



MARG

Modular Applet Robot GUI

Forfattere:

Bjørn Truelsen

Daniel Freiling

Kristina Hansen

**Hovedopgave
5. semester, Datamatiker
Roskilde Handelsskole
November 2009**

Vi giver hermed tilladelse til at denne projektrapport må benyttes af Roskilde Handelsskole og DTU og frit må kopieres af disse organisationer, så længe indholdet forbliver det samme.

Underskrifter:

Dato: **10/11/09**

Titel: **MARG (Modular Applet Robot GUI)**

Søgeord:

hovedopgave, datamatiker, robot, applet, java, systemudvikling, programmering,

Resumé:

Dette er en rapport til hovedopgaven på datamatiker studiet. Den omhandler udviklingen af en GUI til robotter på DTU der afvikles som en applet. Projektet er udviklet ved hjælp af udviklingsmetoden eXtreme Programming, og inddragede UML artefakter. Systemet er implementeret i Java og berører emner så som XML, Socket forbindelser og plugin arkitektur.

Forord

Som afslutning på Datamatiker uddannelsen skal der laves en hovedopgave. De studerende skal igennem denne og en efterfølgende eksamination vise at de kan anvende og har forståelse for de forskellige praktiske og teoretiske dele lært under uddannelsen på egen hånd.

Vores hovedopgave er lavet ud fra en opgave stillet af DTU Elektro om at lave en modulær brugergrænseflade hvor der kan overvåges variabler og styres forskellige dele af deres robotter. I den forbindelse vil vi gerne takke DTU Elektro for et godt samarbejde samt et interessant og udfordrende projektoplæg.

Vi har valgt at kalde det udviklede system for Modular Applet Robot GUI (**MARG**)

Rapportens opbygning er lavet således:

Projekt Etablering: Gruppe og projekt etablering, herunder vores problemformulering og valg af udviklingsmetode.

XP Prerelease: Teorien omkring XP artefakter, ekstra Artefakter vi har trukket ind og relevant forundersøgelse og analyse.

XP projekt: 3 Releases omkring udviklingen af **MARG**

Review: Beskrivelse af Review med anden projekt gruppe

Diskussion og Konklusion: Evaluering af udviklingsprojektet, diskussion af udviklingsmetoden, evaluering af produktet og diskussion og konklusion af problemformulering.

Litteraturliste: Liste over anvendt litteratur

Bilag til Rapporten og Kodebilag ligger i et dokument for sig.
CD-rom med MARG og guide findes bag i Bilaget.

Indhold

Forord.....	3
1. ProjektEtablering	8
1.1 Indledning	8
1.2 Plan for projektetableringen	8
1.3 Beskrivelse af projektpartner	8
1.4 Projektoplæg fra projektpartner	9
1.4.1 DTU Robot	9
1.4.2 Systemet bag GUI'en	9
1.4.3 Selve GUI'en	9
1.5 Problemformulering	9
1.5.1 Problemstilling	9
1.5.2 Hypoteser	10
1.5.3 Uddybende spørgsmål	10
1.5.4 Hvordan kan hypoteserne og spørgsmålene besvares?	10
1.6 Projektafgrænsning.....	10
1.7 Brugerindflydelse.....	10
1.8 Andre mål	11
1.9 Valg af udviklingsmetode.....	11
1.9.1 Indledning.....	11
1.9.2 Krav til udviklingsmetode.....	11
1.9.3 Overvejelser om valg af udviklingsmetode:.....	12
1.9.4 Diskussion af overvejelserne.....	12
1.9.5 Konklusion	13
1.10 Projekt gruppen.....	14
1.10.1 Basis proces	14
1.10.2 Styrker og svagheder	14
1.10.3 Risici	15
1.10.4 Konklusion på projektgruppen	15
1.11 Projekt Plan	15
1.12 Software.....	16
1.13 Evaluering af projektetableringsfasen	16
2. XP Prerelease.....	17
2.1 Indledning	17
2.1.1 Projekt Plan for Prerelease.....	17
2.2 Teori	17
2.2.1 Overblik	17
2.2.2 XP Artefakter	17
2.2.3 Ekstra Artefakter	21
2.3 Relevante forundersøgelser og analyse	23
2.3.1 Kvalitets Faktorer.....	24
2.3.2 Risici til XP projektet	26
2.3.3 Domæne Model.....	26
2.4 Evaluering af Prerelease	27

3. eXtreme Programming Release 1	28
3.1 Indledning	28
3.1.1 Projekt Plan	28
3.2 Metafor	28
3.3 User-stories	29
3.4 Spike Investigations	29
3.4.1 Planlagte Spikes	30
3.4.2 Udførelse af Spikes	30
3.4.3 Spike resultater	31
3.5 Releaseplan	32
3.6 Task Cards	33
3.7 Iteration Plan	33
3.8 Design Klasse Diagram	34
3.9 Class, Responsibilities and Collaboration	35
3.10 Acceptance Test	38
3.11 Programmering	39
3.11.1 Beskrivelse af kode	39
3.11.2 Generelle overvejelser om programmeringen	41
3.12 Sekvens Diagrammer	41
3.12.1 Forbind til Modul	42
3.12.2 Send kommando til Modul	43
3.12.3 Modtagelse af XML data	44
3.13 Brugergrænseflade	45
3.14 Opdaterede Task Cards	47
3.15 Testning	49
3.16 Refaktorering	49
3.17 Revideret Design Klasse Diagram	50
3.18 Implementering af ekstra user-story	51
3.18.1 User-story	51
3.18.2 Task Card	51
3.18.3 Acceptance Test	51
3.18.4 Programmering	52
3.18.5 Refaktorering	53
3.18.6 Overvejelser til ekstra user-story	53
3.19 Præsentation af Release 1 hos projektpartner	54
3.19.1 Acceptance tests	54
3.19.2 Evaluering af Release 1 med projektpartner	55
3.20 Velocity Chart	55
3.21 Burn down chart	56
3.22 Evaluering af Release 1	56
4. eXtreme Programming Release 2	58
4.1 Indledning	58
4.1.1 Projekt Plan	58
4.2 User-stories	58
4.3 Revideret Releaseplan	59

4.4 Iterationsplan	60
4.5 Task Cards	60
4.6 Acceptance Tests	61
4.7 Sekvens Diagrammer	61
4.8 Programmering	62
4.9 Brugergrænseflade	67
4.9.1 Opdateret brugergrænseflade	67
4.9.2 Beskrivelse af brugergrænsefladetest	68
4.10 Opdaterede Task Cards	68
4.11 Testning	70
4.12 Refaktorering	71
4.13 Præsentation af Release 2 hos Projektpartner	71
4.13.1 Acceptance tests	72
4.13.2 Evaluering af Release 2 med projektpartner	72
4.13.3 Resultat af brugergrænsefladetest	73
4.14 Velocity Chart	73
4.15 Burn down Chart	74
4.16 Evaluering af Release 2	74
5. Review	75
5.1 Referat af Review	75
5.2 Evaluering af Review	76
6. eXtreme Programming Release 3	77
6.1 Indledning	77
6.1.1 Projekt Plan	77
6.2 Revideret Releaseplan	77
6.3 User-stories	78
6.4 Iterationsplan	78
6.5 Task Cards	79
6.6 Acceptance tests	79
6.7 Programmering	79
6.7.1 Programmering i dette Release	79
6.7.2 Brug af repository model	83
6.8 Brugergrænseflade	83
6.9 Opdaterede Task Cards	85
6.10 Testning	87
6.11 Refaktorering	88
6.12 Præsentation af Release 3 hos Projektpartner	89
6.12.1 Acceptance Tests	89
6.12.2 Evaluering af Release 3 med Projektpartner	89
6.13 Velocity Chart	90
6.14 Burn Down Chart	90
6.15 Evaluering af Release 3	90
7. Diskussion og Konklusion	92
7.1 Evaluering af udviklings projekt	92

7.2 Diskussion af Problemformulering	95
7.3 Diskussion af udviklingsmetode.....	97
7.4 Evaluering af produktet	98
7.4.1 Produktets funktionalitet.....	98
7.4.2. Manglende funktionalitet.....	98
7.4.3 Videreudvikling af produktet.....	98
7.5 Konklusion af hovedopgave.....	100
8. Litteraturliste	101

Bilag og Kode Bilag findes i tilhørende dokument
CDrom er bag i Bilag

1. ProjektEtablering

1.1 Indledning

Udarbejdet af: Bjørn, Daniel, Kristina

I denne indledende fase vil vi beskrive den overordnede opgave, vores afgrænsning af den og hvordan vi har tænkt os at udarbejde den. Den skal også dokumentere vores kontrakt med projektpartner og projektgruppens opbygning og risici.

1.2 Plan for projektetableringen

Udarbejdet af: Kristina

Dette er vores arbejdsplan for projektetableringsfasen. Den er estimeret ud fra tidligere erfaring.

ID	Task Name	Start	Finish	Duration	aug 2009							sep 2009						
					25	26	27	28	29	30	31	1	2	3	4	5	6	7
1	ProjektEtablering	25-08-2009	07-09-2009	10d														
2	ProjektEtableringsplan	25-08-2009	25-08-2009	1d														
3	Indledning	25-08-2009	25-08-2009	1d														
4	Beskrivelse af Projektpartner	26-08-2009	26-08-2009	1d														
5	Besøg på DTU	28-08-2009	28-08-2009	1d														
6	Projekt Oplæg	28-08-2009	28-08-2009	1d														
7	Valg af metode	31-08-2009	04-09-2009	5d														
8	Problemformulering	02-09-2009	02-09-2009	1d														
9	Projekt Afgrænsning	02-09-2009	02-09-2009	1d														
10	Andre mål	02-09-2009	02-09-2009	1d														
11	Godkendelse af problemformulering	02-09-2009	02-09-2009	1d														
12	Brugerindflydelse	03-09-2009	03-09-2009	1d														
13	Projekt Kontrakt	07-09-2009	07-09-2009	1d														
14	Projektgruppen og basis proces	07-09-2009	07-09-2009	1d														
15	Risici til projektarbejdet	07-09-2009	07-09-2009	1d														
16	Projekt Plan	07-09-2009	07-09-2009	1d														
17	Software	07-09-2009	07-09-2009	1d														
18	Evaluering af Projekt Etablerings fasen	07-09-2009	07-09-2009	1d														

1.3 Beskrivelse af projektpartner

Udarbejdet af: Daniel

Vores projektpartner i dette projekt er Danmarks Tekniske Universitet, DTU, Elektro¹. DTU er en selvejende institution som har fokus på uddannelse, forskning og innovation. DTU Elektro er et institut under DTU som har fokus på uddannelse og forskning i elektro-teknologi. I DTU Elektro er der en afdeling der hedder "Automation and Control" som fokuserer på automatiserede systemer og kontrollen af dem. Det er denne afdeling vi skal samarbejde med.

¹ DTU Elektro – Se litteraturliste

DTU Elektro har nogle kurser og projekter hvor de udvikler og bruger robotter. De har dog ikke så gode brugergrænseflader til disse robotter, da de i øjeblikket er meget basale og tekniske at arbejde med. Dette gør at de sjældent har tiden eller ressourcerne til at få robotterne op at køre når de skal demonstreres for andre.

1.4 Projektoplæg fra projektpartner

Udarbejdet af: Bjørn

Fra vores projektpartner har vi fået et projektoplæg, der ser således ud:

1.4.1 DTU Robot

- Der skal laves en brugergrænseflade til en robot
- Det skal være en Java applet der kan afvikles i hvilken som helst internet browser og dermed skal systemet ikke kræve nogen installation udover Java Virtual Machine.

1.4.2 Systemet bag GUI'en

- Det skal være modulært og konfigurerbart så det kan bruges til flere robotter og/eller nye moduler.
- Det skal bruge en socket-baseret TCP/IP forbindelse til at kommunikere med robotten.
- Det skal både kunne sende og modtage data fra robotten i XML.
- Det skal være nemt at specialisere et modul så ny funktionalitet kan tilføjes til et generelt modul. Dokumentation til hvordan dette gøres er vigtig da det skal være muligt at udføre for folk med et begrænset kendskab til Java.

1.4.3 Selve GUI'en

- Denne skal have en standardiseret måde at vise alle variabler på. En træstruktur som en hierarkisk liste er fortrukket.
- Der skal være et vindue for forbindelsen der viser det rå data der kommer fra robotten.
 - Det skal være muligt at sende kommandoer til robotten fra dette vindue.
- Det skal hele tiden være muligt at se de status variabler brugeren ønsker at overvåge i GUI på et givent tidspunkt.

Overvejelser:

I det projektpartner gerne vil at systemet skal være modulært og konfigurerbart, må vi fokusere på skalerbarheden i systemet. Vi må også fokusere på at lave systemet modificerbart og udvidbart, så der kan laves et nyt specialiseret modul.

1.5 Problemformulering

Udarbejdet af: Bjørn, Daniel, Kristina

1.5.1 Problemstilling

For at tilgå deres robotter bruger DTU Elektro, på nuværende tidspunkt terminal vinduer. Dette gør at hvert robot-modul skal åbnes i et nyt vindue hvilket gør det meget uoverskueligt at arbejde med. Derfor er det en kompliceret proces at starte en robot op og få kontakt til alle dens moduler.

Derudover kræver det kendskab til robotens interface og modulernes forskellige sprog og kommandoer for at kunne bruge terminal vinduer til at kommunikere med en robot.

1.5.2 Hypoteser

- Det er muligt at lave en grafisk brugergrænseflade som en Java applet der kan kommunikere med en DTU Elektro robot via socket forbindelser.
- Det er muligt at lave systemet bag brugergrænsefladen så modulært at det kan understøtte DTU Elektros nuværende samt nye robotters unikke funktionalitet.
- Der findes en systemudviklingsmetode der passer godt til at udvikle dette system.

1.5.3 Uddybende spørgsmål

- Kan der laves en brugergrænseflade, som kan konfigureres til at bruges på alle DTU Elektros robotter?
- Kan en brugergrænseflade udvikles så den kan afvikles i alle internet browsere som har Java installeret?

1.5.4 Hvordan kan hypoteserne og spørgsmålene besvares?

For at be- eller afkræfte vores hypoteser skal vi have viden om Java, socket forbindelser og XML. Vi skal også have viden omkring interfacet til de forskellige robotter på DTU Elektro og hvordan man udvikler en plugin arkitektur.

Vi mener vi har viden og erfaring nok omkring Java, socket forbindelser og XML. I forhold til interfacet til robotten skal vi tilegne os viden fra vores projektpartner. Viden omkring plugins skal vi tilegne os fra vores projektparter, vores vejleder eller fra anden IT relateret kilde.

Vi har viden om flere udviklingsmetoder fra vores uddannelse som gør os i stand til at fortage et metodevalg for dette projekt.

1.6 Projektafgrænsning

Udarbejdet af: Bjørn

Da dette projekt skal laves over 10 uger ved vi ikke om vi får tid nok til at teste systemet på mere end en enkelt robot. Men da vi laver systemet modulært og konfigurerbart, kan det bruges på andre robotter, som understøtter de samme interfaces og videreudvikles af DTU Elektro.

En Java applet virker i alle browsere med Java installeret. Vi vil dog kun teste dette i Internet Explorer og Firefox, men sikre os at det fungerer i både Windows og Linux.

Det vil være muligt at lave ny unik funktionalitet til et modul, i det vi laver systemet modulært. Vi forventer at det vil tage en del tid at skulle skrive en fyldestgørende dokumentation og vejledning til udviklere med et begrænset kendskab til Java der beskriver hvordan de kan lave deres egen unikke funktionalitet til et modul. Derfor vil vi kun lave den normale Java dokumentation til koden, men ikke vejledning i form af en udførlig manual.

1.7 Brugerindflydelse

Udarbejdet af: Kristina

DTU Elektro har stor indflydelse på udviklingsforløbet af systemet. De vil løbende give feedback på vores system og komme med eventuelle nye krav, da de ikke fra starten har en fuld kravspecifikation. Projekt Kontrakten ligger i Bilag 12 "Projekt Kontrakt".

1.8 Andre mål

Udarbejdet af: Daniel

Andre mål for dette projekt er at lave en fyldestgørende hovedopgave der demonstrerer vores kompetencer opnået gennem uddannelsen. Disse kompetencer er bl.a. udvælgelsen og brugen af udviklingsmetoder, programmering, analyse og design af et system, projekt planlægning og test af det udviklede system.

Vi vil fokusere på at lave et system der kan videreudvikles og bruges af DTU Elektro efter endt projekt og samtidig få erfaring med hvordan systemudvikling foregår i samarbejde med et firma.

1.9 Valg af udviklingsmetode

Udarbejdet af: Bjørn, Daniel, Kristina

1.9.1 Indledning

“En software udviklingsmetode er en skabelon der bruges til at strukturere, planlægge og kontrollere udviklingen af et IT system.”²

Der findes flere forskellige slags udviklingsmetoder:³

Sekventiel metode: Der udvikles trinvis.

Iterativ metode: Der udvikles trinvis i iterationer.

Agile metoder: Der udvikles iterativt med større fleksibilitet end i de iterative metoder.

De sekventielle metoder og de iterative metoder er gode til komplekse systemer hvor der skal laves meget dokumentation. De agile metoder er bedre til systemer med lille kompleksitet og der er mere fokus på at få udviklet noget hurtigt samt lave mindre dokumentation.

1.9.2 Krav til udviklingsmetode

For at få et overblik over hvilken udviklingsmetode vi skal udvælge til vores projekt, har vi lavet en liste over vores og projektpartners krav til den udviklingsmetode vi skal kunne:

Vores krav til udviklingsmetoden:

- Det skal være muligt at udvikle prototyper regelmæssigt som vi kan få feedback på af projektpartner.
- Det skal være en udviklingsmetode med iterationer, så der kan laves analyse og design i hver iteration for at sikre modularitet og for at vi kan understøtte projektpartners nye krav og ønsker.
- Der skal kunne laves overordnet design og dokumentation af kernestrukturen.

Projektpartners krav til udviklingsmetoden:

- Der skal være tidlige og regelmæssige prototyper af systemet som der kan gives feedback på.
- Det skal være muligt at tilføje nye krav og ønsker senere i udviklingsforløbet.

² **Software Development Methodology** - Se litteraturliste

³ **Software Development Methodology** - Se litteraturliste

1.9.3 Overvejelser om valg af udviklingsmetode:

Vi vil her gennemgå forskellige udviklingsmetoder og deres relevans for vores projekt. På baggrund af vores og projektpartners krav til udviklingsmetoden ser vi bort fra sekventielle metoder, da de ikke gør brug af iterationer og ikke gør det muligt at tilføje nye krav senere i udviklingsforløbet.

Vi har valgt at gennemgå relevans for de to udviklingsmetoder vi har erfaring med fra vores studier, Unified Process (**UP**) og eXtreme Programming (**XP**). Derudover vil vi gennemgå Spiral modellen, da dette var projektpartners forslag til en mulig udviklingsmetode. Til sidst har vi gennemgået Scrum for at undersøge om vi kan bruge nogle af artefakterne fra denne styringsproces til projekt management i vores projekt.

1.9.3.1 Unified Process

- + ⁴Overordnet design udvikles fra starten.
- + Arbejdsprocessen i hver iteration er veldefineret.
- + Udviklingen er brugerorienteret.
- Adskillige artefakter i hver iteration gør at iterationerne bliver for lange til vores projekt.
- Mange roller skal udfyldes hvilket er svært for et lille projektteam.

1.9.3.2 eXtreme Programming

- + ⁵Releases i udviklingen gør at der ofte laves prototyper, der kan gives feedback på.
- Unit tests vil være svære at udvikle til alt implementeringen i vores system da meget af det skal kommunikere med en fungerende robot på netværket. Dette har vi kun adgang til på DTU hvor vi ikke er så tit.
- CRC Card er ikke nok til at give et overblik over det overordnede design.

1.9.3.3 Spiral Modellen

- + ⁶Der er analyse og design i hver iteration.
- + Regelmæssige prototyper fra starten gør at der ofte kan gives feedback fra projektpartner.
- + Ingen fokus på projektteam.
- + God til systemer hvor dele af arkitekturen er ukendt, hvilket er godt da robotterne vi skal kommunikere med kører på systemer og operativsystemer vi ikke kender så meget til.
- Mest brugt til at udvikle store systemer.

1.9.3.4 Scrum

- + ⁷Product-backlog kan bruges som en prioritetsliste.
- + Burn-down Chart kan bruges til tidsestimering under forløbet.

1.9.4 Diskussion af overvejelserne

Det vi har brug for er en metode hvor vi kan have mange iterationer så vi hurtigt og tit kan levere prototyper til projektpartner. I forhold til hvor kort tid vi har, har vi derfor brug for at iterationerne er korte.

⁴ UPEDU – Se litteraturliste

⁵ eXtreme Programming – Se litteraturliste

⁶ Unprecedented systems, Spiral Modellen – Se litteraturliste

⁷ SCRUM – Se litteraturliste

Unified Process

UP har fokus på risici, analyse og design i starten, hvilket understøtter nogle af vores krav til en udviklingsmetode. UP er bruger orienteret hvilket gør at projektpartner (bruger) har indflydelser på hvordan systemet hænger sammen. Der er ikke så meget fokus på at kunne levere brugbare prototyper tidligt og hvis vi skal bruge UP skal vi skære ned på artefakter for at få kortere iterationer. Disse nedskæringer og den korte tid vi har til projektet mener vi ville svække UP i en sådan grad, at andre udviklingsmetoder ville være et bedre alternativ

Spiral modellen

Spiral modellen opfylder mange af de krav som vi og projektpartner har opstillet til udviklingsmetoden.

Men med Spiral modellen skal vi bruge meget tid på analyse og design i starten, hvilket vil gøre at det tager tid før vi har den første prototype at vise projektpartner. Selvom vi har vægtet analyse og design højt i vores krav til udviklingsmetoden, vægter vi dog projektparters krav om en tidlig prototype højere. Derudover har vi ikke erfaring med Spiral modellen så vi skal bruge lang tid på at sætte os ind i hvordan og hvilke artefakter der skal bruges. Taget i betragtning af at vi ikke har så meget tid til projektet, vil det derfor ikke være hensigtsmæssigt at bruge så meget tid på det, hvis en metode vi kender i forvejen kan bruges.

eXtreme Programming

XP opfylder de vigtigste krav som vi og projektpartner har stillet til udviklingsmetoden, særligt kravet om tidlig og regelmæssige prototyper. Hvis vi skal bruge XP kan vi ikke lave Unit Test til hele systemet og CRC Cards vil ikke være nok til at give et overblik over arkitekturen, hvilket er et af vores krav til valget af udviklingsmetoden. Vi kan dog kompensere for dette ved at bruge andre måder at teste på og inddrage UML værktøjer til at beskrive den overordnede arkitektur. Derudover må vi ligge vægt på refaktorering for at styrke designet og modulariteten af systemet. Ved brug af XP skal vi også være opmærksomme på at XPs aktiviteter understøtter hinanden⁸, og derfor kan det være et problem at vi ikke fuldt ud kan bruge Unit Test. Hvis XP understøttes med et overordnet design medfører det mindre refaktorering, hvilket mindsker nødvendigheden for Unit Tests af alt koden.

Scrum

Scrum som et styringsværktøj mener vi ikke ville kunne tilføre noget til en udviklingsmetode så som UP, da UP allerede indeholder mange fyldestgørende styringsværktøjer. Dette mener vi også gælder generelt for de andre store iterative udviklingsmetoder, så som Spiral Modellen. Scrums styrke ligger i at kunne tilføre agile udviklingsmetoder et mere robust styringsværktøj, da dette er et område de agile metoder ikke ligger så meget vægt på. Hvis vi skulle bruge XP som udviklingsmetode, kunne man evt. bruge Scrum, eller nogle af dets artefakter, så som Product Backlog og Burndown chart, til at styrke management af projektet.

1.9.5 Konklusion

På baggrund af den forløbende diskussion omkring metoder, mener vi ikke at der er én metode der er perfekt til vores projekt. Dog er XP den metode som kræver mindst tilpasning for at opfylde vores og projektparters krav til metoden og derfor har vi valgt at bruge den til dette projekt. Kravet om tidlige

⁸ **Kritik af XP** – Se litteraturliste

og hurtige prototyper har vi vægtet højere end up-front analyse og design, og til dette er XP det bedste valg. Derudover giver XP mulighed for mange korte iterationer, så vi kan præsentere en prototype og få feedback fra projektpartner med korte mellemrum.

Spiral Modellen og UP kunne på mange punkter opfylde vores krav, men vores vægtning af kravene var udslagsgivende for at vi ikke valgte disse metoder, da de ville kræve for meget tilpasning. Spiral Modellen har vi ikke nogen erfaring med og den ville kræve kostbar tid at sætte sig ind i. Derudover ville der gå for lang tid før vi fik påbegyndt samarbejdet med projektpartner og kunne få feedback på systemet. Begge metoder har pga. deres mange artefakter normalt lange iterationer, og de ville miste mange af deres fordele hvis vi skulle forkorte dem til de korte iterationer som vi har brug for.

Vi vælger ikke at inddrage Scrum i vores projekt, da vi mener at XP grundlæggende har mange af de samme artefakter og at vores team er af en størrelse hvor Scrum ikke ville tilføre de samme fordele, som det ville gøre hvis teamet var større og kunne deles op i mindre grupper.

1.10 Projekt gruppen

Udarbejdet af: Daniel

I dette afsnit vil vi beskrive projektgruppen og måden vi forventer at arbejde på.

Følgende er en liste over projektgruppen, vejleder og projektpartner.

	Navn	Email
Projekt Udvikler	Daniel Freiling	banelos@gmail.com
Projekt Udvikler	Bjørn Truelsen	nesleurtb@gmail.com
Projekt Udvikler	Kristina Hansen	kh@impactchurch.dk
Projekt Vejleder	Michael Claudius	claudius@rhs.dk
Projekt Partner	Jens Christian	jca@elektro.dtu.dk
Kontakt Person	Andersen	

1.10.1 Basis proces

Da vi har valgt at bruge XP som udviklingsmetode vil vi følge dennes måde at arbejde på. Dette indebærer blandt andet Stand-up meetings hver morgen og parprogrammering.

Vi arbejder på skolen hver dag og har normal mødetid kl. 8.30. Hvis man er syg eller på anden måde forhindret i at komme giver man besked til en i gruppen.

Beslutningstagning i forhold til projektforsløbet tages internt i gruppen og hvis uenigheder opstår tages projektvejleder med på råd. Ved usikkerhed omkring systemets funktionalitet og brugergrænseflade tages beslutninger i samarbejde med projektpartner.

I forhold til projektarbejdet og teamets samarbejde har vi lavet en liste over styrker og svagheder og også set på hvilke risici der kan være til gruppen.

1.10.2 Styrker og svagheder

Styrker og svagheder i gruppen:

- + Tidligere og gode erfaringer som team
- + Erfaring med udviklingsmetoden

- + God faglig viden i forhold til implementering af systemet
- + Kendskab til brug af repository model til backup og deling af kode
- Tendens til en hvis mængde sygefravald i gruppen

1.10.3 Risici

Risici	Konsekvenser	Forebyggelse af problemet	Plan for påkommende tilfælde
Sygdom/Frafald	Færre der arbejder på projektet og dermed er der mindre tid til det der skal nås	Spis sundt ☺	Revurdering af projektplan i forhold til hvad der kan nås, og muligt mere arbejde til de resterende medlemmer
Tab af data	Tab af dele eller hele projektet	Brug af repository model til at opbevare systemets kode på et fælles tilgængeligt sted	Dele eller hele projektet genetableres fra backup eller hentes fra repository
Tab af motivation	En eller flere mister fokus på projektet	Jobrotation	Peptalk fra resterende gruppemedlemmer
Forsinkelser i forhold til deadlines	Manglende tid til at færdiggøre projektet	Detaljeret planlægning af projektets faser	Brug fridage til at indhente det der mangler

På baggrund af denne risici-liste vil vi gøre hvad vi kan for at forebygge problemerne, men disse risici har på nuværende tidspunkt ingen betydning for om vi vil fortsætte projektet.

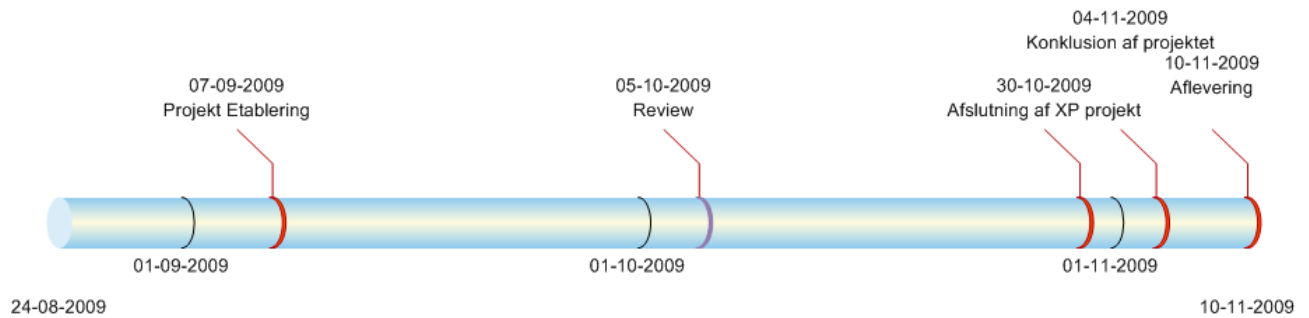
1.10.4 Konklusion på projektgruppen

Kompetencerne i vores gruppe dækker et bredt område i forhold til systemudvikling. Vores risici har ingen betydning for om vi forsætter projektet. Vi har erfaring med den valgte udviklingsmetode og har arbejdet sammen tidligere. På baggrund af dette vurderer vi at vi har gode forudsætninger for at arbejde godt sammen og få et godt projekt ud af det.

1.11 Projekt Plan

Udarbejdet af: Kristina

Følgende er vores projekt plan for hele hovedopgaven. I perioden efter projektetableringen udarbejdes vores XP projekt. XP har både en Releaseplan og iterationsplaner der beskriver forløbet mere detaljeret. For at kunne se på projektet med andre øjne har vi valgt at have et Review i midten af vores projekt. Vi har valgt at afslutte XP projektet d. 30/10/09 for at have god tid til at skrive konklusion og færdiggøre rapporten.



1.12 Software

Udarbejdet af: Kristina

Til dokumentation og rapportskrivning vil vi bruge Microsoft Word, hvilket er en del af Office pakken og tilgængeligt på skolen hvor vi foretager størstedelen af rapportskrivningen. Microsoft Visio vil blive brugt til projekt management, som blandt andet inkluderer projektmilepæle og iterationsplaner. UML og andet design af systemet udføres i Microsoft Visio, Visual Paradigm og Netbeans. Selve implementeringen af systemet forventer vi udelukkende kan udføres i den gratis IDE Netbeans.

1.13 Evaluering af projektetableringsfasen

Udarbejdet af: Bjørn, Daniel, Kristina

I denne opstartsfasen brugte vi meget tid på at skrive problemstilling og vælge udviklingsmetode. Men denne tid var godt givet ud da vi på baggrund af undersøgelser og evaluering af forskellige metoder kunne tage et godt valg af udviklingsmetode. Derudover var det godt at bruge meget tid på at diskutere og tilrettelægge problemdefinitionen da denne giver baggrund for resten af projektets forløb.

Vi har nået at udarbejde de planlagte aktiviteter. Vi har løbende haft møder med vores projekt vejleder omkring valg af metode og opstart af projektet og også besøgt DTU for at få klarhed over projektoplægget.

2. XP Prerelease

2.1 Indledning

Udarbejdet af: Bjørn, Daniel, Kristina

Da beskrivelse af teori for XP og argumentation for valg af ekstra artefakter samt forundersøgelse og analyse, ikke hører hjemme i et XP Release har vi fundet det nødvendigt at lave dette Prerelease.

I afsnittet om Teori, vil vi først se på teorien om de forskellige XP artefakter og derefter teorien for de ekstra Artefakter vi vælger at inddrage.

2.1.1 Projekt Plan for Prerelease

Vores projekt plan for dette Prerelease ligger i Bilag 1.1 "Projekt Planer, Prerelease".

2.2 Teori

Udarbejdet af: Bjørn, Daniel, Kristina

I dette afsnit vil vi beskrive teorien omkring XP. Først vil vi give et overblik over hvordan et XP udviklingsforløb foregår og nogle overvejelser omkring arbejdsformen, samt vores argumentation for hvordan vi vælger at arbejde. Derefter vil vi beskrive de enkelte artefakter og argumentere for hvordan vi har valgt at bruge dem og hvorfor.

2.2.1 Overblik

I XP⁹ starter man med at finde systemets User-stories, som beskriver kravene. Man laver en estimering af disse User-stories og laver derefter en Releaseplan hvor User-stories er fordelt ligeligt i forhold til estimeringen. I XP kan der i hvert Release være flere iterationer. Hver iteration indeholder et givent antal User-stories, som hver bliver opdelt i en række Tasks. Der laves Acceptance Tests og Unit tests til User-stories før man går i gang med at implementere hver Task, og disse tests køres efter endt implementering. På den måde følger XP V-modellen¹⁰ for software udvikling meget godt, med et V for hver User-story.

I XP er der en række artefakter der bruges til at forstå og følge projektets forløb. Der laves en metafor, som på simpel vis forklarer systemets overordnede mål og funktionalitet, hvilket gør det nemt for nye projektmedlemmer at sætte sig ind i systemet. XP har også et Velocity Chart, der viser udviklingshastigheden over tid og et Burn-Down Chart, der viser hvor meget af systemet der mangler at laves.

2.2.1.1 Overvejelser

XP ligger stærkt op til at man skal have et tæt samarbejde med kunden i både definering, planlægning og test af systemet. Vi vil forsøge så godt vi kan, at få projektpartner med i udviklingen af de relevante artefakter, men da vi ikke har en repræsentant fra projektpartner til stede hele tiden og højest kan besøge DTU en gang om ugen, kan dette ikke lade sig gøre for samtlige artefakter.

2.2.2 XP Artefakter

Her vil vi så beskrive teorien for de forskellige dele af XP og argumentere for vores brug af disse.

⁹ **eXtreme Programming** – Se litteraturliste

¹⁰ **V-Model** – Se litteraturliste

2.2.2.1 Metafor

I XP laves en metafor¹¹ for systemet til at beskrive systemets funktionalitet på en simpel måde som hele projektteamet kan forstå. Metaforen hjælper projektteamet til at se systemet fra et andet, nemmere forståeligt, perspektiv. Denne laves oftest i starten af projektet, for tidligt at give overblik over systemets helhed.

2.2.2.2 User-stories

En user-story¹² er en kort beskrivelse af hvordan systemet agerer fra brugerens synsvinkel. De er som regel kun i form af et par sætninger som brugeren kan forstå, uden tekniske udtryk. User-stories bruges til at finde krav til systemet og kort beskrive dem fra brugerens perspektiv. De vil senere blive fyldt ud med mere information når man har præsenteret dem og snakket om dem med brugeren. User-Stories er et kerneartefakt i udviklingsmetoden XP og er essentiel for at det videre forløb af projektet kan estimeres og planlægges.

Vores benyttelse

Normalt laves User-Story som kort på små sedler man skriver på, men vi har valgt at lave dem elektronisk så de kan blive præsenteret i denne rapport.

2.2.2.3 Spike Investigations

Ofte finder man i XP User-Stories som viser sig at være meget svære at estimere på i forhold til implementeringstid, pga. kompleksitet eller manglende viden. Spike Investigations¹³ laves for at få indblik i hvad der kræves for bedre at kunne implementere og estimere en User-Story. Det indebærer en hurtig prototype af kernen af User-Storien der udvikles i løbet af få dage. Denne prototype kan smides væk igen, da den laves primært for at give udviklerne mere viden om problemområdet og dermed bedre forudsætninger for at estimere hele den givne User-Story.

Vores benyttelse

Vi forventer at få brug for mindst en Spike Investigation i vores XP projekt ud fra hvad vi allerede ved om kravene. Præcis hvilken User-story vi laver Spike Investigation til beslutter vi i Release 1, når vi har fundet de første User-Stories.

2.2.2.4 Releaseplan

En Releaseplan¹⁴ udarbejdes som en plan for hvornår user-stories skal implementeres. De med højest forretningsværdi skal implementeres først. Det at have Releases (et vist antal færdigt implementerede user-stories) regelmæssigt giver mulighed for at få feedback fra brugeren ofte og lave tilrettelser. En Releaseplan kan enten udarbejdes ud fra at der skal være Releases på særlige datoer eller ud fra hvor lang tid man regner med de user-stories til det givne Release vil tage.

¹¹ **XP Metafor** – Se litteraturliste

¹² **XP User-stories** – Se litteraturliste

¹³ **XP Spikes** – Se litteraturliste

¹⁴ **XP Releaseplan** – Se litteraturliste

2.2.2.5 Iteration Plan

En iteration plan¹⁵ er en beskrivelse af hvilke user-stories der skal laves i hver iteration af et givet Release. For at kunne lave iterationsplanen skal der estimeres på hver enkelt user-story hvor lang tid det tager at fuldføre den. Hvis en user-story er meget kompliceret kan den deles op i flere tasks.

2.2.2.6 Task Cards

Task cards¹⁶ bliver lavet for at dokumentere hvilke opgaver der er nødvendige for at kunne færdiggøre en user-story. De bliver også brugt ved tidsestimering af de enkelte user-stories, så der kan laves en iterationsplan. Efter endt opgave skrives på kortet hvordan opgaven blev løst og eventuelle mangler og nye opgaver.

2.2.2.7 Class, Responsibilities and Collaboration

Class, Responsibilities and Collaboration (CRC)¹⁷ kort bruges til at give et overblik over klasserne og deres sammenhæng i systemet. Hvert CRC Card repræsenterer et objekt, og der skrives på hvilket ansvar objektet har og hvilke klasser de samarbejder med. Der kan også skrives på hvis objektet har en super- eller subklasse.

Vores benyttelse

CRC Cards laves normalt på papir, men vi har valgt at lægge dem ind elektronisk så de kan være med i rapporten.

2.2.2.8 Programming

Parprogrammering¹⁸ er den teknik der benyttes i XP når kode skal skrives. To personer arbejder på en computer og imens den ene skriver kode gennemlæser den anden samtidig koden og kommer med feedback. Der skiftes regelmæssigt arbejdsroller. Parprogrammering har til formål at give større kvalitet i den udviklede kode uden at afleveringsfrist overskrides.

Vores benyttelse

Så vidt muligt vil vi prøve at følge regelen om parprogrammering. Til backup af den udviklede kode har vi valgt at gøre brug af en repository model.¹⁹

2.2.2.9 Refaktoring

Det er vigtigt i XP at gøre brug af refaktoring²⁰ fordi det er med til at fastholde et enkelt design og dermed en simpel arkitektur. Refaktoring betyder at den udviklede kode omskrives så vidt det er muligt til en mere enkel og læsevenlig kode uden at funktionaliteten ændres.

Vores benyttelse

Refaktoring er vigtigt i vores projekt for at gøre den udviklede kode modulær. Enkelt design og simpel arkitektur kan sikre dette.

¹⁵ **XP Iterations Plan** – Se litteraturliste

¹⁶ **XP Task Cards** – Se litteraturliste

¹⁷ **XP CRC Cards** – Se litteraturliste

¹⁸ **XP Parprogrammering** – Se litteraturliste

¹⁹ **Repository Model** – Se litteraturliste

²⁰ **XP Refaktoring** – Se litteraturliste

2.2.2.10 Velocity Chart

Et velocity chart²¹ viser hvor mange estimerede point der er blevet udviklet i hvert release igennem hele projektet. De estimerede point fra afsluttede user-stories i et givent release tælles sammen og vises som én søjle. Velocity Chart bruges til at evaluere projektets udviklingshastighed over tid.

2.2.2.11 Burn Down Chart

Et burn-down chart²⁰ bruges til at illustrere systemets overordnede fremgang. Det viser det totale antal estimerede point for samtlige user-stories og der tælles ned for hver færdige iteration. Således får man en graf der viser både det arbejde der er tilbagelagt og hvor langt der er igen før at man når ned på 0 estimerede point, hvor systemet gerne skulle være færdigt. Denne graf kan bruges til at se om fremgangen for hele systemet er blevet langsommere eller hurtigere over tid og at estimere hvornår systemet bliver færdigt.

2.2.2.12 Testing

Testning er en fundamental aktivitet i XP. Det sørger for at højne kvaliteten af et system og fungerer som feedback for både kunde og udvikler til om systemet opfylder dets krav. I XP udvikler man efter en test-dreven metode, hvor man først skriver tests og derefter udvikler koden indtil den gennemfører testen.

Der er 3 forskellige slags test i XP:

Unit test

Unit tests er programmerede tests der sørger for at teste koden på et klasse- og metode niveau. Unit Tests skal sørge for at hver metode gør hvad der er tiltænkt. Både i forhold til input/output og om et parameter f.eks. overholder en given grænseværdi. Dermed er Unit tests en black-box test, der ikke tester hvert eneste loop og tilstand, men blot sørger for at klassens metoder agerer som det er tiltænkt.

Integration Test

Dette er tests til at bekræfte at klasserne fungerer som de skal efter at være blevet samlet til et system. Det kan være i form af at samle den nyeste kode fra alle udviklere, kompilere det og køre samtlige tests. Dette sørger for at det bliver opdaget hvis senere implementerede klasser gør at andre dele af systemet ikke længere fungerer korrekt. Integration tests er ofte Unit tests der dog ikke er fokuserede på kun én klasse, men på om flere klasser arbejder sammen korrekt.

Acceptance Test

Acceptance tests er en specialiseret test der laves ud fra hver user-story. Der kan være flere acceptance test til hver user-story. De skal sikre at funktionaliteten bliver implementeret korrekt. En acceptance test beskriver input fra brugeren og det forventede output fra systemet. En user-story er først helt færdig når alle acceptance tests er bestået.

I XP følger man i store træk V-Modellen for hver User-Story der udvikles. Man beskriver først kravene i en User-Story, derefter laver man en acceptance test, der har til formål senere at teste om User-Story kan fuldføres. Efter udviklingen af Task Cards for hver User-Story, udvikler man Unit tests til hver

²¹ **XP Velocity** – Se litteraturliste

klasse og udvikler klasserne indtil de gennemfører deres Unit test. For hver klasse der fuldfører sin Unit test kører man også en integration test der sørger for at koden virker sammen med resten af systemet. Til slut afvikles selve Acceptance Testen på den oprindelige User-Story for at verificere at implementeringen opfylder de oprindelige krav.

Vores benyttelse

I vores projekt har vi fra starten indset at det ville blive meget svært at skulle lave Unit tests til hele systemet. Dette skyldes at store dele af systemet skal kommunikere med moduler på en robot på DTU, som vi kun sjældent har adgang til. Da XP er stærkt præget og får mange af sine styrker som udviklingsmetode fra sin test-proces har vi fra starten overvejet den mulige mangel på Unit tests. Denne mangel er vi dog blevet enige om at vi kan kompensere ved at bruge en simuleret robot og en mock-server at teste imod. En mock-server er et lille serverprogram som vi selv skriver, der simulerer enkelt funktioner fra DTU Elektros robotter.

Vi vil ikke bruge Integration Tests, da vi løbende vil køre systemet for at se om det fungerer samlet. Dette har vi valgt fordi vi som regel kun programmerer på én computer af gangen.

Acceptance tests ser vi som de vigtigste tests i dette projekt, da det er de tests der skal bekræfte at vores system som helhed opfylder de krav som projektpartneren har stillet. Vi vil planlægge at køre acceptance tests de dage hvor vi er ude på DTU da projektpartner og de robotter vi om muligt skal teste systemet med, kun er tilgængelig på disse dage.

2.2.3 Ekstra Artefakter

Udover XP's egne design-artefakter har vi valgt at inddrage et par artefakter, som ofte bruges i udviklingsmetoden UP. Vi har valgt at inddrage en Domæne Model, et Design Klasse diagram og nogle Sekvens Diagrammer. Derudover har vi valgt at inddrage en brugergrænsefladetest og en overordnet Projekt Plan for hver iteration.

Vi vil herunder beskrive teorien omkring de ekstra artefakter.

2.2.3.1 Domæne Model

En domæne-model viser konceptuelle klasser og deres relationer. Modellen viser et billede af hvordan man forestiller sig at begreber ser ud i den virkelige verden og viser derfor ikke et billede af software objekter.

Vores benyttelse

Vi har valgt at inddrage en domæne-model fordi vi gerne fra starten vil have et godt overblik over systemets overordnede design, hvilket vi ikke mener vi kan få i tilstrækkelig grad fra artefakter i XP.

2.2.3.2 Design Klasse Diagram

Et design klasse diagram viser software klasser og deres relationer. Modellen viser software objekter og de associationer der er imellem dem.

Vores benyttelse

Vi har valgt at inddrage et design klasse diagram fordi vi vil styrke det tekniske overblik over systemet og for at kunne illustrere sammenhængen mellem vores CRC Cards.

2.2.3.3 Sekvens Diagram

Et sekvens-diagram er et interaktions diagram der viser hvordan en sekvens af aktiviteter kommunikerer med hinanden og i hvilken rækkefølge det sker.

Vores benyttelse

Til de user-stories hvor det kan give et bedre overblik, vil vi lave et Sekvens Diagram for at kunne designe interaktionen imellem systemets klasser på kodeniveau.

2.2.3.4 Brugergænseflade design

Der findes mange måder at teste om en udviklet brugergænseflade er brugervenlig. I dette afsnit vil vi først beskrive teorien²² omkring brugervenlighed af et system og derefter beskrive hvordan vi vil bruge nogle af disse til at teste brugervenligheden af vores brugergænseflade.

Med brugervenligheds test er der forskellige problemer man ser på:

Program-fejl – Systemet virker ikke som det skal.

Manglende funktionalitet – Det er umuligt at udføre den ønskede handling.

Hvor nemt det er at bruge systemet – kan brugeren finde ud af at bruge systemet?

Man kan klassificere problemerne på følgende måde:

Missing functionality – Systemet kan ikke udføre den ønskede handling

Task failure - Brugeren kan ikke selv udføre handlingen, eller han tror den er udført uden at være det

Annoying – Brugeren synes systemet er irriterende eller besværligt at bruge.

Medium problem – Brugeren finder løsningen efter adskillelige forsøg.

Minor problem – Brugeren at finder løsningen efter nogle få forsøg.

Brugervenligheds tests kan evalueres ud fra disse brugervenlighedsfaktorer:

Fit for use – Systemet kan understøtte de handlinger brugeren har brug for

Ease of learning – Hvor nemt er det at lære at bruge

Task efficiency – hvor effektivt er det at bruge

Ease of remembering – Hvor nemt er det at huske hvordan systemet bruges for en der ikke bruger det så tit

Subjective satisfaction – Hvor tilfreds er brugeren med systemet

Understandability - Hvor nemt er det at forstå hvad systemet gør

Der kan testes på forskellige måder:

Think-aloud Test

Denne test går ud på at få en bruger til at udføre en handling i systemet. Brugeren får at vide hvilken handling der skal udføres og "tænker højt" mens denne handlingen udføres. Der bliver skrevet ned

²² **User Interface Design** – Se litteraturliste

hvad brugeren siger, og ud fra dette kan der evalueres om det gik som systemudviklerne havde forventet. Hvis det var svært for brugeren at udføre handlingen eller den ikke kunne udføres, skal der overvejes at laves ændringer i forhold til designet.

Disse test skal testes på flere personer, så man kan evaluere ud fra gennemsnittet.

Heuristic Evaluation

I denne test får man en brugervenligheds-specialist ind for at teste systemet. De bedømmer systemets brugervenlighed ud fra deres erfaringer. Specialisten afleverer en liste over de fejl der burde rettes ved systemet.

Disse tests kan udføres af flere specialister.

Usability Measurements and requirements

Dette er forskellige små tests:

Task time – måle den tid det tager at udføre en handling.

Problem counts – tælle problemer der opstår undervejs (i en think-aloud test)

Keystroke counts – hvor mange gange skal der "klikkes" før en handling er udført.

Opinion poll – beder forskellige personer om at udfylde et spørgeskema.

Score for understanding – spørger brugeren hvordan de tror systemet virker.

Vores benyttelse

Da vores brug af disse tests går ud på at teste brugervenligheden af brugergrænsefladen, vil vi ligge mest vægt på om systemet er nemt at bruge. Programfejl og manglende funktionalitet bliver testet ved Acceptance test.

Vi vil udvikle nogle think-aloud test, og klassificere de enkelte problemer der opstår undervejs. Efter en bruger har udført vores think-aloud test, vil vi give dem et spørgeskema hvor de kan evaluere vores system ud fra brugervenlighedsfaktorerne. Vi vil også tage tid på hvor lang tid det tager at udføre de enkelte handlinger.

Vi vil afprøve disse tests med projektpartneren, med den gruppe vi laver Review og med andre fra vores klasse.

2.2.3.5 Projekt Plan for hvert Release

I XP udarbejdes der både en iterations- og releaseplan. Men da disse kun har fokus på at understøtte implementeringen af systemet har vi valgt også at lave en overordnet projektplan i iteration 1 for projektet som helhed. Dette vil vi gøre fremadrettet for hver release. Vi mener at det kan bruges som styringsredskab for hele projektet og derved give os et samlet overblik over alle arbejdsopgaver.

2.3 Relevante forundersøgelser og analyse

Udarbejdet af: Bjørn, Daniel, Kristina

Før vi starter på Release 1 i vores XP projekt vil vi lave nogle forundersøgelse, der kan give os et bedre overblik over systemet. Vi vil overveje om der kan laves en Business Analyse. Vi vil se på de overordnede Kvalitets Faktorer for systemet, hvilke Risici der kan være for systemet og lave en Domæne Model for at kunne få overblik over design arkitekturen.

2.3.1 Kvalitets Faktorer

På baggrund af projektpartners krav har vi overvejet hvilke kvalitets faktorer vi skal vægte højt, for at sikre at systemet opnår en god kvalitet. Dette har vi gjort på en skala fra 1-5, med 5 som det af højest betydning. Under tabellen har vi beskrevet vores overvejelser omkring de forskellige kvalitets faktorer.

Kvalitets Faktorer	5	4	3	2	1
Usability	X				
Correctness		X			
Flexibility		X			
Interoperability		X			
Portability		X			
Reusability			X		
Maintainability			X		
Efficiency				X	
Reliability				X	
Testability				X	
Integrity					X

Usability

I øjeblikket bruges Telnet terminaler til at tilgå robotterne, hvilket ikke er så brugervenligt. Det skal være muligt at bruge systemet for forskerne på DTU men også for de studerende som ikke har så meget kendskab til robotterne. Dette kræver høj usability. Derfor har vi også tænkt os at lave brugergrænsefladetest.

Correctness

Vi vil stræbe efter at udvikle systemet korrekt i forhold til projektpartners krav. Men da vi kun har 10 uger til hele projektet kan vi dog ikke garantere at vi kan opfylde alle krav.

Flexibility

Vi har vægtet flexibility højt fordi det i systemet skal være muligt at ændre og tilføje nye moduler. Systemet skal også understøtte plugins og tolke data i XML-format.

Interoperability

Da systemet skal kunne kommunikere med mange forskellige robotter med varierende interfaces, kræves der en høj grad af indbyrdes kompatibilitet.

Portability

Portability skal vægtes højt da det er et krav fra projektpartner at systemet ikke skal installeres og køres fra et specifikt operativsystem. Java og Java applet sikrer høj portability, fordi Java er platform-uafhængigt.

Reusability

Projektpartner ser gerne at store dele af systemet kan genbruges, så de nemmere kan lave specialiserede moduler og kan bruge det vi har lavet i deres videreudvikling af systemet. Derfor har vi valgt at vægte det til 3.

Maintainability

Da systemet ikke er så stort og komplekst vil det ikke kræve meget at vedligeholde. Derfor vil vi ikke ligge så meget vægt på at dette skal være nemt.

Efficiency

Da hurtig responstid af systemet ikke er et krav fra projektpartner, vil vi umiddelbart ikke fokusere på at optimere koden i systemet så dette opnås.

Reliability

Systemet skal kunne det der er tiltænkt hver gang den bliver bedt om noget uden der kommer noget overraskende nyt. Da vi ikke kan ligge så meget vægt på testability, er det forventeligt at der vil komme nogle enkelte fejl, og derfor også mindre reliability.

Testability

Da vi ikke kan teste alt koden 100%, på grund af vores begrænsede adgang til DTU Elektros robotter, kan vi ikke vægte dette så højt. Vi vil dog teste så meget af koden vi kan, og teste systemet på en robot når vi besøger DTU.

Integrity

Der er intet krav fra projektpartner til bruger-validering eller rettighedsopdeling af systemet og det vil kun blive brugt af forskere og studerende på DTU Elektro. Derfor antager vi at alle der bruger systemet har rettighederne til det og at fuld kontrol over den forbundne robot er i projektpartnerens bedste interesse og derfor vil vi ikke ligge så meget fokus på integritet.

Overvejelser om kvalitetsfaktorerne

Det er brugervenlighed vi har vægtet højest af kvalitetsfaktorerne, hvilket ikke er overraskende da projektets hovedformål er at designe en ny brugergrænseflade. Fleksibilitet og genbrugelighed skal vi fokusere særligt på. Disse 2 kvalitetsfaktorer ønsker vi at opnå ved hjælp af et godt design, fokus på modularitet og brug af Model, View, Control (**MVC**) designet. Vi vil også have fokus på de andre kvalitetsfaktorer der er vægtet 4, da de har væsentlig betydning for at systemudviklingen overholder de opstillede krav fra projektpartner. Resten vil vi ikke ligge så meget vægt på i udviklingen af dette system.

2.3.2 Risici til XP projektet

Risici	Konsekvenser	Forebyggelse af problemet	Plan for påkommende tilfælde
Manglende Korrekthed – pga få Unit tests.	Der kommer fejl i systemet og det vil tage længere tid at implementere det.	Lave detaljerede Acceptance Tests.	Lave nogle dele af systemet forfra.
Ingen forbindelse til robot	Basis for hele systemet kan ikke opbygges	Spike Investigations	Få mere info om omkring robotten fra projektpartner
Dårlig kommunikation med projektpartner	Usikkerhed omkring krav. Projektpartner får ikke det ønskede resultat.	Besøg dem så ofte som muligt og hold kontakt over email.	Involvere projektvejleder til mægling.
Manglende viden i forhold til krav.	Længere undersøgelsestid og længere implementeringstid. Mulighed for uopfyldte krav.	Kan ikke forbygges	Lave undersøgelser for at tilegne os den manglende viden. Opsøge specialister.

Overvejelser:

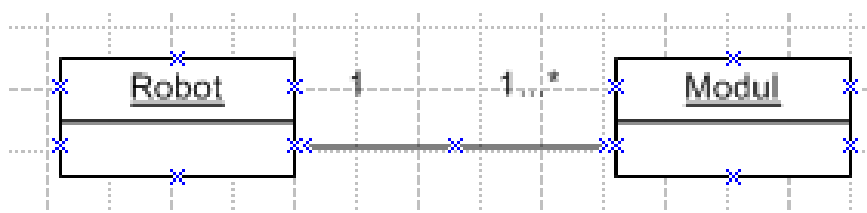
Vi skal sørge for at lave acceptance test detaljerede da det ikke vil være muligt at lave Unit test til alle user-stories. Dermed skulle vi gerne sikre at kvalitetsfaktoren correctness forbliver høj. Det er vigtigt at vi hurtigt får afklaret om der kan etableres forbindelse til en robot ved fx at benytte en spike investigation. Hvis vi ikke kan etablere forbindelse kan vi heller ikke køre vores acceptance tests.

Efter at have afholdt det første møde med projektpartner vurderer vi at kommunikationen er god. Vi skal dog være opmærksomme på at bevare den gode kommunikation da vi er afhængige af projektpartners feedback, der udgør et vigtigt parameter i vores valgte udviklingsmetode.

På nuværende tidspunkt ser vi ingen problemer omkring manglende viden, udover udviklingen af en plugin struktur, i forhold til de opstillede krav til systemet.

2.3.3 Domæne Model

For at få et overblik over design arkitekturen har vi lavet en domæne-model. Men da systemet består af meget få konceptuelle klasser, blev domæne modellen meget lille og gav os ikke noget nyt indsigt i forhold til designet. Systemet består grundlæggende af 2 konceptuelle klasser, Robot og Modul. En robot kan have en eller flere moduler og et modul tilhører kun en robot.



Ved at lave denne domæne model forstod vi at vores system primært handler om at udveksle information med en eksisterende arkitektur. Vi har derfor mere brug for at fokusere på at lave et Design Klasse diagram, der beskriver *kodens* arkitektur. Dette vil vi lave sammen med vores CRC cards i Release 1.

2.4 Evaluering af Prerelease

Udarbejdet af: Bjørn, Daniel, Kristina

Det var den rigtige beslutning vi tog da vi valgte at lave dette Prerelease. For det første synes vi ikke det passede ind at beskrive teorien i et Release hvor det handler om at anvende teorien. For det andet så har det rustet os bedre til at anvende XP ved netop at gennemgå teorien igen før vi starter på Release 1.

De ekstra forundersøgelser og analyser har klarlagt på hvilke områder vi skal supplere XP med ekstra artefakter og hvordan vi kan anvende dem. Det viste sig værd da vi fandt ud af at en domæne-model alligevel ikke giver os den information vi har brug for omkring design arkitekturen.

Vi har diskuteret om vi skulle lave en business analyse af DTU Elektro, men da det ikke har så meget relevans for vores projekt og da DTU ikke er et firma med et marked og konkurrenter, har vi valgt at udelade det.

3. eXtreme Programming Release 1

3.1 Indledning

Udarbejdet af: Bjørn, Daniel, Kristina

På baggrund af det valg vi tog omkring udviklingsmetode i etableringsfasen og de undersøgelser vi lavede omkring XP teori og ekstra artefakter i Prerelease, vil vi gå i gang med at udvikle systemet efter XP udviklingsmetoden. Vi vil i dette afsnit udvikle de enkelte artefakter, forklare hvordan vi har anvendt dem og hvad vi har fået ud af at bruge dem.

I dette Release har vi valgt at skrive overvejelser efter hvert artefakt hvor det er relevant. Disse overvejelser vil indgå i den samlede evaluering af Release 1.

3.1.1 Projekt Plan

Før vi starter med at udvikle artefakterne til Release 1 har vi lavet en projektplan, for at kunne få overblik over alt der skal laves i dette Release. Vi har lavet denne plan på baggrund af vores estimering om at en Release skal være på 2 uger (se 3.5 Releaseplan). Programmeringen starter fra den første dag, da vi skal lave spike investigations.

Vores projekt plan for Release 1 ligger i Bilag 1.2 "Projekt Planer, Release 1".

3.2 Metafor

Udarbejdet af: Bjørn, Daniel, Kristina

Til vores projekt har vi valgt en "Flyveleder" som metafor for vores system, Modular Applet Robot GUI (MARG). En flyveleder samler information om alle fly i hans luftrum. Han kan sende og modtage beskeder fra hvert fly og også se om der er noget galt med flyet. Flyvelederen kan ikke direkte styre flyene, men han kan give simple kommandoer og information til et fly om hvor det skal lande og hvordan forholdene er.

MARG skal kunne samle information om flere robot moduler på én gang og skal kunne modtage og sende beskeder til dem. MARG skal også kunne give simple kommandoer til robottens moduler, så som start, stop osv. Den vil også være i stand til at overvåge om der er noget galt med de enkelte moduler på robotten.

Sammenligning af Metafor og MARG:

- Flyveleder overvåger flyene i hans luftrum -> MARG overvåger modulerne i en robot
- Flyveleder modtager data fra hvert fly -> MARG modtager data fra hvert modul
- Flyveleder giver simple ordrer til et fly -> MARG giver simple kommander til et modul
- Flyveleder overvåger flyets status -> MARG overvåger modulets status

Overvejelser

Metaforen har givet os en bedre forståelse for den overordnede arkitektur af systemet. Vi havde dog alle en god forståelse for projektet som helhed før vi valgte en metafor. Derfor kunne vi egentlig godt have fravalgt at medtage metafor i projektet. Men vi mener at den vil være et godt værktøj til hurtigt at introducere andre til vores projekt, f.eks. hvis en ny udvikler tilknyttes projektet eller når der skal laves et review af projektet. Vi har derfor valgt at beholde metaforen i projektet.

3.3 User-stories

Udarbejdet af: Daniel, Kristina

Vi har lavet vores user-stories på baggrund af vores første møde med projektpartner, projektoplægget og vores afgrænsning af dette. Da robotterne på DTU består af flere moduler, forbinder man til en robot ved at oprette forbindelse til en eller flere af dens moduler. Derfor har vi ikke en user-story der hedder at forbinde til en robot, men i stedet at forbinde til et modul.

Følgende er vores udvalg af user-stories. Ved Estimering gives point fra 1-10 hvor 10 er de user-stories vi mener der tager længst tid at implementere. Business value er inddelt i Low-High efter hvor vigtig de er for projektpartner. Overvejelser omkring hver user-story har vi skrevet på selve kortet.

Vi har valgt at vise 2 user-stories her. Udover disse har vi lavet "Tilføj variabel til status", og "Tilføj modul".

Alle user-stories ligger i Bilag 2.1 "User-stories, Release 1"

Forbind til modul
Story Number: 1 Estimation: 10 pts (baseret på spike 1) Business value: High
Brugeren forbinder til et modul på robotten igennem GUI'en. GUI'en informerer brugeren om forbindelsen blev oprettet og viser den umiddelbart efterfølgende kommunikation.
Overvejelser: Denne user-story er meget omfattende, og indebærer hovedfunktionaliteten for vores system. Ved iterationsplanlægning skal denne deles op i mange individuelle tasks. Det at forbinde til et modul indebærer også at modtage XML dataen løbende og gøre den tilgængelig for GUI'en.

Vis variabel-træ
Story Number: 2 Estimation: 2 pts Business value: High
Brugeren beder om at få præsenteret alle variabler fra et tilsluttet modul. Variablerne bliver vist som en træstruktur.
Overvejelser: Dette er en grafisk præsentation af den interne datastruktur på det givne modul. En meget vigtig funktionalitet med stor business value for kunden.

3.4 Spike Investigations

Udarbejdet af: Daniel, Bjørn

Vi ønskede at lave en spike investigation til user-story "Forbind til modul", fordi denne var meget omfattende og vanskelig at estimere. Vi var usikre på hvor svært det ville blive at få kontakt til et robot-modul og modtage data. Derfor var det at lave en hurtig prototype der etablerede en forbindelse til et robot-modul. Vi brugte kun et par dage på at udvikle denne prototype, men det er muligt at vi vil bruge resultatet i vores videre udvikling af systemet. Vi lavede også en spike investigation til user-story "Send kommando til modul". Denne kan forhåbentlig bygges oven på Spike 1.

3.4.1 Planlagte Spikes

Spike 1: Forbind til Modul

Opret en socket forbindelse til et modul i Java og modtag det umiddelbart efterfølgende data som tekst.

Spike 2: Send kommando til Modul

Send en kommando til et forbundet modul over en socket forbindelse i Java og modtag det umiddelbart efterfølgende data som tekst.

3.4.2 Udførelse af Spikes

I dette afsnit om udførelsen af spikes vil vi forklare i dybden hvordan vi har programmeret vores spikes.

Til "Spike1: Forbind til Modul" valgte vi at lave en enkelt klasse ModClient og holde den så simpel som muligt. Den første metode i klassen som vi lavede var **connect**. Denne metode skulle tage imod en *host* og en *port* og derefter oprette socket forbindelsen og åbne de nødvendige streams.

Dette blev implementeret således:

```
public void connect(String host, int port) {
    socket = new Socket(host, port);
    is = socket.getInputStream();
    out = new PrintWriter(socket.getOutputStream());
    if (!threadRunning) {
        startUpdateThread();
    }
    //try-catch delen er undladt for nemmere læsning.
}
```

De robot-moduler som vi skal kommunikere med bruger normale socket forbindelser, derfor oprettes der i Java blot et Socket objekt, med en given *host* og *port* til et modul. Efter denne er blevet oprettet laves der en PrintWriter *out*, for nemmere at kunne skrive tekst ud til det forbundne modul. Da vi ikke på dette tidspunkt ved hvordan den modtagne data skal behandles gemmer vi blot inputstreamen i en instansvariabel, så vi kan bruge den i resten af klassen. Til sidst i denne metode startes en opdateringstråd, som løbende skal læse data fra modulet. Dette gøres kun hvis den ikke allerede er startet. Med dette har vi allerede formået at oprette en forbindelse til robotten.

Vi ville i denne spike også se om vi kunne læse det data der kommer tilbage fra det forbundne modul. Til dette skulle vi bruge den tidligere nævnte opdateringstråd. Opdateringstråden er blot et implementeret Runnable interface, dvs. et stykke kode der kører i sin egen tråd, som så hele tiden skal sørge for at tage imod det data der kommer ind fra det forbundne modul. I denne spike valgte vi blot at udskrive alt det modtagne data som tekst til konsollen.

Følgende er vores implementering af dette:

```
public class ModClient implements Runnable {
    ...
    private void startUpdateThread() {
        threadRunning = true;
        Thread t = new Thread(this);
        t.start();
    }
}
```

```
public void run() {
    while (true) {
        if (is != null) {
            int bytesToRead = is.available();
            byte[] buffer = new byte[bytesToRead];
            is.read(buffer);
            String output = new String(buffer);
            if (!output.equals("")) {
                System.out.print(output);
            }
        } //try-catch delen er undladt for nemmere læsning
    }
}
...
```

Metoden **startUpdateThread** opretter blot et Thread objekt med det nuværende ModClient objekt som parameter. Derved bliver ModClients **run** metode kaldt så snart den netop oprettede tråd startes. Grunden til at alt dette overhovedet kræver sin egen tråd er at data fra robotten modtages asynkront og derfor skal der ikke kun læses data når der afsendes en kommando. Man kan f.eks. abonnere på data fra et modul og så vil data ankomme med regelmæssige mellemrum uden nogen afsendt kommando. I implementeringen af **run** metoden lavede vi et uendeligt loop, så data bliver læst indtil programmet stoppes. Inde i dette loop tjekker vi først om inputStream er null for at sikre os vi har et objekt at læse fra. Derefter sørger vi for kun at læse det antal bytes fra inputStreamen som vi kan læse uden at streamen blokerer. Denne del af implementeringen var den der tog længst tid i udviklingen af denne spike. Vi havde først brugt en BufferedReader til at læse fra inputStream, men den gav mange problemer da den konstant blokerede for videre afvikling når den ventede på mere data end der kom. Derfor brugte vi den rå inputStream i stedet for.

Til "Spike 2: Send kommando til Modul", var det oplagt at bruge den implementerede ModClient klasse fra Spike 1 og bare lave en ekstra metode til den, så man også kan sende en kommando ud til modulet. Dette gjorde vi, og det viste sig hurtigt at være en nemt implementeret spike, uden meget af den kompleksitet som vi havde forventet.

Dette er vores implementering:

```
public void toServer(String cmd) {
    if (out != null) {
        out.println(cmd);
        out.flush();
    }
}
```

Variablen **out** er den PrintWriter som blev oprettet i **connect** metoden (se Spike1 implementering) og gemt så andre metoder kan tilgå den. Derfor skulle vi i denne **toServer** metode blot tage imod en String cmd, skrive den ud til **out** og derefter sørge for at bufferen er blevet flushed.

3.4.3 Spike resultater

Spike 1

På baggrund af implementeringen af Spike1 har vi estimeret at user-story "Forbind til Modul" bliver den vanskeligste af alle og dermed vurderet til 10 point. Det at forbinde til et modul i sig selv tog ikke

så lang tid at udvikle, men det at kunne læse det modtagne data som XML og kunne gemme det på en forsvarlig måde og samtidig have en stor grad af modularitet i systemet bliver en stor udfordring.

Spike2

Spike2 var langt nemmere at implementere end regnet med. Da Spike1 allerede var implementeret var det at tilføje muligheden for også at skrive en kommando ud til en socket rimeligt overkommeligt. Det krævede dog at vi havde en separat tråd der hele tiden tømmer det data som kommer ind fra det forbundne modul. På baggrund af denne Spike estimerer vi user-story "Send kommando til Modul" til 1 point.

Overvejelser

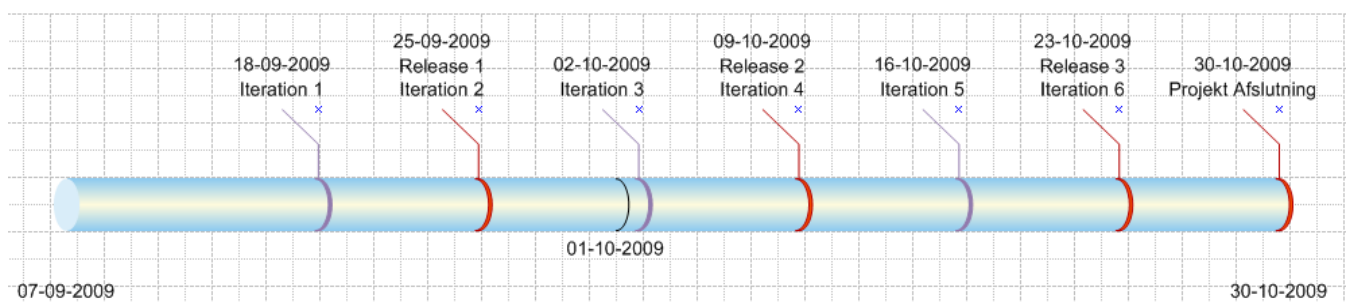
Tilsammen var disse 2 Spike Investigations meget gavnlige at udføre. Produktet er et "Proof of Concept", som fungerer meget som en Telnet klient. Den kan forbinde til hvilken som helst host og port og skrive tekst til den forbundne socket, samtidig med at alt det modtagne data fra socket bliver skrevet ud til skærmen i tekstform. Vi kan med fordel bruge denne Proof of Concept eller elementer fra den i vores senere udvikling af systemet.

3.5 Releaseplan

Udarbejdet af: Kristina

Da vi regelmæssigt skal besøge DTU og vise hvad vi har fået implementeret har vi valgt at lave et Release hver anden uge og have 2 iterationer i hvert Release. Normalt ville man stoppe efter et færdigt Release og hvis der så var flere krav fra kunden ville man planlægge et nyt Release. Men da vi laver et skoleprojekt med en fastlagt deadline, kan vi ikke nå at lave flere releases. Dog har vi valgt at have en ekstra uge efter sidste release hvor der er tid til at lave de sidste småændringer.

Følgende er vores Releaseplan:



Release 1 – User-stories: 11 pts Release Date: 25-09-09	Release 2 – User-stories: 12 pts Release Date: 09-10-09	Release 3 Release Date: 23-10-09
Forbind til Modul (10 pts) Send kommando til Modul (1 pt)	Vis variabeltræ (2 pts) Tilføj variabel til status (4 pts) Tilføj Modul (6 pts)	(Kommende Krav)

Overvejelser

Vi har endnu ikke user-stories nok til at kunne udfylde Release 3. Ud fra projektpartners feedback vil vi kunne lave user-stories til Release 3 senere i forløbet.

3.6 Task Cards

Udarbejdet af: Bjørn

User-stories til hvert Release skal fordeles over 2 iterationer. Derfor vil vi lave Task Cards før iterationsplanen, så vi bedre kan få et overblik over hvor lang hver user-story er og dermed bestemme hvilke der skal være i hver iteration. Vi har også her estimeret hvor lang tid vi regner med hver task vil tage. Tidsestimering er for parprogrammering.

Vi har valgt at vise 2 Task Cards her.

Alle Task Cards ligger i Bilag 3.1 "Task Cards, Release 1".

Task Card 1, Iteration 1, User-story: <u>Forbind til Modul</u>
Dato: 15-9/09
Task: Lav JUnit test til XMLClient
Tid: 2 time
Kommentar:

Task Card 2, Iteration 1, User-story: <u>Forbind til Modul</u>
Dato: 15-9/09
Task: Lav XMLClient
Tid: 4 timer
Kommentar:

Overvejelser

I forhold til use-casen Forbind til Modul har vi måtte opdele den i mange tasks da den er meget kompleks, og også dele den over 2 iterationer. Den task under user-story "Forbind til Modul", som har størst forretningsværdi er at lave XMLClient, da den kommer til at danne bunden for store dele af systemet. På baggrund af de tidligere Spike Investigations ved vi at task "Send kommando til Modul" vil være hurtig at implementere.

3.7 Iteration Plan

Udarbejdet af: Daniel

Da den første user-story vi skal implementere skal deles op i flere tasks og dermed strækker sig over 2 iterationer har vi valgt at lave vores iterationsplan for hele første Release, så den viser de 2 første iterationer. I forhold til tid mener vi det passer fint at skulle bruge omkring 12 timer pr iteration til programmering, da vi ud over dette også skal udarbejde artefakter, lave refaktorering af koden, teste og skrive rapport. Vores tidsestimering er i timer pr. person der skal bruges til programmering. Da vi programmere i par vil det kun være 2 der programmerer af gangen. Den tredje vil arbejde på dokumentet.

Følgende er vores Iteration Plan for 1. og 2. Iteration:

Release 1	Estimeret Tid/timer	Iteration 1	Iteration 2
Forbind til Modul	23 t.	12 t.	11 t.
Send kommando til Modul	1 t.	1 t.	0 t.
Total:	24 t.	13 t.	11 t.

Overvejelser

Iterationsplanen giver et godt overblik over hvor lang tid vi skal bruge på implementering i dette Release. Dermed kan vi bedre planlægge rapport skrivning og udarbejdelse af andre artefakter.

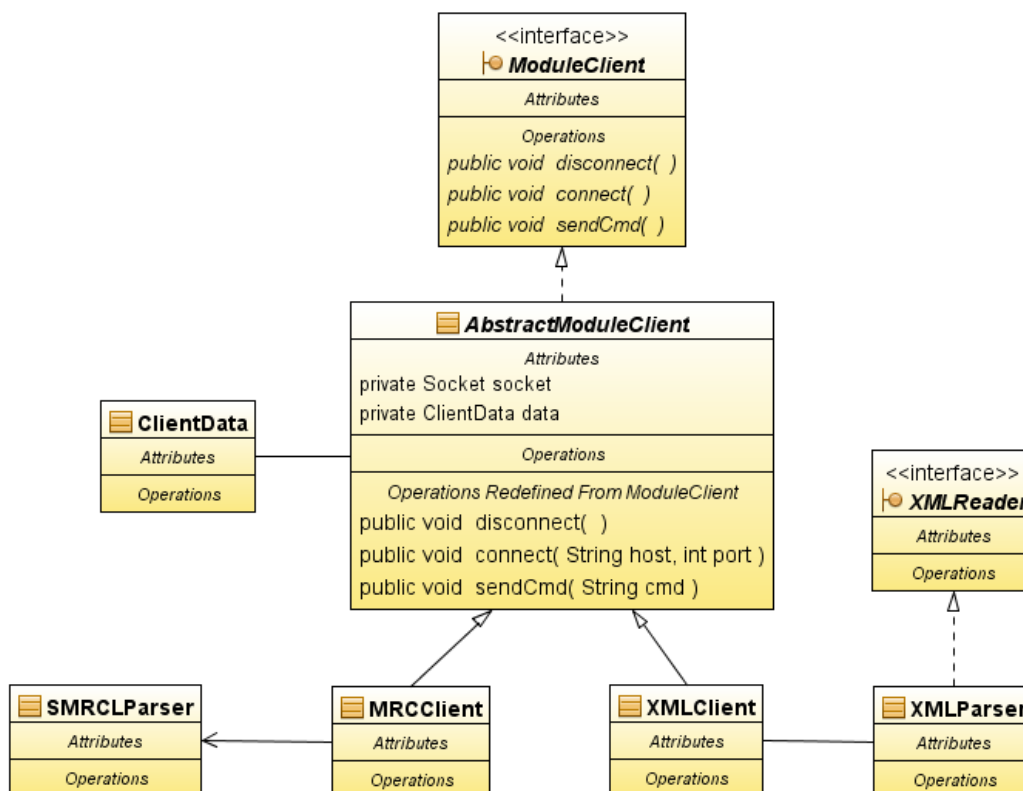
3.8 Design Klasse Diagram

Udarbejdet af: Bjørn, Daniel, Kristina

I forhold til systemets arkitektur har vi først udarbejdet vores CRC cards på papir og sat dem sammen. Men for at kunne vise dette elektronisk og få et overblik over systemets objekter og deres relationer på kode-niveau har vi lavet et Design Klasse Diagram. Den skal ikke repræsentere hele systemet, men primært den del af systemet som skal kommunikere med et robot-modul. Vi har valgt at fokusere på modul-delen af systemet, da den indeholder mest kompleksitet, og derfor kræver et bedre visuelt design og overblik.

I denne rapport har vi valgt at ligge vores Design Klasse Diagram ind først, så det ikke bliver forvirrende at skulle læse alle CRC cards igennem uden at have et overblik over dem. De enkelte metoder vi har vist i Design Klasse Diagrammet har vi beskrevet sammen med en udførlig beskrivelse af de forskellige klasser under CRC cards.

Følgende er vores Design Klasse Diagram:



3.9 Class, Responsibilities and Collaboration

Udarbejdet af: Bjørn, Daniel

Vi har valgt at give alle klasser og metoder engelske navne da vi programmerer på engelsk, så videreudvikling ikke er begrænset til danskere. Efter hvert CRC Card har vi beskrevet selve klassen mere grundigt og beskrevet metoderne under de enkelte klasser i Design Klasse Diagrammet ovenfor. For en samlet oversigt af CRC cards se Bilag 4 "CRC Cards".

ModuleClient

Class Name: ModuleClient	
Superclasses:	
Subclasses: Implementeret af AbstractModuleClient	
Responsibilities:	Collaboration:
Fastlægge en kontrakt for alle Modul Klienter	

Beskrivelse af ModuleClient

Interfacet ModuleClient er grundskabelonen for vores Client klasser. Den foreskriver en række metoder som skal være til stede i alle implementeringer.

Metoder:

- public void disconnect();* Denne skal implementeres til at afbryde forbindelsen til robot modulet af de klasser der implementere den.
- public void connect();* Denne skal implementeres til at oprette forbindelse til et robot modul af de klasser der implementere den.
- public void sendCmd();* Denne skal implementeres til at sende en kommando til robot modulet af de klasser der implementere den.


AbstractModuleClient

Class Name: AbstractModuleClient	
Superclasses: (IF)ModuleClient	
Subclasses: XMLClient, MRCClient	
Responsibilities:	Collaboration:
Oprette en socket forbindelse til et modul Lukke en socket forbindelse til et forbundet modul Sende tekst/kommandoer til modulet	

Beskrivelse af AbstractModuleClient

Denne klasse er en abstrakt klasse der implementerer ModuleClient. AbstractModuleKlient er en generel implementering, der kan nedarves fra. Den har til ansvar at implementere metoderne fra ModuleClient.

AbstractModuleClient foreskriver også en abstrakt run-metode, som hver subklasse til AbstractModuleClient skal overskrive. Idéen med denne er at hver subklasse har hver sin måde løbende at læse data fra modulet på. Derfor har de også brug for deres egen implementering af run metoden. Læsningen af data skal ske i en tråd og derfor bruges run-metoden fra interfacet Runnable.

 AbstractModuleClient
Attributes
private Socket socket private data
Operations
public void run()
Operations Redefined From ModuleClient
public void disconnect() public void connect(String host, int port) public void sendCmd(String cmd)

Måden at *sende* data til modulet ændrer sig ikke og derfor er `sendCmd` implementeret i `AbstractModuleClient`.

Metoder

<code>public void disconnect():</code>	Denne skal afbryde forbindelsen til robot-modulet.
<code>public void connect(String host, int port):</code>	Denne skal oprette forbindelse til et robot-modul på en given host og port.
<code>public void sendCmd(String cmd):</code>	Denne skal sende en kommando til robot modulet ud fra en given string.
<code>public void abstract run():</code>	Denne skal overskrives af hver subklasse til at læse data.

XMLClient

Class Name: XMLClient	
Superclasses: AbstractModuleClient, (IF) ContentHandler	
Subclasses:	
Responsibilities:	Collaboration:
Læse XML løbende og omsætte det til data	XMLReader, ClientData
Sende XML-pakket kommando til modul	

Beskrivelse af XMLClient

XMLClient bliver den primære subklasse til `AbstractModuleKlient`, som løbende skal kunne læse XML data fra serveren. Data ankommer asynkront. Den skal bruge Javas egen `XMLReader`. Dette gøres ved at pakke `sockets.InputStream` ind i en `InputSource` og parse denne med `XMLReader` i sin egen tråd. `SendCmd` metoden skal overskrives i denne klasse så kommandoer bliver pakket ind i XML tags under alle omstændigheder.

XMLReader

Class Name: XMLReader	
Superclasses:	
Subclasses:	
Responsibilities:	Collaboration:
Læse og konvertere XML	XMLParser

Beskrivelse af XML Reader

Denne klasse er Javas egen SAX2 XML reader. Den læser data fra en `InputSource` og pakker den ud af XML tags, så det bliver muligt at behandle det modtagne data. Efter man har oprettet en instans af `XMLReader`, bør man angive en `ContentHandler`, som modtager det indhold `XMLReader` har pakket ud fra det tilsendte XML data.

XMLParser

Class Name: XMLParser	
Superclasses:	
Subclasses:	
Responsibilities:	Collaboration:
Modtage udpakket data fra XML Reader	XMLReader

Beskrivelse af XML Parser

For at holde en høj binding af vores klasser har vi lagt implementeringen af ContentHandler i sin egen klasse. ContentHandler er den der modtager data der er blevet pakket ud af XML af XMLReader. Vi har valgt at XMLClient skal indeholde en XMLParser, som skal stå for selve behandlingen af data. Dette gør at XMLClient ikke bliver for bred og kompliceret en klasse og vil forhåbentlig gavne os senere i implementeringsforløbet.

ClientData

Class Name: ClientData	
Superclasses:	
Subclasses:	
Responsibilities:	Collaboration:
Opbevare data mens programmet kører	

Beskrivelse af ClientData

Dette er klassen som skal indeholde det data som bliver læst i XMLParser. Det er endnu ikke bestemt hvordan den skal gemme dataen, men det er med i vores overvejelse at det skal være modulært.

MRCClient

Class Name: MRCClient	
Superclasses: Abstract Klient	
Subclasses:	
Responsibilities:	Collaboration:
Læse SMRCL	SMRCL Parser,
Skrive kommando i SMRCL	ClientData

Beskrivelse af MRCClient

Dette er den anden subklasse af AbstractModuleClient. Den skal være en implementering af ModuleClient der skal snakke med et modul som ikke kommunikerer i XML. Fra vores kravspecifikation og det vi har fået fremvist på DTU foregå kommunikationen som en telnet klient og er mest tekstbaseret. Dermed er det ikke sikkert at vi skal pakke den modtagne data ud, men vi har sikret at vores arkitektur ville understøtte dette om nødvendigt.

Class Name: SMRCLParser	
Superclasses:	
Subclasses:	
Responsibilities:	Collaboration:
Læse og konvertere SMRCL	

Beskrivelse af SMRCLParser

Dette er parseren til MRCClient. Den skal læse det modtagne data, der ifølge vores kravspecifikation er i DTUs SMRCL sprog. Dog så det ud til fra feedback på DTU at data ikke er pakket ind på nogen måde der umiddelbart ville kræve en parser. Vi har dog forbeholdt os muligheden for dette i vores design.

3.10 Acceptance Test

Udarbejdet af: Kristina

Vi har lavet disse Acceptance test på baggrund af projektpartners krav til systemet. Efter endt implementering af user-stories til dette Release vil vi køre dem på DTU for at sikre os at de giver det forventede output.

Vi vil bruge acceptance test til at teste forbindelse til modul, XML konvertering og læsning af data. Da en stor del af hver test omhandler det bagvedliggende kode og ikke er noget der kan ses fra GUI'en, har vi skrevet et afsnit under hver test for bedre at kunne forklare hvad der sker og hvad vi tester.

Forbind til modul 1: Opret forbindelse til modul

Test Trin	Input/Handler	Forventet Output	Resultat
1	Klient får besked på at skulle forbinde til et modul	Besked fra Server om at forbindelsen er oprettet.	

Forklaring

Denne test skal teste om der kan oprettes forbindelse til et modul. Hvis der er forbindelse til modulet, modtages der en enkel besked med modulets navn fra robotten. Sådant en besked kunne f.eks. se således ud: `<auClient name="AuClientNox" version="2.04"/>`. Svaret kommer an på hvilket modul man forbinder til, men kravet er at den modtagne besked indeholder modulets navn og version.

Forbind til modul 2: Konverter XML

Test Trin	Input/Handler	Forventet Output	Resultat
1	Kommando sendes til et forbundet modul Klienten sender: "var core.version"	XML der er konverteret til Strings GUI RawData vinduet viser: <code><var name="core.version" value="x.xx"></var></code>	

Forklaring

Denne test går ud på at teste om det XML kode der kommer tilbage fra modulet, når en kommando er afsendt, er blevet konverteret korrekt via vores XMLParser klasse.

Forbind til modul 3: Gem data

Test Trin	Input/Handler	Forventet Output	Resultat
1	Konverteret XML gemmes i intern hukommelse Klient sender: "var allcopy"	Konverteret XML er gemt i intern hukommelse Alle variabler bliver vist i RawData vinduet.	

Forklaring

Denne test går ud på at teste om det konverteret XML data er blevet gemt korrekt. Dette skal kunne ses i RawData vinduet.

3.11 Programmering

Udarbejdet af: Daniel, Kristina

I dette afsnit om programmering vil vi forklare i dybden hvordan vi har programmeret forskellige dele af systemet. For at gøre det nemmere at forstå vil vi illustrere det med kodeuddrag og grafiske illustrationer. Derudover vil vi diskutere nogle generelle overvejelser om programmeringen.

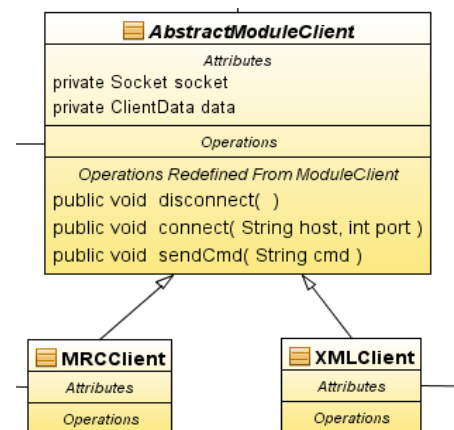
3.11.1 Beskrivelse af kode

I dette Release skulle vi lave de 2 user-stories "Forbind til Modul" og "Send Kommando til Modul". Disse udgør kernen af systemet. Allerede i dette release var der User-stories der gjorde brug af en plugin struktur. Derfor var det vigtigt at vi fra start fokuserede på at reducere kompleksiteten og gøre systemet så modulært som muligt.

Til at starte med i denne iteration havde vi lavet 2 Spike Investigations som gav os viden og en bund at bygge vores Socket implementering ud fra.

Socket forbindelse

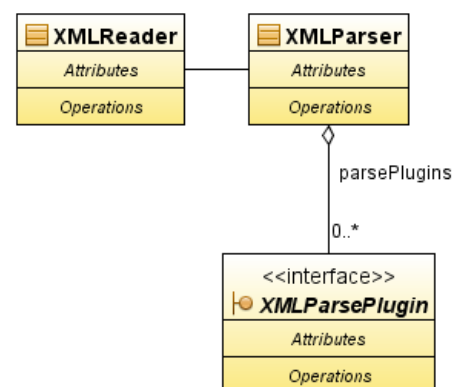
Vi generaliserede til AbstractModuleClient alt der havde med selve forbindelsen til modulet at gøre, så som oprettelse og lukning af Socket og streams. Dette reducerede drastisk kompleksiteten af andre modul klienter, så som XMLClient og MRCCClient, da de nedarver fra AbstractModuleClient, og i store træk kan abstrahere fra alt omkring Socket forbindelsen. Denne arkitektur kom vi frem til ved brug af Design Klasse Diagrammet og CRC cards, som var til utrolig stor hjælp for det overordnede design at systemet.



Plugins

Vi har valgt at lave store dele af vores system i en såkaldt plugin-struktur. Dette har vi gjort da det er et krav fra projektpartner at de skal kunne lave deres egne moduler.

I forhold til plugin-struktur har vi i dette Release fokuseret på læsningen af XML data. Vi har lavet det sådan at XMLParser klassen er en slags "container" for XMLParserPlugins, som alle får at vide når der er ny data. XMLReader har allerede pakket alt dataen ud af XML tags, så det er Plugin's opgave at tolke dette data og gemme det. Vi har valgt ikke at have en central database til at holde alt data fra alle plugins, da det ville kræve at hver plugin-udvikler, skulle ind og rette i denne og indsætte sine egne datastrukturer. I stedet må hvert plugin selv bestemme hvordan det vil gemme og udbyde data til de GUI der har brug for det.



I interfacen XMLParsePlugin, som foreskriver en række metoder for modtagelse af data, har vi dog valgt også at lave en `setPropertyChangeSupport` metode som hvert plugin skal implementere (se kodeudtrage nedenfor). Denne skal plugins implementere så de sætter en intern

PropertyChangeSupport variabel og kalder firePropertyChange på denne når de vil lave callbacks til klasser som abonnerer på det givne plugins data.

Følgende er kodeuddrag til at illustrere den implementerede plugin struktur for XML parsing.

Interface XMLParsePlugin:

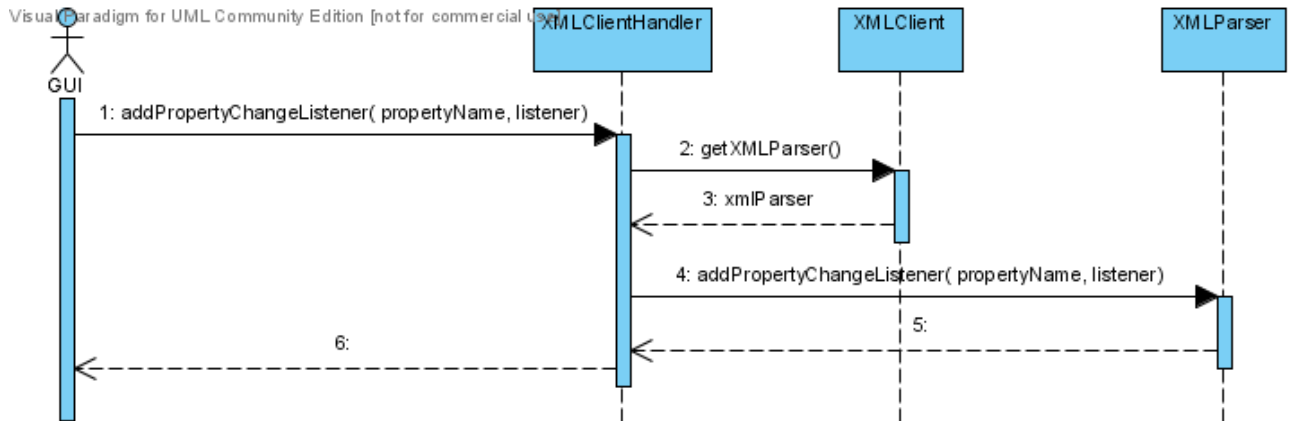
```
public interface XMLParsePlugin {  
  
    public void startElement(String tagName, Attributes atts);  
  
    public void endElement(String tagName);  
  
    public void tagContents(String contents);  
  
    public void setPropertyChangeSupport(PropertyChangeSupport prop);  
}
```

Dette er et kode-uddrag fra VarDataPlugin, som er en af vores egne implementeringer af XMLParsePlugin:

```
public class VarDataPlugin implements XMLParsePlugin {  
  
    public final static String SUBSCRIBE_VARDATA = "varData";  
    private PropertyChangeSupport prop;  
    private ModuleVariable lastVariable;  
  
    public VarDataPlugin() {  
    }  
  
    public void setPropertyChangeSupport(PropertyChangeSupport prop) {  
        this.prop = prop;  
    }  
  
    public void startElement(String tagName, Attributes atts) {  
        if (tagName.equals("var") && atts.getLength() > 2) {  
            ModuleVariable oldVar = lastVariable;  
            String varName = atts.getValue("name");  
            String varType = atts.getValue("typ");  
            String varValue = atts.getValue("value");  
            ModuleVariable newVar = new ModuleVariable(varName, varType, varValue);  
            lastVariable = newVar;  
            prop.firePropertyChange(SUBSCRIBE_VARDATA, oldVar, newVar);  
        }  
        ...  
    }  
}
```

Metoden firePropertyChange bruges til at kalde tilbage til alle der har abonneret på data. Det første argument i metoden firePropertyChange "varData" er den String som GUIs skal bruge når de vil abonnere på netop denne propertyChange. Det andet argument "oldVar", er den gamle værdi og det tredje "newVar" er den nye værdi.

Da det oftest er et enkelt stykke data et plugin modtager, vil det være igennem firePropertyChange metoden at det nye data sendes. Men der er også mulighed for at et plugin selv laver en datastruktur og metoder hvorpå denne gøres tilgængelig for GUIs der har abonneret på den. En GUI kan abonnere på data igennem XMLClient, ved at kalde getXMLParser() og på denne kalde addPropertyChangeListener(propertyName, listener). Se følgende SD:



Følgende er et uddrag fra vores GUI ModuleTabs, som abonnerer på 2 forskellige slags data:

```
client.getXMLParser().addPropertyChangeListener(RawDataPlugin.SUBSCRIBE_RAWDATA, listener);
client.getXMLParser().addPropertyChangeListener(VarDataPlugin.SUBSCRIBE vardata, listener);
```

Se Sekvens Diagram "Modtagelse af XML Data" for en mere overordnet forståelse af den efterfølgende proces når en GUI har abonneret og data modtages fra et modul.

3.11.2 Generelle overvejelser om programmeringen

Som XP forskriver lavede vi parprogrammering det meste af tiden. Dog blev prototypen af brugergrænsefladen, som blev brugt til at forevise projektpartner vores idéer i forhold til GUI, tegnet i hånden af en enkelt person. Implementeringen af brugergrænsefladen lavede vi parvis. Det virkede fint med parprogrammering fordi vi fik overvejet nogle ting omkring modularitet og opsætning af klasser samtidigt. Senere kan det dog godt være at det vil være okay at programmere enkeltvis, da vi jo kun er 3 i vores gruppe og det ikke er et kæmpe projekt vi arbejder på.

Det virkede fint at bruge en repository model til at opbevare vores system på. På den måde var det nemt for alle i gruppen af få adgang til den nyeste version af MARG.

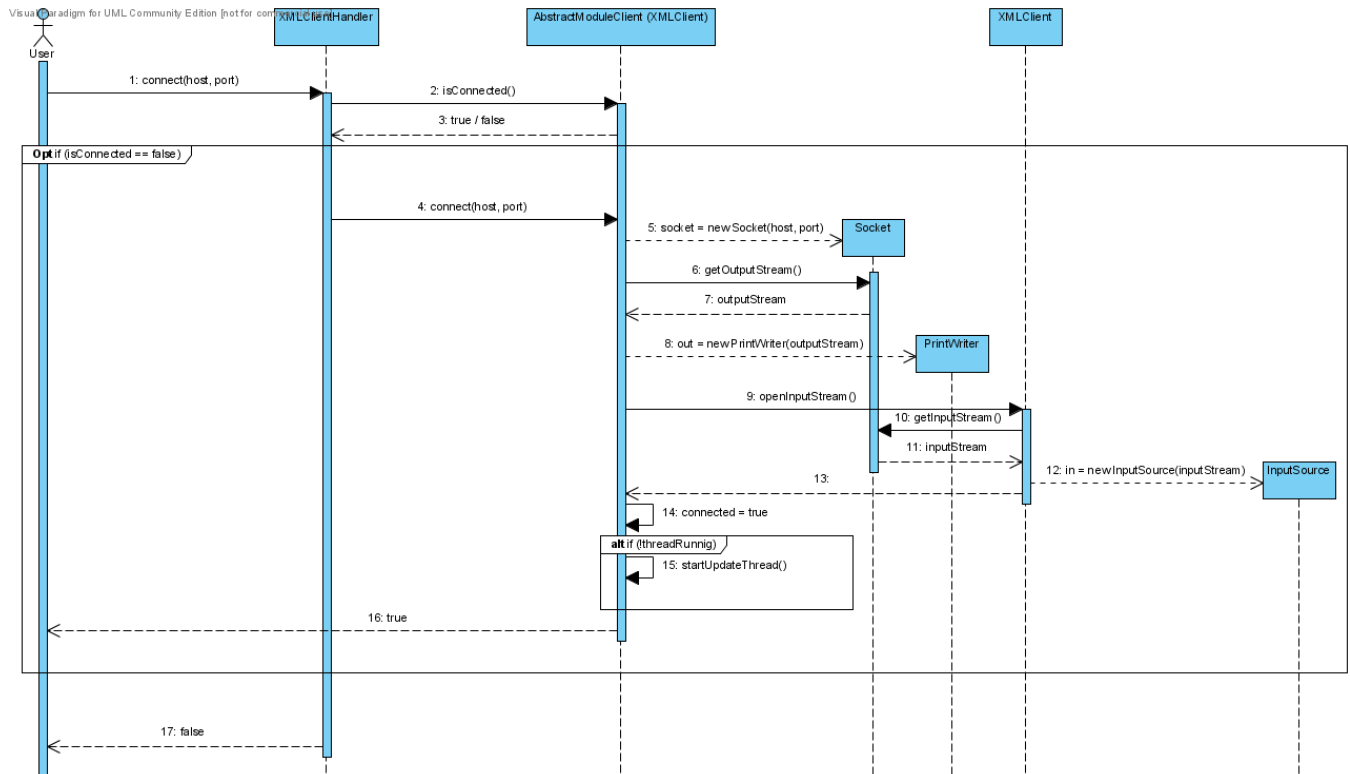
3.12 Sekvens Diagrammer

Udarbejdet af: Bjørn, Daniel

I forhold til Sekvens Diagrammer (SD) havde vi ikke behov for at udvikle nogen før implementeringen af vores user-stories til dette Release. Vi har dog valgt at lave de vigtigste efterfølgende da vi i gruppen har forskelligt kodeniveau, og dette ville give et bedre overblik over hvordan koden hænger sammen for alle. Vi har derfor også valgt at beskrive vores Sekvens Diagrammer grundigt, så det er forståeligt hvad der sker.

Vi har valgt at lave 3 Sekvens Diagrammer. Først og fremmest har vi lavet SD til de 2 brugerinteraktioner som er mulige i vores første release, nemlig "Forbind til Modul" og "Send Kommando til Modul". Derudover besluttede vi at lave en enkel utraditionel SD der viser en interaktion fra modulets perspektiv, nemlig når ny asynkron XML data modtages og skal behandles af et plugin. I dette tilfælde valgte vi at det var klassen VarDataPlugin der skulle behandle det modtagne data og opdatere en træstruktur med det nye data.
For større SD'er se Bilag 8.1 "Sekvens Diagrammer, Release 1".

3.12.1 Forbind til Modul



Beskrivelse af Forbind til Modul

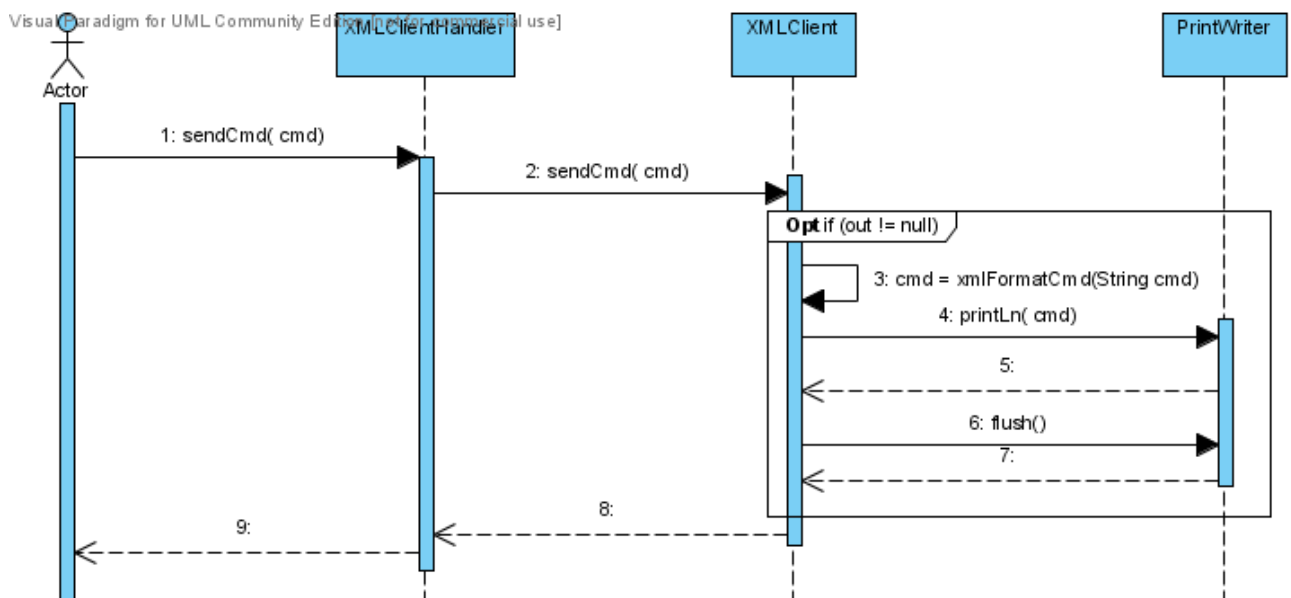
Dette SD beskriver hvordan der oprettes forbindelse til et modul:

AbstractModuleClient er en abstrakt klasse og kan derfor ikke tilgås direkte. Den er i virkeligheden en del af XMLClient, som er en subclasses til den. Vi havde behov for at kunne skelne imellem hvad der skete i den generelle implementering (AbstractModuleClient) og hvad der skete i den specialiserede implementering (XMLClient) og derfor er de to delt op i vores SD.

1. Fra GUI'en bliver der sendt et kald, connect(), til XMLClientHandler, med en host og port til det modul man vil oprette forbindelse til.
2. XMLClientHandler tjekker først om der allerede eksisterer en forbindelse ved at sende et kald, isConnected(), til AbstractModuleClient.
3. Denne sender true eller false tilbage, om forbindelsen allerede eksisterer.
4. Hvis XMLClientHandler får false tilbage sender den en connect() kommando med host og port til AbstractModuleClient, hvilket er superklassen for XMLClient.

- 4B: Hvis der kommer true tilbage betyder det at der allerede er en kørende forbindelse, resten af eksekveringen springes over og der returneres false i den nederste linje; 17. Dette skyldes at man først skal kalde disconnect() før at en ny forbindelse kan oprettes.
5. AbstractModuleClient opretter en ny socket med det host og port den har modtaget, for at kunne få forbindelse til modulet på robotten.
 6. Derefter kalder den getOutputStream() til den nye socket, for at få dennes outputstream.
 7. Den nye socket sender dens outputstream tilbage.
 8. AbstractModuleClient opretter en ny variabel "out" af typen PrintWriter og sætter denne til en new PrintWriter() med den modtagne outputstream som parameter. Dette gør det muligt for klienten at sende kommandoer til modulet igennem "out".
 9. AbstractModuleClient kalder sin egen abstrakte metode openInputStream(). Denne abstrakte metode er implementeret af dens subclasses, hvilket i dette SD er XMLClient.
 10. XMLClient sender getInputStream() til socket.
 11. Socketen sender sin inputStream tilbage til XMLClient
 12. XMLClient opretter en ny variabel "in" og sætter den til en ny InputSource med inputStream som parameter. Dette gøres da Javas egen XMLReader, som vi bruger i XMLClient, bruger en InputSource til løbende at læse XML data, modtaget fra det forbundne modul.
 13. Eksekvering sendes tilbage til AbstractModuleClient
 14. AbstractModuleClient sætter sin variabel connected = true, fordi der nu er oprettet forbindelse.
 15. XMLClient tjekker threadRunning, som angiver om en opdateringstråd kører i forvejen. Hvis der ikke kører en tråd i forvejen, kalder den sin egen metode startUpdateThread().
 16. Der sendes true tilbage til brugeren. Forbindelsen er blevet oprettet.

3.12.2 Send kommando til Modul

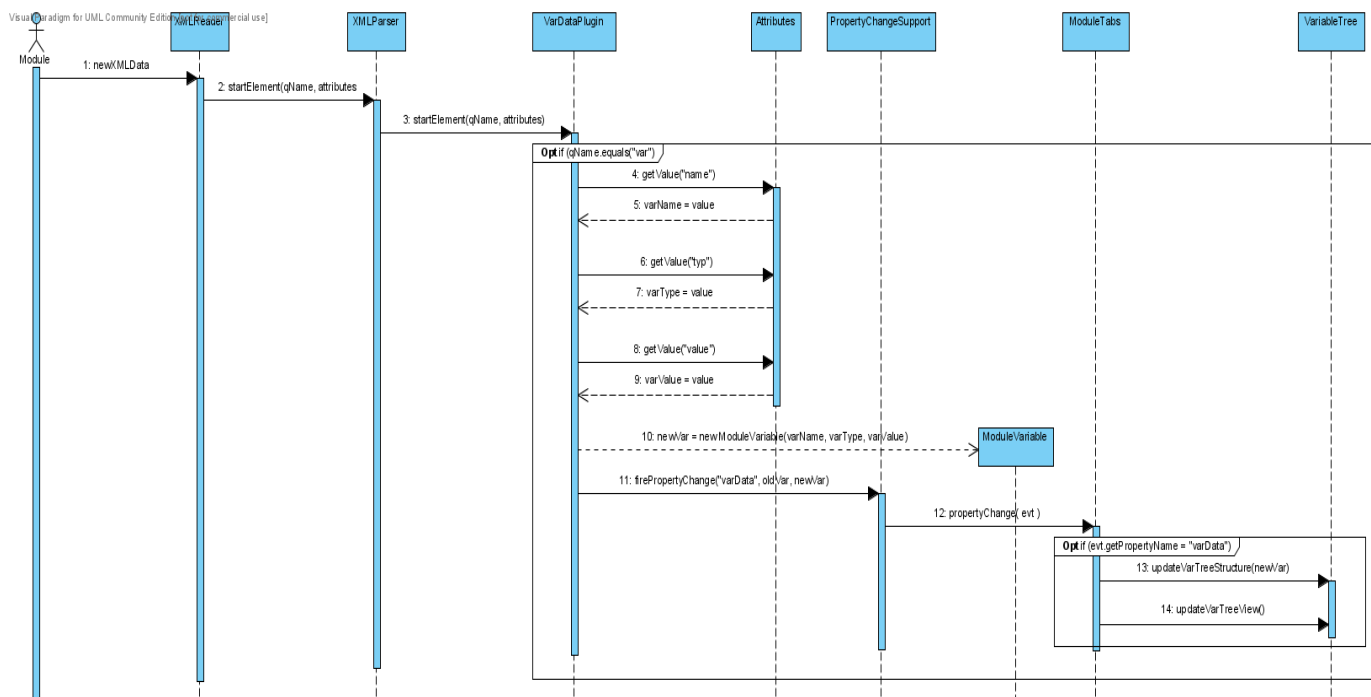


Beskrivelse af Send Kommando til Modul

Dette SD beskriver hvordan der sendes en kommando til et modul:

1. Fra GUIen sendes et kald `sendCmd()` til `XMLClientHandler` med brugerens kommando som en string med navnet `cmd`.
2. `XMLClientHandler` sender kaldet videre til `XMLClient`
3. `XMLClient` tjekker om `out` variabelen, som er en `PrintWriter` forbundet til `Sockets outputStream`, ikke er null. Hvis den ikke er null kalder `XMLClient` sin egen metode `xmlFormatCmd` med kommandoen som parameter, for at indsætte `< og />` rundt om kommandoen.
4. Derefter sender `XMLClient` `println` til dens `PrintWriter (out)` med den nye xmlFormaterede string som parameter.
5. Eksekvering returneres til `XMLClient`
6. `XMLClient` kalder `flush()` til dens `PrintWriter` for at sikre at alle beskeder til modulet er sendt tilbage.
7. Eksekvering afsluttes

3.12.3 Modtagelse af XML data



Beskrivelse af 'Modtagelse af XML Data

Dette utraditionelle SD beskriver hvad der skal ske når der bliver modtaget XML data fra et modul. XML data som `VarDataPlugin` tager imod kan eksempelvis se således ud:

`<var name="core.version" typ="d" value="202"/>`

1. Modulet sender noget XML data til `XML Reader`
2. `XMLReader` kalder `startElement()` med `qName` og attributter som parameter til `XMLParser`. Dette gør den da `XMLParser` implementerer `ContentHandler` og er blevet sat som `XMLReader's contenthandler` med `xr.setContentHandler(xmlParser);`

3. XMLParser sender kaldet videre til alle dens plugins, der i dette eksempel kun er VarDataPlugin.
VarDataPlugin tjekker først selve XML elementets navn (qName), på det modtagne data. Hvis navnet er "var", er det variabeldata som er modtaget og VarDataPlugin fortsætter.
4. VarDataPlugin sender getValue() med "name" som parameter til det modtagne Attributes objekt. Attributes indeholder alle attributter på det givne element, som her er "var" elementet.
5. Attributes sender værdien på "name" tilbage og det bliver gemt som varName.
6. XMLParser kalder getValue() med "typ" som parameter til Attributes
7. Attributes sender værdien på "typ" tilbage og det bliver gemt som varType.
8. XMLParser kalder getValue() med "value" som parameter til Attributes.
9. Attributes sender værdien på "value" tilbage og det bliver gemt som varValue.
10. XMLParser opretter en ny ModuleVariable som er en simpel datastruktur der indeholder varName, varType og varValue som variabler.
11. XMLParser kalder firePropertyChange til PropertyChangeSupport med teksten "varData" for at angive hvad der er sket ændringer med, den tidligere ModuleVariable som "oldData" og den nye som "newVar".
12. PropertyChangeSupport sender en propertyChange med de nye ændringer til ModuleTabs, så den kan reagere på det nye data.
13. ModuleTabs tjekker om propertyName er lig "varData". Hvis den er det, kalder den updateVarTreeStructure med den nye ModuleVariable til VariableTree, for at indholdet af variabeltræet kan blive opdateret.
14. Til sidst kalder ModuleTabs updateVarTreeView() så de nye ændringer bliver vist i GUIen

Overvejelser

Udviklingen af Sekvens Diagrammer hjalp os med at få et overblik over interaktionen imellem klasserne, i stedet for at blive væk i kodens små detaljer. "Forbind til Modul" var godt for vores egen forståelse af hvordan vi havde lavet systemet og fået XML klienten udviklet. Den mere utraditionelle SD der beskriver modtagelsen af asynkron XML data var også utroligt god for os, da den gav os et overblik over en af de meget komplicerede og centrale interaktioner i vores system som ikke direkte har noget med en enkel bruger-interaktion at gøre.

3.13 Brugergrænseflade

Udarbejdet af: Bjørn

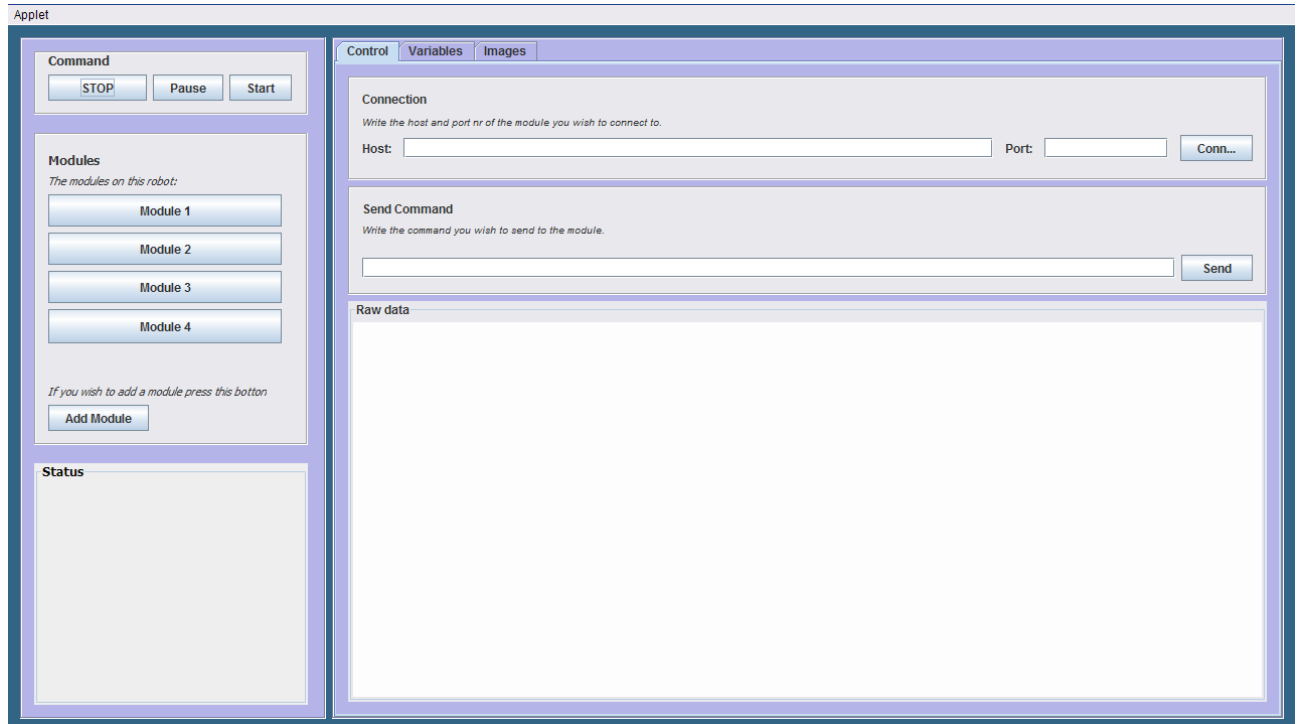
Task Card nr 4 gik ud på at lave en GUI prototype som vi kan bruge til at fremvise de 2 fuldførte User-stories for projektpartner ved Release 1. Følgende er hvad vi kom frem til ud fra vores første krav fra projektpartner.

Til venstre har vi lavet en Menu til Modulerne. Der skal være en knap for hvert modul man kan have forbindelse til. Derudover kan der oprettes nyt modul ved at bruge "Add Module" knappen. Øverst til venstre er der Kommando knapper til at styre selve roboten; "STOP", "Pause" og "Start". Nederst til venstre har vi lavet et lille vindue til de status-variabler der altid skal kunne overvåges.

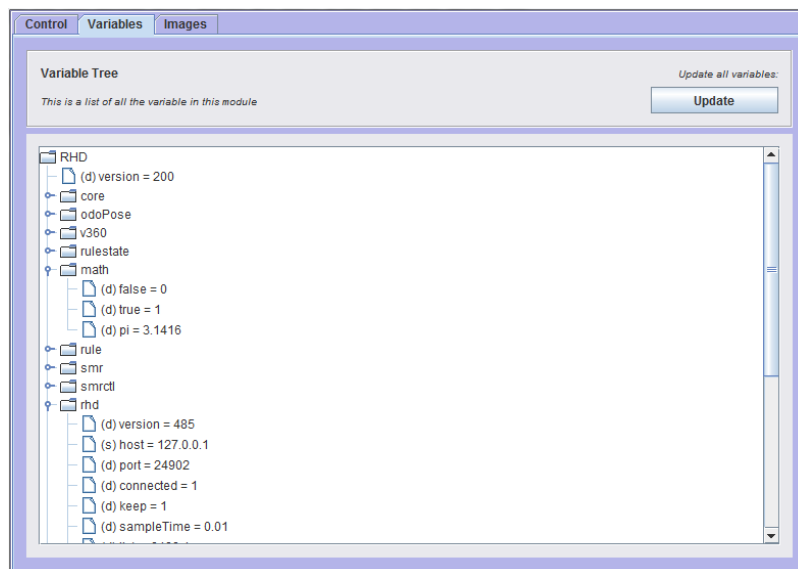
Til højre ser man så de enkelte faneblade der er under et givent modul. Hvis man fx har klikket på "Module 1" ser man de faneblade der hører til dette modul. Her er vist et control faneblad, hvor man kan oprette forbindelse til modulet, sende kommandoer og se alt det data der kommer tilbage som

tekst. Under Module 1's variabel faneblad kan man se variablerne i en træstruktur. Og så kan der være flere andre faneblade så som images mm til de forskellige moduler.

MARG Hovedmenu



Variables Faneblad



Overvejelser

Da dette er vores første udkast til systemets GUI, kan den ændres hen af vejen, i forhold til projektpartners feedback. Det er dog en prototype vi regner med at arbejde videre på, og bruge til det endelige resultat. I senere Releases vil vi beskrive eventuelle ændringer.

3.14 Opdaterede Task Cards

Udarbejdet af: Daniel

Efter endt task har vi skrevet nogle kommentarer på hvert task card. Følgende er vores opdaterede task cards.

Task Card 1, Iteration 1, User-story: <u>Forbind til Modul</u>
Dato: 15-9/09
Task: Lav JUnit test til XMLClient
Tid: 2 time
Kommentar: <ul style="list-style-type: none">- JUnit blev lavet til XMLClient, som er vores primære implementering af ModuleClient og den klasse der sikrer forbindelse til et modul på robotten.- Til testning af forbindelsen brugte vi en såkaldt mock-server, der er en hurtig server implementering der hver gang sender det samme resultat tilbage til den tilsluttede klient.- Denne mock-server brugte vi til at bekræfte at vores modul kunne oprette en socket forbindelse og at input/output virkede som det skulle.- Vi lavede også boundary tests på connect(host, port) metoden, hvor vi testede for om port ligger inden for grænseværdien 1025-65535.- Med vores mock-server kunne vi også teste om serveren/modulet modtog den kommando som blev sendt afsted.

Task Card 2, Iteration 1, User-story: <u>Forbind til Modul</u>
Dato: 15-9/09
Task: Lav XMLClient
Tid: 4 timer
Kommentar: <ul style="list-style-type: none">- AbstractModuleClient er lavet som en fælles implementering af socket forbindelsen for alle senere implementeringer ModulKlienter, så som MRCCClient. Dvs. at man i selve implementeringen af en XMLClient eller MRCCClient i store træk kan abstrahere fra den underliggende socket forbindelse.- Vi udviklede XMLClient efter JUnit testen.

Task Card 3, Iteration 1, User-story: <u>Send kommando til modul</u>
Dato: 15-9/09
Task: Send Kommando til modul
Tid: 1 timer
Kommentar: <ul style="list-style-type: none">- Udviklet sammen med XMLClient- Meget simpelt, implementering tog under en time.

Task Card 4, Iteration 1, User-story: <u>Forbind til Modul</u>
Dato: 16-9/09 – 21-9/09
Task: Lav prototype af overordnet GUI
Tid: 8 timer
Kommentar: <ul style="list-style-type: none">- Prototypen for vores overordnede GUI blev lavet over flere dage.- Første del af prototypen blev vist frem og afprøvet på DTU, som var tilfredse med det vi havde lavet.- Vi afventer nærmere feed-back omkring den endelige GUI prototype.

Task Card 5, Iteration 2, User-story: <u>Forbind til Modul</u>
Dato: 16-9/09
Task: Læsning af XML
Tid: 3 timer
Kommentar: <ul style="list-style-type: none">- Brugte Javas egen XMLReader klasse til at tolke XML data.- Der kræves flere overvejelser omkring denne reader, når det skal laves modulært.

Task Card 6, Iteration 2, User-story: <u>Forbind til Modul</u>
Date: 16-9/09
Task: Gøre læsning af XML modulært
Tid: 3 timer
Kommentar: <ul style="list-style-type: none">- For at gøre læsningen af XML modulært og samtidigt understøtte en plugin struktur, lavede vi et XMLParsePlugin interface, der foreskriver en række metoder til læsning af XML. Vores egen XMLParser klasse, som er den der implementerer ContentHandler og dermed får alle kald fra XMLReader når der er ny data, er klassen som man tilføjer plugins til. XMLParser vil så videresende det data den modtager fra XMLReader til alle plugins den har fået tilføjet.- Således kunne vi lave et RawDataPlugin og et VarDataPlugin, der begge kunne få adgang til den data de har brug for.- Hvert plugin skal også implementere en setPropertyChangeSupport metode således at det er muligt at sende events fra hvert plugin samtidigt med at der kan subscribes til disse events fra XMLParser klassen.

Task Card 7, Iteration 2, User-story: <u>Forbind til Modul</u>
Date: 21-9/09
Task: Opbevar Data modtaget fra XML
Tid: 2 timer
Kommentar: <ul style="list-style-type: none">- I den foreløbige implementering af plugin strukturen styrer plugin'et selv hvordan den opbevarer det modtagne data. De fleste plugins kan lade være med at opbevare noget og blot sende det videre i en firePropertyChange() metode, da der normalt kun er brug for det nyeste data. Hvis det bliver nødvendigt at gemme data på en anden måde, må det implementeres individuelt i hvert plugin.

Task Card 8, Iteration 2, User-story: <u>Forbind til Modul</u>
Date: 21-9/09
Task: Gøre Opbevaring af Data modulær
Tid: 3 timer
Kommentar: <ul style="list-style-type: none">- Modularitet af det opbevarne data opnås ved at hvert plugin selv styrer hvilket data det bibeholder. Tilgængelse af denne data sker igennem propertyChange events, men et plugin kan også skrive sine egne metoder så udefrakommende klasser kan tilgå data på andre måder.

3.15 Testning

Udarbejdet af: Bjørn

I dette Release har vi kun lavet en enkelt JUnit test. Vi har lavet den til den centrale klasse XMLClient, som er en del af begge de 2 implementerede user-stories. Vi har kun lavet denne, da vores andre klasser er af betydelig mindre kompleksitet og for det meste kun har med GUI eller Robotten at gøre, hvilket er svært at teste og ikke decideret nødvendigt i XP.

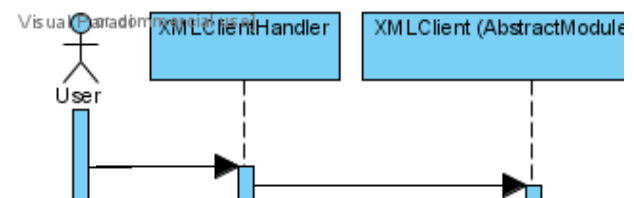
I forhold til andre tests i dette Release har vi også vores Acceptance Tests som vi vil afprøve ved vores besøg på DTU ved præsentation af Release 1.

3.16 Refaktorering

Udarbejdet af: Daniel, Bjørn

I dette afsnit vil vi beskrive i dybden hvordan vi har lavet refaktorering i dette Release.

Ved refaktorering i dette Release har vi lavet en handler klasse, XMLClientHandler imellem vores GUI og XMLClient, for at opretholde det ønskede MVC design. Dette skaber større modularitet og mulighed for senere ændringer.



Vi har også lavet refaktorering af plugin strukturen undervejs efter input fra projektpartner. F.eks. blev metoden tagContents() i XMLParsePlugin lavet om:

```
public void tagContents(char ch[], int start, int length);
```

Før denne ændring lavede vi i XMLParser denne character array om til en String og sendte den videre til hvert plugin. Det viste sig dog at der i visse tilfælde ville komme binært data som tagContent, og det ville derfor ikke være hensigtsmæssigt at dette binærdata bliver lavet om til en String. Derfor må hvert plugin nu selv behandle det modtagne tagContents som det ønsker.

Da vi fra starten har vægtet Modularitet højt og har inddraget mere up-front design i form af UML artefakter, har der også vist sig at være betydeligt mindre refaktorering end forventet. Det kan dog blive nødvendigt med mere refaktorering i kommende iterationer, da user-stories skal bygge videre på den etablerede kode, hvilket kan kræve refaktorering af koden fra dette Release.

3.17 Revideret Design Klasse Diagram

Udarbejdet af: Bjørn, Kristina

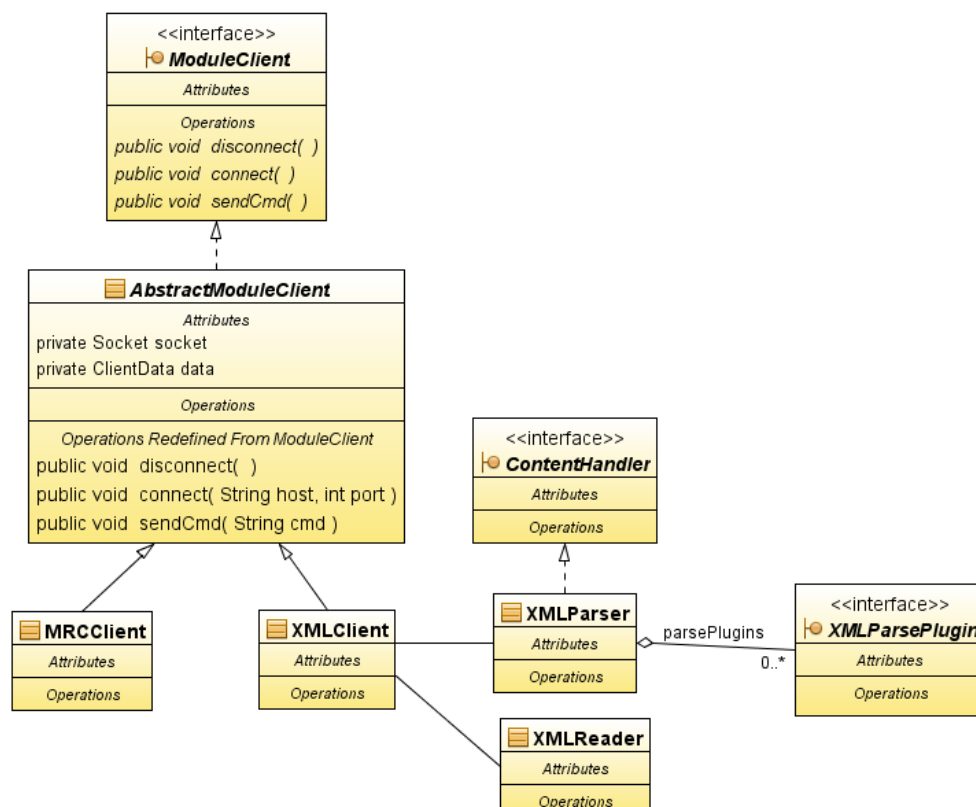
Under implementeringen af vores user-stories skete der ændringer i forhold til det overordnede design, derfor har vi valgt at revidere vores Design Klasse Diagram. Følgende er en beskrivelse af ændringerne.

Først og fremmest er ClientData forsvundet. Den var oprindeligt tiltænkt som en slags central database for alt det modtagne data. Vi har dog i forbindelse med tilføjelse af plugin strukturen besluttet at det ville være bedst at hvert plugin i stedet selv opbevarer sit data i ønskede datastrukturer. Så behøver vi heller ikke foreskrive en specifik datastruktur eller på anden måde begrænse fremtidige plugins. Det gør også at man ikke skal ind og rette i en helt anden klasse når man skriver et nyt plugin der har brug for at gemme data. Den eneste ulempe er at det i visse tilfælde kan gøre det mere vanskeligt for andre dele af systemet at tilgå nødvendige data, men vi mener overordnet at det giver et bedre design.

I et tidligere billede af dette Design Klasse Diagram i programmeringsafsnittet var der en forbindelse imellem XMLReader og XMLParser. Denne er ikke vist her, da forbindelsen sker igennem XMLClient og dermed er underforstået.

Vi har også tilføjet den tiltænkte plugin struktur i form af et kompositions forhold imellem XMLParser og XMLParserPlugin. Dette vil sige at XMLParser kan indeholde 0 eller flere objekter som implementerer interfacet XMLParsePlugin. Samtidigt har vi vist at XMLParser implementerer interfacet ContentHandler. Til sidst har vi også fjernet SMRCLParser klassen, da det efter feedback fra projektpartner ikke virkede som om at der ville blive brug for denne.

Følgende er vores opdaterede Design Klasse Diagram for vores system.



3.18 Implementering af ekstra user-story

Udarbejdet af: Bjørn, Daniel, Kristina

De 2 user-stories som vi havde planlagt at lave i Release 1 blev hurtigere færdige end forventet. Derfor besluttede vi at det ville være en god idé at inddrage den første user-story fra næste Release; "Vis Variabel-træ", da den har høj forretningsværdi og kun er estimeret til 2 ud af 10 point. (Se User-stories).

Her under har vi lavet de forskellige artefakter til denne user-story:

3.18.1 User-story

Vis variabel-træ
Story Number: 2 Estimation: 2 pts Business value: High
Brugeren beder om at få præsenteret alle variabler fra et tilsluttet modul. Variablerne bliver vist som en træstruktur.
Overvejelser: Dette er en grafisk præsentation af den interne datastruktur på det givne modul. En meget vigtig funktionalitet med stor forretningsværdi for projektpartner.

3.18.2 Task Card

Task Card 9, Iteration 2, User-story: <u>Vis variabel-træ</u>
Dato: 16-9/09
Task: Lav variabel-træ
Tid: 2 time
Kommentar: <ul style="list-style-type: none">- Brugte Javas JTree, lavede en ny klasse VariableTree- VariableTree tager imod et oprettet JTree og "dekorerer" det med den data der modtages fra VarDataPlugin.- Man kan i GUI Builderen sætte et normalt JTree ind og så bare dekorere det med vores VariableTree i koden efterfølgende.

3.18.3 Acceptance Test

Vis variabel-træ

Test Trin	Input/Handler	Forventet Output	Resultat
1	Bruger klikker på variabel-træets faneblad i GUIen kaldet "Variables".	Variables faneblad i GUI viser alle variabler som en træstruktur. Om alle variabler er til stede kan tjekkes ved at se om den indeholder alt data der blev modtaget i RawData vinduet.	

Forklaring

Denne test går ud på at teste om Variabel træet bliver vist når man klikker på variabel fanebladet i GUIen. Der kan tjekkes i RawData vinduet om variabeltræet indeholder alle variabler der er blevet modtaget.

3.18.4 Programmering

I dette afsnit vil vi beskrive i dybden hvordan vi har programmeret den ekstra user-story.

Før implementeringen af denne user-story, havde vi allerede lavet et VarDataPlugin. Dette plugin er en implementering af XMLParsePlugin, som tager imod XML data fra et forbundet modul og laver det om til variabeldata. Den kigger efter XML elementet "var", og hvis det eksisterer laver den de medfølgende attributter; name, typ og value om til en simpel datastruktur kaldet ModuleVariable. Da man ved hjælp af dette plugin allerede kunne abonnere på alle modtagne modulvariabler manglede der implementeringsmæssigt kun at få samlet disse variabler i en træstruktur.

Følgende er den implementerede ModuleVariable klasse:

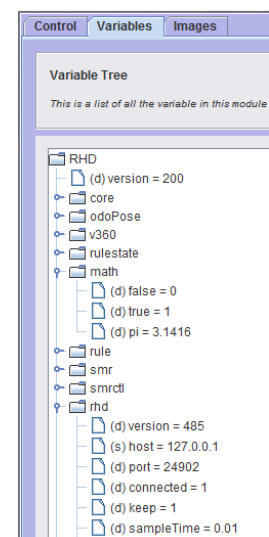
```
public class ModuleVariable {  
  
    private String varName;  
    private String varType;  
    private String value;  
  
    public ModuleVariable(String varName, String varType, String value) {  
        this.varName = varName;  
        this.varType = varType;  
        this.value = value;  
    }  
  
    public String getShortVarName() {  
        int offset = varName.lastIndexOf('.');  
        return varName.substring(offset+1);  
    }  
  
    public String toString() {  
        return "(" + varType + ") " + getShortVarName() + " = " + value;  
    }  
    ... Udeladt get/set metoder for hver variabel og equals for nemmere læsning  
}
```

Struktureringen af en træstruktur skulle laves på baggrund af "varName" attributten, som angiver variabelens fulde navn. Det fulde navn er en punktumopdelt tekststreng, der angiver hvor i træstrukturen variabelen ligger og til sidst hvad selve variabelens navn er. Følgende er et eksempel for bedre forståelse: "core.version", "cam.connected", eller endda "core.server.clients".

I f.eks. "core.server.clients", ville clients være variabelens navn. Variablen clients ville så ligge i mappen server, som selv ligger i mappen core.

Denne viden om træstrukturen brugte vi til at udvikle en klasse kaldet VariableTree, som kan tage imod en ModuleVariable og fylde den ind i en træstruktur på baggrund af dens varName attribut.

Vi valgte at bruge Javas egen træstruktur fra JTree til at forme en træstruktur. Hver node i et JTree har et userobject, hvis toString metode præsenterer noden. Derfor var det meget nemt blot at ændre ModuleVariable's toString metode, så data blev præsenteret som ønsket og



ligge dem ind som userobject i de yderste noder i træstrukturen. Se toString implementeringen i kodeuddrag ovenfor.

Resultatet af dette blev en flot og nemt læselig træstruktur.

3.18.5 Refaktorering

Vi havde i første omgang implementeret VariableTree klassen således at den nedarvede fra Javas egen JTree klasse og lavede om på dens indhold derfra. Vi fandt bagefter ud af at dette gjorde at VariableTree ikke var så nemt at placere i vores GUI. Derfor refaktorerede vi VariableTree klassen, så den blot tager imod et oprettet JTree og derefter "dekorerer" det med andet indhold. Dette gør at vi kan placere og designe et JTree i Netbeans GUI Builder og derefter blot kan dekorere det placerede JTree i den bagvedliggende kode.

```
public class VariableTree {

    private DefaultMutableTreeNode varTop;
    private JTree jTree;
    private String moduleName;

    public VariableTree(JTree jTree, String moduleName) {
        this.jTree = jTree;
        this.moduleName = moduleName;
    }

    public void updateVarTreeStructure(ModuleVariable var) {
        ... (udeladt kode)
        if (varTop == null) {
            varTop = new DefaultMutableTreeNode(moduleName);
            //varTop noden repræsenterer selve modulet
            jTree.setModel(new DefaultTreeModel(varTop));
        }
        ... (udeladt kode)
        if (foundExistingLeaf) {
            ModuleVariable moduleVar =
                (ModuleVariable) node.getUserObject();
            moduleVar.setValue(var.getValue());
        } else {
            DefaultMutableTreeNode leafNode = new
                DefaultMutableTreeNode(var);
            node.add(leafNode);
        }
    }
}
```

Dette uddrag fra koden viser i store træk hvordan et eksisterende JTree bliver givet til VariableTree klassen og derefter dekoreret med en ny træstruktur når nyt ModuleVariable data modtages.

For komplet kode se Kode Bilag.

3.18.6 Overvejelser til ekstra user-story

I XP går man som regel ikke ud over hvad der er forskrevet til den enkelte Release. Vores tanke var at det ikke krævede så meget at implementere denne ekstra user-story. Det gjorde det heller ikke, men hvis man skal følge XPs regler skal man udvikle de forskellige artefakter for en use story før og efter man har implementeret denne, hvilket også kræver tid. Det kom derfor til at tage længere tid end vi

havde regnet med. Derfor bør det overvejes mere grundigt hvis vi skal inddrage en ekstra user-story fremover.

3.19 Præsentation af Release 1 hos projektpartner

Udarbejdet af: Bjørn

Vi var på besøg hos DTU torsdag d. 24. september for at præsentere vores system og køre vores acceptance tests. Følgende er et kort referat af dagen.

"Vi startede mødet ud med at præsentere vores systems GUI. Vi viste de funktionaliteter som var blevet implementeret; Connection, RawData, Variabel træ. Projektpartner virkede tilfreds med GUI designet og især funktionaliteten med variabeltræet. De savner grafisk visning for nogle specifikke variabler, så som IRSensor og LineSensor, hvilket vi nok må overveje at prioritere højere i planlægningen af kommende iterationer.

Vi havde en del diskussion og feedback i forhold til plugin strukturen og hvordan denne skulle fungere. Der var nogle overvejelser omkring parsing af XML der indeholder binært data eller store mængder af data.

Efter mødet testede vi vores system på en SMR robot, som vi også kunne køre vores acceptance tests imod. Resultatet af vores acceptance tests var en succes. Vi testede en masse metodekald til MRC og RHD modulerne på SMR robotten, for at mappe interfacet af robotten og bedre kunne skrive acceptance tests i de næste iterationer.

Vi aftalte at mødes igen torsdag d. 8. oktober, hvor vi vil præsentere Release 2."

Vi fik lavet en del noter omkring detaljer til GUI og systemet bagved som vi kan bruge fremadrettet fra Release 2. Endvidere fik vi diskuteret ideer til en kommende Release 3. Se Bilag 10.1 "Release 1 d. 24/9/09".

3.19.1 Acceptance tests

Alle vores acceptance tests blev godkendt. Ved "Konverter XML" opstod der dog en lille fejl, men den kunne hurtigt rettes. Derfor blev det ikke en fejl der skulle tages med i overvejelser omkring udvikling af nye User-stories til næste Release.

Forbind til modul 1: Opret forbindelse til modul

Test Trin	Input/Handler	Forventet Output	Resultat
1	Klient får besked på at skulle forbinde til et modul	Besked fra Server om at forbindelsen er oprettet.	GODKENDT

Forbind til modul 2: Konverter XML

Test Trin	Input/Handler	Forventet Output	Resultat
1	Kommando sendes til et forbundet modul Klienten sender: "var core.version"	XML der er konverteret til Strings GUI RawData vinduet viser: <var name="core.version" value="x.xx"></var>	GODKENDT med en lille fejl: "xxx" i stedet for "x.xx"

Forbind til modul 3: Gem data

Test Trin	Input/Handler	Forventet Output	Resultat
1	Konverteret XML gemmes i intern hukommelse Klient sender: "var allcopy"	Konverteret XML er gemt i intern hukommelse Alle variabler bliver vist i RawData vinduet.	GODKENDT

Vis Variabel-træ 1

Test Trin	Input/Handler	Forventet Output	Resultat
1	Bruger klikker på variabel-træets faneblad i GUIen kaldet "Variables".	Variables faneblad i GUI viser alle variabler som en træstruktur. Om alle variabler er til stede kan tjekkes ved at se om den indeholder alt data der blev modtaget i RawData vinduet.	GODKENDT

3.19.2 Evaluering af Release 1 med projektpartner

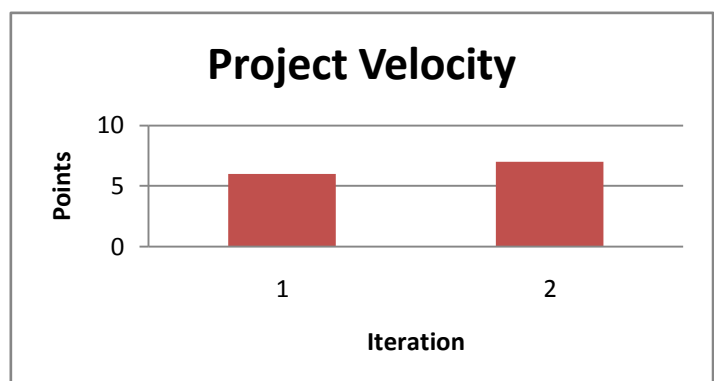
Der var tilfredshed med Release 1 hvilket betyder at vi kan gå i gang med Release 2 uden problemer. Projektpartnerens ønske om at kunne se en grafisk status på udvalgte variabler bør vi dog fremskynde til at være indeholdt i Release 2. Diskussioner og feedback omkring vores opbygning og struktur af koden er meget givende og belyser nogle måder at se tingene på som vi ikke selv ville gøre det. Feedback til GUI og den bagvedliggende kode betyder at vi har en del forslag til den kommende Release, hvilket er rigtig godt. Vi har også fået ideer nok til at kunne lave et Release 3, hvilket gør at vi kan planlægge bedre og nemmere kan prioritere indholdet i de enkelte iterationer.

Projektpartner har mange ideer og forslag, hvilket betyder at vi skal være gode til at få projektpartner til at prioritere de enkelte ideer og forslag. Vi mener at kommunikationen med projektpartner er god og foregår på et ligeværdigt plan.

3.20 Velocity Chart

Udarbejdet af: Kristina

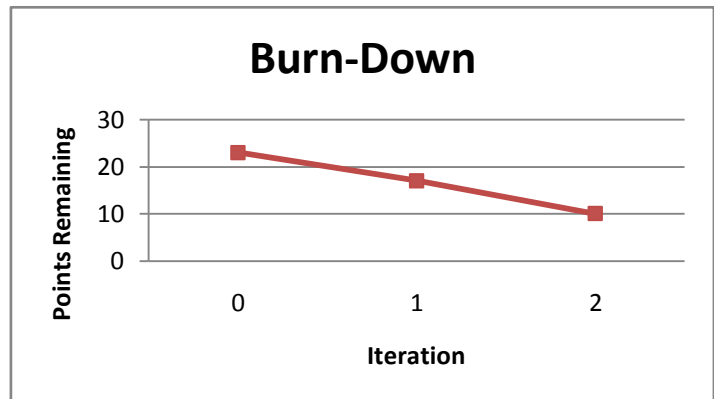
Følgende er vores Project Velocity chart efter første Release, hvilket indebar 2 iterationer. Over disse 2 iterationer fik vi i alt udarbejdet 13 estimerede point, i det vi også implementerede "Vis variabel-træ". Vi fik udviklet et point mere i iteration 2, men blev mere presset for tid. Derfor var den reelle hastighed nok den samme for de 2 iterationer.



3.21 Burn down chart

Udarbejdet af: Kristina

Dette er vores Project Burn-Down chart, som det ser ud efter første Release. Punkt 0 angiver antal point for hele projektet før vi startede. Første Release indebar 2 iterationer hvor der i alt blev lavet 13 estimerede point. Tilbage er nu 10 point af hele projektet. Dette virker ikke som så meget, men vi regner med at få flere krav fra projektpartner til det planlagte Release 3.



3.22 Evaluering af Release 1

Udarbejdet af: Bjørn, Daniel, Kristina

Release 1 gik rigtig godt. Der var en del der skulle udvikles, i og med at vi skulle lave Metafor, CRC cards, Design Klasse Diagrammer og Releaseplan for hele projektet. Disse skal vi ikke udvikle i de næste Releases, men det kan dog forekomme at vi skal lave nogle ændringer.

Vi har ikke selv fået det store udbytte af metaforen 'Flyveleder' hvilket skyldes at vi alle fra starten havde en god forståelse for projektet. Vi har aftalt at lave et review af projektrapporten med en anden projektgruppe og håber derfor at metaforen kan være en hjælp for dem til at opnå en hurtig forståelse af projektet.

I forhold til User-stories, vil vi på baggrund af projektpartners feedback omkring prioritering af udvikling, gennemse Releaseplan omkring hvilke User-stories der skal udvikles først og eventuelt også lave nye User-stories i begyndelsen af Release 2.

Spike Investigations hjalp os til bedre at estimere hvor lang tid implementeringen af user-stories ville tage. Det gav os også en god viden og bund for selve udviklingen af user-stories.

Brugen af de ekstra UML artefakter som normalt ikke er en del af XP, har mest hjulpet os til at få en intern forståelse i Release 1. De har været med til at give os et visuelt overblik over design og arkitektur af systemet. Selvom vi fra starten havde regnet med at skulle lave Sekvens Diagrammer først, blev det ikke nødvendigt i dette Release. Vi lavede derimod nogle af de vigtigste efter vi havde programmeret dem for at kunne holde et visuelt overblik over koden og sørge for fælles forståelse.

I forhold til Kvalitets Faktorer, hvor vi vægtede Usability højest har vi i dette Release brugt megen tid på at designe den grafiske brugergrænseflade. Fleksibilitet i form af modularitet som også var vægtet højt har vi fokuseret meget på i det overordnede design og på kodeniveau.

Vi har i vores risici til projektet beskrevet at der kunne forekomme problemer med manglende Unit tests. Denne risiko har vi i dette Release taget højde for ved at lave detaljerede acceptance tests. Der har endnu ikke været problemer med manglende viden til at opfylde de givne krav.

Vores Estimering af de forskellige User-stories og Task Cards til dette Release viste sig at holde forholdsvis godt. Vi havde dog ekstra tid og besluttede derfor at udvikle en user-story mere. Selve implementeringen tog ikke lang tid, men arbejdet med artefakter før og efter implementeringen som XP foreskriver gjorde at det samlede tidsforbrug blev større end forventet og at vi derfor måtte udskyde planlagt arbejde med projektrapporten. Vi må derfor erkende at vi ikke burde være gået i gang med den ekstra user-story 'Vis variabel træ'.

4. eXtreme Programming Release 2

4.1 Indledning

Udarbejdet af: Bjørn, Daniel, Kristina

Vi er nu klar til at gå i gang med Release 2. Der er dog nogle artefakter der skal ses på igen, da projektpartners prioritet omkring hvilke user-stories der skal udvikles først, har ændret sig. Derfor vil vi i dette Release se på vores nuværende user-stories og udvikle nye, samt lave en revideret Releaseplan.

Med hensyn til de nye user-stories kan det blive nødvendigt at lave nye spike investigations, for at kunne estimere hvor omfangsrige disse vil være.

Af UML artefakter vil vi igen inddrage Sekvens Diagrammer hvis det bliver nødvendigt, for at få overblik over hvordan de forskellige klasser i koden skal kommunikere med hinanden. Det kan også være det bliver nødvendigt at gå tilbage og revidere vores Design Klasse Diagram, for at få overblik over de nye klasse vi eventuelt har brug for, til de næste user-stories.

I dette Release har vi valgt at skrive overvejelser efter hvert artefakt hvor det er relevant. Disse overvejelser vil indgå i den samlede evaluering af Release 2.

Vi starter vores Release med at lave en Projekt Plan.

4.1.1 Projekt Plan

Vores Projekt Plan for Release 2 ligger i Bilag 1.3 "Projekt Planer, Release 2".

Overvejelser

Dette Gantt Chart har været godt at bruge, da det giver overblik over alle de forskellige artefakter der skal udvikles i dette Release. Det hjælper os til at estimere hvor meget vi skal nå pr. dag for at kunne blive klar til Release 2 på DTU d. 8-10-09.

4.2 User-stories

Udarbejdet af: Bjørn, Daniel, Kristina

På baggrund af feedback fra projektpartner ved Release 1 (Se bilag 10.1 "Release 1 d. 24/9/09/") har vi fået udarbejdet 7 nye user-stories. Projektpartner var meget interesseret i at vi udviklede "Vis grafer for SMR variabler" og "Juster SMR variabler" som det første i dette Release. Normalt ville man først ligge vægt på at opbygge systemets kerne og lave det modulært. Det har disse 2 user-stories ikke fokus på, men da det er af høj prioritet for projektpartner vil vi gå i gang med at udvikle dem. Efter udvikling af "Vis grafer for SMR variabler" og "Juster SMR variabler" vil vi fokusere på "Tilføj Modul" da denne er med til at opbygge kernen i systemet.

Projektpartner kom også med ideer til user-stories til Release 3. Men vi har valgt først at fokusere på at estimere dem i Release 3.

Vi havde 2 user-stories i forvejen som ikke var implementeret endnu: "Tilføj variabel til status" og "Tilføj Modul". Disse 2 har vi revideret så det passer med det antal point vi på nuværende tidspunkt estimerer dem til.

Vi har ikke haft brug for spikes til at estimere de nye user-stories.

Vi har her valgt at vise 1 af de user-stories vi vil udvikle i dette Release. Både de nye og reviderede user-stories findes i Bilag 2.2 "User-stories, Release 2".

Vis grafer for SMR variabler
Story Number: 6
Estimation: 4
Business value: High
Brugeren får præsenteret SMR robotens status ved hjælp af grafer. En graf for linie-sensor og en for den infrarød-sensor. Der vises også grafisk om hjulene på robotten kører.
Overvejelser:

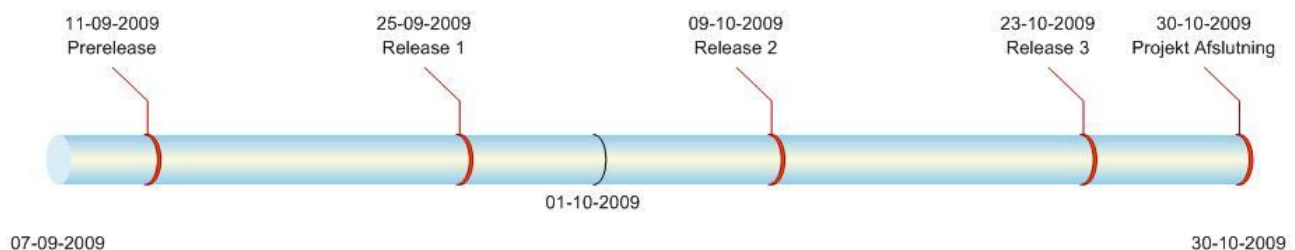
4.3 Revideret Releaseplan

Udarbejdet af: Bjørn

På baggrund af de nye user-stories og projektpartners prioritet til udviklingen har vi revideret vores Releaseplan. Da vi i forrige Release fik udviklet 2 pts mere end de estimerede 11 pts, har vi valgt at ligge user-stories ind i Release 2 til 13 pts. Resten har vi lagt i Release 3, så denne på nuværende tidspunkt har 8 pts. Der vil komme flere user-stories til Release 3, når vi får feedback på Release 2.

Vi har også valgt at have 1 iteration i hvert kommende Release i stedet for 2, da vi kun får feedback fra projektpartner ved Release. Uden feedback til hver iteration kan det være svært at gå tilbage og lave ændringer i den følgende iteration.

Herunder ses vores reviderede Releaseplan



Release 1 – User-stories: 13 pts Release Date: 25-09-09	Release 2 – User-stories: 13 pts Release Date: 09-10-09	Release 3 – User-stories: 8 Release Date: 23-10-09
Forbind til Modul (10 pts) Send kommando til Modul (1 pts) Vis variabeltræ (2 pts)	Vis grafer for SMR variabler (4 pts) Justér SMR variabler (2 pts) Tilføj Modul (4 pts) Ændre variabelværdi i variabeltræ (2 pts) Genbrug af sendte kommandoer (1 pts)	Kontrol knapper (3 pts) Tilføj variabel til status (2 pts) Modul Status Lampe (2 pts) Titel (1 pts)

4.4 Iterationsplan

Udarbejdet af: Kristina

Følgende er vores Iterationsplan for Release 2. Da vi har valgt kun at have en iteration i dette Release viser denne plan kun en iteration.

Release 2	Estimeret Tid/timer
Vis grafer for SMR variabler	5 t.
Justér SMR variabler	2 t.
Tilføj Modul	9 t.
Ændre Variabel værdi i variabeltræ	2 t.
Genbrug af sendte kommandoer	1 t.
Total:	19 t.

Overvejelser

Selvom vi har estimeret en del flere point til dette Release har vi dog kun udregnet det til 19 timers arbejde, i modsætning til de 24 timer i Release 1. Dette skyldes at der var mange tasks der skulle udføres til "Forbind til modul" som gjorde at det blev 23 timer kun til denne ene user-story i Release 1. I dette Release er der ikke så mange tasks til hver user-story, og dermed tager de ikke lige så lang tid.

4.5 Task Cards

Udarbejdet af: Daniel, Bjørn

Da vi i dette Release kun har en iteration blev det ikke nødvendigt at lave task cards før iterationsplanen.

Vi har valgt kun at vise 1 ud af de 9 task card til dette Release her. Alle task cards ligger i Bilag 3.2 "Task Cards, Release 2".

Lav et modul plugin

Task Card 10, Release 2, User-story: Vis grafer for SMR variabler
Date:
Task: Lav et modul plugin til SMR robotten
Tid: 2 t.
Kommentar:

Overvejelser

Vi har i dette Release haft user-stories der primært har med GUIen at gøre. Derfor er der ingen task cards til JUnit tests. Når vi har delt user-stories op Task Cards har vi som regel delt dem op i det at udvikle den bagvedliggende kode og præsenteringen i GUI.

4.6 Acceptance Tests

Udarbejdet af: Kristina, Daniel

Vi har lavet disse Acceptance test på baggrund af projektpartners krav til systemet. Efter endt implementering af user-stories til dette Release vil vi køre dem på DTU for at sikre os at de giver det forventede output.

Vi vil bruge acceptance test til at teste det nye SMR plugin og dens funktioner, tilføjelsen af et nyt modul, ændring af variabelværdi og genbrug af sendte kommandoer. I dette Release er vores tests meget fokuseret på brugergrænsefladen og brugerinteraktioner, og er forklarende i sig selv. Derfor har vi ikke yderligere forklaringer under hver test.

Vi har valgt at vise 1 af de 7 test til dette Release her. Alle acceptance tests kan ses i Bilag 5.2 "Acceptance Tests, Release 2".

Vis grafer for SMR variabler

Test Trin	Input/Handlering	Forventet Output	Resultat
1	Bruger klikker på SMR faneblad under et modul	I GUIen vises grafer for ir sensor og line sensor fra SMR robotten.	

4.7 Sekvens Diagrammer

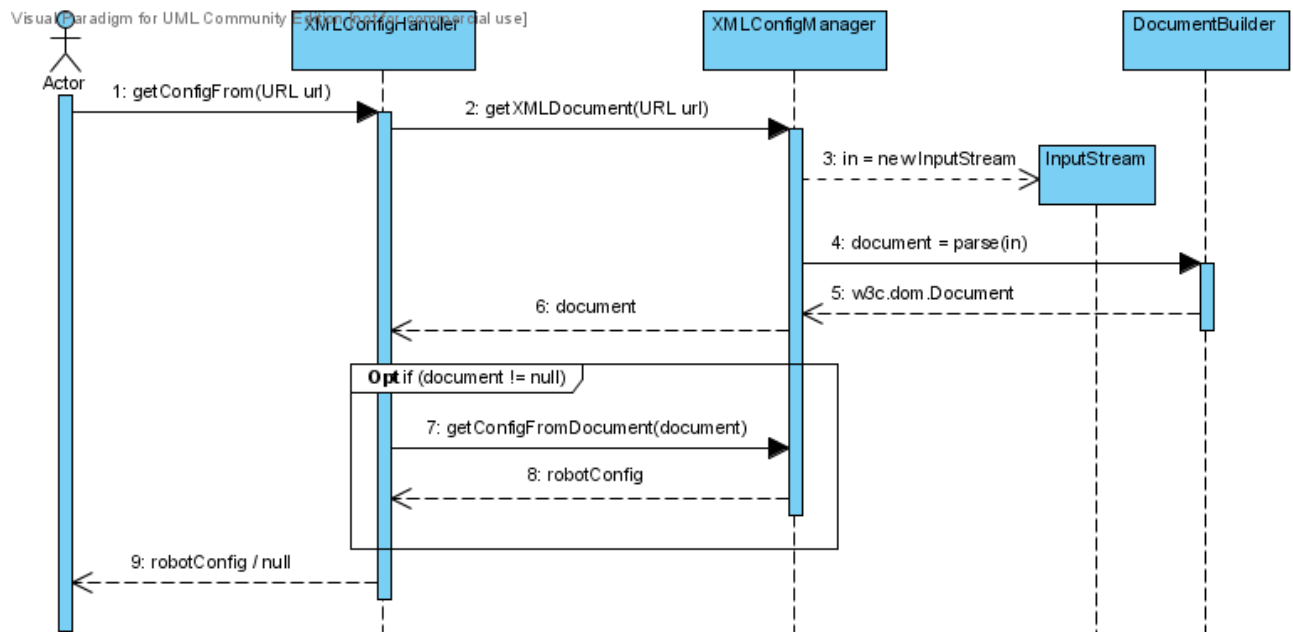
Udarbejdet af: Daniel, Bjørn

I dette Release har vi valgt at udvikle et Sekvens Diagram før vi programmerede til et af vores Task Cards.

Dette valgte vi fordi "Tilføj modul ved konfigurerings", umiddelbart virkede som en ret kompleks Task og derfor ville vi gerne designe den overordnede interaktion først. Denne task går ud på at der ved start af systemet skal indlæses en XML fil fra en given adresse og at der ud fra denne XML fil skal læses information for hvilke moduler der skal startes med det samme.

Vores beskrivelse af dette SD er ikke lige så detaljeret som vores SD i det foregående Release, da vi mente det i sig selv var nemt at forstå.

For større SD se Bilag 8.2 "Sekvens Diagrammer, Release 2".



Beskrivelse af Sekvens Diagram

Vi valgte at der kun skal kaldes en enkelt metode i XMLConfigHandler. Handleren skal derefter stå for alle videre kald til XMLConfigManager og til sidst returnere det læste configurations information, som er blevet læst ud fra den givne URL.

2. XMLConfigHandler beder først XMLConfigManager om at hente og omforme et XML dokument fra den givne URL. Hvis der ikke findes noget dokument på den modtagne URL eller læsning af filen fejler, så returneres der null i stedet for et Document objekt.
7. Handleren tjekker dette, og hvis det returnerede document ikke er null beder den XMLConfigManager om at læse configurations data ud fra det givne Document objekt. Denne læste data returneres.

Overvejelser

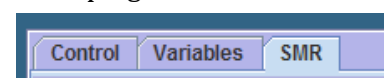
Vi har i dette SD brugt betegnelsen "robotConfig" for det data der sendes tilbage til GUIen, da vi ikke endnu er sikre på selve datastrukturen for de indlæste moduler. Vi har overvejet at bruge en arraylist af Module klasser der indeholder den nødvendige information for hvert modul, evt. pakket i et Robot objekt, men har endnu ikke taget en beslutning om dette.

4.8 Programmering

Udarbejdet af: Daniel

I dette afsnit vil vi forklare i dybden hvordan vi har udviklet de forskellige user-stories til dette Release.

Vores user-stories "Vis grafer for SMR variabler" og "Juster SMR variabler" er begge del af et samlet plugin, som vi er blevet bedt af projektpartner om at implementere. Dette plugin skal laves til én specifik slags af DTUs robotter, kaldet Small Mobile Robot (**SMR**). Pluginet skal konstrueres således at det kan tilføjes som et ekstra faneblad til hvilket som helst modul.



Billede: SMR som et ekstra tab

ModulePlugin

For at implementere den første user-story, "Vis grafer for SMR variabler", var der derfor et par andre ting i forhold til Plugin-strukturen som skulle på plads først. Det var på dette tidspunkt allerede muligt at lave ens egne plugins til læsning af XML, men det skulle nu også være muligt at lave et plugin som kan tolke data der allerede er blevet læst fra XML og vise dette på en ønsket måde. Til dette formål lavede vi et interface kaldet `ModulePlugin`. `ModulePlugin` skulle fastsætte en skabelon for alle plugins, som skulle kunne tilføjes til et modul. Hvert enkelt modul skulle så indeholde plugins, der ville blive tilføjet som `JPanel`s til det enkelte moduls `JTabbedPane`.

Dette er `ModulePlugin` interfacet:

```
public interface ModulePlugin {  
  
    public void setXMLClientHandler(XMLClientHandler handler);  
    public String getPluginName();  
    public JPanel getJPanel();  
    public void startPlugin();  
    public void closePlugin();  
    public void doUpdate();  
}
```

Metoden `setXMLClientHandler()` bruges til at tilbyde den handler som selve modulet bruger til at kommunikere med et modul på robotten. Dette gøres så det tilføjede plugin har mulighed for at sende kommandoer og tilføje nye `XMLParsePlugin`. `ModulePlugin` foreskriver `startPlugin()` og `closePlugin()` metoder, så et plugin kan vente med at initialisere ressourcer til det er nødvendigt og lukke dem når de ikke længere er nødvendige. Der er også foreskrevet en `doUpdate()` metode, der er tiltænkt at skulle kaldes med et fast interval fra modulet, så hvert eneste plugin ikke selv behøver at have en tråd kørende til at klare regelmæssige opdateringer.

Selve den grafiske repræsentation af et plugin skal som sagt bestå af et `JPanel`. Vi har derfor lavet en metode kaldet `getJPanel()` i `ModulePlugin` som blot skal returnere et `JPanel` objekt. Dette objekt vil så blive tilføjet til modulets `JTabbedPane` med det navn som returneres fra `getPluginName()` metoden. Ved at bruge dette simple design kan et implementeret plugin returnere en instans af en anden `JPanel` klasse, eller selv være en `JPanel` og returnere en reference til sig selv. Dette gør at man kan lave plugins på en meget modulær måde med høj binding, ved at have læsning af XML, behandling af data og GUI for sig. Men samtidig er muligheden der for at man kan samle det hele i én klasse.

Overvejelser

At samle det hele i én klasse kan være en fordel hvis studerende fra DTU skal lave deres eget plugin og ikke kender så meget til Java. Hvis de f.eks. skal lave en ny måde at læse data på og præsentere denne igennem et plugin, kan de blot lave en ny klasse der extender `JPanel` og implementerer `XMLParsePlugin` og `ModulePlugin` på samme tid. Hvis dette laves i en IDE som Netbeans, vil man i den samme klasse både have adgang til at implementere de 2 interfaces i source, og grafisk at designe visningen af ens data.

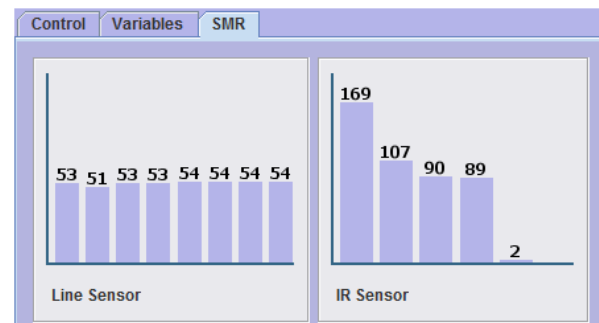
SMRPlugin

SMRPlugin skulle være vores første implementering af ModulePlugin. Dens mål er at kunne vise en graf der repræsenterer data for SMR robotens liniesensor, dens infrarøde sensor og vise visuelt om hvert baghjul kører. Dette omfatter Task Card 10: "Lav et modul faneblad plugin til SMR roboten". Data for SMR robotens liniesensor og infrarøde sensor kan modtages ved hjælp af det allerede udviklede XMLParsePlugin: VarDataPlugin. Derfor skal vores plugin blot abonnere på variabel-data og så tjekke om variabelen hedder "rhd.irsensor" eller "rhd.linesensor".

Det modtagne data for de 2 sensorer, skal dernæst præsenteres i hver sin søjlegraf. Vi kunne ikke finde nogen måde at bruge Javas egne klasser til at tegne grafer på og vi ville helst ikke bruge andre libraries, da vores system skal køres i en applet og ikke fylde for meget. Derfor valgte vi at lave vores egen klasse til at tegne en søjlegraf ud fra et array af heltal.

Denne kaldte vi LineGraph. Den blev implementeret ved at bygge videre på et JPanel og overskrive dens

paintComponent metode til at tegne en søjlegraf ud fra data der sættes med en setData metode. Denne blev lavet fleksibel nok til at den kunne bruges til at repræsentere data for begge sensorer.



Billede: Implementeret SMR plugin

Tilføj Modul

En vigtig user-story i dette Release var "Tilføj Modul". Den var primært udtænkt ud fra den ønskede funktionalitet; at kunne trykke på en knap i GUIen, udfylde en pop-up boks med informationer for et nyt modul, som derefter bliver tilføjet til GUIen. Det nye modul skulle repræsenteres af en knap i venstre side af GUIen og ved aktivering af denne knap skulle det valgte modul vise sit indhold i den højre side af GUIen. Arbejdet med at implementere første task card til denne user-story (Task Card 13 "Gør det muligt fra GUI at tilføje et modul"), lå primært i små ændringer i MargGUI klassen og ændring af konstruktørerne i ModuleTabs klassen, så der kan gives et navn, host og port ved oprettelse af et ModuleTabs.

```
public ModuleTabs(String name, String host, int port, boolean autoConnect) {  
    ...  
}
```

Som ses ovenfor er det også muligt at angive om der skal forsøges at forbinde til modulet med det samme.

I MargGUI klassen, som er selve grundklassen for vores applet, blev den højre del af GUIen ændret så den bruger et CardLayout til at organisere sit indhold. CardLayout skal ses som en stak af kort, hvor kun det øverste kort er synligt. Dette gør at der kan lægges flere ModuleTabs oven i hinanden i højre side, men kun et enkelt vil være synligt af gangen. Hvilket der er synligt styres af knapperne for hvert modul i venstre side af GUIen.

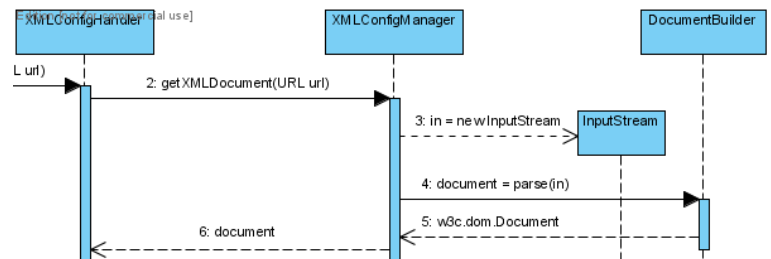
Der blev til denne task lavet en knap i venstre side af MargGUI menuen, "Add Module". Når bruger trykker på denne, vises der et pop-up vindue hvor man kan udfylde Navn, Host, Port og om der skal forbindes til modulet med det samme. Dette pop-up vindue kan ses i brugergrænseflade afsnittet.

Hvis pop-up vinduet bekræftes vil vores GUI herefter tilføje et nyt modul med de indtastede informationer. En knap vil også blive tilføjet i menuen i venstre side af GUI'en.

```
if (dialog.wasApproved()) {
    String moduleName = dialog.getModuleName();
    String host = dialog.getHost();
    int port = dialog.getPort();
    boolean autoConnect = dialog.getAutoConnect();
    if (!moduleName.equals("")) {
        RobotModule module = new ModuleTabs(moduleName, host, port,
        autoConnect);
        this.addModuleButton(module);
    } else {
        JOptionPane.showMessageDialog(mainGUI, "Cannot create a module
        without a name");
    }
}
```

Tilføj XML konfiguration af modul

Vores user-story "Tilføj Modul" bestod også af Task Card 14: "Tilføj XML konfiguration af modul". Denne indebærer muligheden for at konfigurere hvilke moduler der skal tilføjes til vores Applet fra start, via en XML fil. Til denne task havde vi allerede designet hvilke klasser der var behov for og hvordan de skulle interagere. Vi implementerede i store træk vores klasser efter vores design. Den eneste udfordring vi stødte på var forskellen på at skulle åbne en XML fil, når programmet køres lokalt og når det køres over et netværk som et applet. Dette løste vi med en smule kode som kan detektere om applet køres lokalt eller over netværket og rette URL hvis det køres lokalt.



Vi har valgt at XML konfiguration skal hentes fra filen *robot.xml* som skal findes ved siden af den html fil der kører vores Applet. Da det ikke altid er den samme adresse vores Applet køres fra, har vi som udgangspunkt brug for at vide hvor vores Applet bliver kørt. Til dette brugte vi metoden `getCodeBase()` der returnerer en URL til hvor ens applet eksekveres fra. Følgende er vores kode i `MargGUI` klassen der fixer URL, henter og læser XML konfigurationsfilen og til sidst kalder `addLoadedModules` på de returnerede `Module` objekter:

```
private void autoLoadModules() {
    XMLConfigHandler configHandler = new XMLConfigHandler();
    java.net.URL codebaseURL = MargGUI.this.getCodeBase();
    if (XMLConfigHandler.isLocalURL(codebaseURL)) {
        codebaseURL = XMLConfigHandler.fixLocalUrl(codebaseURL);
    }
    URL configFile = XMLConfigHandler.getFileURL(codebaseURL,
        ROBOT_CONFIG_FILENAME); //Points to String "robot.xml"
    List<Module> loadedModules = configHandler.getConfigFrom(configFile);
    addLoadedModules(loadedModules);
}
```

XMLConfigHandler er en mellemmand for XMLConfigManager og dens brugere. Selve XMLConfigManager blev implementeret således at hvis der går noget galt undervejs returnerer den blot en tom liste af moduler.

XMLConfigManager bruger primært metoden `getElementsByTagName()` til at gennemgå det hentede XML dokument og finde den ønskede information om hvert modul.

Dette er et eksempel på en *robot.xml* fil der definerer 2 moduler:

```
<robot>
  <module>
    <name>SMR RHD</name>
    <host>10.0.2.2</host>
    <port>24929</port>
    <plugin>SMRPlugin</plugin>
    <plugin>NoneExistantPlugin</plugin>
    <autoConnect>true</autoConnect>
  </module>
  <module>
    <name>Empty</name>
    <host></host>
    <port></port>
  </module>
</robot>
```

Læsningen af *robot.xml* er gjort meget fejltolerant, så der kun minimum skal være et udfyldt "name" felt. Resten kan undlades og udfyldes når applet kører. AutoConnect, der indikerer om der skal oprettes forbindelse til modulet med det samme er som standard sat til *false* hvis andet ikke er angivet.

Efter at have implementeret læsningen af XML konfiguration, kunne vi se at vi havde behov for at finde ud af hvilke implementeringer af ModulePlugin der er til rådighed ved systemets start. Når der f.eks. angives i XML at SMRPlugin skal tilføjes til et modul, så skal vi i koden kunne tjekke om der rent faktisk findes en klasse der hedder SMRPlugin der implementerer ModulePlugin interfacet. Denne funktionalitet er både nødvendig for Task Card 14: "Tilføj XML konfiguration af modul" og Task Card 15: "Tilføj plugin til modul", da den sidste også skal kunne detektere hvilke ModulePlugin implementeringer der er til rådighed, så bruger kan vælge en og derefter få den tilføjet til det aktive modul.

Til denne funktionalitet valgte vi at lave en utility klasse kaldet PluginManager. Denne har primært til opgave at opdage hvilke ModulePlugin implementeringer der findes i systemet. Til dette formål lavede vi en metode der tager imod et navn på en pakke i systemet og en Class der angiver interfacet som klasserne skal implementere. Denne metode returnerer så en liste af klasser i den givne pakke der implementerer det givne interface. Metoden i sig selv er rimeligt kompliceret, men for at give en bedre forståelse af hvordan den fungerer er her metodens signatur:

```
private static List<Class> getClassesOfInterface(String thePackage, Class theInterface);
```

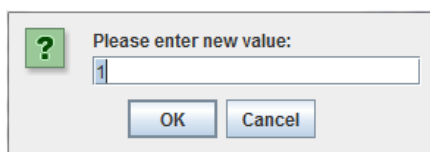
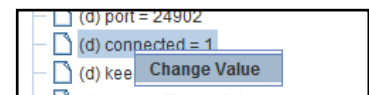
Denne metode brugte vi i PluginManager til fra start at initialisere en List indeholdende en reference til hvert ModulePlugin der findes i pakken *marg.gui.plugin*.

```
public class PluginManager {  
  
    List<Class> modulePluginClasses = new ArrayList<Class>();  
  
    public PluginManager() {  
        initModulePlugins();  
    }  
  
    private void initModulePlugins() {  
        modulePluginClasses = getClassessOfInterface("marg.gui.plugin",  
                                                    ModulePlugin.class);  
    }  
  
    public List<Class> getAvailableModulePlugins() {  
        return modulePluginClasses;  
    }  
  
    ...  
}
```

Med denne klasse kan vores applet på kørselstidspunktet finde ud af hvilke ModulePlugin der er til rådighed og tilføje dem til et modul hvis det er konfigureret i robot.xml eller tilføjet af brugeren. Hvis projektpartner og andre brugere senere vil udvikle og tilføje deres eget plugin, lad os kalde det "NytPlugin", skal de blot placere det i pakken `marg.gui.plugin` og angive `<plugin>NytPlugin</plugin>` under et modul i robot.xml filen. På denne måde vil vores system helt automatisk kunne opdage nye plugins og tilføje dem på kørselstidspunktet. Denne funktionalitet kunne også bruges til automatisk at finde XMLParsePlugins, men da XP foreskriver at man ikke bør lave mere end hvad man har planlagt, vil vi på nuværende tidspunkt ikke implementere dette.

Ændre variabelværdi

Der blev i dette Release også implementeret muligheden for at ændre værdien for en variabel i variabeltræet, som alle moduler har. Dette blev gjort ved at lave en lille menu ved højreklik af en variabel hvor brugeren kan vælge "Change Value". Hvis brugeren vælger dette kommer der et pop-up vindue



frem hvor brugeren kan indtaste en ny værdi. Hvis brugeren trykker "OK" i pop-up vinduet, bliver den nye værdi automatisk sendt til modulet og variabeltræet opdateret.

Det blev også i dette Release gjort muligt for brugeren at genbruge sendte kommandoer i "Control" fanebladet i hvert modul.

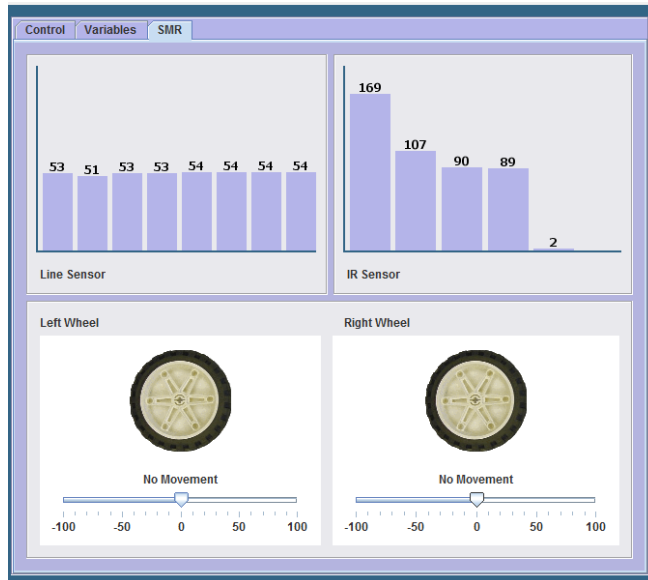
4. 9 Brugergrænseflade

Udarbejdet af: Kristina

4.9.1 Opdateret brugergrænseflade

Da projektpartner var godt tilfreds med vores brugergrænseflade har vi valgt at arbejde videre på den prototype vi lavede i Release 1.

Det først billede herunder viser vores implementering af SMR faneblad der viser grafer for Linie- og IR- sensorene, og højre og venstre baghjul på robotten.



The dialog box has two sections. The first section is titled 'Module Name' and contains a text input field with the placeholder text 'Write the name of the new module. Keep it short'. The second section is titled 'Connection' and contains a text input field with the placeholder text 'Write the host and port nr of the module you wish to connect to.'. Below this are two input fields labeled 'Host:' and 'Port:'. At the bottom of the dialog box are two buttons: 'Cancel' and 'Add Module'. There is also a checkbox labeled 'Connect Now:' which is checked.

Derudover har vi gjort det muligt at tilføje et modul, ved et pop-up vindue, som set ovenfor:

Der indtastes navnet på modulet og hvilken host og port man vil lave en forbindelse til. Og så sætter man hak ved om man vil oprette forbindelse til modulet med det samme.

Overvejelser

Dette er måden hvorpå man kan forbinde til et nyt modul når man allerede er inde i GUI'en. Man kan også vælge at konfigurere i XML filen at en forbindelse til et modul skal oprettes hver gang man starter GUI'en. Dermed behøver man ikke manuelt at tilføje moduler hver gang systemet startes.

4.9.2 Beskrivelse af brugergrænsefladetest

Da kvalitetsfaktoren brugervenlighed er vægtet højt for vores system, har vi valgt at inddrage test af brugergrænsefladen. Som beskrevet i Prerelease har vi udviklet nogle think-aloud test, hvor vi vil klassificere de enkelte problemer der måske opstår når disse udføres. Efter en bruger har udført en think-aloud test, vil vi give dem et spørgeskema hvor de kan evaluere ud fra brugervenlighedsfaktorerne. Vi vil også tage tid på hvor lang tid det tager at udføre en handling. Se test og spørgeskema i Bilag 9 "Brugergrænsefladetest".

Vi vil afprøve disse tests med projektpartner, andre fra vores klasse og med den gruppe vi laver Review med i midten af projektet. Resultatet vil vi beskrive i afsnittet om Præsentation af Release 2.

4.10 Opdaterede Task Cards

Udarbejdet af: Bjørn, Daniel

Følgende er vores opdaterede task cards. Vi har på hvert kort indskrevet kommentarer omkring hvordan vi har fået udviklet de enkelte tasks.

Da vi manglede information fra projektpartner omkring hvilke variabler der bruges til hjulene på en robot, kunne vi ikke implementere Task 12 "Juster SMR variabler" og har derfor flyttet denne til Release 3. Derudover har vi i dette Release prioriteret at bruge en dag på at lave et Review med en

anden gruppe (se afsnit 5. Review), og har derfor også overflyttet task 15 "Tilføj plugin til modul" til Release 3.

Task Card 10, Release 2, User-story: <u>Vis grafer for SMR variabler</u>
Date: 2-10-09
Task: Lav et modul plugin til SMR robotten
Tid: 2 t.
Kommentar: <ul style="list-style-type: none">- Vi lavede et interface ModulePlugin som SMRPlugin implementerede.- Alle modul plugins skal have metoderne start og stop plugin, så de ikke initialiserer data før de skal bruges.- ModuleTabs har en ArrayList modplugin, der indeholder mange ModulePlugin.

Task Card 11, Release 2, User-story: <u>Vis grafer for SMR variabler</u>
Date: 2-10-09
Task: Tegn GUI til SMR plugin
Tid: 3 t.
Kommentar: <ul style="list-style-type: none">- Vi har lavet klasserne LineGraph og RobotWheel hvor vi med kode tegnede de nødvendige GUI elementer til en bar-graf og et hjul der kan kører rundt- De to klasser har høj binding så de kun fokuserer på at tegne det de bliver bedt om. De har ingen forbindelser til andre klasser.

Task Card 13, Release 2, User-story: <u>Tilføj modul</u>
Date: 5-10-09
Task: Gør det muligt fra GUI at tilføje et modul
Tid: 1 t.
Kommentar: <ul style="list-style-type: none">- Lavede vores egen dialog-boks hvor navn, host og port på et modul kan indtastes og derefter kan vinduet bekræftes eller annulleres.- Dialogboksen er en subklasse til Javas egen JDialog.- Ændrede ModuleTabs klassen så den kan instantieres med navn, host og port.- Gjorde det muligt for ModuleTabs automatisk at forbinde til et givent modul ved opstart

Task Card 14, Release 2, User-story: <u>Tilføj modul</u>
Date: 6-10-09
Task: Tilføj XML konfiguration af moduler
Tid: 2 t.
Kommentar: <ul style="list-style-type: none">- Vi lavede en XMLConfigManager klasse som står for selve arbejdet og en XMLConfigHandler der agerer som mellemmand imellem klienter og ConfigManager.- Manager står for at hente selve XML filen, læse og omforme den til en ønsket datastruktur.- Som ønsket datastruktur lavede vi en simpel klasse inde i XMLConfigManager kaldet Module, der indeholder navn, host, port, autoconnect og en liste af navne på plugins.- Vi valgte at konfig skulle hentes fra robot.xml som skal ligge ved siden af applet- Der var problemer med at lokalisere robot.xml når projektet kørtes lokalt og skulle testes, derfor lavede vi et par metoder i XMLConfigHandler der detekterer om en URL referer til en lokal fil, og hvis den gør, tilretter den.

Task Card 16 , Release 2, User-story: Tilføj modul
Date: 5-10-09
Task: Gør det muligt at have forbindelse til flere moduler
Tid: 3 t.
Kommentar: <ul style="list-style-type: none">- Vi har lavet et Interface RobotModule, som en afkobling mellem MainGUI og dens moduler, så den kan tilgå de enkelte på en generel måde.- Vi brugte Javas JPanel Cardlayout til at have alle moduler på et panel, så der kan skiftes i mellem hvilke der bliver vist.- Til modulmenuen hvor modul knapperne bliver tilføjet brugte vi Javas Grid Bag layout for at knapperne nemt kunne tilføjes løbende og ligges i samme kolonne.- Vi lagde panelet med knapperne for hvert modul inde i et scrollpane, så der er understøttelse for mange moduler

Task Card 17, Release 2, User-story: Ændre variabelværdi i variabeltræ
Date: 2-10-09
Task: Gør det muligt at ændre variabelværdi i variabeltræ
Tid: 2 t.
Kommentar: <ul style="list-style-type: none">- Brugte Javas egne GUI objekter til at lave pop-up vindue, hvor den nye værdi kan indtastes- Det at ændre værdien på en variabel er i sin egen metode, så den kan senere bruges til genvejstaster, som fx F2.

Task Card 18, Release 2, User-story: Genbrug af sendte kommandoer
Date: 5-10-09
Task: Gør det muligt at genbruge sendte kommandoer.
Tid: 1
Kommentar: <ul style="list-style-type: none">- Vi brugte JComboBox til at opbevare tidligere sendte kommandoer.

Overvejelser

I XP er det normalt af højeste værdi at få udviklet alle tasks til et bestemt Release, men da vi også laver en hovedrapport sammen med vores XP projekt, har vi alligevel vægtet det højere at lave et Review af rapporten, og dermed overflyttet nogle tasks til næste Release.

4.11 Testning

Udarbejdet af: Kristina

Vi havde i dette Release ikke planlagt nogen JUnit test, da vores User-stories primært handlede om GUI orienteret funktionalitet som ikke er så nemt at teste med en automatiseret JUnit test. I stedet har vi igennem dette Release draget flittig brug af interne Main metoder i hver klasse, der har til formål at teste klassen i et lokalt omfang. På denne måde kunne vi som udviklere hurtigt og nemt teste den enkelte klasses funktionalitet uden nødvendigvis at skulle integrere den med resten af systemet først.

Dette bryder lidt med XP udviklingsmetoden, men vi mener at vores klasser er blevet testet godt imens de blev udviklet og at vi har forsøgt at teste vores klasser på andre måder. Dette omfatter primært

Acceptance Tests og Brugergrænsefladetests, som har været gode værktøjer til at vurdere om vores system rent faktisk opfylder vores og projektpartners krav.

Vi har efter endt implementering af dette Release dog fundet ud af at et par af de udviklede klasser nok godt kunne have haft gavn af en JUnit test før vi var gået i gang med implementeringen. Dette gælder blandt andet for en klasse så som PluginManager, der i sig selv ikke var del af en specifik user-story. Det viste sig dog at der var behov for denne, som udviklingen af andre User-stories skred frem. Vi mener på nuværende tidspunkt ikke at det er nødvendigt at bruge tid i Release 3 på at gå tilbage og lave en JUnit test til denne klasse, da den allerede har gennemgået en række andre tests og de User-stories der bruger den har også gennemført deres Acceptance Tests.

4.12 Refaktorering

Udarbejdet af: Daniel, Kristina

I dette Release har der ligesom i sidste ikke været så meget refaktorering. Et godt design og en gennemdiskuteret arkitektur har gjort at de fleste User-stories ved endt implementering allerede har besiddet en høj grad af fleksibilitet og genbrugelighed. To kvalitetsfaktorer som vi fra start har vægtet rigtigt højt.

Vi har igen i dette Release adskillige steder, brugt designet med at skyde en Handler klasse imellem GUI og Model kode. Dette kan f.eks. ses i vores implementering af XMLConfigManager, hvor en XMLConfigHandler blev indført imellem Manager og de klasser der bruger den. Handler klassen sørger for lav kobling til Manager og indeholder enkelte utility metoder der simplificerer interaktionen med Manager.

Til Task Card 17: "Gør det muligt at ændre variabelværdi i variabeltræ", gik vi ind og lavede en refaktorering så GUI klassen ModuleTabs ikke selv håndterer det at sende kommandoer. I stedet lavede vi i XMLClientHandler klassen en metode kaldet `setVariable(varName, varValue)`; som ModuleTabs kan bruge når den skal sætte værdien for en variabel. Dette simplificerede og afkoblede interaktionen imellem GUI og XMLClient.

En lille ting som der blev gjort sideløbende med implementeringen af andre User-stories i dette Release, er brugen af en Log utility klasse. Denne klasse har som eneste formål at tilbyde en GlobalLogger, dvs. et Logger objekt der gælder for hele systemet. Denne bruges forskellige steder i vores system til at udskrive handlinger af forskellig vigtighed til konsollen. Dette har givet os en bedre måde at overvåge hvad der sker bag systemet når det kører og finde frem til fejl, skulle vi have behov for det. Før denne refaktorering, skete logging med normale udprint til konsollen, men der var alt for mange af dem, hvilket gjorde det uoverskueligt at følge med i.

4.13 Præsentation af Release 2 hos Projektpartner

Udarbejdet af: Kristina

Vi var på besøg hos DTU torsdag d. 8. oktober for at præsentere vores system til Release 2 og køre vores acceptance tests. Følgende er et kort referat af dagen.

Vi startede med at præsentere den opdaterede system GUI og den implementerede funktionalitet 'Vis grafer'. Projektpartner var meget tilfreds med 'Vis grafer for SMR

variabler', for det grafiske betyder at systemet vil blive brugt. Vi havde ikke nået at implementere 'Juster SMR variabler' som har betydning for hjulenes værdier mht. hastighed. Vi havde også været lidt i tvivl omkring kravet til denne user-story. Projektpartner oplyste de detaljer der var nødvendige for at fastsætte kravet til user-story 'Juster SMR variabler'. Den videre præsentation af de implementerede funktionaliteter foregik ved at vi fik projektpartner til at udføre en think-aloud test. På den måde fik vi lavet en test af brugergrænsefladen og samtidig vist resten af de implementerede user-stories. Efterfølgende gennemgik vi noget af den implementerede kode. Projektpartner have ingen indvendinger til koden. Projektpartner var generelt tilfreds med Release 2. Vi aftalte at mødes igen torsdag d. 22. oktober, hvor vi har Release 3.

Efter præsentationen etablerede vi forbindelse til en SMR robot og fik afprøvet 'Vis grafer for SMR variabler'. Det viste sig at vi skal have et loft for grafværdier da det ikke var muligt at aflæse udsving tydeligt nok. Vi uploadede vores GUI til en SMR robot og tilgik efterfølgende SMR robotten. Det virkede efter hensigten og GUI startede automatisk. Ved tjek med tilgang fra andre computere på DTU viste det sig desværre at det ikke kunne lade sig gøre, måske pga. at Java version var forældet på de afprøvede computere. Projektpartner ville kontakte systemadministrator for en afklaring.

Vores system viste en alvorlig fejl under forbindelsen til SMR robot som fik systemet til at gå ned. Heldigvis fik vi lokaliseret fejlen som værende et overflow af data i raw data vinduet. Løsningen vil være at tilføje løbende sletning af data i raw data vinduet.

Vi fik endvidere lavet nogle noter omkring projektpartners ønsker til kommende implementeringer i release 3. Se bilag 10.2 "Release 2 d. 8/10/09".

4.13.1 Acceptance tests

Vores acceptance test blev alle godkendt. Dog kunne vi ikke køre 3 tests da de ikke blev implementeret i dette Release 2, men er overført til release 3. Alle tests ligger i Bilag 7.1 "Kørte Acceptance Tests, Release 2".

4.13.2 Evaluering af Release 2 med projektpartner

Projektpartner var tilfreds med Release 2 og synes det var godt vi havde prioriteret at lave "Vis grafer for SMR variabler".

Det var godt at se at vi kunne få Java-applet frem ved at tilgå SMR robot. Mindre godt var det at de Java versioner som er installeret på DTU's computere er gamle versioner – det lykkedes i hvert fald ikke at starte Java-applet ved at tilgå SMR robot. Vi håber derfor at DTU vil få opdateret deres Java version snarest.

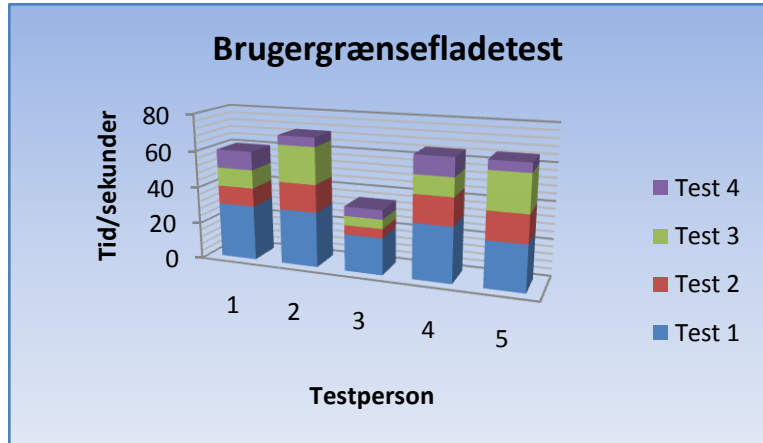
Det var heller ikke så godt at se systemet gik ned. Til gengæld var det rigtig godt at vi fik lokaliseret fejlen med det samme til at være en overflow i raw data vinduet og allerede ved hvordan en løsning kan implementeres.

Vi fik projektpartner til at prioritere kommende og ekstra implementeringer i Release 3, hvilket giver os et godt udgangspunkt for at revidere vores Releaseplan til Release 3.

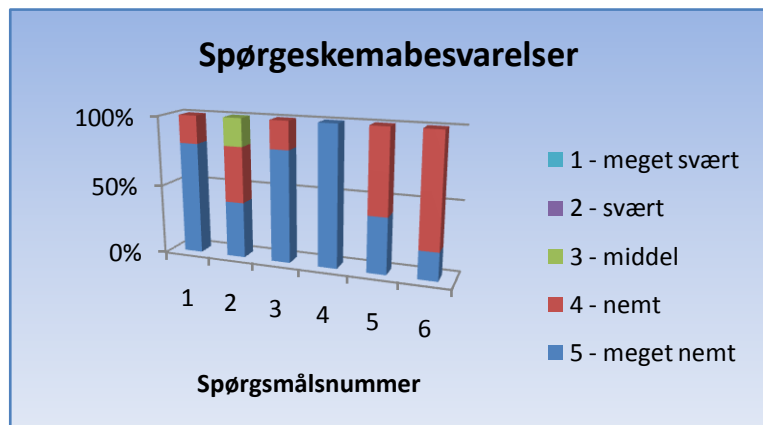
4.13.3 Resultat af brugergrænsefladetest

Vi fik lavet nogle think-aloud test vedrørende vores brugergrænseflade og på den baggrund kan vi konkludere at alle testpersoner kunne løse de 4 stillede opgaver uden nævneværdig hjælp.

Brugergrænsefladetesten viser at alle testpersoner bruger lidt længere tid på test 1 end på de resterende, hvilket nok kan skyldes at det er første kontakt til systemet og at der måske lige skal dannes et overblik af hele brugergrænsefladen. Test 2 kræver lidt søgen efter 'SMR grafer' og her kan ordet faneblad hjælpe de pågældende. Der skal indtastes en kommando i test 3 som kan være med til at forklare forskellen i anvendt tid. Generelt bruger testpersonerne mindre tid på test 4 hvilket kan skyldes en allerede opnået fortrolighed med brugergrænsefladen. Testperson 3 bruger halv så lang tid på testene end de andre testpersoner hvilket kan forklares med at det er vores projektpartner.



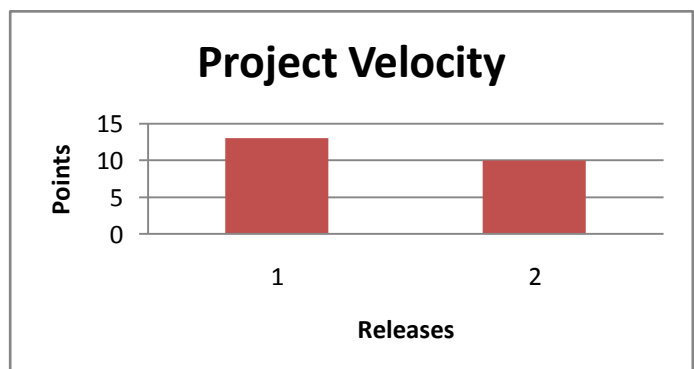
Endvidere vurderede testpersonerne vores brugergrænseflade til overvejende at ligge i kategorierne '5 - meget nemt' og '4 - nemt'. Det foregik via et spørgeskema omhandlende brugervenlighedsfaktorer som testpersonerne fik udleveret umiddelbart efter think-aloud testen. Vi er sammenfattende blevet bekræftet i at vores brugergrænseflade er forståelig og let at navigere rundt på og vil derfor fortsætte med samme design.



4.14 Velocity Chart

Udarbejdet af: Bjørn

Til Release 2 ser vores Velocity Chart således ud. Vi har valgt at lave det om så det viser Releases i stedet for iterationer, da vi efterfølgende kun har en iteration i hvert Release. Dermed kan den overordnede hastighed bedre aflæses. Da vi overflyttede task "Juster SMR variabler" som var 2 point og "Tilføj plugin til modul" som vi vurderede til at være 1 point (denne indgik i user-story "Tilføj Modul" som var 4 point) til Release 3,

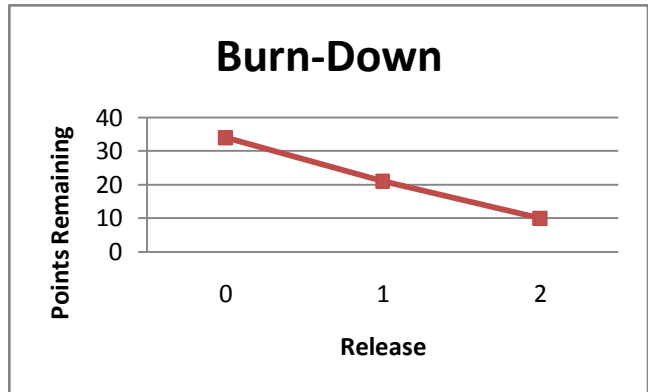


fik vi kun udviklet 10 point i dette Release. Dermed gik vi ned i udviklingshastighed i forhold til Release 1, hvilket også skyldes vores inddragede Review.

4.15 Burn down Chart

Udarbejdet af: Bjørn

For at skabe et regelmæssigt overblik har vi ændret vores Burn-Down chart til at vise hvor meget der blev lavet i hvert Release i stedet for hver iteration. Med de nye krav fra projektpartner er vi kommet op på 34 point der skal laves. Vi har udarbejdet 13 pts i Release 1 og 10 pts i Release 2. Dermed er der 11 pts tilbage til Release 3.



Overvejelser

Da vi ikke fik lavet lige så mange point i dette Release som i Release 1 er vi lidt bagud i forhold til kurven. Men siden der kun er 11 point tilbage at lave, burde det godt kunne nås i Release 3.

4.16 Evaluering af Release 2

Udarbejdet af: Bjørn, Daniel, Kristina

Vi synes dette Release gik rigtig godt. Projektpartner var tilfredse med det der vare blevet udviklet til systemet. På grund af sygdom i dette Release blev vi dog nødsaget til at lave om på vores plan, så Release 3 bliver udskudt med en enkelt uge. Vi inddrager den sidste uge i XP projektet til at færdiggøre Release 3. Dette vil vi vise i en opdateret Releaseplan i Release 3. I forhold til tilføjesen af et review, hvilket gjorde at vi måtte udskyde en del af en user-story til næste Release, kunne vi have inddraget overvejelser omkring tiden det ville tage at udføre det i estimering og planlægning af Release 2.

Det blev også nødvendigt at udskyde en anden user-story til Release 3, på grund af manglende information fra projektpartner om dens implementering. Denne information fik vi ved mødet med projektpartner i slutningen af Release 2 og kan derfor udvikle den i Release 3.

Det blev ikke nødvendigt i dette release at revidere vores Design Klasse Diagram.

Det blev heller ikke nødvendigt med Spike investigations for at estimere User-stories i dette Release.

Vi valgte at lave et Sekvens Diagram til en af vores User-stories, og lavede det denne gang før selve implementeringen som det normalt gøres. Vi mente at udviklingen af SD'et hjalp os med at lave et godt design som alle forstod før vi gik i gang med programmeringen.

På baggrund af dette Release har vi indset at det vil være en god ide at nærmere overveje om der kan udvikles JUnit tests til de enkelte User-stories, i næste Release. Dette skyldes at vi ved endt implementering af dette Release indså at vi godt kunne have lavet JUnit test til nogle af vores User-stories, hvis vi havde overvejet hvordan de skulle implementeres mere grundigt fra start af.

5. Review

5.1 Referat af Review

Udarbejdet af: Bjørn, Daniel, Kristina

²³For at kunne evaluere et projekts forløb kan der i løbet af udviklingen af et software system laves et eller flere Reviews. Ved et Review ses der på hvordan det går med projektet ved at sammenholde kravene til produktet og selve produktets indhold på det givne tidspunkt.

En reviewer får kravspecifikation og produkt noget tid før og laver forberedelse ved at sammenholde disse to. Ved selve Reviewet fremligger reviewer de ting der skal ændres for en eller to udviklere og der tages noter. Dette er ikke et tidspunkt hvor der diskuteres. Udviklerne tager bare imod kritik.

Reviewer kan bedømme udfaldet af projektet til at være:

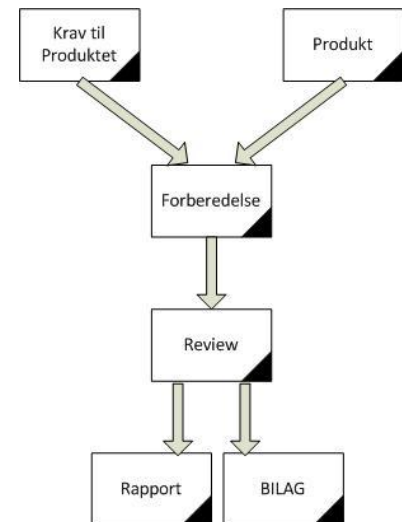
Ubetinget godkendt

Betinget godkendt – mangler nogle småting

Ikke godkendt – noget skal laves om før det er godkendt

Ubrugeligt – skal forkastes

Review kan ikke gennemføres – hvis dokumentet evt. er ubrugeligt eller der ikke var tid til forberedelse.



De ting der skal ændres kommer så i selve rapporten eller i bilag.

Vores benyttelse

Vores formål med at lave et Review var at kunne få noget feedback på vores rapport, ud over det vi har fået fra vores projektvejleder. Produktet som vi skulle have feedback på, var derfor ikke så meget selve software-systemet, men rapporten. Kravspecifikationen for dette review var kravene til at lave en Datamatiker Hovedopgave.

Vi udvekslede rapporter nogle dage før og fra vores side læste vi deres rapport igennem hver for sig, og snakkede sammen før selve reviewet om hvilken feedback vi skulle give. Ved selve Reviewet skiftedes vi så til at give feedback til hinanden. Vi baserede feedback på at give hinanden gode idéer til at gøre rapporten bedre og ikke på om rapporten kunne godkendes eller ej.

Følgende er et referat af den kritik vi fik. Først står der kort om de gode ting ved vores rapport og derefter står der om de ting der evt. skulle ændres. Efter hver ting der evt. skal ændres har vi skrevet hvad vores respons er.

Feedback fra anden projektgruppe

- *God forside*
- *Godt med begrundelser for hvorfor I gør ting*

²³ Professional Systems Development Kapitel 6 – Se Litteraturliste

- *Gode planer*
- *God metafor – var nemt at forstå systemets opbygning ud fra denne*

Ting der skal ændres

- *Der er for mange numre i jeres indeksering*

Vi mener selv dette er okay da det giver et bedre overblik over hvordan rapporten er delt op. Derfor vil vi ikke ændre dette.

- *Der er 2 typer af Spikes i XP – arkitektur spikes og estimerings spikes.*

Vores kilder beskriver ikke noget om nogen opdeling. En spike er en forundersøgelse - om det så er på arkitektur eller estimering af user-story. Derfor finder vi det ikke vigtigt at skulle dele spikes op. Vores spike "Forbind til modul" var både til arkitekturen og estimering af en user-story.

- *Har I brug for CRC cards?*

Da vi arbejder ud fra udviklingsmetoden XP og det er måden at få overblik over de forskellige klasser på, mener vi det er vigtigt at have dem med. Vi har udarbejdet disse på papir først og for at kunne få et bedre overblik over hvordan de hang sammen har vi lavet vores Design Klasse Diagram som et ekstra artefakt. Vores Design Klasse Diagram er der for at understøtte vores CRC cards og ikke omvendt.

- *I har mange forkortelser som I ikke har forklaret hvad betyder.*

Dette er vi allerede klar over og vil gennemgå rapporten så snart som muligt for at forklare de enkelte forkortelser. Men det var godt at få det udpeget igen.

- *Det kunne måske give et bedre overblik over de enkelte user-stories og deres udvikling at I lagde alle artefakter efter hinanden, i stedet for at gruppere artefakterne. Hvis I fx til user-story "Forbind til Modul" havde user-story, Task Cards og Acceptance Test lige efter hinanden.*

Dette er en rigtig god idé, fordi det som sagt ville give et bedre indblik de enkelte user-stories udvikling. Det ville dog gøre at man mister overblikket over hele systemet så vi har valgt at beholde vores artefakter grupperet. Vores rapport følger også udviklingen punkt for punkt, og en opstilling på denne måde ville ikke følge udviklingen. Men da vi synes godt om denne idé vil vi lave et afsnit i vores bilag hvor man kan få dette indblik hvis det er ønsket. Se Bilag 6. "User-stories udvikling."

5.2 Evaluering af Review

Udarbejdet af:

Selvom det gik ud over XPs regel om at vægte udvikling af de User-stories der er blevet planlagt højest, var det godt at have et Review med en anden gruppe, da det var en stor hjælp for vores overordnede hovedopgave.

Feedback fra den anden projektgruppe fik os til at gennemtænke de beslutninger vi havde taget og gjorde at vi fik bekræftet mange af vores beslutninger. De steder hvor eventuelle ændringer skulle ske fik vi diskuteret og udført de ændringer som vi var enige i.

Det var godt at høre at de kunne lide vores metafor og at det hjalp dem til at få indsigt i systemets opbygning, da det bekræftede vores formodning om at den kunne hjælpe andre der ikke på forhånd kendte til projektet.

6. eXtreme Programming Release 3

6.1 Indledning

Udarbejdet af: Bjørn, Daniel, Kristina

Vi tager her hul på den sidste planlagte Release 3. Vi vil forsøge at implementere så mange ønsker til systemet som muligt på baggrund af seneste feedback fra Projektpartner, men vil dog prioritere ønsker ud fra relevans i forhold til kernen af systemet.

I dette Release har vi valgt at skrive overvejelser efter hvert artefakt hvor det er relevant. Disse overvejelser vil indgå i den samlede evaluering af Release 3.

6.1.1 Projekt Plan

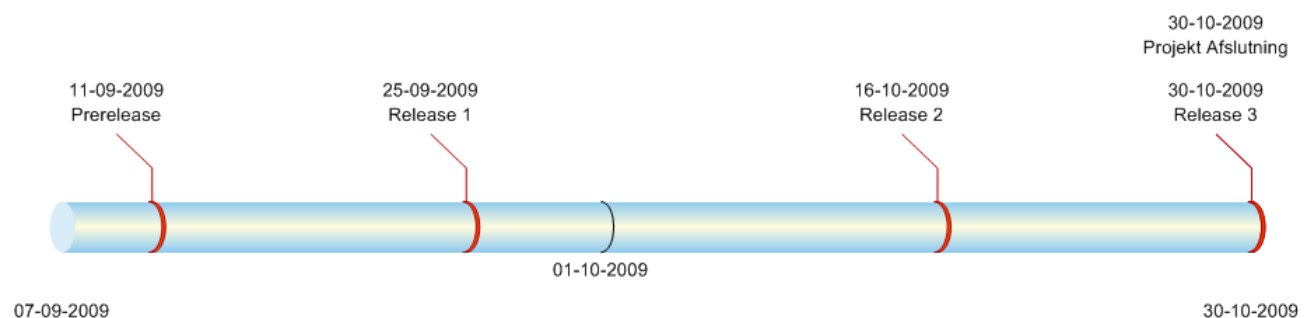
Vores projekt plan for Release 3 ligger i Bilag 1.4 "Projekt Planer, Release 3".

6.2 Revideret Releaseplan

Udarbejdet af: Daniel

Da vi har måttet rykke vores Release 3 en uge frem har vi her lavet en revideret Releaseplan. Vi har estimeret Release 3 til 11 pts. Her er inkluderet de 3 pts vi ikke fik udviklet i Release 2 fra user-story "Justér SMR variabler" og task card "Tilføj plugin til modul". På baggrund af projektpartners prioritet til udviklingen har vi fastlagt hvilke User-stories ville være bedst at have med i Release 3.

Herunder ses vores opdaterede Releaseplan:



Release 1 – User-stories: 13 pts Release Date: 25-09-09	Release 2 – User-stories: 10 pts Release Date: 09-10-09	Release 3 – User-stories: 11 Release Date: 30-10-09
Forbind til Modul (10 pts) Send kommando til Modul (1 pts) Vis variabeltræ (2 pts)	Vis grafer for SMR variabler (4 pts) Tilføj Modul (3 pts) Ændre variabelværdi i variabeltræ (2 pts) Genbrug af sendte kommandoer (1 pts)	Justér SMR variabler (2pts) Tilføj plugin til modul (1pts) Tilføj variabel til status (2 pts) Modul Status Lampe (2 pts) Kontrol knapper (3 pts) Titel (1 pts)

Overvejelser

"Tilføj plugin til modul" valgte vi at lave som en user-story for sig selv i dette Release, da denne task oprindeligt var en del af "Tilføj Modul" som blev lavet i Release 2. Vi kunne ikke basere vores estimering på hvor mange points blev lavet i Release 1, da points mere antyder størrelse af en user-

story end hvor lang tid det tager at implementere denne. "Forbind til Modul" havde 10 pts men tog 23 timer at udvikle. Derfor baserede vi i stedet for estimering på Release 2 og regnede med at vi kunne nå lidt mere i dette Release da vi ikke har Review.

6.3 User-stories

Udarbejdet af: Bjørn, Kristina

I dette Release har vi lavet en enkelt ekstra user-story "Tilføj plugin til modul". Denne var oprindeligt en del af "Tilføj Modul" men blev ikke implementeret i Release 2.

Vi har ikke haft brug for spikes til at estimere den nye user-story.

Følgende er de user-stories vi vil udvikle i dette Release. Vi har valgt kun at vise 1 user story her. Alle user-stories findes også i Bilag 2.3 "User-stories, Release 3".

Tilføj variabel til status
Story Number: 4
Estimation: 2 pts
Business value: Medium
Brugeren markerer en variabel, og tilføjer den til status. Status er en række variabler der bliver overvåget og hjælper brugeren til at se status for robotten.
Overvejelser: Sværhedsgraden af implementeringen afhænger af om brugeren skal kunne angive grænseværdier for status variabler og hvordan brugeren vil have dette til at fungere.

Overvejelser

Da projektpartner havde nogle små GUI ændringer de gerne ville have lavet til det system de får udleveret efter Release 3, prøvede vi om disse kunne laves som user-stories. Vi blev dog enige om at de var for små, og derfor bliver lavet sideløbende med de andre user-stories. Vi vil beskrive disse små ændringer i afsnittet "Brugergrænseflade"

6.4 Iterationsplan

Udarbejdet af: Bjørn

Følgende er vores Iterationsplan for Release 3. Vi har igen kun en iteration, da det virkede fint i Release 2.

Release 2	Estimeret Tid/timer
Justér SMR variabler	3 t.
Tilføj plugin til modul	2 t.
Tilføj variabel til status	3,5 t.
Modul status lampe	2 t.
Kontrol knapper	2 t.
Titel	2 t.
Total:	19 t.

Overvejelser

I dette Release kom vi op på 19 timers estimeret arbejde – det samme som i Release 2. Dette skulle passe godt da der også var nogle små GUI ændringer til dette Release, men vi skal ikke bruge tid på et Review.

6.5 Task Cards

Udarbejdet af: Kristina, Daniel

I dette Release har vi også lavet task cards efter udviklingen af vores Iterations plan som normalt forskrevet i XP. Til user-story "Tilføj Variabel til Status" har vi valgt at det skal være muligt at tilføje en grænseværdi. Derfor har vi lavet en JUnit til at teste dette.

Vi har valgt kun at vise 1 task card her. Alle task cards kan ses i Bilag 3.3 "Task Cards, Release 3".

Task Card 19 , Release 3, User-story: Tilføj variabel til status
Date:
Task: Lav JUnit til at tjekke grænseværdier for status variabler
Tid: 2 t.
Kommentar:

Overvejelser

I dette Release sørgede vi for at tænke bedre over hvilke tasks der eventuelt kunne laves nogle JUnit tests til. Der var ikke mange steder det kunne gøres, men vi fandt dog 2: "Lav JUnit til at tjekke grænseværdier for status variabler" og "Lav JUnit til præsentering af overvågede variabler".

6.6 Acceptance tests

Udarbejdet af: Bjørn

Vi har igen lavet Acceptance tests på baggrund af projektpartners krav til systemet. Efter endt implementering af user-stories til dette Release vil vi køre dem på DTU for at sikre os at de giver det forventede output. Acceptance test "Tilføj plugin til modul" har vi revideret fra Release 2.

Vi har valgt kun at vise 1 test her. Se alle Acceptance tests i Bilag 5.3 "Acceptance Tests, Release 3".

Tilføj variabel til status 1: Tilføj variabel

Test Trin	Input/Handler	Forventet Output	Resultat
1	Brugeren højreklikker på en variabel i variabeltræet.	En pop-up menu vises hvor der kan vælges "Add to Status"	
2	Brugeren vælgerne "Add to Status"	Variabel bliver tilføjet status i Status faneblad	

6.7 Programmering

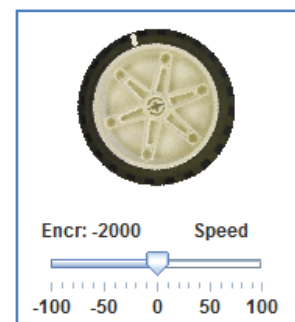
Udarbejdet af: Bjørn, Daniel

I dette afsnit vil vi først trække frem og gennemgå i dybden dele af implementeringen. Derefter vil vi beskrive vores brug af Repository Model.

6.7.1 Programmering i dette Release

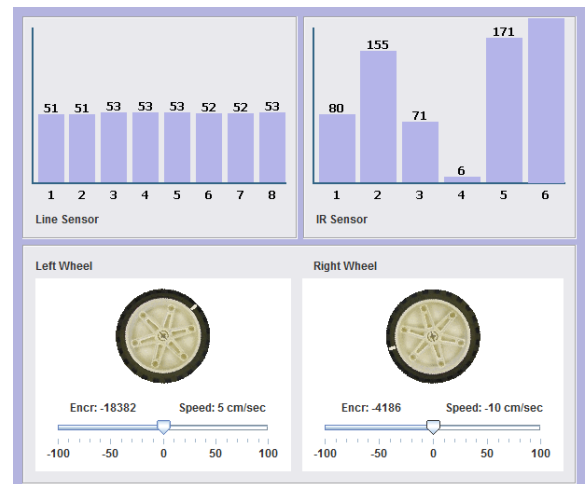
Justér hjulenes hastighed i SMR Plugin

I SMR plugin, som vi begyndte at udvikle i sidste release, manglede vi stadig at lave det så brugeren kan justere hastigheden på robotens hjul. Denne user-story kaldte vi "Justér SMR variabler". Vi fik i forrige release lavet en RobotWheel klasse, som blot tegner et billede af et hjul og har en metode til at rotere dette.



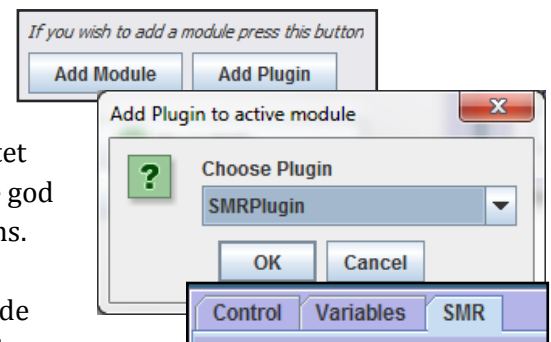
Til at styre hjulet og styre dets hastighed lavede vi en WheelControl klasse. Den bruger variabel-data ved navn "rhd.encyr" og "rhd.encyl", der angiver SMR robotens omdrejningstæller. Ifølge projektpartner har robotens hjul drejet en hel omgang når en encoder værdi er ændret med 2000. Hvis den bliver ændret positivt kører hjulet fremad og negativt bagud. Vi fik udviklet klassen så den fint animerede hjulet baseret på dens sidste 2 modtagne encoder-værdier for hvert hjul. Dette gjorde at hjulet altid vil animere videre indtil ny data modtages, og ikke bare en enkel gang når ny data modtages. Dette gav en meget god animation og så ud til at ville passe godt sammen med den reelle rotation af robotens hjul. Til at styre selve hastigheden brugte vi en slider der kan gå fra -100 til 100. Denne vil når slideren sættes til x, sende kommandoen "var rhd.speedr[1] = x" (eller speedl hvis det er venstre hjul). Hvis det hele fungerer korrekt, vil den forbundne SMR robots hjul begynde at køre den givne hastighed og denne vil på baggrund af encoder værdier blive reflekteret i hastigheden af hjulene over slideren.

SMR Plugin endte med at indeholde en LineGraph for robotens linie- og infrarøde sensor og et WheelControl for hvert af de 2 baghjul. Tilsammen blev det et godt og funktionelt plugin, der både opfylder projektpartners ønsker, og demonstrerer hvad der er muligt at lave ved hjælp af vores pluginstruktur.



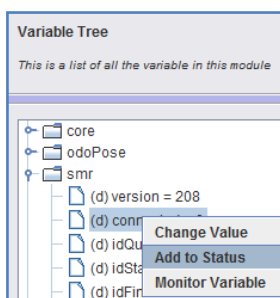
Tilføj plugin til modul

Til user-story "Tilføj plugin til modul" lavede vi en knap i menuen i venstre side af vores GUI "Add Plugin". Denne knap kan bruges til at tilføje en implementering af ModulePlugin til det aktive modul. Det vil sige, det modul som på kørselstidspunktet bliver vist i højre side af GUIen. Til koden bag dette kunne vi gøre god brug af PluginManager til at få en liste over de tilgængelige plugins. Denne liste valgte vi at vise for brugeren når der trykkes på "Add Plugin" med en simpel JOptionsPane. Hvis brugeren vælger en af de præsenterede plugins bliver det herefter tilføjet det aktive modul.

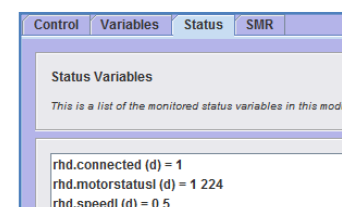


Tilføj variabel til Status

I denne release tilføjede vi et nyt faneblad til alle moduler kaldet "Status". I variabeltræet tilføjede vi i højreklikks-menuen "Add to Status", så en variabel kan tilføjes til Status faneblad. Status faneblad er ment som en samling af de vigtigste variabler for det enkelte modul på roboten, der giver et hurtigt overblik over dens generelle status.

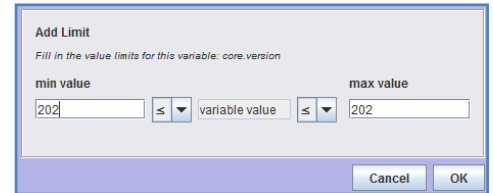


Vi implementerede en klasse kaldet StatusVariable der indeholder en reference til det ModuleVariable objekt man højreklikkede på i variabeltræet, når man vælger "Add to Status". Det smarte ved dette er at når en variabel



bliver opdateret i træet, vil den automatisk også blive opdateret i Status faneblad, da de bruger en reference til det samme objekt.

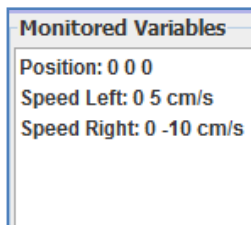
Vi udviklede statusvariablerne så der også kan sættes en grænseværdi for hver variabel. På den måde kan brugeren sætte en max og en min værdi som den enkelte variabel skal opholde sig indenfor. Hvis variabelens værdi bevæger sig udenfor grænseværdierne bliver det markeret med en "(Outside Value Limit)" besked. Grænseværdien sættes ved hjælp af en pop-up boks hvor der kan indtastes en min og max værdi for variabelen og vælges hvilken sammenlignings-metode der skal bruges.



Vi implementerede alt dette ved at tilføje en ValueLimit klasse til StatusVariable, som hvis den er sat, kan bruges til at tjekke om det givne StatusVariable objekt befinder sig inden for sin grænseværdi.

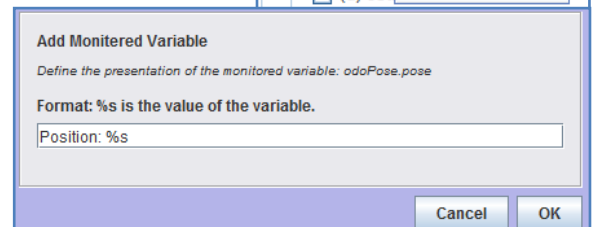
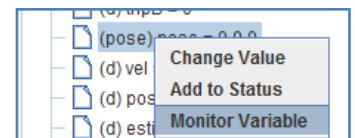
Tilføj Variabel til overvågningsvindue

Der blev i dette release også udviklet et centralt overvågningsvindue til variabler fra alle moduler. Dette overvågningsvindue skulle for brugeren vise vigtige variabler fra hele robotten. Heldigvis viste vores design af datastrukturen ModuleVariable sig at være os til gavn igen. Som ved implementeringen



af StatusVariable lavede vi en ny klasse kaldet MonitoredVariable, der på samme måde tog imod en ModuleVariable reference. Forskellen var denne gang at vi ikke havde brug for nogen grænseværdi, men at det skulle være muligt at indstille præsentationen af en overvåget variabel. For brugeren vil det ikke være hensigtsmæssigt at have en masse ubeskrivende variabelnavne på række i

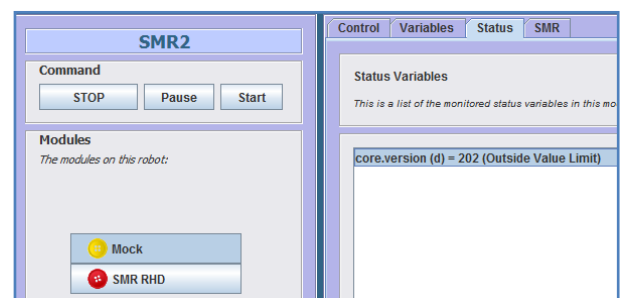
overvågningsvinduet. Derfor besluttede vi at det skulle være muligt at kunne indstille præsentation for en overvåget variabel. Vi brugte en præsentations String, der i `toString` metoden erstatter "%s" med variabelens værdi. På denne måde kan en variabel som f.eks. "rhd.speedr = 5" blive præsenteret som "Speed Right: 5 cm/s", hvilket er meget mere sigende og informativt for brugeren af systemet.



Modul status-lampe og kontrollknapper

Menuen i venstre side af GUI modtog mange forbedringer i dette release. Først og fremmest blev statuslamperne for hvert modul implementeret, således at der på knappen ud for hvert modul er et farvet symbol der angiver det givne moduls status. Denne status bliver opdateret jævnligt ved at bruge metoden `getModuleStatusColor` på hvert modul.

Hvis denne metode returnerer `Color.RED`, bliver et rødt symbol sat osv. Et modul vil returnere `Color.RED` hvis det givne modul ikke er forbundet, `Color.GREEN` hvis der er forbindelse og alt er OK og til slut `Color.YELLOW` hvis der er forbindelse, men



en af statusvariablerne er uden for dens grænseværdi. På den måde har brugeren på hvilket som helst tidspunkt et overblik over om modulerne er forbundet og om de kører som de skal.

Følgende er koden i ModuleMenu klassen der opdaterer hvert moduls statuslampe

```
Collection<RobotModule> modules = MainGUIHandler.getInstance().getAllModules();  
for (RobotModule module : modules) {  
    String moduleName = module.getModuleName();  
    Color statusColor = module.getModuleStatusColor();  
    JToggleButton moduleButton = MainGUIHandler.getInstance()  
        .getMenuButtonByName(moduleName);  
  
    if (moduleButton != null) {  
        if (statusColor == Color.GREEN) {  
            moduleButton.setIcon(greenIcon);  
        } else if (statusColor == Color.YELLOW) {  
            moduleButton.setIcon(yellowIcon);  
        } else if (statusColor == Color.RED) {  
            moduleButton.setIcon(redIcon);  
        }  
    }  
}
```

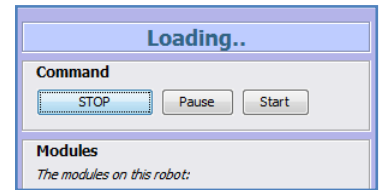
Denne kode afvikles gentagende i en tråd, således at knappernes status bliver opdateret regelmæssigt.

Menuens kontrolknapper; start, stop og pause, blev også implementeret så de nu virker. Hver kontrolknap repræsenteres af et ControlButton objekt. Objektet indeholder navnet på et modul og den kommando der skal sendes til modulet. De 3 ControlButtons blev også gjort konfigurerbare ved igen at videreudvikle XMLConfigManager. Vi stod dog nu i den situation at vi havde konfiguration der var generelt for robotten og ikke for et enkelt modul. Derfor måtte vi refaktorere den måde vi fortolker konfigurationsfilen på, sådan at vi modtager et Robot objekt. Dette Robot objekt indeholder både generel konfigurering for robotten og også dens moduler. Der kan læses mere om dette i refaktorings afsnittet. Efter refaktoreringen kunne vi load de 3 ControlButton objekter fra start ved at lave en robot.xml fil der ser således ud:

```
<robot>  
  <name>SMR2</name>  
  <controlButtons>  
    <start moduleName="Mock" command="var rhd.start 1 1"/>  
    <pause moduleName="SMR RHD" command="var rhd.pause 1 1"/>  
    <stop moduleName="SMR RHD" command="var rhd.stop 1 1"/>  
  </controlButtons>  
  <module>  
    <name>Mock</name>  
    <host>localhost</host>  
    <port>24930</port>  
    <plugin>SMRPlugin</plugin>  
    <autoConnect>true</autoConnect>  
  </module>  
</robot>
```

Attributten for en kontrolknap "moduleName" skal være lig med det der er skrevet i "name" elementet for et modul, for at kommandoen bliver sendt til det. Grunden til at dette blev gjort konfigurerbart, er at det vil være meget forskelligt fra robot til robot, hvilken kommando og modul der skal sendes til for at starte, stoppe og pause robotten.

Som det kan ses i robot.xml eksemplet ovenfor, gjorde vi det også muligt at konfigurere robottens titel. Dette gjorde vi for hele tiden at kunne vise i GUI menuen hvilken robot man er forbundet til. Vi lavede det så dette "name" element er krævet i XML filen, og at der i GUI bliver vist "Loading.." indtil XML filen er blevet hentet og indlæst.

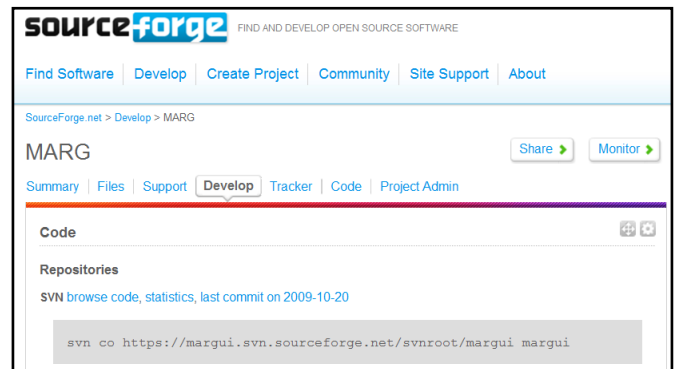


6.7.2 Brug af repository model

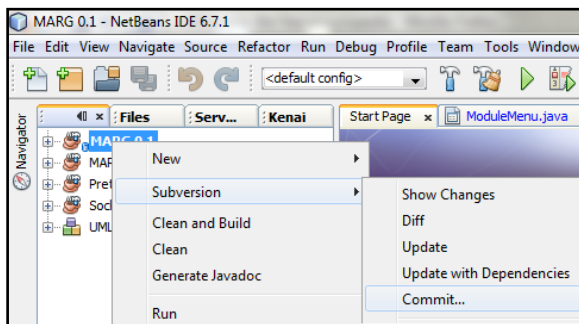
Til vores projekt var vi fra starten enige om at bruge Repository Model til at opbevare vores kode. Dette beskrev vi i under forebyggelse af tab af data i vores risikoliste til projektet.

En repository model ville sørge for et centralt sted at opbevare vores kode, som alle projektgruppens medlemmer kan tilgå. Det ville også fungere som backup, i tilfælde af mistet data.

Vi valgte at bruge SourceForge, som er et website hvor man gratis kan oprette et software projekt. Når man opretter et projekt på SourceForge følger der et Subversion



Repository, samt en masse andre software udviklings værktøjer med. Dette Subversion (SVN) repository er hostet af SourceForge og kan tilgås hvis man har installeret en Subversion klient. Alle 3 medlemmer af projektgruppen bruger NetBeans IDE'en, der direkte understøtter brugen af et SVN repository. Dette gjorde brugen af vores repository nemt at integrere med den daglige udvikling af systemet. Alle medlemmer af gruppen kan på hvilket som helst tidspunkt hente den nyeste version af systemets kode fra deres IDE, det eneste der er krævet er en internetforbindelse. For at uploade/commit vores ændringer af koden skulle vi derudover blot have en bruger på SourceForge og være medlem af det oprettede projekt.

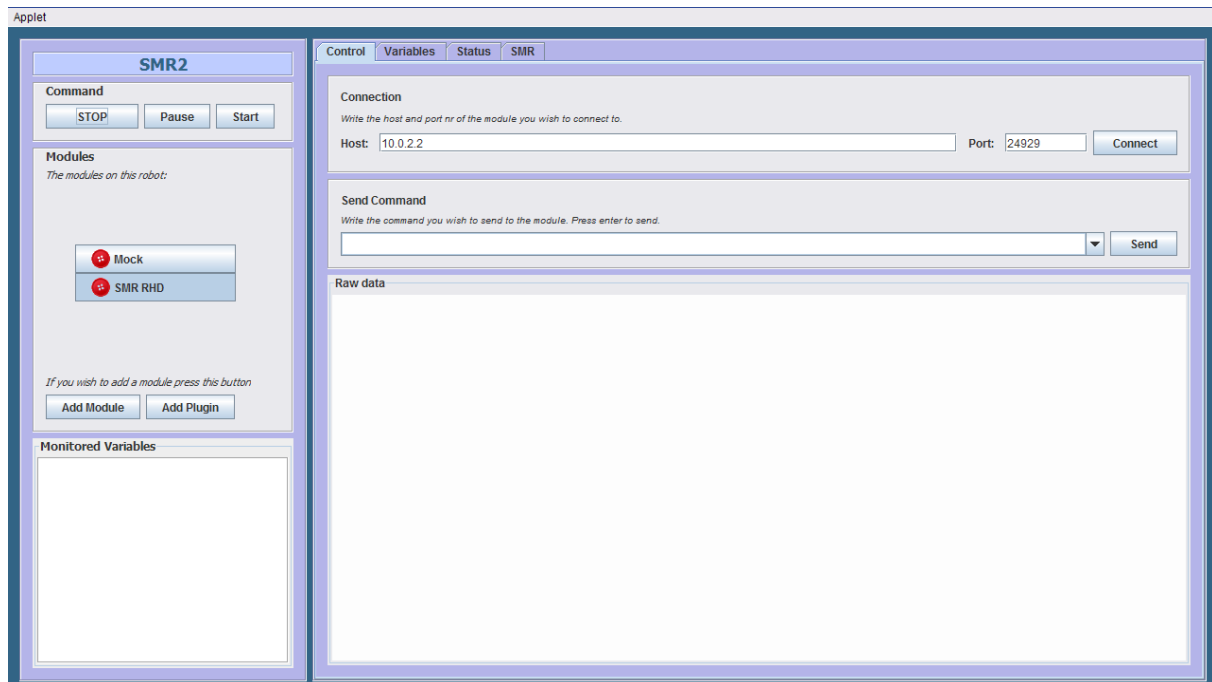


Brugen af et SVN repository har indtil videre været en rigtig stor hjælp for projektgruppen. Det har sparet os for en stor mængde tidskrævende arbejde med at integrere og distribuere den udviklede kode. Selvom vi for det meste har programmeret i par, som XP foreskriver, har der været enkelte implementeringsopgaver som vi har lavet individuelt. Her har vores SVN repository også været til stor hjælp og gjort det hurtigt og nemt at samle og distribuere den nyeste version af vores systems kode.

6.8 Brugergrænseflade

Udarbejdet af: Daniel

Selve brugergrænsefladen blev der ikke lavet de store ændringer til i dette Release. Det færdige system ser således ud når man åbner det:



Øverst til venstre har vi placeret navnet på robotten. Derved kan man hele tiden se hvilken robot man har forbindelse til. I dette tilfælde SMR2. Moduler, forbindelse til disse og plugins vises som konfigureret i XML filen. I dette tilfælde er modulerne: "Mock" og "SMR RHD" (som ikke er forbundet automatisk) og plugin "SMR". Fanebladene Control, Variables og Status er generelle for alle moduler og vises altid.

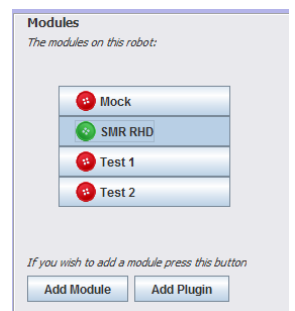
Der var nogle små ting som projektpartner gerne ville have ændret. Disse ændringer fik vi lavet sideløbende med at vi implementerede vores user-stories til dette Release. Ændringerne var:

"Når et modul er valgt skal det kunne ses på knappen, så man kan se hvilket modul man bruger".

Dette lavede vi sådan at det når man har trykket på en modul knap ser det ud som om knappen er trykket ind.

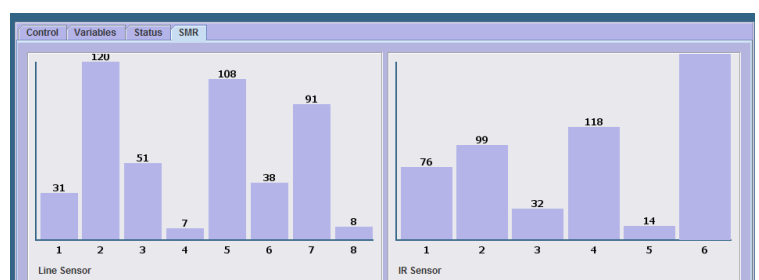
"Så snart man starter systemet og har kontakt til et modul, skal alle variabler kunne ses i variabeltræet".

Førhen blev der ikke vist noget i variabeltræet før brugeren selv havde sendt en "var allcopy" kommando til modulet. Nu har vi lavet det sådan at denne kommando bliver sendt når systemet starter hvis der er forbindelse til et modul. Dermed bliver alle variabler vist i variabeltræet fra starten.



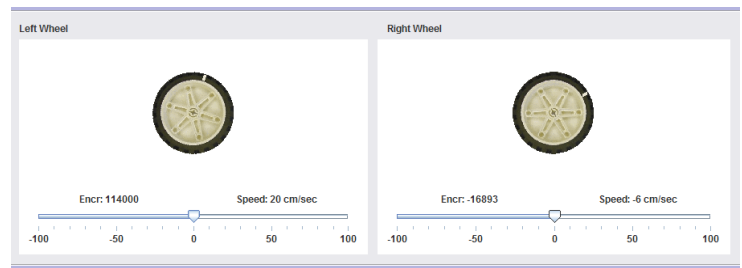
"Der skal stå tal under hver sensor i graferne".

Dette har vi lavet så der under Line sensor grafen står 1-8 og under IR sensor grafen står 1-6.



"Der må gerne stå encoder og speed værdier ved hjulene"

Dette har vi også fået lavet så man kan se hvor hurtigt hjulene kører og hvor mange omdrejninger de laver.



6.9 Opdaterede Task Cards

Udarbejdet af: Daniel, Kristina

Følgende er vores opdaterede task cards med kommentarer omkring hvordan vi implementerede dem. I dette release var der igen flest Tasks der primært havde med GUI'en at gøre og derfor ikke var så oplagte at lave Unit tests til. Vi fandt dog 2 Tasks hvor det ville være muligt og gavnligt at lave Unit Test til.

Vi implementerede de 2 Tasks som blev udsat fra det forrige release først; "Justér SMR Variabler" og "Tilføj plugin til modul".

Task Card 12, Release 3, User-story: <u>Justér SMR variabler</u>
Date: 20-10-09
Task: Gør det muligt at justere SMR variabler, så det kan ses om hjulene kører.
Tid: 3 t.
Kommentar: <ul style="list-style-type: none">- Vi rettede ikke i RobotWheel klassen- Vi lagde selve funktionaliteten til at dreje hjulene i WheelControl klassen- Brugte en slider-bar til at justere robotens hastighed for hvert hjul

Task Card 15 , Release 3, User-story: <u>Tilføj plugin til modul</u>
Date: 20-10-09
Task: Tilføj plugin til modul
Tid: 2 t.
Kommentar: <ul style="list-style-type: none">- Funktionaliteten fra de forrige releases og vores modulære opbygning af systemet, gjorde denne task nemmere at implementere end først antaget- Vi hentede de plugins der var til rådighed fra PluginManager og viste dem i en pop-up boks hvor et kan vælges i en drop-down boks. Det valgte plugin tilføjes til det aktive modul.

Task Card 19 , Release 3, User-story: <u>Tilføj variabel til status</u>
Date: 21-10-09
Task: Lav JUnit til at tjekke grænseværdier for status variabler
Tid: 2 t.
Kommentar: <ul style="list-style-type: none">- Lavede en JUnit der tjekker om ValueLimit klassen er korrekt implementeret.- ValueLimit blev designet så den har en min og en maxvalue og så den har en metode der tjekker om et givent tal ligger inden for denne grænse.

Task Card 20 , Release 3, User-story: <u>Tilføj variabel til status</u>
Date: 21-10-09
Task: Lav grafisk variabel-status faneblad – inkl. mulighed for tilføjelse i variabeltræ
Tid: 1,5 t.
Kommentar: <ul style="list-style-type: none">- Vi byggede videre på højre-klik menu i variabeltræ og tilføjede "Add to Status"- Vi tilføjede et faneblad til ModuleTabs hvor de tilføjede variabler bliver vist- Lavede en StatusVariable der er en slags fremviser for ModuleVariable, som blot omskriver dens toString metode.- Gjorde det muligt at tilføje en grænseværdi til en StatusVariable i Status fanebladet via. en højreklik-menu. Klassen ValueLimit blev implementeret på baggrund af JUnit (Task 19).

Task Card 21 , Release 3, User-story: <u>Tilføj variabel til status</u>
Date: 22-10-09
Task: Lav JUnit til præsenteringen af overvågede variabler.
Tid: 2 t.
Kommentar: <ul style="list-style-type: none">- Lavede en JUnit test der tjekker om klassen MonitoredVariable er implementeret korrekt.- MonitoredVariable blev lavet til at skulle bestå af en ModuleVariable og en tekst string der skal præsentere denne modulvariabel på en ønsket måde.

Task Card 22 , Release 3, User-story: <u>Tilføj variabel til status</u>
Date: 22-10-09
Task: Lav behandling og tjek af status variabler
Tid: 2 t.
Kommentar: <ul style="list-style-type: none">- For det ikke blev forvirrende med et status faneblad og et status vindue, kaldte vi dette status vindue for "Monitored Variables".- Vinduet blev lavet så det indeholder en liste af MonitoredVariable objekter.- MonitoredVariable klassen blev implementeret efter den udviklede JUnit i Task 21.- Hver MonitoredVariable indeholder en ModuleVariable og et stykke tekst der bruges til at præsentere værdien af ModuleVariable på en ønsket måde.

Task Card 23 , Release 3, User-story: <u>Modul status lampe</u>
Date: 22-10-09
Task: Tilføj mulighed for at tjekke status på et modul
Tid: 2 t.
Kommentar: <ul style="list-style-type: none">- Status for hvert modul bliver vist med et billede af et Rødt, Gult eller Grønt symbol på det givne moduls knap i venstre side af GUIen.- Status blev lavet til at opdatere to gange i sekundet ved at bruge RobotModule's getModuleStatusColor der returnerer Grøn hvis alt er okay, Gul hvis en status variabel er ude for sin grænseværdi og Rød hvis der ingen forbindelse er.

Task Card 24, Release 3, User-story: <u>Kontrol knapper</u>
Date: 23-10-09
Task: Lav det konfigurerbart i XML, hvor og hvilke kommandoer der skal sendes.
Tid: 2 t.

Kommentar:

- Vi byggede videre på XMLConfigManager så den også kan læse et controlButtons element og dens indeholdende start, pause og stop elementer.
- Start, pause og stop indeholder en moduleName attribut og en command attribut, der tilsammen fortæller vores system hvilket modul den skal have fat i og hvilken kommando der skal sendes når den givne knap trykkes på.

Task Card 26 , Release 3, User-story: Titel

Date: 23-10-09

Task: Lav robottens titel konfigurerbar i XML.

Tid: 2 t.

Kommentar:

- Lavede et felt i venstre side af GUIen hvor robottens navn skal stå.
- Byggede videre på XMLConfigManager så et "name" element der indeholder navnet på robotten der forbindes til også kan læses. Dette navn bliver vist i førnævnte felt, når konfigurationen er blevet indlæst.

6.10 Testning

Udarbejdet af: Daniel

I dette Release lavede vi JUnit test til vores klasser, "ValueLimit" og til "MonitoredVariable". Vores Acceptance Tests kan ses i afsnit 6.5

Følgende er et kodeuddrag fra JUnit test til "ValueLimit" der viser vores test af dens centrale metode: *checkCorrect*

```
@Before public void setUp() {
    limitLEQ = new ValueLimit(-50.5, ValueLimit.LESS_THAN_EQUALS, 50.5,
        ValueLimit.LESS_THAN_EQUALS);
    limitL = new ValueLimit(-50.5, ValueLimit.LESS_THAN, 50.5,
        ValueLimit.LESS_THAN);
}
@Test public void testCheckCorrect() {
    System.out.println("checkCorrect");
    double value = -50.5;
    assertEquals(true, limitLEQ.checkCorrect(value));
    assertEquals(false, limitL.checkCorrect(value));

    value = 0;
    assertEquals(true, limitLEQ.checkCorrect(value));
    assertEquals(true, limitL.checkCorrect(value));

    value = 50.5;
    assertEquals(true, limitLEQ.checkCorrect(value));
    assertEquals(false, limitL.checkCorrect(value));
}
```

Overvejelser

I dette Release fik vi tænkt lidt mere i dybden før vi implementerede, så vi fandt ud af at vi kunne lave 2 JUnit Tests. Dette hjalp os til at holde XPs standard omkring det at lave test før man programmerer.

6.11 Refaktorering

Udarbejdet af: Bjørn, Daniel

Den største refaktorering i dette Release var ændringen af hvordan vi indlæser konfiguration fra XML fil. I stedet for at returnere en liste af Module objekter, blev XMLConfigManager lavet om så den returnerer et Robot objekt, der både indeholder generel information om robotten og listen af moduler. Denne ændring stemmer stadig overens med det tidligere SD "Modtagelse af XML data" fra release 1, da vi på det tidspunkt endnu ikke havde besluttet os for hvilken datastruktur konfigurationen skulle returneres i. Vi havde allerede på det tidspunkt diskuteret at dette kunne blive en nødvendig ændring og ville derfor ikke binde os til én specifik datastruktur. Det viste sig at være en klog beslutning da vi nu har implementeret ændringen.

Følgende er Robot datastrukturen som den læste konfiguration pakkes i:

```
public class Robot {  
  
    private List<Module> modules = new ArrayList<Module>();  
    private ControlButton startButton;  
    private ControlButton pauseButton;  
    private ControlButton stopButton;  
    private String robotName;  
  
    ...  
}
```

Følgende er koden i MargGUI der indlæser det hentede Robot objekt:

```
Thread loadThread = new Thread(new Runnable() {  
    public void run() {  
        final Robot loadedRobot = loadRobot();  
        loadTitle(loadedRobot);  
        loadControlButtons(loadedRobot);  
        addLoadedModules(loadedRobot.getModules());  
    }  
});  
loadThread.start();  
  
...  
private Robot loadRobot() {  
    XMLConfigHandler configHandler = new XMLConfigHandler();  
    java.net.URL codebaseURL = MargGUI.this.getCodeBase();  
    System.out.println("CodeBase: " + codebaseURL);  
    if (XMLConfigHandler.isLocalURL(codebaseURL)) {  
        codebaseURL = XMLConfigHandler.fixLocalUrl(codebaseURL);  
    }  
    URL configFile = XMLConfigHandler.getFileURL(codebaseURL,  
                                                ROBOT_CONFIG_FILENAME);  
    return configHandler.getConfigFrom(configFile);  
}
```

Det kan tage lang tid at hente konfigurationen og indlæse alle modulerne, derfor har vi lagt koden til dette i sin egen tråd, så GUI også kan bruges på samme tid.

Metoden *loadRobot* er den refaktorerede version af metoden *autoLoadModules* som vi havde kodeuddrag fra i programmeringsafsnittet i release 2.

Overvejelser: Efter lidt reflektering omkring refaktoreringen i dette Release, indså vi at vi faktisk havde endt med at implementere de objekter som vi oprindeligt havde identificeret i vores Domænemodel. Objekterne Robot og Modul.

6.12 Præsentation af Release 3 hos Projektpartner

Udarbejdet af: Daniel

Vi var på besøg hos DTU torsdag d. 29. oktober for at præsentere vores system og køre vores acceptance tests. Det var sidste og afsluttende Release på projektet. Følgende er et kort referat af dagen.

Vi startede med at udlevere en cd-rom til projektpartner indeholdende alle systemfiler inklusiv java-doc og en guide forbeholdt de studerende på DTU på hvordan de kan implementere nye moduler og plugins. Vi gennemgik indholdet på cd-rom med uddybende forklaring til hvert emne²⁴. Derefter overførte projektpartner den nyeste version af systemet til SMR robot så vi kunne præsentere de nyeste implementeringer fra release 3. Projektpartner var tilfreds med resultatet og at systemet var brugbart for dem. De var meget tilfreds med den måde som sensorer og hjul blev vist på grafisk. Derudover var de godt tilfreds med at vi havde lavet det muligt at tilføje grænseværdier til udvalgte variabler. Vi viste at systemet kunne afvikles på både en version af Windows og Linux. Endvidere at systemet kunne afvikles med forskellige browsere som Internet Explorer og Firefox.

Efter præsentationen etablerede Projektpartner forbindelse til en anden type robot og det virkede og variabeltræ blev vist som det skulle. Projektpartner havde dagen forinden testet om der kunne etableres forbindelse til en tredje type robot og det kunne der.

Projektpartner havde et ønske om at teste om kompilering af systemet kunne ske på deres Linux computer. Det kunne det ikke. Men fejlen bestod formentlig i at version af Netbeans eller Java SDK var forældet. Det samme var tilfældet når der blev forsøgt at etablere forbindelse til robot men denne gang var det Java som var forældet. Det var en Java version 1.5.

Projektpartner ville teste om der kunne etableres en forbindelse til robot på deres Windows computer. Det kunne godt lade sig gøre. Her var Java min. version 1.6. Det samme var tilfældet da der blev forsøgt at etablere forbindelse fra deres bærbar med Windows.

Projektpartner ville kontakte systemadministrator for en afklaring med hensyn til opdatering af Java og Netbeans/Java SDK på deres Linux computere.

6.12.1 Acceptance Tests

Alle acceptance tests blev endnu engang godkendt hvilket var forventeligt da det var sidste og afsluttende release. Alle Acceptance tests ligger i Bilag 7.2 "Kørte Acceptance Tests, Release 3".

6.12.2 Evaluering af Release 3 med Projektpartner

Projektpartner var meget tilfreds med release 3 og at systemet var brugbart for dem. De var især tilfredse med den grafiske visning af hjul og sensorer. Endvidere at det var blevet muligt at tilføje grænseværdier til udvalgte variabler og kunne overvåge dem på forsiden i deres eget lille vindue. Projektpartner syntes vi havde arbejdet godt med udviklingsprojektet og at vi havde nået et flot resultat. Et brugbart system som er nemt at videreudvikle og som let kan konfigureres.

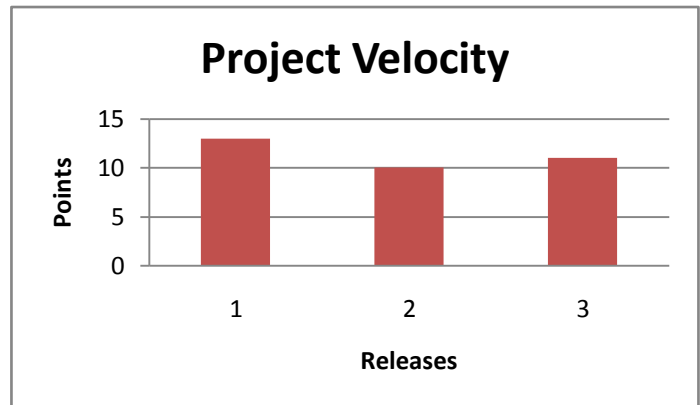
²⁴ **GUIDE: Creating Plugins for MARG** – Se Bilag

Vi har været glade for den gode introduktion til projektet og den løbende feedback vi har fået undervejs ved afslutning af et release. Det har gjort det meget nemmere at komme i gang med næste release.

6.13 Velocity Chart

Udarbejdet af: Kristina

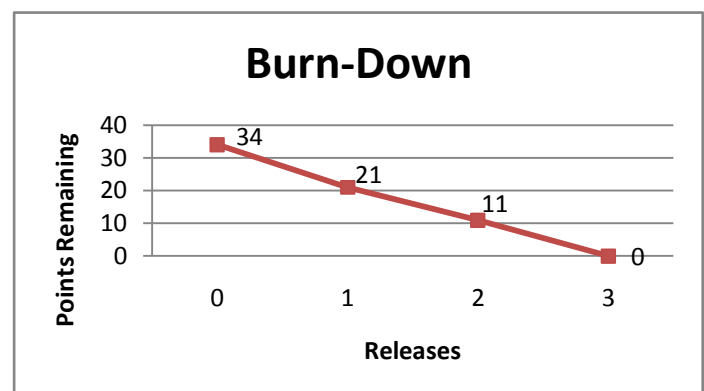
Dette er vores Velocity Chart efter Release 3. Vi steg en smule i udviklingshastighed i forhold til Release 2, men kom dog ikke op på den samme hastighed som Release 1. Dette kan skyldes at vi havde inddraget en ekstra user-story i Release 1, hvilket viste sig ikke at være et godt valg på grund af tidsmangel. Derfor har vores udviklingshastighed generelt ligget på nogenlunde samme niveau.



6.14 Burn Down Chart

Udarbejdet af: Daniel

Dette er vores burn down chart for Release 3. Det viser at vi har fået implementeret alle 34 pts. 13 pts i Release 1, 10 pts i release 2 og 11 pts i Release 3. Dermed er projektet færdigt.



Overvejelser

Vi har ikke haft så meget gavn af dette Burn Down chart da vi godt kunne holde overblikket over hvor langt vi var nået uden. Det var dog en god motivationsfaktor da vi kunne se projektet gik fremad og vi kom tættere på mål.

6.15 Evaluering af Release 3

Udarbejdet af: Bjørn, Daniel, Kristina

Dette release gik rigtig godt, og vi var hurtige til at implementere vores user-stories. Dette skyldes at vi nu var kommet ind i en rytme med hvordan og hvornår forskellige artefakter skulle laves, og kunne dermed hurtigere få lavet de forskellige dele.

Det blev ikke nødvendigt med Sekvens Diagrammer i dette Release, da vi ikke havde besvær med at holde overblik over koden.

I forrige Release fik vi ikke implementeret task card "Tilføj plugin til modul" i user historien "Tilføj modul". Så da denne måtte overflyttes til dette Release besluttede vi at lave den til en user-story for sig selv. "Tilføj plugin til modul" var også stor nok til at kunne være en user-story for sig selv og dermed var det et godt valg at gøre dette.

Vi forsøgte at tænke mere i dybden omkring hvordan vi ville implementere vores user-stories i dette Release, for bedre at kunne finde evt. JUnit tests. Dette resulterede også i at vi fandt to JUnit Tests vi kunne implementere.

Som implementeringen af de forskellige user-stories skred frem i dette Release, fik vi faktisk implementeret den Domæne Model vi havde lavet i XP prerelease. Vi lavede en klasse "Robot" og en klasse "Modul", som jo var de eneste to klasser vi havde identificeret i vores Domæne Model.

I forhold til vores SMR plugin fik vi både opfyldt projektpartners ønsker, og også demonstreret hvad der er muligt at lave ved hjælp af vores pluginstruktur.

Release 3 blev afslutningen på vores XP projekt da vi havde rykket vores Releaseplan en uge og dermed ikke havde ekstra tid til små ændringer. Men det gjorde ikke noget at vi havde rykket det, for vi fik implementeret alle de user-stories som var vigtige for kernesystemet og som var krav fra starten af. Det var også fint nok for projektpartner at vi rykkede det. Vi havde endda også tid til nogle mindre GUI ændringer i dette Release.

Der er stadig ting der kan videreudvikles på MARG men vi fik afleveret et generisk system med hele kernearkitekturen og ekstra plugin som projektpartner var meget begejstret for. De kan bruge MARG som det er og selv videreudvikle det og lave nye plugins.

Vi løb dog ind i problemer med hvilken Java version MARG kan køre på. Dette burde vi eller projektpartner nok have tænkt på at undersøge før vi gik i gang med systemet. På nuværende tidspunkt har projektpartner brug for at få deres Java version opdateret på deres Linux computere før de kan bruge systemet. Disse er nemlig deres primære arbejdscomputere. En anden gang vil det derfor have været det værd at lave en undersøgelse omkring dette fra starten af.

7. Diskussion og Konklusion

Udarbejdet af: Bjørn, Daniel, Kristina

Efter endt XP projekt er vi nu klar til at lave diskussion og konklusion af vores projekt og problemformulering. Først vil vi evaluere udviklingsforløbet og diskutere problemformulering og udviklingsmetoden. Dernæst vil vi evaluere produktet og lave konklusion af hovedopgaven.

7.1 Evaluering af udviklings projekt

Vi har brugt udviklingsmetoden XP til vores projekt og har med projektet fået en endnu bedre forståelse af udviklingsmetoden og hvorfor vi bør anvende en udviklingsmetode. Vi har løbende kommenteret vores overvejelser omkring brugen af XP i projektrapporten igennem de tre Releases og vil her beskrive de væsentligste.

Ekstra Artefakter

Ved valget af udviklingsmetoden XP vidste vi allerede at vi havde behov for at inddrage ekstra artefakter til at understøtte vores projekt. Vi havde en forventning om at en Domæne Model kunne give os et godt overblik af det overordnede design af arkitekturen til systemet. Det viste sig ikke at holde stik da systemet grundlæggende kun indeholdt 2 konceptuelle klasser, nemlig Robot og Modul. Vi kan dog efterfølgende konstatere at vores Domæne Model afspejler vores færdige system.

Et bedre overblik af systemet kunne vi derimod få ved anvendelsen af et Design Klasse Diagram som kunne understøtte brugen af CRC cards ved at give et bedre visuelt billede af kodens arkitektur. Samtidigt er det hurtigere at hente et Design Klasse Diagram frem end at skulle tage CRC cards frem på bordet igen. Vi synes at CRC cards er bedst at arbejde med da det er nemt at flytte rundt med kortene, mens et Design Klasse Diagram er bedre til at præsentere resultatet.

Vi synes at XP mangler et artefakt til at understøtte vores kode så vi kan få et bedre overblik af vores user-stories. Derfor valgte vi løbende at inddrage SD hvor vi fandt det nødvendigt. Det viste sig dog allerede i Release 1 at der måske alligevel ikke var behov for et eller flere SD da user-stories var nemmere at implementere end forventet. Vi besluttede dog at lave 3 SD efter at implementeringen var sket. Grunden til det var at vi i Projektgruppen er på forskelligt kodeniveau og for at sikre en god forståelse af koden hos alle blev SD altså lavet efter implementeringen. Vi havde dog behov for at lave et SD i Release 2 som gav os en bedre forståelse af en user-story før implementeringen.

Sammenfattende synes vi derfor at det var godt og hensigtsmæssigt at vi supplerede XP med de ekstra UML artefakter. Vi brugte dem på vores egen måde men de gav os en bedre forståelse for udvalgte dele af projektet hvilket også var formålet.

Vi supplerede XP med et ekstra artefakt i form af en Projekt Plan. Vi synes der var behov for at kunne følge den daglige udvikling af systemet. Dette var et godt valg da vi samtidig med at dokumentere alt arbejdet i XP udviklingsforløbet skulle dokumentere hele projektforsløbet i denne projektrapport.

Spikes, User-stories , Estimering og Releaseplan

Det stod klart med det samme at vi stod over for en vanskelig opgave med at skulle estimere user-story "Forbind til modul". Vi besluttede derfor at lave en spike investigation med det formål at afdække om det var muligt at oprette forbindelse til et modul på en robot og om vi kunne modtage data fra modulet. Forbindelsen skulle ske via en socket med host og port. Da vi ikke havde mulighed

for at oprette en forbindelse direkte til et robot modul, anvendte vi vores egen testserver. Vi fandt ud af at det at oprette forbindelsen ikke medførte problemer. Derimod lå der flere udfordringer i at læse modtaget data i XML og viderebehandle dette samtidig med at systemet skulle forblive modulært. Den anden spike investigation gik ud på at undersøge om det var muligt at sende en kommando til et robot-modul. Den var rimelig overkommelig da vi faktisk allerede havde fået afklaret at vi kunne oprette forbindelse til robot modul via en socket og modtage data tilbage.

De to spike investigations var trods alt en god hjælp til at vi kunne få estimeret user-story. Vi har dog senere i udviklingsprojektet været inde på muligheden for at opdele user-story "Forbind til modul", så der nemmere kunne laves en estimering.

Vi fik udviklet og implementeret alle user-stories. Vi lavede dog enkelte ændringer undervejs på nogle user-stories. "Tilføj modul" blev delt til 2 user-stories - "Tilføj modul" og "Tilføj plugin til modul". Det viste sig at være uhensigtsmæssigt at have "Tilføj plugin til modul" som et task card under "Tilføj modul" da det er forskellige funktionaliteter de repræsenterer.

Der blev også ændret i vores Releaseplan flere gange. Dette tilskrives i høj grad projektpartners ønsker om ændret prioritering undervejs i udviklingsforløbet. Vi gik på kompromis ved ikke at udvikle kernesystem i Release 2 til fordel for user-story "Vis grafer for SMR variabler".

Vi blev hurtigere færdige end forventet med at implementere user-stories i Release 1. Derfor gik vi i gang med at udvikle og implementere user-story "Vis variabeltræ" som var planlagt til Release 2. Det må vi konstatere var en fejl. Hensigten var at udnytte den ekstra tid til hurtigt at implementere en user-story som endda havde høj forretningsværdi og som var estimeret lavt. Det viste sig dog at tage længere tid end forventet hvilket betød at vi måtte udskyde en del af den sideløbende projektdokumentation. Vi var klar over at det ikke var efter XP forskrift at inddrage ekstra user-stories til en fastlagt Releaseplan og har derfor lært at det kan være meget uhensigtsmæssigt at bryde med en udviklingsmetodes principper.

Baggrunden for at vi blev fristet til at inddrage en user-story mere, når vi nu havde ekstra tid, skyldes at vi stadig som studerende ikke har den store erfaring med estimering. Havde vi estimeret korrekt i Release 1 så havde vi ikke kommet ud i den situation. Vi burde nok have opdelt user-story "Forbind til modul" til flere mindre da det ville have været nemmere at estimere.

Vores manglende erfaring med estimering har selvfølgelig medvirket til at vi har revideret vores Releaseplan en del gange i udviklingsforløbet. Dette skyldes også sygedage undervejs i projektgruppen. I Release 2 skyldes udskydelsen af user-story "Justér SMR variabler" at vi ikke havde fået afklaret kravene grundigt nok omkring højre/venstre hjul. Vi kunne ikke nå at indhente nye oplysninger til at kunne få user-story implementeret og derfor blev den flyttet til Release 3. Endvidere burde vi nok også havde medtaget det planlagte review med anden projektgruppe og lavet en estimering af review. Det havde givet et bedre samlet overblik af XP udviklingsforløbet og projektdokumentation i forhold til hvor meget tid vi skulle bruge totalt i Release 2.

Brugergrænseflade

Den udviklede brugergrænseflade til systemet fik vi afprøvet ved at inddrage nogle testpersoner. Vi havde i vores analyse af kvalitetsfaktorer vægtet brugervenlighed højest og vi valgte derfor at lave en brugergrænsefladetest i form af en think-aloud test. Resultatet bekræftede os i at vores udviklede

brugergrænseflade er brugervenlig. Vi havde dog kun fem testpersoner og udkast til brugergrænsefladen var allerede blevet diskuteret med projektpartner ved første møde. Det var selvfølgelig en fordel at vi fik så hurtig en feedback på brugergrænsefladen så vi havde nok kunne få mere udbytte af en brugergrænsefladetest hvis dette ikke var tilfældet.

Programmering

Selve programmeringen gik rigtigt godt i dette projekt. Vores valg om at inddrage design-artefakter i begyndelsen af projektet gjorde at vi allerede fra start havde et overordnet design at implementere systemet ud fra. Systemet blev primært udviklet ved hjælp af parprogrammering, kun meget få og mindre opgaver blev udviklet på egen hånd når projektgruppen ikke var samlet.

Vi fik i dette projekt udviklet nogle rigtigt gode værktøjer. Disse omfatter blandt andet XMLConfigManager, der kan hente, læse og omdanne en XML fil til konfigurationsinformation og PluginManager, der kan bruges til at finde klasser internt i den kompilerede kode som implementerer en given interface. Disse værktøjer var med til at vi overordnet set kunne opfylde projektpartners krav om at systemet skal kunne konfigureres til at køre på forskellige robotter og at der kan udvikles plugins til at understøtte en robots unikke funktionalitet. Vi havde ikke før udviklet et system hvor ny funktionalitet skulle kunne tilføjes efter at systemet er afleveret, så vi er glade for at programmeringen gik så fint at vi har kunnet aflevere et system med en fungerende plugin-struktur, der automatisk finder nye plugins og gør det muligt fra GUI og XML fil at tilføje dem til et modul. Samlet set er programmering gået over al forventning og vi er blevet bedre til at programmere i Java og til at implementere avancerede arkitekturer, som f.eks. vores plugin-struktur.

Refaktorering

Den førnævnte PluginManager var et meget vigtigt værktøj at få udviklet. Den gjorde at vi automatisk kunne finde plugins i vores system. Denne blev refaktoreret til at være en singleton, som alle klasser i systemet kan tilgå og få information omkring hvilke plugins der er til rådighed. En anden utility klasse, som vi lavede under refaktorering var Log, der fungerer som en udbyder af det centrale Logger objekt. Dette brugte vi i systemet til at skrive info og fejlbeskeder ud til systemudviklerne. Brugen af denne hjalp til hurtigt at kunne lokalisere eventuelle fejl og få et overblik over hvad der sker bag GUIen når systemet kører. Vi lavede undervejs i dette projekt mange mindre refaktoreringer der ofte gik ud på at dele store metoder op i mindre dele eller hive fælles implementering ud i sin egen klasse. Nogle steder indsatte vi handler klasser imellem GUI og de klasser GUIen brugte, som foreskrevet i Model-View-Control arkitekturen. Alt dette hjalp med at holde en høj binding og lav kobling imellem vores klasser hvilket tilsammen har gjort vores færdige system mere fleksibelt og modulært. I det hele taget synes vi at vi har fået brugt refaktorering godt og brugen af det har undervejs forbedret vores design betydeligt.

Test

Allerede ved beskrivelsen af risici til XP projektet var vi opmærksomme på den manglende korrekthed ved brug af få Unit tests. Vi havde en forventning om at det nok ville blive svært at kunne lave Unit tests til alt, da det var en brugergrænseflade vi skulle udvikle. Det bryder med princippet i XP udviklingsmetoden som netop foreskriver brugen af test før implementering. Vi har derfor i stedet for lagt vægten på at få lavet detaljerede Acceptance Test, hvilket vi faktisk også mener, er den rigtige

fremgangsmåde når det er en brugergrænseflade vi har at gøre med. For at kompensere de manglende Unit test har vi lavet main metoder til hver GUI klasse for at teste om de virker hver for sig.

Velocity og Burn down Charts

Vi fik ikke det store udbytte af at benytte Velocity Chart og Burn Down Chart. Forklaringen er at vi ikke havde problemer med at bevare overblikket over de udviklede user-stories og task cards. Ser vi på Velocity Chart skyldes det nok også at vi kun havde en iteration i hvert Release. Havde vi haft flere iterationer i hvert Release så havde det været et godt artefakt at have med. Med Burn Down Chart ville udbyttet også have været bedre hvis udviklingsforløbet havde strakt sig over en længere periode og indeholdt flere estimerede point. Vi synes dog stadig at et Burn Down Chart er berettiget i et XP udviklingsforløb da det kan være en god motivationsfaktor, fordi man kan se at man kommer tættere på målet.

Releases på DTU

De 3 Releases som vi havde planlagt i XP udviklingsforløbet gik alle rigtig godt. Vi har fulgt XP udviklingsmetoden som den foreskrives så godt som muligt og samlet set mener vi det er lykkedes for os. Efter Release 2 havde Projektpartner et brugbart system som kunne afvikles på en robot. Alle vores acceptance tests bestod. Vi havde en god kontakt og dialog med Projektpartner igennem hele forløbet og synes virkelig at vi fik en god feedback på det vi havde udviklet, som vi kunne bruge fremadrettet i udviklingsforløbet.

Review

Uden den store sammenhæng til XP udviklingsforløbet valgte vi undervejs at lave et review med en anden projektgruppe. Det viste sig desværre at vi måtte gå på kompromis med XP udviklingsmetoden, om at vægte udvikling af planlagte user-stories højest, for at vi kunne få tid til at lave dette review. Men vi synes det var en god beslutning for vi mener at det gav os mulighed for at reflektere over det vi havde dokumenteret i projektrapporten og dermed også XP udviklingsforløbet. Feedback fra den anden gruppe var med til at bekræfte mange af vores beslutninger og overvejelser.

Projekt vejledning

Løbende under hele projektforsløbet har vi fået feedback af vores projektvejleder. Det var en stor hjælp at projektvejleder var med tidligt i forløbet, da vi derfor hurtigere kunne tage beslutninger om hvilken vej vores udviklingsprojekt skulle gå. Vi synes vi har fået en konstruktiv feedback under hele forløbet med mulighed for at diskutere enkelte emner. Dialogen til projektvejleder har været god.

7.2 Diskussion af Problemformulering

I dette afsnit vil vi gennemgå vores problemformulering og diskutere om vi fik løst denne og hvordan vi i så fald gjorde. Vi vil gøre dette en del af gangen. Den del vi vil diskutere står først i kursiv og så kommer vores diskussion under denne.

Hypoteser

"Det er muligt at lave en grafisk brugergrænseflade som en Java Applet der kan kommunikere med en DTU Elektro robot via socket forbindelser. "

Denne hypotese kan vi nu bekræfte, da det *var* muligt for os at lave en grafisk brugergrænseflade som en Java Applet der kan kommunikere med DTU Elektros Robotter. Forbindelse blev oprettet via socket forbindelser. Vi lavede vores egne test servere som vi brugte mens vi udviklede systemet på skolen og testede systemet på de rigtige robotter når vi var på DTU. De tests vi lavede af brugergrænsefladens brugervenlighed sikrede os også at det blev nemt for projektpartner og andre eventuelle brugere at benytte systemet.

"Det er muligt at lave systemet bag brugergrænsefladen så modulært at det kan understøtte DTU Elektros nuværende samt nye robotters unikke funktionalitet."

I det vi fik lavet et meget generisk værktøj som kan bruges på nuværende tidspunkt og også videreudvikles senere, kan denne hypotese også bekræftes. Vi fik afprøvet MARG på 3 forskellige robotter og vores opbygning af systemet burde sikre at det kan bruges til alle robotter der kommunikere i XML og understøtter Java. Plugin strukturen sikrer at der kan tilføjes nye plugins på en nem måde. Dermed kan vores system understøtte fremtidige og unikke funktionaliteter.

"Der findes en systemudviklingsmetode der passer godt til at udvikle dette system."

XP passede rigtig godt til at udvikle MARG, derfor kan denne hypotese bekræftes. Det var rigtig godt at vi fik analyseret forskellige udviklingsmetoder i begyndelse af denne hovedopgave, da det hjalp os til at finde frem til det helt rigtige styreledskab til dette projekt. XP passede godt fordi vi hurtigt kunne få udviklet en prototype så projektpartner kunne få en idé om hvordan systemet skulle være, og dermed give feedback til de forskellige Releases. Der kunne udvikles nye user-stories til hvert Release hvis projektpartner havde nye krav, og prioritering af hvilke krav der blev udviklet først kunne hurtigt ændres.

Uddybende spørgsmål

"Kan der laves en brugergrænseflade, som kan konfigureres til at bruges på alle DTU Elektros robotter?"

Vi har udviklet vores brugergrænseflade til at kunne konfigureres i en XML fil, til hvilken ip og port der skal skabes kontakt til for hvert modul. Dette er den information der varierer for hver robot på DTU Elektro der skal oprettes forbindelse til. Derfor kan spørgsmålet bekræftes. Da vi har haft mulighed for at teste MARG på 3 robotter forventer vi det kan bruges på alle robotter der kommunikere i XML. Hvis en robot kommunikere i et andet sprog end XML er det muligt at videreudvikle systemet til at understøtte det. Dette er muligt da systemet er modulært nok til at der kan videreudvikles uden at lave om på det der i forvejen virker.

"Kan en brugergrænseflade udvikles så den kan afvikles i alle internet browsere som har Java installeret?"

Da vi har udviklet vores system som en Java Applet kan den afvikles i alle internet browsere der har Java 1.6 installeret. Derfor kan dette spørgsmål kun tildels bekræftes. Ved afprøvning af systemet på DTU fandt vi frem til at der er behov for minimum at bruge Java 1.6 for at kunne køre det. Dette burde dog ikke være et problem da det er gratis for brugere at opgradere Java og Java 1.6 har været udgivet længe.

"Hvordan kan hypoteserne og spørgsmålene besvares?"

For at be- eller afkræfte vores hypoteser skal vi have viden om Java, socket forbindelser og XML. Vi skal også have viden omkring interfacet til de forskellige robotter på DTU Elektro og hvordan man udvikler en plugin arkitektur.

Vi mener vi har viden og erfaring nok omkring Java, socket forbindelser og XML. I forhold til interfacet til robotten skal vi tilegne os viden fra vores projektpartner. Viden omkring plugins skal vi tilegne os fra vores projektparter, vores vejleder eller fra anden IT relateret kilde.

Vi har viden om flere udviklingsmetoder fra vores uddannelse som gør os i stand til at fortage et metodevalg for dette projekt."

Det viste sig at vores fælles viden omkring Java, socket forbindelser og XML var tilstrækkelig for at kunne udvikle MARG. Vi fik nemt og hurtigt den viden vi havde brug for omkring forbindelse til robotten fra projektpartner og den viden vi havde brug for, for at designe en plugin struktur havde vi faktisk selv meget af i forvejen. Det vi skulle undersøge var om hvordan man gennemfører de forskellige pakker i systemet for plugins når systemet bliver startet og kører i en Applet. Dette fandt vi information om på internettet²⁵

7.3 Diskussion af udviklingsmetode

Valget af XP som vores udviklingsmetode for MARG viste sig at være et godt valg. Et af kravene fra både os og projektpartner til udviklingsmetoden var at der skulle kunne laves tidlige og regelmæssige prototyper. Dermed passede XP godt til dette udviklingsprojekt og der var mulighed for ofte at give feedback omkring systemet.

Vi har forsøgt at følge XP så meget som muligt og havde kun nogle enkelte afvigelser fra principperne i XP. Da vi mener at en udviklingsmetode ikke skal afgøre systemets implementering, men derimod være et redskab til at styre udviklingen af systemet, vil der nok af og til være nogle små afvigelser fra udviklingsmetodens principper. Vi vil ikke gå på kompromis med implementering af systemet for at kunne overholde udviklingsmetodens principper.

Vi mener også vi fik kompenseret for de dele af XP som vi ikke benyttede under udviklingsforløbet. Vi havde fokus på lave vores Acceptance Test mere grundige og testede GUI klasser ved hjælp af *main* metoder, da vi ikke kunne lave så mange Unit tests. Vi lagde vægt på at refaktorere koden løbende for at styrke designet og modulariteten af MARG. Her var det også en hjælp at inddrage et Design Klasse Diagram der gav os et godt overblik over designet af arkitekturen.

Vi lærte af de afvigelser til XP som vi lavede. Det var en fejl at inddrage en ekstra user-story i et Release, men det var tværtimod godt at inddrage nogle ekstra UML artefakter.

Det var rigtig godt at vi brugte tid på at finde frem til hvilken udviklingsmetode vi skulle anvende til vores projekt. Vi mener ikke at vi kunne have anvendt en anden udviklingsmetode lige så effektivt når vi ser tilbage på udviklingsforløbet. XP viste sig endvidere at være fleksibel når vi havde behov for det, i det at vi kunne inddrage ekstra artefakter.

²⁵ **Finde klasser i en jar fil** – Se litteraturliste

7.4 Evaluering af produktet

7.4.1 Produktets funktionalitet

Som et færdigt produkt, blev vores system meget omfangsrigt. Det endte med at indeholde den oprindeligt ønskede funktionalitet og opfyldte også langt de fleste af de krav som projektpartner løbende fandt frem til. Systemet kan forbinde til flere moduler på en robot og kan konfigureres ved hjælp af en XML fil. Grundlæggende indeholder systemet kontrolknapper, menu for robotens moduler, mulighed for at ændre hvert moduls opsætning og direkte adgang til at skrive kommandoer og læse modtaget data til det enkelte modul.

Derudover fik vi udviklet nogle rigtigt gode grafiske værktøjer til at overvåge variabler, som bliver modtaget fra hvert modul på roboten. Dette gør det meget mere brugervenligt og nemt at arbejde med robotten end det før var for projektpartner.

Hvad der virkeligt er værd at understrege omkring vores produkt er dets arkitektur og design. Denne gør det muligt ved hjælp af en pluginstruktur at videreudvikle vores leverede produkt udover dets nuværende krav. Systemet er på den måde et generisk værktøj, der gør det muligt senere i produktets levetid at understøtte nye krav. Helt konkret er der i vores system mulighed for at udvikle plugins der læser XML data og plugins der præsenterer data grafisk i et modul.

Vi fik udviklet 2 plugins til at læse XML; RawDataPlugin og VarDataPlugin, som blev brugt i udviklingen af den fælles funktionalitet for alle moduler.

SMRPlugin blev vores eneste plugin til et modul, men dets funktionalitet viste hvor meget der kan gøres med vores pluginstruktur. SMRPlugin kan modtage data om hjulomdrejninger og hastighed og vise denne både som tekst og som et billede af robotens hjul der roterer. SMRPlugin kan endda ved hjælp af en slider sende kommandoer til robotten, der indstiller hjulenes hastighed.

Vi er meget glade og stolte over at kunne aflevere ikke blot et færdigt produkt, men et generisk værktøj, som projektpartner kan bruge og videreudvikle i den retning de ønsker.

7.4.2. Manglende funktionalitet

Vi fik ikke udviklet alle de klasser vi havde med i vores Design Klasse Diagram. Alternativet til XMLClient; MRCCClient blev aldrig implementeret. Dette skyldes dog at den ikke blev prioriteret så højt fra Projektpartner og derfor blev sat på standby. Vores forståelse i starten for MRCCClient gik på at det var et specielt sprog der skulle konverteres, men det viste sig på nærmere forespørgsel at det bare var ren tekst. Derfor havde vi indarbejdet MRCCClient og SMRCLParser i vores Design Klasse Diagram, så vores arkitektur kunne understøtte udviklingen af disse. Hvis Projektpartner senere ønsker at implementere MRCCClient kan det med rette gøres da vores arkitektur i høj grad understøtter dette.

7.4.3 Videreudvikling af produktet

Ud fra Evaluering af XP, Diskussion af problemformulering og feedback fra projektpartner har vi lavet en liste over ting der kan videreudvikles på MARG. Denne kan bruges af projektpartner og studerende på DTU der skal arbejde med MARG.

Videreudvikling af kernesystemet

Vis andet faneblad ved start end Control

Projektpartner var ikke så meget for at det var Control siden man så først når man åbnede MARG. Derfor ville det være godt at lave om på rækkefølgen af hvilke sider der bliver vist først. Det kunne evt. udvikles således at et plugin som SMR blev vist først.

Konfigurering af Monitored Variabler

Projektpartner ville gerne at man kan konfigurere hvilke variabler der skal vises i Monitored Variables, sådan at man ikke skal ind og vælge de samme hver gang MARG åbnes.

Konfigurering af Variabler til Status

Projektpartner så også gerne at variabler der skal overvåges på Status fanebladet kunne konfigureres fra starten. Dermed er variabler man altid vil overvåge allerede på Status fanebladet når MARG åbnes.

Automatisk opdagelse af XMLParsePlugins

Det kunne være smart hvis systemet også automatisk kunne finde XMLParsePlugins ligesom med ModulePlugins. Dette kunne implementeres ved at bygge videre på PluginManager klassen.

Tilføjelse af genvejstaster i GUI

For brugere der ikke ønsker at benytte musen så meget ville det forbedre brugervenligheden af systemet at tilføje flere genvejstaster til GUI.

MARG på mobil

Projektpartner mente det kunne være smart ikke at skulle have en stor skærm med ud, når en robot skulle køres. Det ville være nemmere at kunne aflæse og sende kommandoer på en mobiltelefon.

Nye plugins

Følgende er ny funktionalitet til systemet, som vi mener, kan udvikles ved hjælp af vores plugin struktur. For det meste vil udviklingen af disse bestå af et XMLParsePlugin der fortolker ny XML data og et ModulePlugin der abonnerer på og præsenterer denne data.

Image plugin

Nogle af DTU Elektros robotter har et kamera installeret. Der kunne udvikles et plugin der viser billederne fra dette kamera.

Map plugin

De fleste af robotterne på DTU Elektro indeholder data om robottens placering og omgivelser, hvilket kunne modtages fra robotten og tegnes som et kort.

Laser Scanner plugin

Et plugin der kan teste om laser scanneren virker. Til at starte med kunne det bare være til at se om der leveres noget data. Senere kunne dette præsenteres som et billede eller graf.

Højtalere plugin

Et plugin kan udvikles der kan indstille og sende kommandoer til SMR robottens højtalere.

Motor Status plugin

Et plugin der visuelt viser status for robottens motor kan udvikles.

Robot Test plugin

Der kunne udvikles et plugin der ved tryk på en knap udfører en række test på robotten og overvåger om nogle udvalgte variabler ændrer sig som forventet.

7.5 Konklusion af hovedopgave

Vores problemformulering består af nogle hypoteser og uddybende spørgsmål som vi her vil liste op og konkludere på.

"Det er muligt at lave en grafisk brugergrænseflade som en Java Applet der kan kommunikere med en DTU Elektro robot via socket forbindelser. "

Ja. Systemet kan kommunikere med DTU Elektros robotter og fungere som en Java Applet.

"Det er muligt at lave systemet bag brugergrænsefladen så modulært at det kan understøtte DTU Elektros nuværende samt nye robotters unikke funktionalitet."

Ja. Vi har lavet et generisk værktøj som understøtter DTU Elektros nuværende robotter og systemets plugin struktur samt konfiguration understøtter nye robotters unikke funktionalitet.

"Der findes en systemudviklingsmetode der passer godt til at udvikle dette system."

Ja. Vi synes XP passede rigtig godt til vores projekt og de krav som vi og projektpartner stillede til udviklingsmetoden.

"Kan der laves en brugergrænseflade, som kan konfigureres til at bruges på alle DTU Elektros robotter?"

Ja. Systemet kan konfigureres i en XML fil. Dermed kan det tilpasses til at fungere på alle deres robotter som tilgås via en socket forbindelse og som kommunikerer i XML.

"Kan en brugergrænseflade udvikles så den kan afvikles i alle internet browsere som har Java installeret?"

Ja. Den udviklede Java Applet kan afvikles i alle internet browsere som har Java installeret. Dog minimum Java version 1.6.

Vi synes vi har fået lavet en fyldestgørende hovedopgave der demonstrerer vores brede kompetencer opnået gennem uddannelsen til datamatiker.

Samtidig har vi fået udviklet en grafisk brugergrænseflade der opfylder projektpartners krav.

Brugergrænsefladen er lavet så modulær og generisk at den umiddelbart kan anvendes og videreudvikles af DTU Elektro.

8. Litteraturliste

DTU Elektro

www.DTU.dk
www.Elektro.DTU.dk

eXtreme Programming

<http://www.extremeprogramming.org/>

Finde klasser i en jar fil

<http://forums.sun.com/thread.jspa?threadID=341935&start=15>

Kritik af XP

http://www.softwarereality.com/lifecycle/xp/safety_net.jsp

Professional Systems Development

Prentice Hall – Niels Erik Andersen, Finn Kensing ...

Repository Model

[http://en.wikipedia.org/wiki/Subversion_\(software\)](http://en.wikipedia.org/wiki/Subversion_(software))

SCRUM

<http://da.wikipedia.org/wiki/Scrum>
<http://www.controlchaos.com/about/>

Software Development Methodology

http://en.wikipedia.org/wiki/Software_development_methodology

Spiral Modellen

http://www.computer.org/portal/cms_docs_computer/computer/homepage/misc/Boehm/r5061.pdf

Unprecedented systems

Artikel af Richard J. Sylvester, The MITRE Corporation og Marilyn J. Stewart, Logicon Inc.

UPEDU

<http://www.upedu.org/upedu/>

User Interface Design- A software Engineering perspective

Addison -Søren Lauesen

V-Model

<http://en.wikipedia.org/wiki/V-Model>

XP CRC Cards

<http://www.extremeprogramming.org/rules/crccards.html>

XP Iterations Plan

http://www.mayford.ca/xp/iteration_plan.html

XP Metafor

<http://www.mayford.ca/xp/metaphor.html>

XP Parprogramming

<http://www.mayford.ca/xp/pairprogramming.html>

XP Refaktoring

<http://www.mayford.ca/xp/refactoring.html>

XP Releaseplan

http://www.mayford.ca/xp/release_plan.html

XP Spikes

<http://www.mayford.ca/xp/spike.html>

XP Task Cards

<http://www.extremeprogramming.org/rules/iterationplanning.html>

XP User-stories

<http://www.mayford.ca/xp/stories.html>

XP Velocity

<http://www.mayford.ca/xp/velocity.html>