
Primary Actor: Manager
Main Success Scenario:
1. The Manager opens the system and logs into registration system.
2. The Manager chooses to create a new user in the system.
3. The system asks for the establishment of a new user.
4. The Manager chooses the profile to be created.
5. The system will ask the manager to enter information about the user.
6. The Manager saves the information.
7. The Manager gives the user his credentials.

Alternative Flow (Extensions):
1a. At all times, if the system fails:
- The system reboots, the manager logs in and requests recovery.
- The system returns to the previous stage.
- The system can not return to the previous stage.

- The system shows an error.
- The Manager starts over again.

Alternative Flow (Extensions):
2a. A user must be deleted from the system.

- The manager selects appropiate .txt document in the system.
- The manager finds the right user and deletes the user.
- The system asks the manager to confirm deletion.
- The user is deleted and is no longer part of the swim club

2b. A user has changes to their profile.

- The manager selects appropiate .txt document in the system.
- The Manager finds the right user.
- The Manager edits the information and saves.
- The user's information is now updated.

Use Case #2

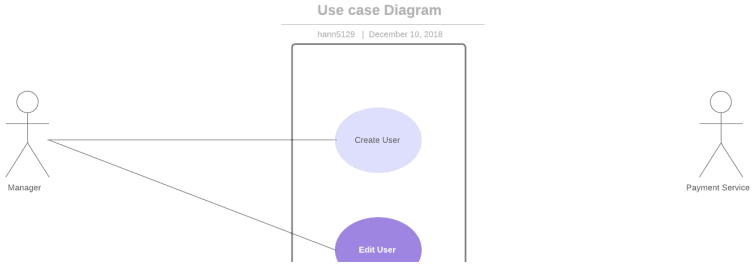
UC #2: Oversigt over medlemmer i restance/ikke betalt
Primary Actor: Cashier
Main Success Scenario:
1. The Cashier opens the system and logs into registration system
2. The Cashier goes to user administration in the system.
3. The Cashier clicks View Member Debt.
4. The Cashier clicks a “send mail to user with missing payment” to user..
5. The System sends mail and the user is notified.

Alternative Flow (Extensions):
1a. At all times, if the system fails:

- The system reboots, the manager logs in and requests recovery.
- The system returns to the previous stage.
- The system can not return to the previous stage.
- The system shows an error.
- The Manager starts over again.

Use case Diagram

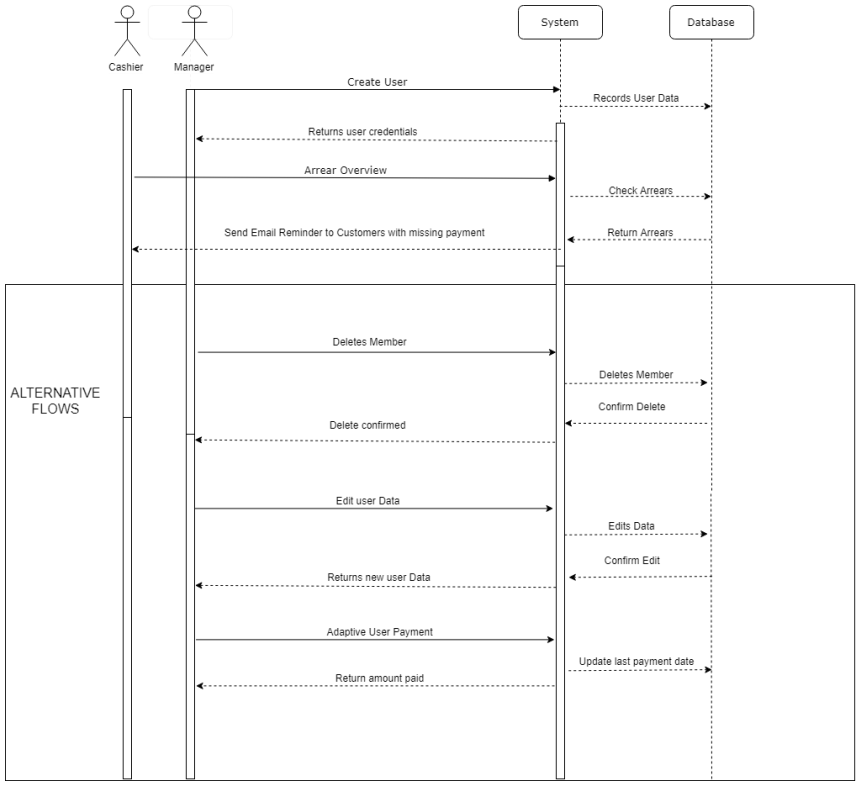
Efter man har lavet sin(e) use case(s), skal man nu lave et Use Case Diagram, for at skabe et overblik over hvem der skal have
adgang, til hvilke funktioner.
Her udvides der lidt, så man også får



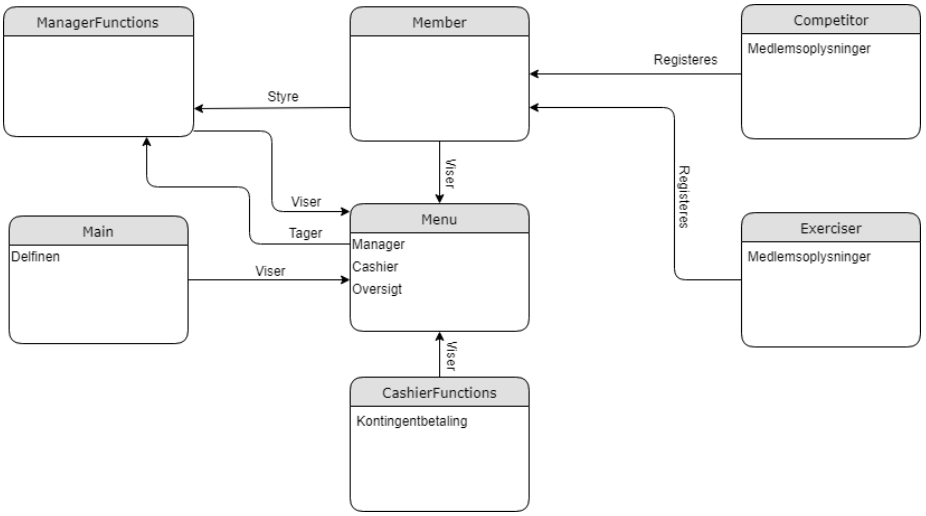
at se hvilke aktører der kan være, udover kun den primære.
Der er nu 3 mulige aktører. Hvis vi tager udgangspunkt i vores Use Case, kan de forklares således:
Main Actor: Dette er den primære aktør. De(n) kan ses på venstre side af boksen. Her er det Manager og Cashier som er vores primære aktører.
Supporting Actor: Supporting Actor er typisk en aktør som ikke er helt i fokus, men samtidig nødvendig for at den pågældende Use Case kan gennemføres. I vores eksempel er det betalingssystemet.
Offstage Actor: Denne aktør holder sig helt i baggrunden, og er typisk en part man ikke tænker over. De skal dog tages med, da de er en vigtig del af en success scenario. I dette eksempel er det SKAT der er vores Offstage Actor, idet de gerne vil vide nogle ting om økonomien vedr. vores svømmeklub, men ikke har interesse i at blive draget ind i den specifikke sag og være en aktiv part.

Sequence flow

Et System Sequence Diagram har til formål at kortlægge alle de processer som skal foregå i ens system, mellem de klasser ens kode består af.
Her skriver du dine primære aktører på venstre side, og ens system på højre side.
Herfra skriver man alle ens processer op, som skal ske ifølge ens Use Case(s).
Det er vigtigt at inkludere både ens success scenarier, og alternative flows.
Som det fremgår af diagrammet til højre, er de solide streger, dem som primære aktører aktiverer, ved at lave en handling, hvortil der så kommer et retursvar, som vises via en stiplet linje.



Domæne model

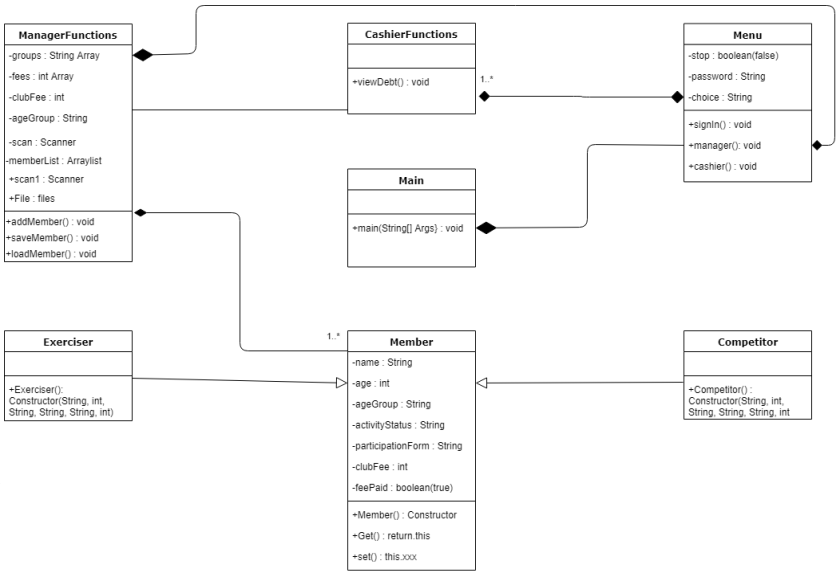


En domænemodel bruges som en analyse over hvordan ens kode skal udformes, i forhold til hvilke klasser man skal bruge, og hvilke attributter de skal have. Kendes også som konceptuelle klasser. Ligesom i en Use Case, er det vigtigt at lægge fokus på at bruge dagligdags sprog, frem for software termer. Dette er for at man kan præsentere sin domæne model for sin kunde, der muligvis ikke har en teknisk baggrund.
Via en gennemgang af sin(e) use case(s) laver man en liste af alle navneord og udsagnsord, som vil danne grundlag for ens klasser og metoder.

Klassediagram

Bruges som design og skabelon for hvordan ens kode skal se ud
Kendes også som software klasser.

Et klassediagram adskiller sig fra en domænemodel, ved at man også oplister de metoder man skal bruge.
Derudover beslutter man om ens attributter og metoder skal være private/public eller protected.
Til sidst forbinder man ens klasser via forskellige former for logiske forbindelser, såsom Association, Composition, Aggregation mm.



I vores klassediagram har vi sat Exerciser og Competitor til at benytte Inheritance/nedarvning, fra

Member.
Fra Member, og op til ManagerFunctions, går der en Composition da et medlem ikke kan eksisterer uden ManagerFunctions klassen. Samtidig har vi noteret at der kan være fra 1 til uendelige instanser af Member, som er visualiseret ved “1..*”.
ManagerFunctions og Menu er forbundet via en dobbeltsidet composition, da manager metoden i Menu er afhængig af at ManagerFunctions eksisterer, da manager ikke ville kunne lave noget uden, og omvendt. Hvis ikke Manager eksisterer er der ikke nogen til at bruge ManagerFunctions metoderne.

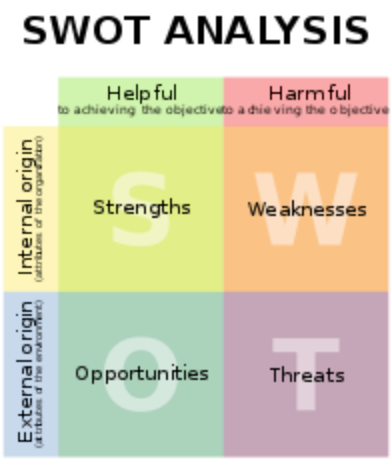
I stil med ManagerFunctions, er vores klasse CashierFunctions også forbundet med Menu via en dobbeltsidet composition. På klassen CashierFunctions er der også noteret at der kan være 1 til uendelige instanser, men dette er med henblik på at der kan være flere personer med en restance.

Derudover er CashierFunctions og ManagerFunctions forbundet af en simpel Association, da CashierFunctions skaber en ny Arrayliste, debtList, ud af ManagerFunctions.getMemberList.
Til sidst er Menu forbundet til Main med en composition, da der uden Main, ikke kan køres noget program.

ITO

I det kommende afsnit vil vi beskrive arbejdet med ITO metoderne. Vi lægger vægt på en beskrivende forklaring af, hvad, og hvorfor vi har gjort, som vi har gjort.

SWOT-analyse



Kilde: [url]

En SWOT-analyse er en nyttig værktøjs-metode til at forstå en virksomhed/organisations styrker og svagheder og til at identificere både de åbne muligheder, der findes for virksomheden og samtidigt de trusler, virksomheden står overfor. Når man bruger den i en business-context er den med til at afklare bæredygtigheden af din virksomhed på et marked.

Vores SWOT-analyse til dette projekt baserer sig på en gennemgang af virksomhedens ønsker, men set ud fra vores organisation/som udvikler af systemet.

Ift. styrkerne for udarbejdelse af systemet, har vi identificeret følgende:

- **En person der administrerer nye medlemmer** - dette er en styrke, da det sikre, at der kun er en person, der varetager opgaverne, samt at der derved mindsker fejlene.
- **On-going registering af data sikre valid data til analyser** - Svømmeklub delfin vil gerne have, at man løbende skal registrere data på medlemmer. Dette er en styrke, da det sikre, at Delfinen hele tiden har opdateret data på deres medlemmer, og vil have nemmere ved at udarbejde visual & predictive analytics.
- **Nyt system afhjælper personalet ift. kontingentbetaling** - systemet vil være med til at frigive personalet en del ressourcer, da de vil kunne trække en liste over medlemmer, der endnu ikke har betalt deres kontingent for året.

Ift. svagheder har vi identificeret følgende:

- **En person der administrerer nye medlemmer** - Ligesom det er en fordel, at der kun er en, der står for administrationen af nye medlemmer, er det ligeså en ulempe, da det vil være risiko for at denne person er syg, siger op eller ligne. Der vil derfor være risiko for mangelfulde oplysninger og informationer fx ved overdragelse.
- **Kasserer tager sig af kontingentbetaling** - Hvis der kun er en kasserer, der tager sig af kontingentbetalinger kan dette være en ulempe, da der kan opstå fejl, hvis der ikke fremgår et fire-øjen princip.
- **Manuel registrering af kontingentbetaling**: Der kan ved registerering af kontingentbetaling fremkomme mangelfulde registreringer, da dette skal opdateres i systemet manuelt.

- **Vedligeholdelse og registrering af medlemmers træningsresultater -** Vedligeholdelsen af den on-going data-registering kan være mangelfuld, hvis der glemmes at registrere data.

Ift. muligheder har vi identificeret følgende:

- **Oversigt over medlemsdata** - Der er en del muligheder i at kunne trække en liste over medlemsdata: 1) identificer aktive medlemmer, 2) se en liste over dem, som mangler at betale kontingent og opkræve udestående, 3) fremadrettet se hvilke kategorier virksomheden har flest af, og hvem der stopper hurtigt i klubben ved brug af systemet.
- **God segmentering af data** - Over et par års data, vil Svømmeklub Delfin kunne lave gode udtræk af data, hvor de kan udarbejde en segmenteringsanalyse, de vil kunne bruge til markedsføringsstrategi.
- **Send påmindelser ud til medlemmer ved nyt år via system** - fremadrettet kan virksomheden udsende påmindelser om kontingentbetaling ved årsskifte, hvis dette implementeres i systemet.
- **Indbetalt elektronisk og modtag kvittering for kontingent betaling** - Man kan i fremtiden gøre det muligt for brugere selv at indbetale kontingent elektronisk og modtage kvittering. Dette ville også eliminere tastefejl fra kasserer.
- **Få medlemmer til at registrere deres træningsresultater selv:** Dette ville lette personalet ansvar, og gøre medlemmer i stand til selv at se og tracke deres udvikling. Man kan derfor nøjes med at få personalet til at dobbelttjekke det indtastede for at sikre korrekt registrering.

Ift. trusler har vi identificeret følgende:

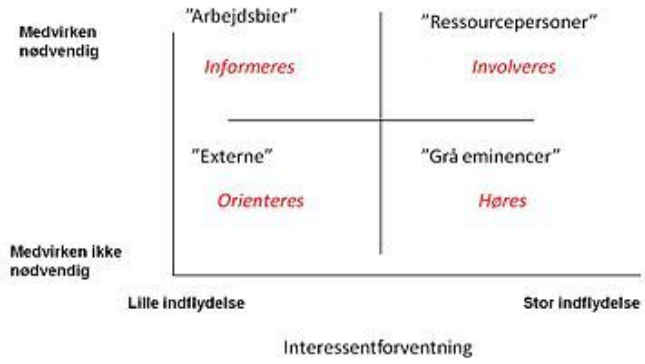
- **Oversigt over medlemsdata, så der leakes persondata** - der kan pga. Mængden af oplysninger i systemet opstår katastrofale fejl, såsom leaking af persondata. Dette skal der selvfølgelig tages højde for i systemet, således at systemet er beskyttet mod udefrakommende trusler.
- **Brug af data forkert af medarbejdere eller til analyser uden samtykke** - Pga. den nye lovgivning GDPR er det nødvendigt for Svømmeklubben Delfin at der udarbejdes et utvetydigt samtykke ved registrering af nyt medlem, samt at disse bruges.
- **Konkurrenter udarbejder et bedre og mere brugervenligt system for administrator** - der er en trussel for at andre softwareudbydere kan tilbyde et bedre produkt end vores system.

Interessent-analyse

En interessentanalyses formål er, at sikre, at alle på projektet ved, hvilke personer der har interesse i projektet. Som regel vil der være mange personer, der vil have en form for interesse i projektet fx gavne deres arbejdsmæssige situation, fremme en KPI/Mål, eller lignende. Derfor

vil disse mennesker også være interesseret i at arbejde for projektet. Ligesom der er interessenter, der er for projektet, vil der ligeledes også være nogle personer, der er modstridende interesse. Disse vil kunne forstyrre projektforløbet. Det er derfor vigtigt at analysere, hvilke personer der vil være i berøring med projektforløbet, før under og efter.

Med en interessentanalyse kan man sikre at projektets mål og succeskriterier er attraktive for de væsentligste personer, så interessenterne ser en fordel i projektet og derved arbejder med projektet. Derudover kan interessentanalysen også bruges til at planlægge hvem der skal indgå i beslutningsprocesserne, fx ved testning, implementering osv. (Se figur 1: Gantt Chart). På den måde kan man sikre, at der er fastsat en relevant informationsstrategi til de relevante interessenter; hvilke interessenter skal informeres om hvad og hvornår.



Kilde: [url]

Som del af vores interessentanalyse har vi identificeret følgende interessenter, der vil være relevant for projektet:

- Medlemmer (mindre indflydelse)
- Delfin adm (stor indflydelse)
- Systemudvikler (stor indflydelse)
- Projektleder af systemudv. (Stor indflydelse)
- Delfin personale (lille indflydelse, stor medvirken)
- Skat (mindre indflydelse).

Som resultat af vores interessentanalyse har vi identificeret ressourcepersoner som delfin administration/personale, da det er dem vi kan trække på, hvis der opstår situationer, hvor vi mangler input. De er bl.a. Ressourcepersoner, da de er med til at støtte op omkring produktudviklingen.
Derudover har vi også projektlederen og systemudvikleren som er en kombination af ressourcepersoner og arbejdsbier. Det skyldes, at deres medvirken er nødvendig, men samtidig har de ikke den store indflydelse på, hvad der skal laves. Modsat kan man sige, at de har stor indflydelse, da de er med til at udforme hele systemet, og derved kan sætte et stort præg på, hvor godt projektet bliver.
Endelig har vi medlemmer, som skal orienteres. Det skyldes at de ikke har direkte stor indflydelse, men naturligvis en medvirken i produktet. Denne er dog ikke nødvendig.

Risiko-analyse

En risikoanalyse er en måde, hvorpå man kan vurdere de risici, der er de mest alvorlige ift. Projektets gennemførsel, og hvordan man kan finde afviklingsstrategier for at overkomme og mitigere risici. En risiko er en sandsynlighed for at noget sker, og samtidig konsekvensen af selvsamme problem/risiko. En risikoanalyse har derved formål at afdække, forebygge og reducere risici for et givent projekt. Derudover kan man bruge en risikoanalyse til at prioritere dem og finde mulige løsninger til at mitigere risici på, så projektet forløber på bedste vis.

Efter en brainstorming baseret på vores projekt, har vi identificeret følgende risici (risikomomenter):

- **Udvikler dokumenterer ikke kode ordentligt.**
  Der kan forekomme risici for at udvikler ikke får dokumenteret sit arbejde ordentligt. Det kan være en risici i det øjeblik hvor udvikler stopper på projekt, og der er en anden der skal overtage dette. Eller når kunden får leveret systemet og har en inhouse udvikler til at tage sig af de små ændringer. Hvis dokumentation ikke er på plads, kan det være svært for andre at forstå, hvad det går ud på.
- **Tekniske risici: mangel på test**
  Det er vigtigt at man undervejs i koden får testet produktet, således at man undgår evt. Fejl og finder de steder, hvor der er mangler. Hvis ikke systemet bliver testet tilstrækkeligt, kan det medfører at kunden vil udsættes for problemer med systemet, som leverandøren ikke har afdækket tilstrækkeligt.
- **Organisatorisk risici: afdelingen ændre struktur, sammensætning.**
  Der kan forekomme organisatoriske ændringer hos kunden, som gør, at der opstår nye risici. Det kan bl.a. skyldes at kontaktpersonen hos kunden bliver opsagt/fyret eller stopper. Eller at ressourcepersoner stopper. Derfor er det vigtigt at sørge for at interessentanalyse hele tiden er opdateret, og man hele tiden har en finger på pulsen ift. kundens organisationsstruktur.
- **Ressource risici: Projektleder forlader projekt.**
  Der kan være fare for, at projektlederen stopper eller forlader projektet. Det kan medfører en stor risiko for projektet. Derfor skal man sørge for at have en nødplan, og sikre sig, at der er en kompetent afløser til at overtage opgaven for at minimere vigtige informationer, der kan gå til spilde ved overdragelse af projekt.

Risikoanalyse tabel

Baseret på ovenstående har vi vurderet, hvor stor sandsynlighed der er, for at risikomomenterne indtræffer, samt overvejet hvilke konsekvenser det vil have for projektet.

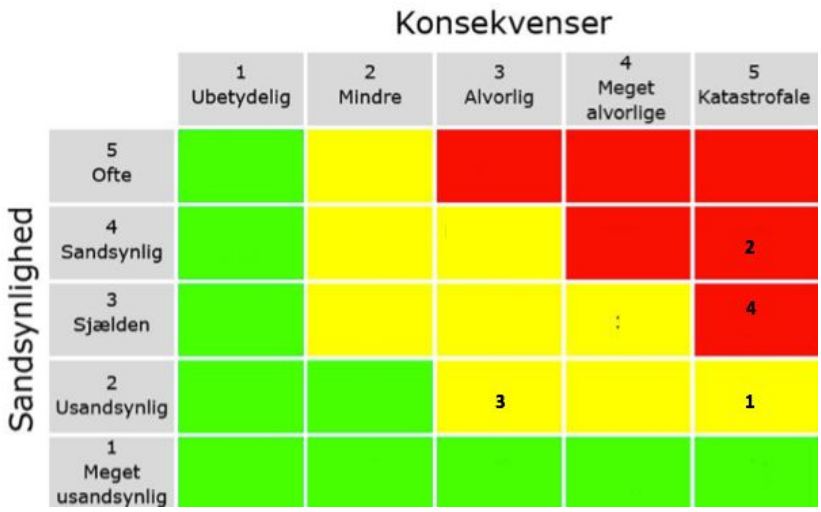
	Sandsynlighed	Konsekvens	Produkt
1: Udvikler dokumentere ikke kode ordentligt.	2	5	10
2: Tekniske risici: mangel på test	4	5	20
3: Organisatorisk risici: afdelingen ændre struktur, sammensætning.	2	3	6
4: Ressource risici: Projektleder forlader projekt.	3	5	15

Sandsynligheden: 1 meget lav, 5 meget høj
Konsekvens: 1 = ubetydeligt, 5 = katastrofal
Risikoen (Produktet): Sandsynlighed * konsekvens
Kilde: fra undervisningsmateriale ITO ppt. 5.

Risikoprofil

Baseret på vores udarbejdet risiko-tabel har vi opsummeret risici i en risikoprofil. Her kan man se risci og konsekvenser for overstående risikomomenter beskrevet i projektet.

Udfra ovenstående fire identificeret risici, placerer to af risikomomenterne sig indenfor det røde felt. Derudfra kan vi se, hvilke vi skal prioritere højest og udarbejde en Risk management plan ud fra.



Kilde: fra undervisningsmateriale ITO ppt. 5.

Som eksempel på en risiko planlægning af overstående, kan følgende tiltag være med til at afværge to af identificeret rissici:

- **Nr. 3: organisatorisk risiko**: sikre at der hele tiden er samspil og afstemning ml. Kunde og leverance, så man ikke går glip eller overser detaljer. I tilfælde af ting ændres er der på forhånd udarbejdet en plan for hvordan dette imødekommes bedst ved fx at få

afstemt med kunden evt. forsinkelser vil forekomme og samt eksempel på hvilke.
- **Nr. 4: Ressource**: sikre at der er et godt arbejdsmiljø og alle bliver hørt, som alle er ansvarlige for. I tilfælde af projektleder forlader projektet, er der allerede udarbejdet en plan for, hvem der overtager projektet og der er udarbejdet en onboarding guide.

Man skal dog være opmærksom på, at risici kan ændre sig i projektforløbet, hvorfor det er vigtigt at tage risikoanalysen op til vurdering løbende af de medarbejdere, der har indblik i den faglige del, for at kunne foretage en reel vurdering af risici.

Construction

I det kommende afsnit vil vi beskrive vores udarbejdet kode, som baserer sig på ovenstående arbejde. Vi lægger vægt på en beskrivende forklaring af, hvad, og hvorfor vi har gjort, som vi har gjort.

Member

"Member" klassen er vores "parent" klasse for alle slags medlemmer, der udelukkende indeholder attributter, en constructor samt setter- og getter-metoder. Nedenunder kan man se, hvilke typer af attributter, vi har valgt at arbejde med:

```
public class Member{

   // Attributter
   private String name;
   private int age;
   private String ageGroup;
   private String activityStatus;
   private String participationForm;
   private int clubFee;
   private boolean feePaid = true;

   // Constructor
   public Member(String name,int age, String ageGroup, String activityStatus, String participationForm, int clubFee){
      this.name = name;
      this.age = age;
      this.ageGroup = ageGroup;
      this.activityStatus = activityStatus;
      this.participationForm = participationForm;
      this.clubFee = clubFee;

   }
```

Klasserne "Competitor" og "Exerciser" er vores "child" klasser, der arver attributter fra "Member", dog med egne constructors og super constructors. En superconstrutor refererer til "parent" klassens constructor. Begge "child" klasser ser næsten identiske ud pånær navn, da der skal kunne differentieres mellem de forskellige instanser af objekterne. Dette kan ses herunder:

```
public class Exerciser extends Member{

   // Sub Class constructor
   public Exerciser(String name,int age,String ageGroup, String activityStatus,String participationForm, int clubFee){
      // Super Class constructor
      super(name, age, ageGroup, activityStatus, participationForm, clubFee);
   }
}
```

ManagerFunctions

Klassen “ManagerFunctions” består af tre metoder, som kun bruges af formanden.

I loadMembers indlæses medlemmer fra to .txt filer (FileNotFoundException throw i tilfælde af filerne ikke eksisterer) for at sørge for, at data ikke går tabt. Scannerne går begge filer igennem og lagrer filernes indhold i en ArrayList. Derefter lukkes scanner objekterne, så der ikke lækkes hukommelse. Herunder ses et while-loop, hvor den indlæser data i filerne (competitor og exerciser .txt), og gemmer dem i en liste i systemet.
Vi har i dette tilfælde valgt at lave en while funktion for at sikre, at data ikke går tabt, når man lukker programmet.

```
public void loadMembers()throws FileNotFoundException{

      Scanner scan1 = new Scanner(new File("competition.txt"));

   // sÃ¥ lÃ¦nge der er input til scan1 kÃ¸res while loopet
   while (scan1.hasNext()) {
      String name = scan1.next();
      int age = scan1.nextInt();
      String ageGroup = scan1.next();
      String activityStatus = scan1.next();
      String participationForm = scan1.next();
      int clubFee = scan1.nextInt();
      memberList.add(new Competitor(name, age, ageGroup, activityStatus, participationForm, clubFee));
   }
   Scanner scan2 = new Scanner(new File("exercise.txt"));

   // samme som overstÃ¥ende bare for scan2 obj.
   while (scan2.hasNext()) {
      String name = scan2.next();
      int age = scan2.nextInt();
      String ageGroup = scan2.next();
      String activityStatus = scan2.next();
      String participationForm = scan2.next();
      int clubFee = scan2.nextInt();
      memberList.add(new Exerciser(name, age, ageGroup, activityStatus, participationForm, clubFee));
   }
   scan1.close();
   scan2.close();
}
```

I addMember metoden tilføjes nye medlemmer til en arrayList for medlemmer. På baggrund af den indtastede info deles medlemmer op i forskellige kategorier som fx aldersgruppe. Herunder vises scopet af de oplysninger et medlem skal oprettes under, bl.a. Alder for at kunne kategorisere medlemmer i systemet:

```
public void addMember(){
   try{

     // input igennem scanneren for variable
      System.out.print("Type in member's name: ");
      String name = scan.next();

      System.out.print("Type in member's age: ");
      int age = scan.nextInt();

      System.out.print("Type in active/passive ");
      String activityStatus = scan.next();

      System.out.print("Type in exercise/competitive: ");
      String participationForm = scan.next();
```

I tilfælde af de rigtige filer ikke eksisterer thrower vi "FileNotFoundException", for at sikre at programmet kan kører og programmet ikke crasher.

Til sidst bruges saveMember metoden til at skrive til en fil alt efter, hvilken type obj (enten exerciser eller competitor) det er:

```
public void saveMember()throws FileNotFoundException{

   PrintStream write1 = new PrintStream("exercise.txt");
   PrintStream write2 = new PrintStream("competition.txt");

   // for each loop, der gÃ¥r igennem ArrayListen(memberList) og sortere objekterne alt efter type, hvorefter de skrives
   for(Member i : memberList){
      if (i instanceof Exerciser){
         write1.println(i.getName()+" "+i.getAge()+" "+i.getAgeGroup()+" "+i.getActivityStatus()+" "+i.getParticipationForm()+"
      }
      else if (i instanceof Competitor){
         write2.println(i.getName()+" "+i.getAge()+" "+i.getAgeGroup()+" "+i.getActivityStatus()+" "+i.getParticipationForm()+"
      }
   }
   write1.close();
   write2.close();
```

ManagerFunctions har også egne attributter, da der undervejs skal indtastes information om nye medlemmer, som skal lagres og listes. Herunder ses vores udvalgte attributter og getter-metode:

```
public class ManagerFunctions{

    // Attributter
    private String groups[] = {"Junior", "Senior"};
    private int fees[] = {500,1000,1600};
    private static ArrayList<Member> memberList = new ArrayList<Member>();
    private int clubFee;
    private String ageGroup;
    private Scanner scan = new Scanner(System.in);

    // memberList getter
    public static ArrayList<Member> getMemberList(){
           return memberList;
    }
```

CashierFunctions

"CashierFunctions" klassen bruges kun af kasserer. Den indeholder en metode, der kan vise en oversigt over medlemmer i restance. Attributter er en *int* og en *arrayList*, da der skal tildeles en random værdi igennem Math.random-metoden *(math random kan tildele en random værdi i intervallet af størrelsen på listen. Fx hvis der er 6 medlemmer, så tager Math-random funktionen en random værdi mellem 1-6)* og der skal loopes igennem en liste. Herunder vises funktionen inden for CashierFunctions, der vælger en tilfældig member i databasen og sætter ham/hende til ikke-betalt:

```
public class CashierFunctions{

	// Visning af alle klubbens medlemmer som skylder kontingent.
	// Der udvælges KUN en tilfældig person på memberList, som udskrives.
	public void viewDebt(){

		// En ArrayList af typen Member kaldet debtList tildeles indholdet af memberList fra ManagerFunctions klassen

		ArrayList<Member> debtList = new ManagerFunctions().getMemberList();

		// en random værdi inden for intervallet af ArrayListens størrelse vælges.
		// Der type castes til int, fordi returtypen på Math.random metoden er double.
		int random = (int) (Math.random() * debtList.size());

		System.out.println();
		System.out.println("List of Members in debt:");
		// tilfældigt medlem vælges og sættes til ubetalt
		debtList.get(random).setFeePaid(false);

		// for each loop som udskriver liste af medlemmer i restance
		for(Member i : debtList){
		   if (i.getFeePaid() == false){
			System.out.println("-----------------------------");
			System.out.println("Name: "+i.getName()+"\n"+
					   "Age: "+i.getAge()+"\n"+
					   "Age Group: "+i.getAgeGroup()+"\n"+
					   "Activity Status: "+i.getActivityStatus()+"\n"+
					   "ParticipationForm: "+i.getParticipationForm()+"\n"+
					   "Debt: "+i.getClubFee()+" kr.");
	  }
	 }
    }
}
```

Menu

"Menu" klassen står for al navigation i systemet og har tre metoder. En "sign in" menu, en menu kun for formand og en menu kun for kasserer. Attributterne er her to Strings og en scanner, så der kan tages input fra brugeren, samt en boolean til at holde et loop kørende.

signIn() metoden byder én velkommen og giver mulighed for at logge ind. Det er her muligt at logge ind som enten manager eller cashier alt efter, hvilket kodeord der indtastes.

Metoderne manager() og cashier() er specifikke menuer for hhv. formanden og kasserer.
Vi thrower fejlene, hvis filerne ikke eksisterer vha. "FileNotFoundException" og hvis der indtastes "invalid" input (altså hvis det præcis ikke er den string vi beder om, så) fanger vi "InputMismatchException". Herunder ses et eksempel på Manager menuen:

```
public void manager()throws FileNotFoundException{

    System.out.println("Now logged in as 'Manager'");
    System.out.println("1 - Add new Member");
    System.out.println("2 - Quit Manager menu");

    try{
        System.out.print("Choice: ");
        choice = scan.next();

        if(choice.equals("1")){
            managerFunctions.addMember();
            managerFunctions.saveMember();
        }
        else if(choice.equals("2")){
            System.out.println("Manager signing off");
            stop = false;
        }else{
            System.out.println("Invalid input!");
            manager(); //recursive
        }

    }catch(InputMismatchException i){
        System.out.println("Invalid input!");
        System.out.println("");
    }
}
```

Main

"Main" klassen sker der egentlig ikke så meget i, andet end at klassen kalder signIn metoden, så metoderne i de andre klasser kan kaldes. Det er her programmet startes. Vi fanger "FileNotFoundException" fejlen, i tilfælde af, at der ingen txt fil eksisterer. Herunder ses vores udarbejdet Main klasse:

```
public class Main{

    //
    public static void main(String[] args)throws FileNotFoundException{

        // Menu objekt skabes
        Menu menu = new Menu();

        // Programmet startes
        menu.signIn();

    }
}
```