

Automatisk tabeloprettelse og fejltjek til forsøgsresultater på
Rigshospitalet

3. Delrapport - ProjDat 2014

Instruktor: Markus Lund Wittorf

Gruppemedlemmer:

Mikkel Aleksander Høgh Rasmussen - 100888

Mathias Fall Christensen - 020693

Markus Visvaldis Ingemann Thieden - 170594

Indhold

1	Litteraturreview	3
1.1	Brooks, F. P., No silver bullet	3
1.1.1	Hovedpunkter	3
1.1.2	Tanker og spørgsmål	4
1.2	Ehn og Kyng, Cardboard computers	5
1.2.1	Hovedpunkter	5
1.2.2	Tanker og spørgsmål	6
2	Delrapport	7
2.1	Abstract	7
2.2	IT-projektets formål og rammer	7
2.2.1	Functionality	7
2.2.2	Application domain	7
2.2.3	Conditions	7
2.2.4	Technology	7
2.2.5	Objects	7
2.2.6	Responsibility	8
2.3	Kravspecifikation for IT-løsningen	8
2.3.1	Funktionelle krav	8
2.3.2	Ikke-funktionelle krav	8
2.3.3	Use-case model	9
2.3.4	Specificerede use-cases	9
2.3.5	Klassediagram	11
2.3.6	Sekvens-diagrammer	12
2.4	Systemdesign sammenfatning	13
2.4.1	System-design resume	13
2.4.2	Udestående design- og implementationsopgaver	13
2.5	Program- og systemtest	13
2.6	Brugergrænseflade og interaktionsdesign	14
2.7	Versionsstyring	16
2.7.1	Vigtigste ændringer	17
2.8	Projektsamarbejdet	17
2.8.1	Hvad går godt?	17
2.8.2	Hvad går mindre godt?	18
2.8.3	Effektivisering af udviklingsarbejde	18
3	Referencer	18

1 Litteraturreview

1.1 Brooks, F. P., No silver bullet

1.1.1 Hovedpunkter

Artiklen beskriver en søgen efter en generel måde at løse problemer ifm. softwareudvikling; en så kaldt sølvkulge.

Først beskriver han problemet med en analogi over Aristoteles, hvori han deler problemerne op i to dele "essensen" og "uheld". Essensen er selve dataopbygningen og programarkitekturen. Essensen har følgende egenskaber:

- Komplexitet
- Overensstemmelse
- Omskiftlighed
- Usynlighed.

Komplexitet består i at software har utroligt mange stadier, og en meget kompleks opbygning, hvilket besværliggør kommunikationen mellem udviklere.

Eftersom software bliver udviklet af mennesker, er der mange holdninger til, hvordan programmer skal opbygges. Derfor er det vigtigt, at få sin software tilpasset med tidligere standarder, for at undgå yderligere kompleksitet.

Omskiftlighed gør det muligt at ændre softwaren efter udgivelse, hvilket er en funktion der er essentiel for enhver succesfuld udgivelse, da hardware opdateres med jævne mellemrum. Software er usynligt og u håndgribeligt, fordi det er så komplekst. Man kan derfor ikke visualisere programmer, og de metoder der findes er overfladiske.

Uheld, som var den anden del af problemerne, er utilsigtet hændelser, hvor deres løsninger ofte bringer fremskridt. Uheld forekommer bl.a. når flere programmer skal snakke sammen. Efterfølgende beskriver Brooks mulige håb for "sølvkulgen" - hvilke veje man kunne gå, for at få løst problemerne i softwareudvikling. Kunstig intelligens er en af de mulige løsninger. Han mener, at man lave en form for "expertsystem", som hjælper programøren med løsningen af eventuelle problemer, men han kan ikke komme med et gæt på, hvornår dette bliver en realitet.

En anden mulighed er at købe færdiglavet programmer. Da der er en vækst af software-distributører, kan man spare penge på at købe færdiglavet software, i stedet for selv at udvikle det, da programmører er dyre i drift.

Den iterative proces beskrives også som en anden mulig "sølvkulge". Det er rigtig svært for en kunde at beskrive, præcist hvad det er de gerne vil have. Man kan løse dette problem ved at arbejde på, at lave tidlige prototyper, som kunden så kan forholde sig til. På denne

måde angriber man essensen af softwaren, ved at udvikle og videreudvikle tilgangen til et problem.

1.1.2 Tanker og spørgsmål

Umiddelbart ville vi mene at artiklen er en anelse redundant. De problemer der bliver stillet op, er ganske vidst noget, der opstår inden for softwareudvikling, men når Brooks ikke selv kan komme på en ordenlig, realistisk løsning, falder artiklen til jorden. Bevares, det bliver nævnt, at den iterative proces er en god metode til at mindske antallet af problemer, og det er bestemt også en holdning, vi i gruppen deler.

Bogen beskriver den iterative arbejds metode scrum, der deler essensen af denne holdning. Metoden går ud på at inddele arbejdet i såkaldte sprints, som typisk er på omkring 30 dage. Til hvert sprint har man et scope, som er en forudsigelse af mål som skal nåes indefor sprint perioden. Efter hvert sprint holdes et møde hvor nye scopes fastsættes, alt efter hvordan det var gået i sidste sprint. På den måde arbejder man iterativt imod et endeligt produkt¹. I gruppen mener vi, at Brooks har nogle gode pointer i sin beskrivelse af software - det han kalder "essencen". Selvfølgelig giver det god mening, at den software man udvikler, skal indrette sig efter software standarder og normer, og idéen om at, at din software skal være omskiftelig, har jo gjort at opdateringer til div. programmer udkommer så ofte som de gør. Det er måske det sidstnævnte, som vi vil bide os mest fast i, iforhold til artiklen, og det ligger klart og fast, at vi har tænkt os at benytte os af dette "råd".

¹Se evt. s. 599-603 i OOSE[1]

1.2 Ehn og Kyng, Cardboard computers

1.2.1 Hovepunkter

Artiklen argumenterer for, at der er både fordele og ulemper ved "abstraherede" prototyper, eksemplificeret ved deres papcomputere. Fordelene opsummeres overordnet i fire punkter:

- "Hands-on experience"
- Forståelighed
- Lav omkostning
- Sjovt (og dermed engagerende)

Forståelighed og "hands-on experience" går "hånd i hånd", da denne mockup metode er baseret på en kontrakt imellem designerne af mockup og de deltagende brugere. Denne kontrakt er i grunden en leg eller et "rollespil", da den kræver at deltagerne lever sig ind i den fantasi der er opstillet. Uden indlevelsen falder metoden til jorden. Hvis indlevelsen derimod er til stede kan brugeren få en kraftig fornemmelse af deltagelse i processen, og kan relatere deres "hands-on" oplevelser langt mere direkte, end hvis de skulle forholde sig abstrakt til det endelige programs teoretiske funktionaliteter.

Derudover medfører denne metode et intimt samarbejde mellem designere og brugere, og da det fysiske mockup kognitivt er så langt fra et computersimuleret mockup, risikerer brugeren ikke at forveksle tekniske problemer med designbeslutninger.

Endnu en fordel ved at lege med pap er, at det lægger op til en form for "super-rapid-prototyping". Når brugeren sidder med så fysiske manifestationer af funktionerne, lægger det op til eksperimentation og kritik på et andet niveau, end hvis de sad overfor en computerskærm, og de foreslåede ændringer kan implementeres og testes med det samme, blot ved at finde et andet dejligt stykke pap og en tusch.

Artiklen bevæger sig også ind på grænseområdet mellem den ovennævnte "pap-metode" og en mere "konservativ" mockup metodologi. Den argumenterer for, at efterhånden som man bevæger sig mod et mere "repræsentativt simulacrum" vil brugerens indlevelsesevne stige, især for brugere der kun deltager kort. Til gengæld begynder fordelene ved "klippeklistre" metoden at forsvinde - det er for eksempel ikke lige så sjovt at sidde overfor et realistisk computermockup, som det er at sidde og lege med pap. Derved vil brugerens indlevelsesevne højnes på bekostning af blandt andet engagementet.

Denne form for dialog med brugeren udgør en form for falsifikation, hvor brugeren over en længere periode konfronteres med den tidssvarende version af programmet (eller mockupet). Brugerens feedback bliver brugt til forbedring og fejlrettelse af designet, og forhåbentlig kan man igennem en iterativ proces komme frem til ikke blot en mere præcis model, men

i sidste ende et mere finpudset produkt².

1.2.2 Tanker og spørgsmål

Artiklen er tydeligt præget af, at være skrevet i det sidste årtusinde. På trods af, at den sandsynligvis engang har været relevant (muligvis indenfor en anden sektor), er dens nuværende værdi allerhøjest tvivlsom. Nu om dage falder langt de fleste anvendelsesområder indenfor deres særkategori af ”virksomheder der allerede har den påkrævede IT-infrastruktur”, og hele paphistorien mister derfor en stor del af sin relevans. På trods af dette er der nogle meta-egenskaber ved designmetoden som med lidt god vilje kan overføres til den moderne verden. Vi har for eksempel designet et ”tilsvarende” mockup i form af et billede af interfacet, som brugeren så kan kritisere. Mange af artiklens pointer kan dog ikke overføres ved denne overgang, som for eksempel den barnlige fornøjelse ved ”legens sprog” og rollespillet, eller den rapid-prototyping man ville kunne få ud af, at brugeren blot kunne gribe det nærmeste stykke pap og tilføje en ny feature.

²S. 41 i OOSE[1]

2 Delrapport

2.1 Abstract

Vi laver en desktopapplikation til automatisk oprettelse af tabeller med henblik på forsøgsresultatindsamling for Professor Lars Rasmussen fra anæstesi afd. på Rigshospitalet. Målet med projektet er en simplificere processen fra indledende forsøgsprotokol til klargjort tabel. Dette vil vi gøre ved at analysere protokollen, og automatisk oprette et dokument med de i protokollen angivne parametre. Outputtet er et dokument der kan læses af excel eller lignende programmer, som fungerer som en skabelon til det specifikke forsøg. Derudover har vi en fejltjek-funktion, der kan undersøge om et udfyldt skema overholder specifikationerne.

2.2 IT-projektets formål og rammer

2.2.1 Functionality

Opretter automatisk skemaer til forsøgsresultater ud fra protokoller. Tjekker indtastede værdier for varians fra angivne grænseværdier.

2.2.2 Application domain

Forskere på rigshospitalet, evt. sekretærer der står for dataindtastning.

2.2.3 Conditions

Programmet udvikles som led i projektkurset. Programmet vil blive anvendt til at reducere indtastningsfejl i forbindelse med forskningsresultater.

2.2.4 Technology

Programmet skrives i Java. Skrives til at være kompatibelt med rigshospitalets Windows-systemer.

2.2.5 Objects

Protokoller, skemaer og CRF'er (Case Report Files).

2.2.6 Responsibility

Automatisering af oprettelse af skemaer. Fejltekning af indtastet data.

2.3 Kravspecifikation for IT-løsningen

De funktionelle krav udgør specifikationer af funktioner, som programmet skal understøtte. Ikke funktionelle krav er kvalitative begrænsninger, som indirekte er relateret til programets funktionalitet.³

2.3.1 Funktionelle krav

- Skal i praksis kunne håndtere et stort antal parametre til oprettelse af tabeller.
- Skal kunne oprette et dokument med forsøgsparametre, som kan læses af Excel.
- Skal kunne ”oversætte” en forsøgsprotokol og opfange parametre samt begrænsninger.
- Skal kunne sammenligne et udfyldt excelskema med forsøgsprotokollens krav, og advare brugeren hvis de ikke er overholdt.

2.3.2 Ikke-funktionelle krav

- Skal kunne oprette tabeller hurtigere end manuel indtastning.
- Simpelt og overskueligt design.
- Skal være brugervenligt.

³Se evt. s. 12-13 i OOSE[1]

2.3.3 Use-case model

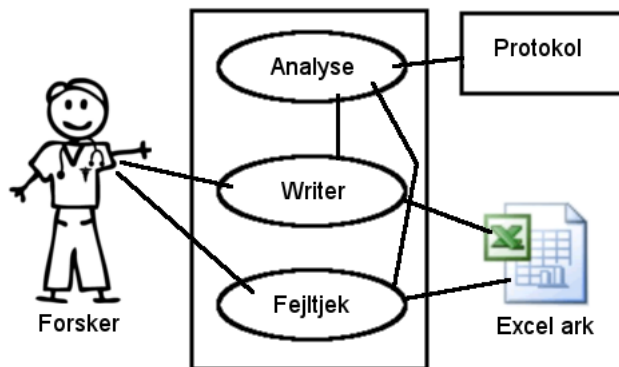


Fig. 1: Use-case model

Forskeren kører Writer funktionen for at oprette et Excel ark, eller Fejltjek for at tjekke et skema. Analyse indsamler information fra Protokollen. Writer tager input fra Forskeren, henter information fra Analyse, og skriver Excel arket. Fejltjek modtager parametre fra Forskeren, og sammenholder informationen fra Analyse med skemaet angivet af Forskeren.

2.3.4 Specificerede use-cases

I opgavebeskrivelsen står der, at vi skal udforme 3 use-cases, men da vores program i realiteten kun skal kunne de ting, vi har beskrevet nedenfor, har det kun været muligt for os at lave disse to:

USE CASE 1: Opret Excel Ark

Participating actors: Forsker

Flow of events:

1. Forsker vælger stien for Protokol.
2. Forsker vælger stien for skema output.
3. Forsker indlæser protokollen.
4. Programmet læser værdier fra protokollen og viser dem i brugergrænsefladen
5. Forsker gennemlæser og bekræfter de indlæste værdier i brugergrænsefladen

6. Forsker opretter skemaet ved tryk på "Opret Skema"
7. Programmet skriver værdierne til en excel fil på den i skridt (2) valgte sti.

Entry condition: Forskeren har startet programmet.

- Exit condition:
- Skemaet er blevet oprettet.
 - Programmet afsluttes.
-

- Quality requirements:
- Programmet skal oprette skemaet hurtigere end Forsker ville kunne gøre det på egen hånd.
 - Programmet skal finde alle relevante attributter og grænseværdier.
-

USE CASE 2: Fejltjek Excel Ark

Participating actors: Forsker

- Flow of events:
1. Forsker vælger stien for Protokol.
 2. Forsker vælger stien for Excel Ark.
 3. Forsker starter fejltjek ved tryk på "Kør tjek".
 4. Programmet læser grænseværdier fra protokollen og sammenholder dem med værdierne i Excel arket.
 5. Forsker aflæser fejlagtige værdier hvis de er tilstede, og foretager eventuelle rettelser.

Entry condition: Forsker vælger "Tjek Skema" tabben.

- Exit condition:
- Tjekket er kørt igennem.
 - Programmet afsluttes.
-

- Quality requirements:
- Programmet skal vise en oversigt over alle værdier, der ikke ligger indenfor de i protokollen angivne grænseværdier.
-

2.3.5 Klassediagram

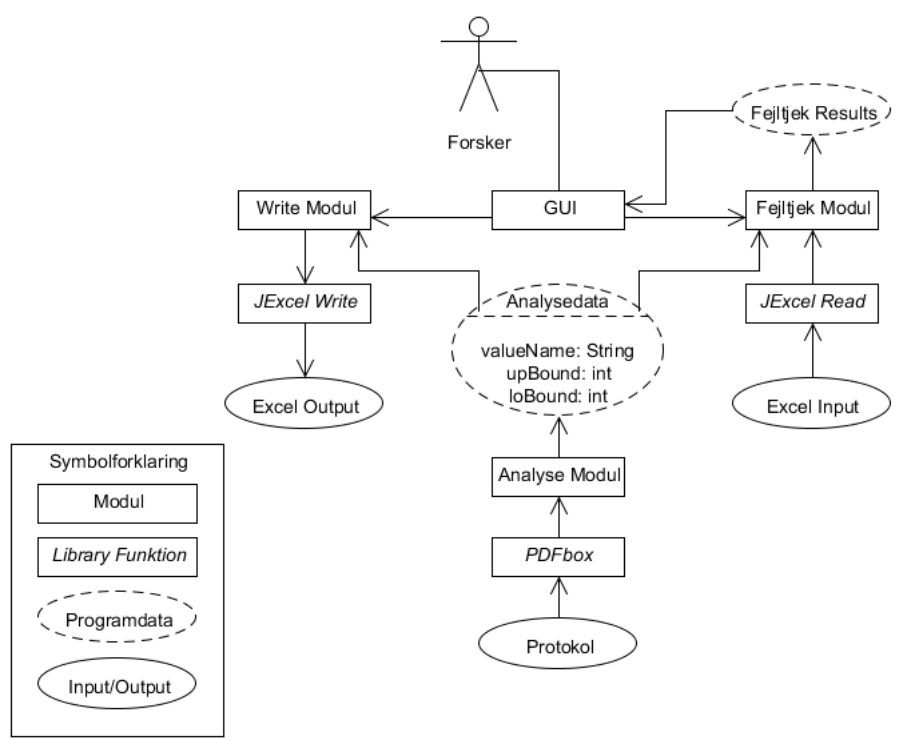


Fig. 2: Klassediagram

Forskeren bruger GUI'en til at køre enten Write eller Fejltjek modulerne. Både Write og Fejltjek modulerne afhænger af analysedataen, som bliver lavet af Analysemodulet ud fra protokollen vha. PDFbox. PDFbox udtrækker tekst fra protokollen (som ligger i .pdf format), og vi kører regex på de relevante dele af denne tekst. Write modulet skriver de udtrukne data (vha. JExcel Write) til et excel dokument. Fejltjek modulet læser et excel dokument vha. JExcel Read. Den læste data sammenholdes med den fra protokollen udtrukne tekst, sammenligner værdierne (tjekker om de ligger indenfor de acceptable værdier) og sender resultaterne videre til GUI'en, som viser dem til

forskerne/brugerne.

2.3.6 Sekvens-diagrammer

Vi har lavet to sekvensdiagrammer, der passer til de to use-cases vi har beskrevet tidligere.

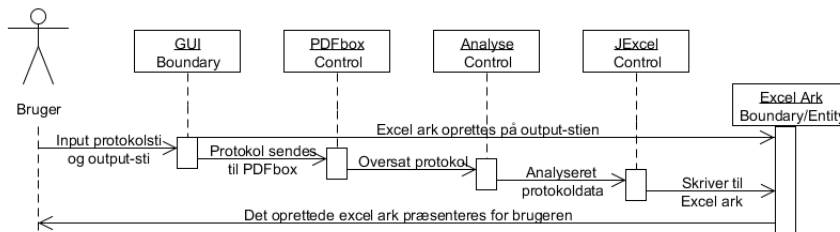


Fig. 3: Sekvensdiagram for Use-case 1

Fig. 3 er et sekvensdiagram over den første use-case, hvor forskeren benytter programmet til at oprette et skema ud fra en protokol. Brugeren angiver stien for protokollen og den ønskede placering for Excel-arket (se fig. 5 for mockup). Et Excel-ark oprettes på den angivne output-sti, og protokolfilen sendes til PDFbox biblioteket, der oversætter det til et læsbart (for programmet) format. Analysemodulet kører regex på denne data, og finder de relevante parametre. Disse bliver sendt til JExcel, som skriver den fundne data ind i Excel-arket. Til sidst vises det færdige Excel-ark for brugeren.

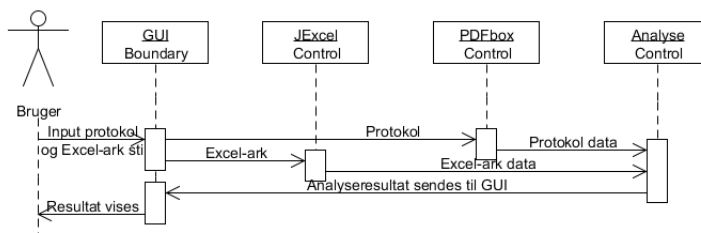


Fig. 4: Sekvensdiagram for Use-case 2

Fig. 4 er et sekvensdiagram over den anden use-case, hvor forskeren benytter sig af den

anden funktionalitet i programet til at tjekke et skema igennem for fejl (se fig. 6 for mockup). Brugeren angiver stierne for både protokollen og Excel-arket. Excel-arket sendes til JExcel, som læser dataen fra de af brugeren indtastede felter. Denne data sendes så til analysemodulet. Protokollen sendes til PDFbox, som ligesom i fig. 3 læser filen og sender dataen til analysemodulet. Analysemodulet sammenligner dataen fra PDFbox og JExcel, og sender resultatet til GUI'et, som viser det for brugeren.

2.4 Systemdesign sammenfatning

2.4.1 System-design resume

Da vi laver et program der skal kunne oprette excel dokumenter og læse PDF filer, har vi benyttet os af to ikke-standard Javapakker; PDFbox og JExcel⁴. Metoderne i disse klasse danner stort grundlag for vores program, og vores opgave ligger i at bearbejde den data som PDFbox kan trække ud for os. Vi har 3 hoved moduler i programmet;

- Analyse, der benytter sig af PDFbox
- Writer, der benytter sig af JExcel til at oprette et excel dokument
- Fejltjek, der benytter sig af begge pakker til at læse en protokol og et excel skema.

De sidstnævnte moduler er pt. ikke implementeret. Analyse klassen er dog udarbejdet i en grov form (se punkt 2.7).

2.4.2 Udestående design- og implementationsopgaver

Fejltjek og Writer klasserne er ikke blevet implementeren endnu, og det samme gælder GUI'en. Mht. til GUI'en, har vi dog en god idé om, hvordan den skal se ud jf. vores mockups. Fejltjek og Write afhænger i stor grad af Analyse klassen, så vi arbejder meget linært - vi kan ikke begynde på det næste modul, før den første er implementeret.

I Analyse klassen, mangler vi en del der kan bearbejde den data der bliver udtrukket fra PDF'en. Dette har vi umiddelbart tænkt os at gøre vha. Scanner klassen fra Javas standard bibliotek.

2.5 Program- og systemtest

Vores tests består hovedsageligt af at køre de enkelte moduler, for at tjekke deres funktionalitet. Vi har planer om at fremvise en prototype af produktet til vores kunde. En kørsel

⁴Kilderne kan findes på hhv. <http://pdfbox.apache.org/> og <http://jexcelapi.sourceforge.net/>

af Analyse klassen er at finde under punkt 2.7.

2.6 Brugergrænseflade og interaktionsdesign

Attribut navn	Tilladte værdier
Alder	Int: 18-35
Køn	Char: M/K
Ryger	Bool: S/F
Vægt	Int: 35-250

Fig. 5: Mockup af brugergrænseflade mht. Use-case 1

Her ses et mockup af vores brugergrænseflade. Protokolstien kan angives, og brugeren kan specificere hvor excelfilen skal oprettes. Tabellen viser en oversigt over de fundne attributter fra protokollen.

Som det ses, kan der navigeres mellem programmets to dele via fanerne i toppen af vinduet.

Attribut	Placering	Aktuel Værdi	Tilladte Værdier	
Alder	B-12	Int: > 17	Int: 18-35	
Køn	C-4	String: Mand	Char: M/K	
Ryger	D-7	String: Ja	Bool: S/F	
Vægt	E-4	Int: < 422	Int: 35-250	

Fig. 6: Mockup af brugergrænseflade mht. Use-case 2

Dette er et mockup af den anden del af programmet. Her er idéen at der er blevet angivet et excel skema og den originale protokol, hvorefter der er blevet kørt et fejltjek. Tabellen viser evt. fejl eller afvigelser fra standard syntaks (f.eks. er der under køn blevet skrevet "Mand" i celle C-4, men protokollen har bedt om et input i form af M/K).

2.7 Versionsstyring

På nuværende tidspunkt, har vi implementeret en grov version af klassen Analyse, der kan læse en PDF, og trække relevant data ud. Da vi pt ikke benytter os af github, kan vi ikke vise en commit-log, men vi har i sinde at begynde på det. I stedet har vi nedenfor den første implementation af Analyse-klassen, der giver et godt indblik i, hvad den skal gøre:

```
1 import java.io.*;
2 import org.apache.pdfbox.pdmodel.*;
3 import org.apache.pdfbox.util.*;
4 import java.util.regex.*;
5
6 public class Analyse {
7
8     public static void main(String[] args){
9         PDDocument pd;
10        String att;
11        try {
12            // PDF fil der skal læses
13            File input = new File("filstien angives her");
14
15            // StringBuilder opbevarer den udtrukkede tekst
16            StringBuilder sb = new StringBuilder();
17            pd = PDDocument.load(input);
18            PDFTextStripper stripper = new PDFTextStripper();
19
20            // Tilføjer text til StringBuilder
21            sb.append(stripper.getText(pd));
22
23            // Udtrækker relevant data fra pdf vha. regex.
24            // Leder efter linjer omsluttet af "//...//"
25            Pattern p = Pattern.compile("//\\S+//");
26
27            // Matcher, der leder i texten
28            Matcher m = p.matcher(sb);
29
30            while (m.find()){
31                // group() metoden sætter teksten ind i variablen att.
32                att = m.group();
33                System.out.println(att);
34            }
35
36            if (pd != null) {
37                pd.close();
```



```

38     }
39 } catch (Exception e){
40     e.printStackTrace();
41 }
42 }
43 }

```

En kørsel af programmet med en protokol-PDF, vil finde og udsrive alle linjer omsluttet med "//...//", og som overholder en bestemt syntaks (e.g. //;Køn;Alder;Blodtype//). Herefter skal en anden del af klassen kunne separere attributterne (f.eks. hva. Scanner klassen fra Javas standard bibliotek)

Den overordnede idé er, at det skal kunne være muligt at "browse" gennem fil stier via GUI'en, for at finde input filen. Dette vil først være muligt når GUI'en er implementeret.

2.7.1 Vigtigste ændringer

Siden vores sidste aflevering, har vi ikke lavet nogle betydelige ændringer i vores kode. Vi forventer at have en endelig version af Analyse klassen, samt en grov implementation af Writer klassen klar indenfor den næste uge.

2.8 Projektsamarbejdet

Samarbejdet fungerer fint, så vi ser ingen grund til at ændre i dette. Vi er gode til at arbejdsfordele, og vores forskellige evner og styrker komplimenterer hinanden udmærket. Vi har fortsat tænkt os at holde møder på ugentlig basis, hvor kodning og evt. rettelser vil finde sted. Dog skal det nævnes at vi, på opfordring fra vores instruktør (og fordi det er et krav til opgaven) har tænkt os at begynde på at bruge github (se mere i afsnit 2.8.3).

2.8.1 Hvad går godt?

Vi er afklarede om målet for vores projekt.

Fremmødet har været upåklageligt.

Arbejdsmorale er høj.

Samarbejdet fungerer stadig godt.

2.8.2 Hvad går mindre godt?

Vi er først lige nu ved at komme i gang med github.

Tidspresset er begyndt at tage til, hvilket kan have en negativ indflydelse på arbejdsmorale. Vi har dog stadig et optimistisk syn på projektet, og sporadiske Cafeen? besøg holder humøret oppe.

2.8.3 Effektivisering af udviklingsarbejde

Vi vil begynde at bruge github, så vi kan arbejde uafhængigt af hinanden. Dette er ikke ensbetydende med at vi vil begynde at arbejde individuelt hele tiden, men hvis vi kan komme undgå at skulle mødes hver gang, vi skal kode, kan de i høj grad få øget tempoet på processen.

3 Referencer

- [1] Object-Oriented Software Engineering Using UML, Patterns, and Java. *Bernd Bruegge, Allen H. Dutoit* 3rd Edition, 2014.