

CDIO opgave forår 2014

Alternativt projekt - Anden iteration
Dataopsamling fra tests af piezoelektriske krystaller.

Morten Hesselbjerg, s017704
Christian Budtz, s134000
Rikke Christina Hansen, s120359
Rúni Egholm Vørmadal, s134004
Martin Nielsen, s123064
Eirik Oterholm Nielsen, s123006

CDIO_51_alternativ							
Time-regnskab	Ver. 2008-09-03						Bemærk
Dato	Deltager	Design	Impl.	Test	Dok.	Andet	Ialt
19/2	Christian					3	3 Kundemøde
	Morten					3	3 -
	Rikke					3	3 -
	Martin					3	3 -
	Rúni					3	3 -
	Eirik					3	3 -
24/2	Christian					2	2 Projektplanlægning
	Morten					2	2
	Rikke					2	2
	Martin					2	2
	Rúni					2	2
	Eirik					2	2
25/2	Rúni						0
	Christian					2	2 RoadMap
26/2	Christian					2	2 RoadMap
1/3/2014	Christian	2					2
2/3/2014	Christian	1			2		3
1/3/2014	Morten		2				2 C# kode
2/3/2014	Morten		2	2			4 C# kode
2/3/2014	Rikke				2		2
3/3/2014	Martin					4	4
3/3/2014	Christian	2					2
4/3/2014	Christian	3					3
4/3/2014	Rúni					2	2 kundemøde
4/3/2014	Morten					2	2 kundemøde
4/3/2014	Martin		4				4
6/3/2014	Christian		1	1			2 GUI
8/3/2014	Morten					2	2 PP præs.
9/3/2014	Christian	0.5	0.5	0.5	0.5		2
9/3/2014	Morten					4	4 PP præs.
10/3/2014	Christian	2				1	3
10/3/2014	Rúni	2				1	3
14/3/2014	Christian		2	1			3
12/3/2014	Rúni		4				4
15/3/2014	Christian		1	1	1		3
17/3/2014	Rúni		6			1	7 GUI
17/3/2014	Martin		6			1	7
17/3/2014	Eirik		2				2
17/3/2014	Morten		6				6
17/3/2014	Christian		3		3		6
18/3/2014	Martin		3			3	6
18/3/2014	Eirik		6				6
18/3/2014	Rúni		9	1			10 GUI
18/3/2014	Morten		5		1		6
18/3/2014	Christian		5				5
19/3/2014	Rúni		3				3
20/3/2014	Rúni		5				5
22/3/2014	Martin				1.5		1.5
22/3/2014	Christian				1.5		1.5
22/3/2014	Rúni		2		5	1	8
22/3/2014	Morten		2		2		4
23/3/2014	Martin		2		1.5		3.5
23/3/2014	Rúni		2		7	1	10
23/3/2014	Christian		2		4		6
23/3/2014	Morten		4		2		6
	Sum	12.5	89.5	6.5	34	57	199.5

Indholdsfortegnelse

[Timeregnskab](#)

[Indholdsfortegnelse](#)

[Indledning](#)

[Kravspecifikation](#)

[Domænemodel](#)

[Use Cases](#)

[Udfør måling af data på batch](#)

[Hent data](#)

[Design](#)

[Implementering](#)

[.control](#)

[MainController](#)

[LoginController](#)

[MainMenuController](#)

[BatchMeasureController](#)

[DatabaseController](#)

[SocketController](#)

[DasyFileReader](#)

[.view](#)

[LoginGUI](#)

[MainMenuGUI](#)

[BatchMeasureGUI](#)

[NewBatchGui](#)

[C# Komponent](#)

[C# komponent - JavaConverter-klasse](#)

[C# komponent - Asynkron socket listener \(socket-klasse\)](#)

[HTML \(Hyper-Text Markup Language\)](#)

[Udviklingsstatus](#)

[Udviklingsplan](#)

[Tredje iteration:](#)

[Bilag](#)

[Bilag 1 - Java/C# protokol](#)

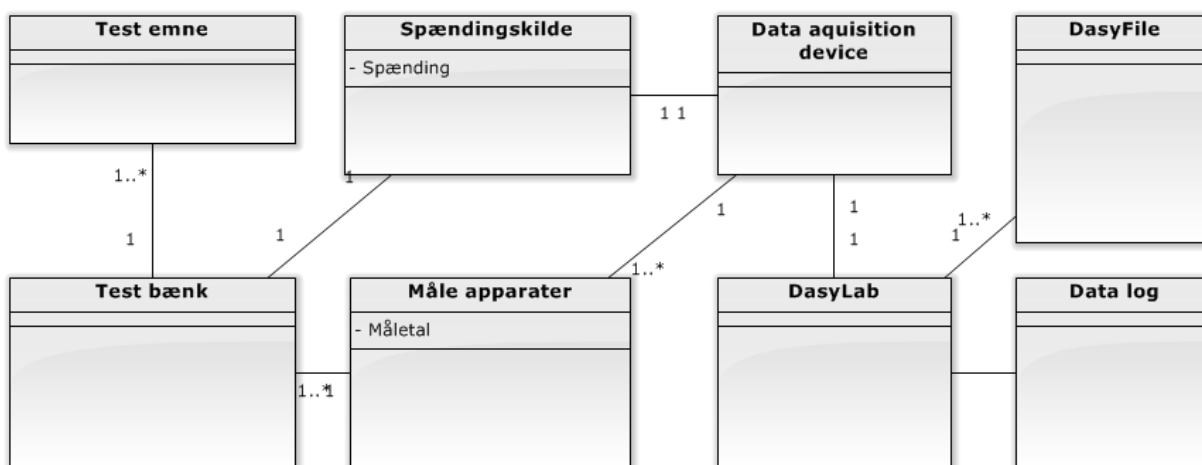
Indledning

Vi er nu halvvejs i projektet og følger i store træk tidsplanen. Vi har forfinet kravspecifikation og design af løsningen - herunder en præliminær protokol for kommunikation mellem vores C# komponent og Java komponent. Desuden har vi udviklet en GUI prototype til vores program og

Kravspecifikation

Domænemodel

Efter endnu en gennemgang med kunden har vi fået udspecificeret domænemodellen. Vi er nu landet på en løsning der ser ud som følger:



Kunden styrer måleapparaterne fra et program, DasyLab, der automatisk opsamler data fra et amperemeter, der måler en Leak current (lækstrøm) med 5 hz. Dataene gemmes automatisk i en fil (DasyFile - flad tekstfil, med en måling per linje). Lækstrømmen til tiden 65 udlæses og skrives i hånden på et datablad.

Kunden foretager ligeledes en måling af Stroke (deformation) som en enkelt måling. Denne måling aflæses direkte fra en mikrometerskrue, der dog kan tilsluttes en analog port og dermed kan aflæses via et Data aquisition device.

Use Cases

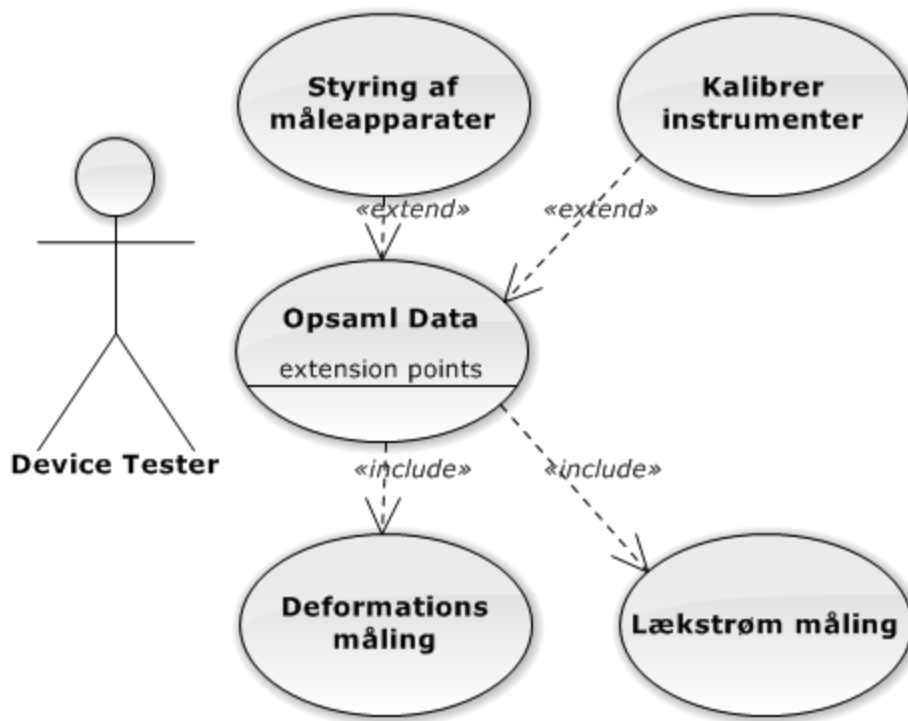
Vores uses cases er ligeledes opdateret til at reflektere hvad kunden finder er det vigtigste at implementere først.

Udfør måling af data på batch

Preconditions:

Der er fremfundet et batch af piezoelektriske elementer der skal testes. Testemnerne er monteret i testbænken. Måleinstrumenter tilsluttet korrekte porte på USB- Data Aquisition Device'et. Måleinstrumenter er tændt. Alle nødvendige apparatindstillinger er rigtige (senere

evt. implementering af styring fra PC).



Aktør	System
<ol style="list-style-type: none"> 1. Opstarter program 2. Logger ind. 3. Opretter Batch måling 4. Starter måling af Leak Current 5. Godkender data eller gentager måling 6. Starter måling af Stroke. 7. Godkender data eller gentager måling 8. Gentager 4-7 indtil alle emner er målt 9. Afslutter Batchmåling og logger ud 	<ol style="list-style-type: none"> 1. Modtager nødvendige brugerindstillinger Tilslutter USB -DAQ. 2. Modtager Brugerdata 3. Opretter Batch i database 4. Opsamler og lagrer data i DasyFile. DasyFile Indlæses og data for lækstrøm præsenteres for bruger. 5. Gemmer data i database eller kasserer data. 6. Måler direkte på mikrometer skrue og præsenterer data for bruger 7. Gemmer data i database eller kasserer data. 8. ds. 9. Logger bruger af

Extensions:

1a: Apparater kalibreres.

4a Alle måleapparater tilsluttes parallelt og softwaren kontrollerer spændingskilde og måleapparatur automatisk, så alle data logges i en arbejdsgang.

5/7b: Data lagres i en enten lokal eller cloud baseret SQL database til senere udtræk.



Hent data

Preconditions: Data er opsamlet og lagret.

Aktør 1. Starter browser og tilgår web-side 2. Vælger udtræk af data	System 1. modtager login 2. Præsenterer data.
---	--

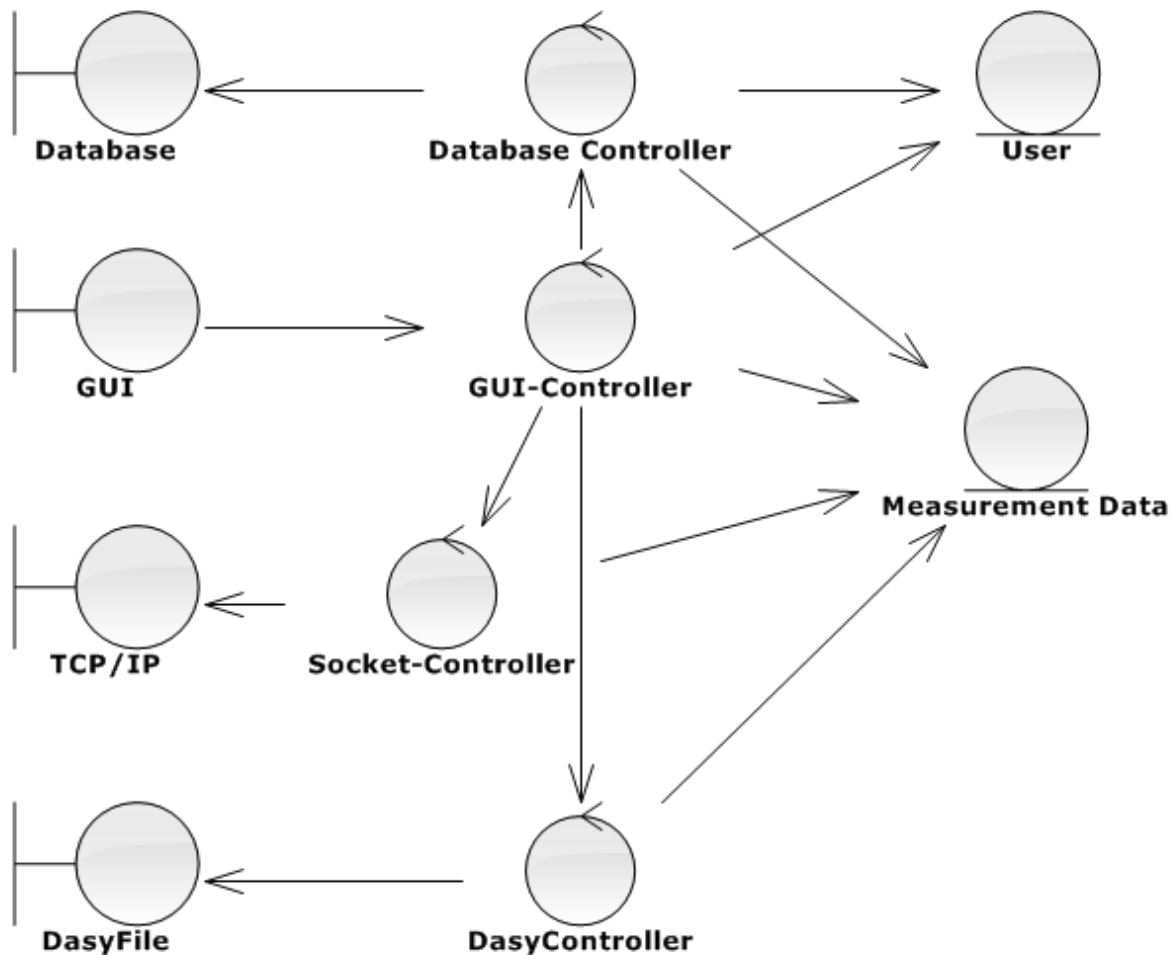
Extensions:

2a) Forskellige brugerdefinerede rapporter/udtræk

BCE-diagrammer

Der er ingen væsentlige ændringer i de eksisterende diagrammer for C# USB dataopsamlingskomponenten og Rapportmodulet. Der er tilføjet et nyt boundary til vores Dataopsamlingsmodul i form af en indlæsning af en DasyLab fil, der parses til vores system.

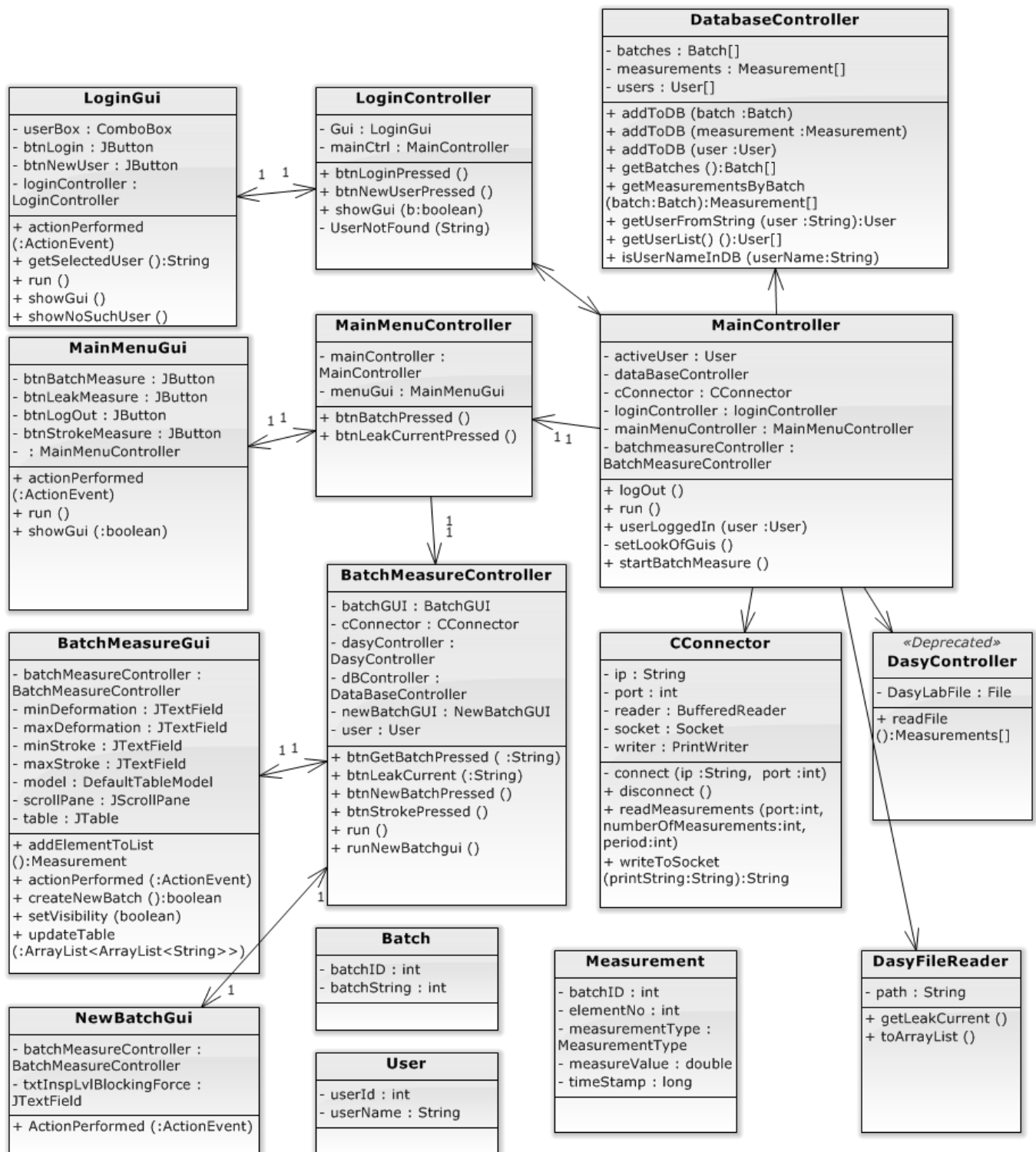
Brugermodule til kontrol af dataopsamlingen. En GUI interagerer med GUI-controlleren, der igangsætter opsamlingen af data fra Dataopsamlingsmodulet over TCP/IP og lagrer data i databasen.



Brugermodule til kontrol af dataopsamlingen. En GUI interagerer med GUI-controlleren, der igangsætter opsamlingen af data fra Dataopsamlingsmodulet over TCP/IP og lagrer data i databasen. Alternativt indlæses data fra en DasyFile, der er genereret af kundens software, DasyLab.

Design

Vi har ekspanderet klassediagrammet til modsvare vores nye behov for at interagere med kundens software (dasyLab). Desuden er relevante metoder tilføjet. Nedenstående diagram afspejler nuværende udseende af vores løsning. Flere klasser afventer refaktorering og Coupling kan optimeres (duplikatreferencer fjernes).



Klassediagram. Afspejler nuværende status - flere attributter er udeladt.

Implementering

Vi har oprettet klasser svarende til vores klassediagram. Indtil videre har vi haft fokus på GUI og Java-C# interaktion. Andre klasser er indtil videre test-driver klasser der returnerer sample data til at teste vores GUI. Således er der eksempelvis ingen databasefunktion i DatabaseController, men blot method-stubs, der returnerer testdata..

.control

MainController

Her bliver der holdt styr på de forskellige controllers. MainControlleren opretter alle de andre controllers og holder styr på når hvilken GUI skal vises.

LoginController

LoginControlleren sørger for at der bliver logget ind før man kan lave målinger og oprette batches. På denne måde kan vi nemt holde styr på hvilken bruger har oprettet batches, godkendt batches og lavet de forskellige målinger. LoginControllere skal senere også implementere at man kan oprette en ny bruger hvis denne ikke findes allerede. Men på nuværende tidspunkt sørger den kun for login.

MainMenuController

Efter man er logget ind bliver ansvaret ført over til denne controller. Idéen med den i første omgang, var at man kunne lave enkelte målinger uden at være tvunget til at lave de forskellige indstillinger først, men det bliver måske unødvendigt kompliceret, og måske kan MainMenuControlleren med dens gui spares væk.

BatchMeasureController

Denne controller styrer al logik bag BatchMeasureGui'en. Dette er altså en af de mere omfattende controllere, da den både skal kunne indhente data fra databasen, dasyControlleren og CConnector'en som er forbundet med C# komponenten, og som skal give os stroke målingerne. BatchMeasureController'en bliver oprettet af MainControlleren som er kernen af controllerne. Herfra bliver referencen til databasen og dasyControlleren parset.

Funktionaliteten er endnu ret begrænset og indtil videre kan vi kun køre programmet med dummydata som er sat ind i de forskellige metoder. Flere metoder kalder metoder i andre klasser som endnu ikke er blevet færdiggjort, men det har resulteret i at vi har fået identificeret flere af de metoder de forskellige klasser skal have.

DatabaseController

Vi har skrevet en dummy DatabaseController, der returnerer relevant data til fejltesting.

SocketController

Vi har skrevet en socket controller der forbinder til C# delen, og defineret en protokol for de to delene at følge. Controlleren udstiller en metode, "readMeasurements", der sender information

til C# delen, som beskrevet i den vedlagte protokol, og returnerer aflæsninger og en public disconnect metode. Forbindelsen bliver sat op når klassen bliver constructed, og hvis den er tabt når readMeasurements bliver kaldt prøver den at genetablere forbindelsen. Hvis det ikke virker, bliver en fejl sendt tilbage. Der er også mulighed for C# at returnere fejlmeddelelser, som fx. der ingen usb komponent tilkoblet. Den præcise protocol findes i bilag 1.

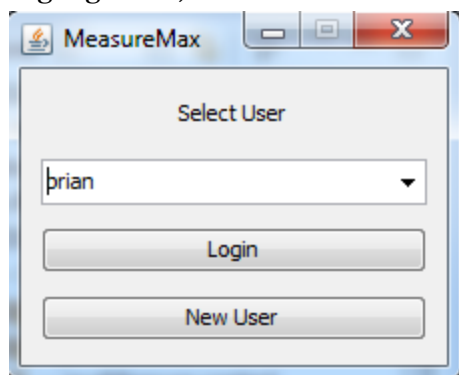
DasyFileReader

DasyFileReader er en klasse, der er tænkt til at læse filer fra DasyLab ind. Dens constructor tager en String "path" som argument, som er adressen på filen, der skal læses. Den bruger en BufferedReader med en FileReader til at læse filen. Der arbejdes på at få klassen til at kunne manipulere med filen. Metoden toArrayList skal kunne tage målingsværdierne fra en fil og sætte dem ind i en arraylist.

.view

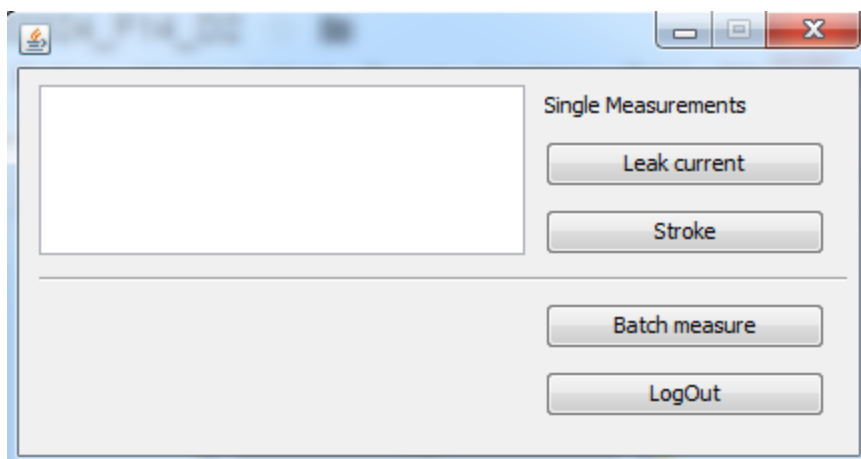
LoginGUI

Når man starter programmet, skal man først logge ind. Vi ser det ikke nødvendigt med adgangskode, derfor er der ikke noget felt til det.

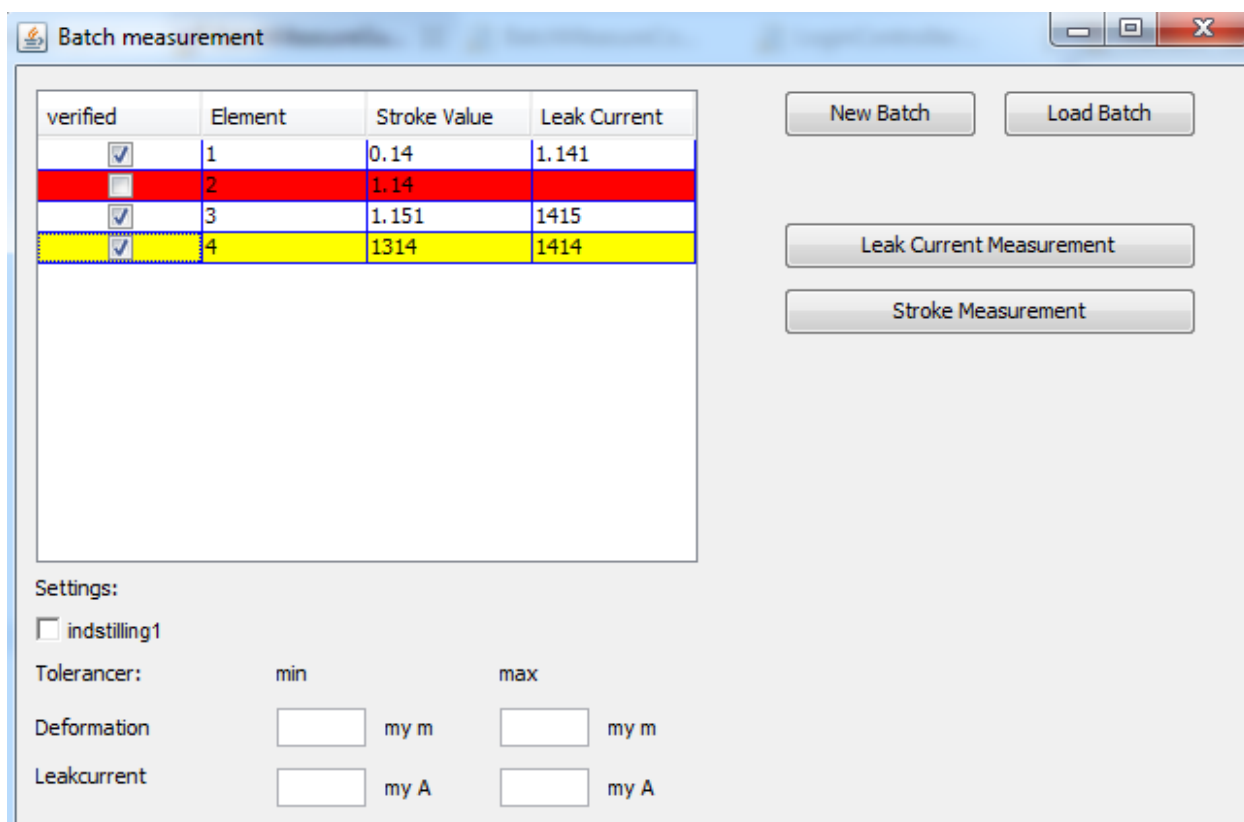


MainMenuGUI

Når man er logget ind kommer vinduet nedenfor frem. Her kan man tage en enkelt test måling eller lignende, hvor en batch ikke er nødvendig. tavlen skal muligvis være lige som den der er i BatchMeasureGui'en.



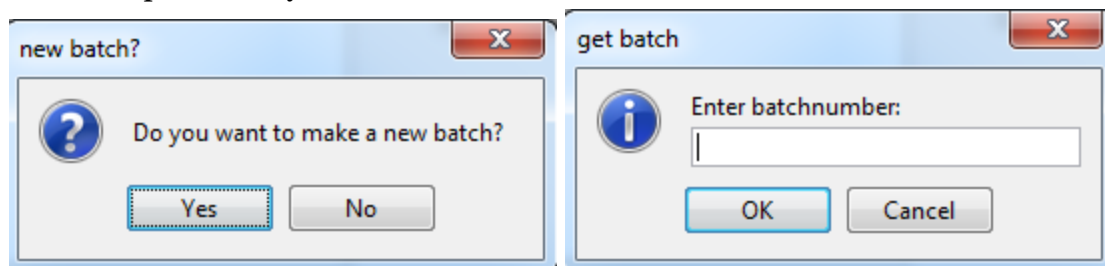
BatchMeasureGUI



Dette er stadig ikke helt færdigt, men det er helt tydeligt ved at tage form. Der er blevet lagt mange kræfter i at få tabellen til at fungere rigtigt, og samtidigt opfylde forskellige brugerkrav, som for eksempel checkbokse til hvert element, som bestemmer om elementet er godkendt eller ej. De forskellige felter der kan udfyldes nederst bliver nok fjernet, og istedet vil der komme felter som ikke kan redigeres som bliver sat da man trykker "load batch", da en oprettet batch har fået specificeret disse indstillinger. De rækker hvor fluebenet bliver fjernet bliver røde for at vise at de ikke er godkendt, mens den gule farve kun viser hvilket element der bliver målt på, og som nu er, er det altid det sidste i rækken. Når der er så mange rækker at de ikke alle kan vises i

vinduet, kommer en scrollbar ud til højre.

Hvis man trykker “load batch” popper vinduet nedenfor til højre op og man kan indtaste batchnummeret. Hvis batchen ikke eksisterer i databasen vil boksen til venstre poppe op, og man kan oprette en ny batch.



NewBatchGui

The 'New Batch' dialog box contains a 'Batch:' label and a text input field, followed by a 'Create Batch' button. Below this is a table with four columns: 'Specifications', 'norm value', 'tolerance', and 'inspection level'. The table lists various specifications for a batch, each with input fields for the values.

Specifications	norm value	tolerance	inspection level
outer diameter:	<input type="text"/>	±	<input type="text" value="min. 5"/>
inner diameter:	<input type="text"/>	±	<input type="text" value="min. 5"/>
thickness:	<input type="text"/>	±	<input type="text" value="min. 5"/>
elekt. marg. OD:	<input type="text"/>		<input type="text"/>
elekt. marg. ID:	<input type="text"/>		<input type="text"/>
number of active layers:	<input type="text"/>	±	<input type="text"/>
layer thickness:	<input type="text"/>	±	<input type="text"/>
capacitans:	<input type="text"/>	±	<input type="text" value="min. 10"/>
stroke:	<input type="text"/>	±	<input type="text" value="AQL 0.65"/>
blocking force:	<input type="text"/>	±	<input type="text"/>
measure voltage:	<input type="text"/>		<input type="text"/>
dynamic test:	<input type="checkbox"/>		<input type="text"/>
leak current, max:	<input type="checkbox"/>		<input type="text" value="min. 20"/>
DC resistance:	<input type="text"/>	±	<input type="text"/>
special visual inspection:	<input type="checkbox"/>		<input type="text" value="AQL 0.65"/>
spectrum:	<input type="checkbox"/>		<input type="text"/>

Dette vindue popper op når man vil oprette et nyt batch. Der er ikke blevet lavet nogle funktioner bag denne endnu, og som nu er bliver den kontrolleret af BatchMeasureControlleren. Disse indstillinger bliver vist på BatchMeasureGui'en. Mange indstillinger er ens hver gang (næsten) og for brugervenlighed har vi udfyldt nogle felter i forvejen (bliver måske flere felter senere).

C# Komponent

Vores C# komponent står for opsamlingen af data fra USB-DAQ'en. Vi har udviklet en rudimentær protokol til:

1. at forespørge om en serie af dataopsamlinger med et givent interval fra en given port på USB-DAQ'en
2. modtage en serie af data med identifikation af port, timestamp og måleværdi. I protokollen er også specificeret format for fejlmeddelser.

Den præcise specifikation for protokollen findes i bilag 1.

C# komponent - JavaConverter-klasse

Klassen JavaConverter indeholder metoder til oversættelse af input og output der modtages og sendes over netværkssocket (dvs. fra og til javakomponenten).

Klassen indeholder indtil videre metoderne javaInputConverter og javaOutputConverter.

Det modtagne data fra Javakomponenten er en string, og metoden javaInputConverter sørger for at konvertere strengen til de korrekte kommandoer.

Metoden javaOutputConverter konverterer den foretagne måling til en streng, så den er klar til at blive sendt til Javakomponenten.

C# komponent - Asynkron socket listener (socket-klasse)

Socket-klassen i C# komponenten er kodet som en asynkron socket listener. Dette betyder at hovedtråden fortsætter med eksekvering, mens en eventlistener afventer at der kommer noget data fra Javakomponenten.

HTML (Hyper-Text Markup Language)

Vi har udviklet en form i HTML, der indtil videre indeholder de basale rapportkriterier som kunden ønsker, når der skal laves udtræk til statistik. Her er der tale om, at det er muligt at indtaste et Batchnummer eller - fra en liste - vælge det ønskede Batchnummer. Derudover vil det være muligt at vælge et udtræk ud fra datoen ved hjælp af et Javascript. Der vil senere blive lavet kode til at generere udtrækket, men da ikke er nogen database kan udtrækket endnu ikke genereres.

Selve HTML-delen består indtil videre af en enkelt side, skrevet i Notepad++, med et <head> indeholdende de informationer, som er standard i et site, og <body> hvor det grafiske forefindes. Overordnet består <body> af en container kaldet "wrapper", der indeholder en header, hvori firmaets logo finde, og den omtalte form med tilhørende Javascript (hvilket vil blive beskrevet længere nede i teksten). Som udgangspunkt er det hele pakket ind i div-kasser, hvor de fleste er angivet med et ID, hvilket blandet andet bruges i forbindelse med CSS.

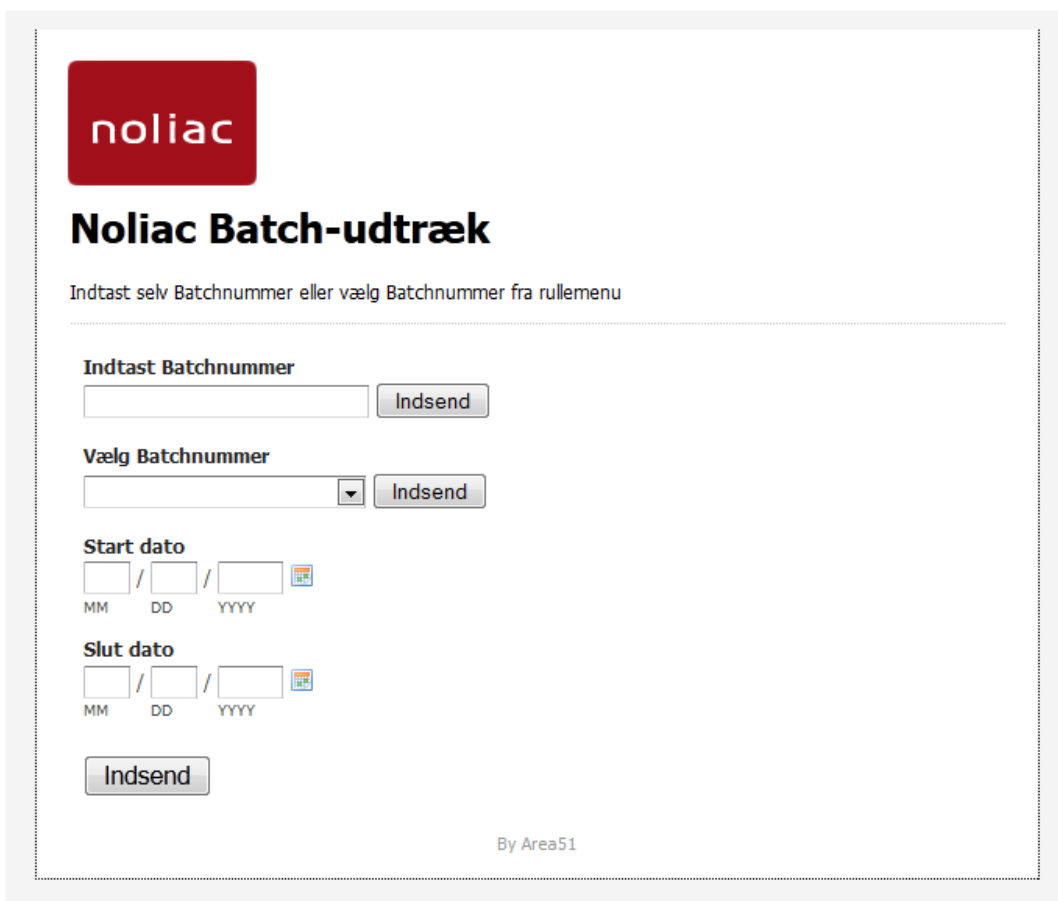
```
<div id="wrapper">

  <div id="header"></div>

</div>
```

Eksemplet ovenfor viser et udsnit af vores “wrapper”, som til start indeholder “header”-en.

Designet er lavet i henhold til Noliacs egen hjemmeside, så farver stemmer overens med hvad der ses på www.noliac.com. Dette har vi gjort ved at gå ind og kigge på kildekoden for firmaets hjemmeside. I henhold til deres CSS er farverne valgt og brugt i vores design.

The image shows a web form titled "Noliac Batch-udtræk". At the top left is the Noliac logo, a red square with the word "noliac" in white. Below the logo is the title "Noliac Batch-udtræk" in bold black text. Under the title is a subtitle "Indtast selv Batchnummer eller vælg Batchnummer fra rullemenu". The form contains three main sections: 1. "Indtast Batchnummer" with a text input field and an "Indsend" button. 2. "Vælg Batchnummer" with a dropdown menu and an "Indsend" button. 3. "Start dato" and "Slut dato" sections, each with MM/DD/YYYY input fields and a calendar icon. At the bottom of the form is a large "Indsend" button. The footer of the form says "By Area51".

Ved brugen af CSS gør vi det lettere at ændre i designet fremover. Vi har valgt at lægge det i en separat fil og importere det i <head>. Skulle det være, at hjemmesiden får nye farver, så er det enkelt at kunne gå ind i CSS-filen og ændre farverne her, modsat at skulle ændre farverne på de enkelte attributter i HTML'en, eller i <head>.

CSS er kreeret ud fra ID og class. Hvor ID definerer det enkelte objekt og derved generelt ikke bruges yderligere, hvorimod class bliver genbrugt. Derudover er det muligt at tilføje designelementer direkte på attributter. Nedenfor er der et udsnit fra vores CSS, hvor der vises de

tre måder hvorpå man kan lægge design på HTML'en.

```
/*Eksempel på design på attribut*/
body
{
  background:#f4f4f4;
  font-family:"Lucida Grande", Tahoma, Arial, Verdana, sans-serif;
  font-size:small;
  margin:8px 0 16px;
  text-align:center;
}

/*Eksempel på design på ID*/
#wrapper
{
  border-left:1px dotted black;
  border-right:1px dotted black;
  border-bottom:1px dotted black;
  width:642px;
  margin: 0 auto ;
}

/*Eksempel på design på class*/
.appnitro
{
  font-family:Lucida Grande, Tahoma, Arial, Verdana, sans-serif;
  font-size:small;
}
```

Eksemplerne er taget direkte fra vores CSS.

Når der skal hentes udtræk ud fra en dato har vi valgt, at dette gøres ud fra en kalender. For at gøre det let for brugeren har vi valgt at bruge et Javascript. Dokumentationen for kalenderen forefindes på <http://www.dynarch.com/jscal/>.

Udviklingsstatus

Jævnfør første rapport havde vi valgt følgende fokusområder for vores første iteration af projektet:

1. Videreudvikling af kravspecifikation.
 - a. Mockup af brugerinterface i samarbejde med bruger (High priority)
 - b. Specificere krav til html-rapport generering.¹
 - c. Identifikation af yderligere use cases.
 - d. Indhentning af specifikationer på måleapparat og
 - e. Afklaring af dataopsamlingsmodul - enkelte opsamlinger/ flere samples over tid.

¹ Sideløbende med relevant undervisning.

- f. Billede og udspecificering af testopstilling.
 - g. Kopier af sample data dokumenter til udvikling af datadesign.
- 2. Design
 - a. Specifikation af interface - low level socket server?/Web server?
 - b. Specifikation af kommunikationsprotokol
 - c. Udvikling af klassediagram for Java-del
 - d. Udvikling af sekvensdiagram for Java-del
- 3. Kode
 - a. Implementering af fungerende interface mellem C# komponent og Java del (High Risk, high priority).
 - b. Udvikling af Java komponent til dataopsamling.
 - i. Begynde på GUI-programmering (bundet af 2a)
 - ii. Implementere threaded datakommunikation med C# komponent (low priority)
 - c. Begynde på HTML del
- 4. Test
 - a. C# komponent testing
 - i. testing mod dummyapparat
 - ii. evt. test mod real life apparatur (high risk)
 - b. Interface testing

Mål: Fungerende interfacing mellem Java og C# komponent

Slut: 23/3 - 14

Ad)

1. Kravspecifikation

- a. Vi illustrerede et interface i Excel og foreviste kunden det. De mente at designet var passende og at vi skulle arbejde videre i den retning, men havde ikke nogle umiddelbare ændringer eller tilføjelser. Vi har udarbejdet vores nuværende GUI udfra disse mockups, da disse blev accepteret, men udfyldningsskemaer som vi har fået fra kunden har dog været med til at vi ændret og tilføjet nogle flere ting, så som newBatchGui'en.

Login	
Bruger	<input type="text"/>
<input type="button" value="Ny bruger"/>	

Noliac Measure Max - Logged in as CB			
Enkelt måling			
Måling	<input type="text" value="Lækstrøm"/>		Kalibrer
Tolerancer	Max	Min	Andre indstillinger
Lækspænding	??	??	
Anden måling			
Tredje måling			
Lækstrøm	<input type="text" value="123"/>		
Måling i gang			
<input type="button" value="Gem Måling"/>	<input type="button" value="Ny måleSerie"/>	<input type="button" value="Hent MåleSerie"/>	<input type="button" value="Afslut Program"/>

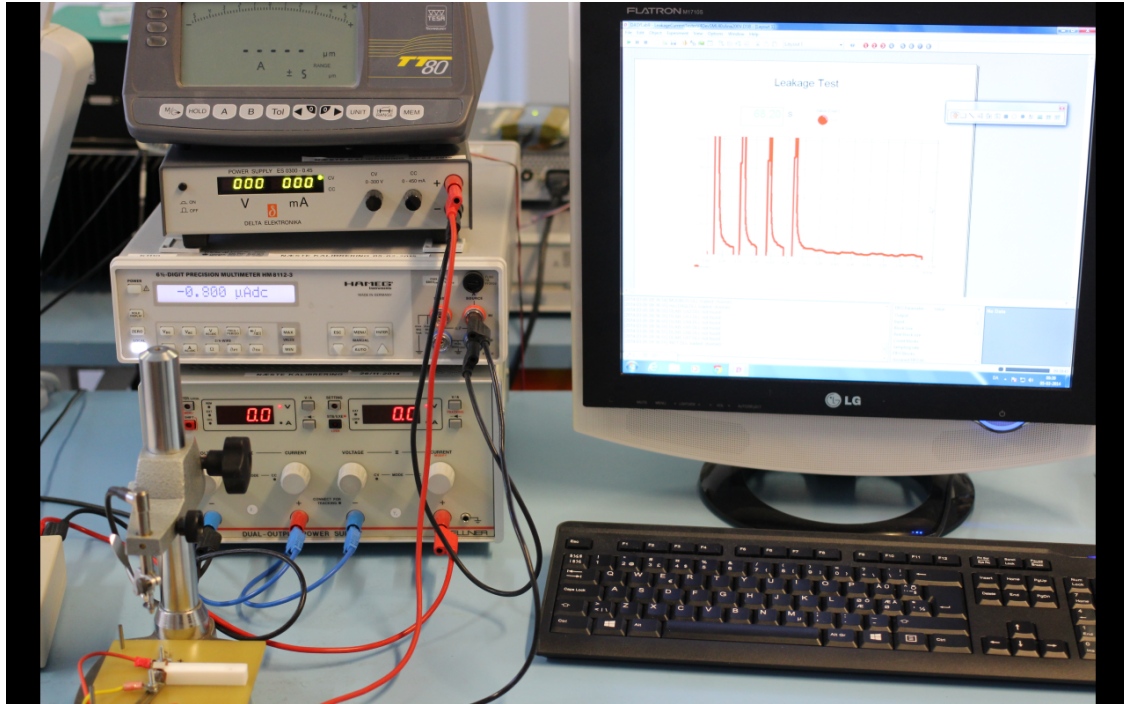
Noliac Measure Max - logged in as TestBruger				
Element	Lækspænding	Anden måling	Tredje måling	
1	2	234234	123	
2				
Måling i gang...				

Batchnummer 123 Elementnummer 1			
Operator	AD		
Tolerancer	Max	Min	Andre indstillinger
Lækspænding	??	??	
Anden måling			
Tredje måling			
<input type="button" value="Godkend måling"/>	<input type="button" value="Kasser måling"/>		<input type="button" value="Afslut måleSerie"/>

- b. Vi er ikke kommet meget længere med specifikation af html-rapport generering, da rapport genereringen er en mindre kritisk use case og vi derfor venter med at implementere mere komplicerede html rapporter. Indtil videre planlægger vi at generere rapporter i csv format, således at de kan bearbejdes eksempelvis i excel. Vi har kun for nylig modtaget nogle af de rapporter som kontrolløren af slut kontrollen udfylder og har heller ikke haft relevant undervisning endnu hvorfor vi har nedprioriteret dette. Vi planlægger at udarbejde et brugervenligt web-interface til at hente rapportdata og hvis tiden tillader det vil vi også implementere forskellige autogenererede rapporter.
- c. Vi har ikke identificeret nye uses cases, men har opdateret de nuværende ud fra kundens ønsker og opdateringer kommer løbende.
- d. Vi er kommet frem til at måling af lækstrøm er nemmest at få fat i ved at læse en tekstfil som bliver genereret af DasyLab som er det program de bruger på

tidspunktet. Da denne måling er ret kompliceret idet at den foregår over tid, og målingen skal accepteres ved visuel inspektion af en graf, forventer vi ihvertfald i første omgang at det bliver en lidt for stor mundfuld. Den anden måling, Stroke målingen, bliver målt med en mikrometer skrue, som vi kan kommunikere med ved hjælp af et lille program skrevet C#. Nogle af problemerne som nu er, er at mange ting foregår manuelt. For eksempel er der en afbryder til strømforsyningen som de bruger til stroke målingerne, og her skal vi finde en smart måde at få fat i de rigtige værdier på det rigtige tidspunkter. status på specifikationer på måleapparatur

- e. Mikrometerskruen har en analog udgang, som vi skal forsøge at få læst noget data fra, næste gang vi er hos kunden. Vi har modtaget manual til apparatet, og det burde være muligt at aflæse signalet via USB DAQen.
- f. Vi har fået fastlagt hvordan data skal opsamles. Lækstrøm målingen bliver hentet fra en fil som DasyLab generer. Vi har med succes formået at indlæse denne fil, og i næste iteration burde behandling af filen være færdig således at vi får den rigtige måling ud af filen. Stroke målingen gennemføres ved hjælp af en C# komponent som kommunikerer med DAQ-devicet, dette er dog ikke blevet testet endnu, så vi er ikke sikre på om det fungerer rigtig.
- g. Testopstillingen ses på billedet nedenfor. På skærmen ses den graf som bliver lavet i dasyLab. På bænken nederst i venstre hjørne ligger det lille element der bliver målt på. Det apparat der er øverst måler stroke målingen, dette er altså dette apparat mikrometerskruen er forbundet til, og som vi skal få data ud fra. Dernæst kommer et amperemåler som bruges til målingerne som ses på skærmen



- h. Vi har fået nogle eksempler af de filer som DasyLab generer når den laver leak current målingerne. Disse filer kan findes i projektet under DasyLab files, og kan åbnes med notesblok eller andet tekst program. Som nu er bliver brugeren selv nødt til at finde disse filer, men muligvis kan vi finde en måde at identificere de rigtige filer automatisk.
2. Design
 - a. Vi besluttede os indtil videre for en low-level løsning til kommunikation mellem C# og Java da det vil være begrænset hvor forskelligartet data vi skal sende mellem de to komponenter. Viser det sig at vi har et behov for mere kompliceret dataudveksling vil vi muligvis skifte til en mere standardiseret protokol eks. XML eller JSON
 - b. Vi har specificeret en protokol for kommunikation mellem java og C#, den beskriver forbindelse, hvilke requests og responses vi har defineret, og hvilke fejl vi behandler.
 - c. Vi har videreudviklet klassediagrammet så det modsvarer den nye sammenhæng i vores program.
 - d. Vi er først nu ved at opnå forståelse af flowet i en målingerne og hvordan vi bedst omsætter det i vores program og vi har derfor ikke omsat det i et sekvensdiagram. Vi vil prioritere det i næste iteration.
3. Kode
 - a. Javadelen er klar til testning, men vi har ikke gjort den klar til at behandle alle

fejlene beskrevet i protokollen. C# komponenten kan nu forbindes med Javakomponenten. Der er lavet JUnit testcase, der tester om der er forbindelse. Der forestår stadig at implementere den fulde protokol beskrevet i bilag 1.

- b. Java komponenten kan endnu ikke opsamle data, som vi havde håbet, da kommunikationen mellem java og C# ikke fungerer endnu. Men vi har kodet nogle af redskaberne som skal behandle de opsamlede data. Også har vi lavet en meget fin brugergrænseflade, som dog mangler nogle detaljer endnu.
 - c. Vi har kodet et mockup Interface i html/css/js som vi vil forelægge vores kunde mhp at få input til at arbejde videre. Mockup'et er vedlagt projektet i HMTL-mappen.
4. Test
- a. C# komponenten er ikke klar til kommunikationstests endnu.
 - b. Der har ikke været så meget at teste, da det meste af det vi har lavet kun er meget simple ting så som JButton og JTextField, men der har været nogle test med hensyn til tabellen i BatchMeasureGui'en, da vi har overvretet nogle metoder i JTable for at få det ønskede display og muligheder som vi skal have med. For eksempel skulle kun den første kolonne kunne ændres, da denne indeholdt checkboxes, mens de andre selvfølgelig ikke skulle ændres da det så ville være nemt at manipulere data.

Mockup af user interface.

Udviklingsplan

Idet vi i store træk følger vores plan, er der ikke større ændringer i vores plan fremadrettet. Vi har dog forfinet vores plan i hht de svar vi har fået fra Noliac.

Tredje iteration:

- 1. Kravspecifikation
 - a. Noliac foretrækker indtil videre en løsning uden password beskyttelse for simplicitetens skyld. Vi lader muligheden for passwordbeskyttede konti stå åben.
 - b. Vi skal planlægge endnu et møde med Noliac, da vi løbende identificerer designbeslutninger, som kunden bliver nødt til at tage stilling til. Vores brugergrænseflade, som vi er kommet godt på vej med, skal godkendes af kunden.
- 2. Design
 - a. Konstruktion af database - skal reflektere de dataobjekter vi indtil videre har modelleret - Users, Batches, Measurements.

- b. Sekvensdiagram for main use case.
- 3. Kode
 - a. Færdiggøre gui
 - b. Færdiggøre C# og java connector, socket programmering(high priority)
 - c. Interface til Database -
 - d. Implementering af SQL/JSP i HTML/CSS grænseflade
 - e. færdiggøre scannerprogrammering til indlæsning og manipulation med Dasy Lab filer
- 4. Test
 - a. Test af C# - java interaktion
 - b. Test af java - C# - USB-DAQ
 - c. Test af dataopsamling mod real-life apparatur (high risk)

Mål: Fungerende dataopsamling og webinterface

Slut: 27/4

Sidste iteration:

Bugfixing, optimering og evt. udvidelse af implementering

- 1. Optimering af kodebase/GUI
 - a. Test af grænsetilfælde
 - b. Inputvalidering
- 2. Evt:
 - a. Implementering af 'custom' rapportgenerering.
 - b. Måske rapportopslag for kundens kunder efter batchnr?
 - c. Statistisk udregning af sample size for at garantere en vis tolerance med en given sandsynlighed.
 - d. Brugerstyring

Mål: Optimering af løsning, evt. implementering af flere use cases

Slut: 11/5

Bilag

Bilag 1 - Java/C# protokol

Protocol for java to C# socket communication.

The C# component should be set to listen, and java will connect when the program starts. By default, java will try to connect to localhost port 4567, but this can be overloaded.

When the connection has been established, the following requests and responses are allowed.

Request 1:

Java sends a string to C# in the following format: port+";"+number+";"+period
where port is the desired port on the DAQ to read from, number is the desired amount of readings, and period is the time between readings
port, number and period are ints.

Response 1:

C# then returns a string in the format: data+";"+timeStamp+";"
this is repeated number times.

Data is the reading, a float value, and timestamp is the time since the previous reading(long).

If an error has occurred, C# returns a string in the format: error+";"+number+";"
error is a string that reads error, and number is an int, that describes what error has occurred.

List of errors:

error 00: The DAQ is not connected

error 01: no connection between java and C#