Queuemonitoring med IBM Websphere MQ, for Region Syddanmark.

En hovedopgave skrevet af Kim Hansen og Christopher Buch

Queuemonitoring with IBM Websphere MQ for Region Syddanmark

A final project by Kim Hansen and Christopher Buch

Indhold

Synopsis 6

Introduction (English)(Kim) 7

Indledning (Kim+Christopher) 7

Problemformulering(Kim+Christopher) 8

Begrænsning af omfang(Kim + Christopher) 8

Virksomheden(Kim + Christopher) 8

Beskrivelse af Websphere MQ(Kim + Christopher) 9

Kravspecifikation(Kim + Christopher) 10

- Global Monitor 10
- Message Queue 10
- Cloverleaf 10
- FURPS 11
  - Functionality 11
  - Usability 12
  - Reliability 12
  - Performance 12
  - Supportability 13

Risikoanalyse(Kim + Christopher) 13

- Risiko: Manglende erfaring blandt udviklingerne 14
  - Løsning: 15
- Risiko: manglende kontakt med kunden. 15
  - Løsning: 15
- Risiko: Software har for stort ressource brug. 16
  - Løsning: 16
- Risiko: Software har en tendens til at 'crashe'. 16

løsning: 16
Risiko: Manglende dokumentation ved projektet afslutning. 16
Løsning: 16
Risikoestimering 17
Arkitektur(Christopher) 18
MVC Teori 19
Model 19
Controller 19
View 20
MVC i praksis 20
Modellaget 21
Controlleren 21
Viewet 22
Arkitektur konklusion 22
Valg af udviklingsmetode(Kim) 23
Konklusion 26
User Stories(Kim + Christopher) 27
Product Backlog(Kim) 29
Estimering af opgaven 31
Planlægning(Kim+ Christopher) 33
Cost-Time-Scope analyse(Christopher) 35
Cost Time Scope Teori 35
Cost 35
Time 35
Scope 36
Cost Time Scope Praksis 36

Time 36
Scope 37
Cost 37
Sprint backlog(Kim) 37
Projektforløbet(Kim + Christopher) 38
Valget af opgaven 38
Opstartsperioden 39
Starten på sprint forløbet 39
Sprint 1 40
Sprint 2 40
Sprint 3 41
Sprint 4 41
Projektafslutningen 42
Sprints(Kim) 42
Sprint 1 backlog 42
Sprint 1 konklusion 42
Sprint 2 backlog 43
Sprint 2 Konklusion 44
Sprint 3 backlog 44
Sprint 3 konklusion 45
Sprint 4 45
Sprint 4 konklusion 46
Konklusion på sprint forløbet 46
Sekvensdiagrammer(Christopher) 48
Sekvensdiagram til Page_Load() 49
Sekvensdiagram til Timer1_Tick() 51

Programmering(Kim + Christopher) 52
Test 52
Controlleren: View.Aspx.cs 53
Page_Load 54
ReadQueue 55
SendToView 57
Outputtet fra SendToView 58
Timer1_Tick 59
ModelMQConnect klassen 61
ConnectMQ metoden 63
DisconnectMQ metoden 64
ReadQueueDetails metoden 65
ModelMQReader 66
ModelXMLReader 67
ModelXMLQueue 70
Løsning på problemet med Tråde 70
Diskussion(Kim + Christopher) 72
Hvad er gået godt 72
Hvad er gået skidt 72
Hvad kunne vi have gjort bedre 73
Hvad kunne de have gjort bedre 74
Afsluttende konklusion(Kim + Christopher) 75
Kildehenvisninger 78

Queuemonitoring med IBM Websphere MQ

En hovedopgave til datamatiker studiet af:

Christopher V. Buch

Kim Hansen

Synopsis

Dette projekt der foregik i perioden fra d.27/10 2014 til d.5/1 2015, omhandler opbygningen af et program til overvågning af Regions Syddanmarks Regional IT's Websphere MQ queues. I løbet af projektet vil vi forklare hvordan et sådan program kan skabes, og hvilke udfordringer der er ved udforme et sådant projekt for Regional IT's integrationsafdeling.

Introduction (English)(Kim + Christopher)

This is a project done for Regionalt IT's integration department, which is a part of Region Syddanmark. Region Syddanmark is the name of the government public company taking care of healthcare and public services in the southern region of Denmark.
The project is aimed at creating a queue monitoring program, for Region Syddanmarks message queue system, which uses IBM Websphere MQ. The system is to be used internally within the IT department, to ensure that Websphere MQ, and all its queues are running smoothly at all times.
The system should be able to alert the IT departments employees in an obvious and visual manner, if one or more of the Websphere MQ queues fails to deliver its messages for whatever reason.
The program was originally intended to be made using Java as the programming language, but due to some issues at the startup phase, we switched to C# and ASP.NET. The switch was made in agreement with the IT department, and although it wasn't optimal for them, the switch was deemed necessary for the project to succeed.
We used SCRUM as our development method, and although we of course couldn't run true SCRUM we did use several of the tools SCRUM provides for an example sprints and product backlogs. The choice of SCRUM as our development method came as a result of us feeling more confident with SCRUM, than with other methods.

Indledning (Kim+Christopher)

I dette projektforløb har gruppen arbejdet for Region Syddanmarks Integration og Planlægningsafdeling. Vi er blevet stillet til opgave at lave et program, til overvågning af Regionens message queue, vi skal såfremt vi bliver færdige med den opgave, udvide systemet så det også kan vise servernes ressourceforbrug. begge opgaver skal kunne vises gennem regionens Global monitor som en widget.

Vi har valgt at bruge de engelske udtryk associeret med Websphere MQ, da vi mener dette giver bedre forståelighed for emnet. Dette betyder blandt andet, at vi bruger queues i stedet

for køer, queue manager i stedet for kø passer. Da disse har dobbelt betydning på dansk, dette gøres for at undgå at der opstår misforståelser.

Problemformulering(Kim+Christopher)

Gruppen blev stillet for opgaven at lave et overvågningsprogram i Java, som kan overvåge Message queues i regionens instans af queue managing softwaren Websphere MQ. Programmet skal være i stand til at se, hvorvidt meddelelser har lagt i deres queues for længe. Hvis meddelelser har lagt for længe i dens queue, skal programmet være i stand til at vise dette, på en måde hvorpå det er tydeligt for afdelingen, at der nu bør gøres noget, for at få meddelelserne sendt ud til deres modtagere.

Yderligere blev gruppen præsenteret for opgaven om at udvide ovennævnte program, så det også kan holde øje med, hvorvidt regionens servere kører som de skal. Her er tiltænkt at programmet skal kunne måle, hvor meget af servernes cpu-kraft, ram og diskplads bliver brugt. Til dette var det tiltænkt at der i programmet, skulle læses fra et andet program, kaldet Zabbix, som er beregnet til at overvåge ressourceforbrug for programmer i netværk og applikationer.

Det er tiltænkt at programmet der udvikles, som skal kunne læse fra Websphere MQ og Zabbix, skal laves som en jsp-løsning, sådan at det kan køre på regionens Global Monitor.

Begrænsning af omfang(Kim + Christopher)

Da programmet skal køre på interne systemer, skal vi ikke tage specielle hensyn til sikkerhed, eller for eksempel login da systemet skal køre på et internt system.

Programmet skal laves som en selvstående løsning, altså skal programmet ikke deles op i f.eks. en client server løsning.

Virksomheden(Kim + Christopher)

Region Syddanmark er en af regionens største arbejdspladser, med mere end 30.000 ansatte. Inden for dette ligger Regional IT, som en større afdeling der tager sig af IT-løsninger til sundhedssektoren i regionen. Dvs. alle offentlige hospitaler i regionen, samt lægehuse og lægeklinikker. Herunder er der yderligere underafdelinger: IT-Drift og Test og

implementering. Under IT-drift er Integration og planlægning, som ledes af Henrik Juul Andreasen.

Denne underafdeling tager sig af programmering af udvidelser til sundheds IT i regionen og yder også support til disse programmer, når der opstår problemer med dem.

Regional IT kører officielt med deres egen udviklingsmetode, men den minder meget om udviklingsmetoden kaldet 'Kanban'.

Kanban er en udviklingsmetode der fokusere på inkrementel og evolutionær udvikling. Det vil sige man fokuserer på at videreudvikle og forbedre eksisterende systemer med små skridt, i stedet for store og måske mere effektive ændringer, da disse har en større chance for at fejle. Dette er kritisk da regionalt IT håndterer sundhedsvæsenets it-systemer. En længerevarende fejl på disse systemer kan have fatale konsekvenser for regionens borgere.

Regional IT er dog begyndt at tage skridt imod indførsel af SCRUM, som udviklingsmetode. Det sker pga. den meget store arbejdsbyrde og de dertilhørende forsinkelser der plager regionen, de har blandt andet. indført et ugentligt SCRUM-møde. Det er tiltænkt at man vil have daglige møde, et såkaldt 'standup meeting' i fremtiden.

Beskrivelse af Websphere MQ(Kim + Christopher)

Websphere MQ er et message queue system, som ligger mellem afsender og modtager af meddelelser i interne systemer.

Grunden til man benytter et message queue system, er for at undgå at meddelelser går tabt, såfremt en modtagers server er nede. Hvis en modtager server er nede, vil beskeden blive ved med at ligge i en queue i Websphere MQ, indtil modtager serveren kommer op at køre igen.

Systemet kan holde øje med mange detaljer omkring beskederne, heriblandt hvem afsenderen er, hvornår meddelelsen blev modtaget i queuen, hvor stor meddelelsen er og hvor det er tiltænkt at beskeden skal sendes hen.

Websphere MQ indeholder også en funktion der kan tage sig af meddelelser, som er for store til at blive sendt igennem, sådan at disse ikke lukker flaskehalsen mellem afsendere og modtagere.

Websphere MQ's opbygning består af queuemanagers, som så indeholder diverse lokale og remote queue. Normalvis benyttes kun en queuemanager, men der er dog mulighed for oprettelse af flere i systemet.

Vores opgave med henhold til Websphere MQ bliver at oprette forbindelse til denne og derefter bruge dens API til at lave forespørgsler til de aktuelle queue alder og dybde.

Kravspecifikation(Kim + Christopher)

I de følgende afsnit vil vi dække tre stykker software, som Region Syddanmark benytter og for forståelsens skyld beskriver vi dem her først. Derefter vil vi beskrive vores kravspecifikation videre via FURPS-modellen.

Global Monitor

Global Monitor er det system der varetager regionens overvågning af systemer, som benyttes af Region Syddanmarks sundhedsstab.

Message Queue

Message Queue systemet er et meddelelsessystem, som kan sætte de mange meddelelser der sendes fra regionens hospitalers IT-systemer i queue. Grunden til at benytte en MQ server, er at webservices kan gå ned og såfremt en meddelelse sendes til en webservice, som er gået ned, ville den kunne gå tabt. Hvis man til gengæld benytter en MQ server, kan beskeden blive lagret der, indtil servicen er online igen, eller parat til at modtage flere meddelelser.

Cloverleaf

Cloverleaf er en messagebroker udviklet af det amerikanske firma Infor. Systemet benyttes til oversættelse og logning af meddelelser mellem forskellige systemer. I Region Syddanmarks tilfælde, benyttes den til at oversætte og logge meddelelser, sendt fra de forskellige systemer, på de forskellige afdelinger, på regionens hospitaler.

Logging delen foregår i en SQL-Server Database, som opbevarer information om meddelelser der er sendt og modtaget. Databasen gemmer desuden information omkring meddelelser som ikke er nået frem til de tiltænkte modtagere og meddelelser der stadig ligger i queuen.

FURPS

I vores kravspecifikation har vi tænkt os at anvende FURPS modellen, en metode kendt til at udspecificere dele af kravene til et system.

Vi har valgt at bruge FURPS fordi vi synes det giver et meget simpelt men stadig godt overblik over hvad vi forventer det endelige system skal kunne.

FURPS hjælper med at uddelegere de enkelte områder af udviklingen og de krav der stilles dertil. Eksempelvis er usability medhjælpende til at sikre at systemet har en god brugerflade, som er nem at forstå og nem at gå til.

FURPS står for:

F: Functionality - der beskriver de ønskede funktioner i systemet, samt hvad der er ønsket at systemet kan arbejde sammen med.

U: Usability - beskriver kravene til programmets brugervenlighed

R: Reliability - beskriver kravene til programmets drift sikkerhed

P: Performance - beskriver kravene til for eksempel svartider, eller ressourcebrug.

S: Supportability - forklarer til hvilket niveau, man ønsker at gøre det er let for andre udviklere at sætte sig ind i programmet bagefter. Det kan være blandt andet involveret arkitektur og den ønskede coupling mellem lagene i systemet.

Functionality

funktionaliteten i softwaren, skal arbejde sammen med Regional IT's Message Queue system (MQ).

Dertil skal der laves et program, som kan se på hvad der ligger af beskeder i MQ, hvor lang tid de har lagt der og hvor mange meddelelser der måtte ligge der.

Det er forskellige krav til hvor ofte de enkelte servere skal kaldes, da nogle beskeder kan være akutte mens andre beskeder godt kan vente lidt længere, derfor er der forskellige intervaller for hvornår serverne skal kaldes for at sikre at informationen på skærmen er tidssvarende, det vil sige at informationen der står på tavlen skal være pålidelig.

De tiltænkte intervaller er henholdsvis 30 sekunder og op til et minut. der kan senere blive tale om en udvidelse til systemet hvor der skal opdateres data fra MQ hvert femte sekund.

Usability

Programmet skal have en simpel brugerflade så man nemt kan se om alle MQ queues kører som de skal, og at der ingen problemer er med dem. Brugerfladen skal kunne informere regionens ansatte om hvor mange meddelelser der ligger i MQ og hvor lang tid de har lagt der.

Det skal være muligt at ændre font størrelse og farve. Andre elementer skal også kunne ændres, det skal gøres fordi brugerfladen skal køres på en monitor med en 4k opløsning, og kunden ønsker en 'homogen og branded brugergrænseflade'.

Hertil kommer farvekodning, så at Regional IT kan se status på servere og på MQ queues. Her tænkes blandt andet at der kan varsles med grøn, når der ingen problemer er, gult når der er mindre forsinkelser på meddelelser og rød, såfremt servere er gået ned eller meddelelser ligger og hober sig op i længere queues. Det er et krav at der automatisk bliver sendt en alarm e-mail hvis der er et kritisk problem.

Reliability

Informationen som står på tavlen skal være korrekt og pålidelig. Dermed skal informationen også være tidssvarende. Du skal kunne stole på at informationen vist på tavlen er opdateret indenfor de givne intervaller.

Performance

Programmet skal foretage serverkald til Websphere MQ servere i forskellige tidsintervaller. det er vigtigt at systemet er i stand til at kontakte Websphere MQ serverne og opdatere informationen på tavlen henholdsvis hvert minut og hver 30. sekund.

Der må ikke være nogen videre impact på performance af MQ queues, der er tale om max 5 % overhead.

Supportability

Det er et ønske fra kunden at brugerfladen skal kunne skiftes ud hvis man finder en bedre visuel løsning. Det vil sige at, informationen skal være nem at sende videre til en anden brugerflade end den leveret af udviklerne, som i dette tilfælde er os.

Dette sikres via vores valgte arkitektur som er MVC

Der skal samtidigt være god dokumentation af projektet, da vi formentlig ikke vil være dem der skal vedligeholde projektet efter levering.

Dokumentationen skal både beskrive funktionaliteten og hvad tiltænkt brug er. Dette i en grad så udenforstående hurtigt vil kunne sætte sig ind i hvad programmets funktion er og hvad det skal bruges til.

Programmet skal kunne køre med regionens Global Monitor, og skal kodes i Java 1.6 selve præsentationenslaget skal laves som JSP sider.

Risikoanalyse(Kim + Christopher)

I de følgende afsnit, vil vi forsøge at forklare informationerne, for nogle udvalgte problemer der kan opstå i løbet af projektperioden. Vi vil forsøge at beskrive de udvalgte problemer nærmere og komme med mulige svar på, hvordan vi kan formindske risikoen ved de udvalgte problemer.

Risk	Sandsynlighed	Alvorlighed
Software har en tendens til at 'crashe'.	10 %	2
Software har et for stort ressourcebrug.	10 %	1
Mindre overskred af ressourcebrug.	15 %	3
Manglende dokumentation ved projektet afslutning.	10 %	2

Manglende erfaring blandt udviklingerne.	50 %	2
Softwaren har mangler i forhold til hvad kunden ønskede.	30 %	3
Projektet overskrider deadline.	15 %	3
Manglende kontakt med kunden	25 %	2

1: katastrofalt

2: kritisk

3: mindre problem

Risiko: Manglende erfaring blandt udviklingerne.

Udviklerne til dette projekt har hovedsageligt tidligere benyttet c# som primære programmeringssprog, derfor vil det tage tid for dem, at sætte sig ind i at skulle udvikle ikke blot i et nyt sprog, men også i et andet udviklingsværktøj.

Tiden det tager at sætte sig ind i Java og jsp burde dog ikke være usandsynligt lang, da sproget ligger meget tæt på C#'s abstraktionsniveau. Derudover har de to sprog også en syntax der minder meget om hinandens, hvilket vil gøre det forholdsvist simpelt for en programmør der har benyttet det ene sprog i en tid, at sætte sig ind i det andet.

Den største forhindring bliver dermed ikke sprogenes forskel, men deres udviklingsmiljøer, som der kan være en del forskel på.

Da begge udviklere på dette projekt dog er vant til at sætte sig ind i nye programmer og er vant til at skulle benytte diverse søgemaskiner for at få forklaret spørgsmål, er barrieren ikke forståelsen, men tiden der kan være nødvendig at bruge for at finde svar på spørgsmål og derefter læse og forstå disse.

Løsning:

Såfremt det skulle ske at tiden der er nødvendig at bruge, for at finde svarene til spørgsmål til Java eller Javas kobling til Websphere MQ tager for lang tid, kan det blive nødvendigt for gruppen at gå tilbage til c#. Hvis dette sker, vil der derefter være brug for mindre tid til at sætte sig ind i sproget og dets kobling til Websphere MQ, hvilket vil gøre projektet hurtigere færdig.

Vi har sat risikoen til denne del forholdsvist højt, da der er givet en forholdsvist kort periode til at udvikle det færdige system i. Såfremt man har en kort tid til at fuldføre et projekt, må opgaven være meget simpel, hvis man skal benytte et nyt programmeringssprog.

Risiko: manglende kontakt med kunden.

Risikoen her ligger i at regionen har meget travlt, med deres mange projekter, det samt sygdom blandt de personer som der agerer kunder for os, kan gøre at de ikke har tid til at svare på for eksempel mails, hvilket kan give forlængede svartider. Manglende kommunikation fra regionen kan i værste fald gøre, at vi udvikler i en forkert retning i forhold til hvad regionen har behov for.

Løsning:

Vi vil forsøge at bruge den evt. downtime vi vil få, hvis vi ikke kan komme i kontakt med kunden, på at dygtiggøre os i bl.a. Java og den generelle brug af Eclipse. Ved at bruge tiden på at dygtiggøre os i Java, undgår vi at projektet køres af sporet. Dette har dog den ulempe, at projektet kan blive holdt tilbage og dermed tage længere tid end først anslået.

Regionen har i produkt kontrakten angivet at de ville have et ugentligt møde med os, så denne risiko burde derfor være minimal. Vi vil hvis det alligevel bliver et problem, have mulighed for at opsøge dem på regionen og venligt kræve et møde med dem.

Hvis det stadig er et problem med kontakten efter alle ovenstående muligheder er opbrugt, er der mulighed for at opsige kontrakten på baggrund af at regionen ikke har opfyldt sin del af kontrakten.

Risiko: Software har for stort ressource brug.

programmet er tiltænkt at køre som en widget på regionens Global monitor, derfor bør der ikke være noget større ressource brug, der er fra regionen sat en grænse til max 5 % overhead på MQ-queues.

Løsning:

Vi vil i samarbejde med kunden finde ud af om vi enten skal skære funktionalitet væk, eller om vi skal prøve at optimere system. Vi vil i øvrigt være i regelmæssig kontakt med regionen, så vi vil kunne komme i kontakt med dem vis vi opdager et problem i løbet af projektforløbet.

Risiko: Software har en tendens til at 'crashe'.

Programmet er tiltænkt at skulle køre konstant over en længere periode, her kan der være tale om måneder af gangen før programmet genstartes. Det er derfor vigtigt for regionen at de kan stole på informationen er up to date og at informationen der bliver vist ikke fryser i løbet af dets brug.

løsning:

Vi vil i løbet af projektforløbet teste så meget som muligt for at sørge for at minimere denne risiko, hvis der mod forventning alligevel er driftsproblemer ved levering, vil vi i samarbejde med regionen finde en god måde at overdrage projektet til regionens egne udviklere. Vi kan dog ikke se hvorvidt programmet vil køre ustabilt på Global monitor, før implementation, vi kan derimod sørge for allerede i programmeringsfasen at programmet kører stabilt i vores udviklingsmiljø.

Risiko: Manglende dokumentation ved projektet afslutning.

Såfremt der ikke er tilstrækkelig dokumentation til koden, ved afslutning af projektperioden, kan dette give problemer for regionens udviklere, som skal overtage koden. Disse vil i et sådant tilfælde, skulle bruge dyrebar tid på at slavisk gennemgå koden, for at forstå, hvordan udviklerne har båret sig ad med at få lavet den til regionen ønskede funktionalitet.

Løsning:

For at undgå dette vil vi løbende dokumentere koden og programmets funktionalitet, det betyder at vi skal afsætte tid til det i løbet af vores sprints. Når vi afsætter tid i sprintene til

dokumentation, vil vi minimere risikoen for manglende dokumentation ved projektets afslutning.

Ved at bruge tid på dokumentation sikrer vi os, at regionens udviklere vil have bedre mulighed for at forstå programmet og dermed ikke skal bruge unødig tid på at sætte sig ind i funktionaliteten. Dermed er den ekstra tid brugt i programmeringsfasen sparet i implementeringsfasen.

Risikoestimering

Risiko estimering	Performance	Support	Schedule
Katastrofalt	Programmet crasher eller fryser.	Ingen mulighed for opsætning af program.	Ingen adgang til kode af program.
Kritisk	Programmet bruger al for mange ressourcer i forhold til hvad der var formodet.	Forsinkelser i release på grund af dårlig dokumentation.	Ingen dokumentation til program.
Marginal	Programmet bruger en smule flere ressourcer end først antaget.	Mindre mangler i dokumentation. Ingen mulighed for kontakt med udviklere.	Mangler i dokumentation til program.
Ubetydelig	Programmet viser ikke alle MQ queues som det var tiltænkt.	Mindre kontakt problemer med udviklere af program.	Ubetydelige mangler, mindre misforståelser.

Udover en procent baseret analyse af risikoer til dette projekt valgte vi også at lave en tabel, der viser risikoer i forhold til det endelige program. Tabellen giver en kort forklaring på problemer der kan opstå i forhold til programmet og vi benytter herefter en usability estimering på hvor kritisk et problem det vil være, såfremt problemet opstår. Problemerne er delt op i tre kategorier:

- Performance: et problem der dækker selve programmet og dets kørsel.
- Support: Mulighederne for at få forklaring fra udviklerne omkring programmet.
- Schedule: Tidsmæssige begrænsninger, der kan have en effekt på brug af programmet.

Som sagt har vi delt risikoerne op i et usability index, for hvor stort et problem de forskellige er.

Vi kan med det indeks se de problemer der helst ikke måtte opstå, når programmet er færdigt.

Eksempelvis vil det for kunden være et katastrofalt problem, hvis programmet hele tiden fryser eller crasher, da dette vil gøre det umuligt at stole på det data, som programmet viser frem.

Til sammenligning af problemer, kan det at programmet bruger en smule flere ressourcer end først antaget, anses for at være et næsten ubetydeligt problem, som ikke vil have en nær så stor effekt på brugen af programmet.

Estimeringens værdier er lavet på baggrund af hvad der for kunden vil give størst tilfredshed og kan bruges af udviklerne til at prioritere opgavers rækkefølge, samt hjælpe dem med at undgå at aflevere et program, som måtte have kritiske eller katastrofale problemer.

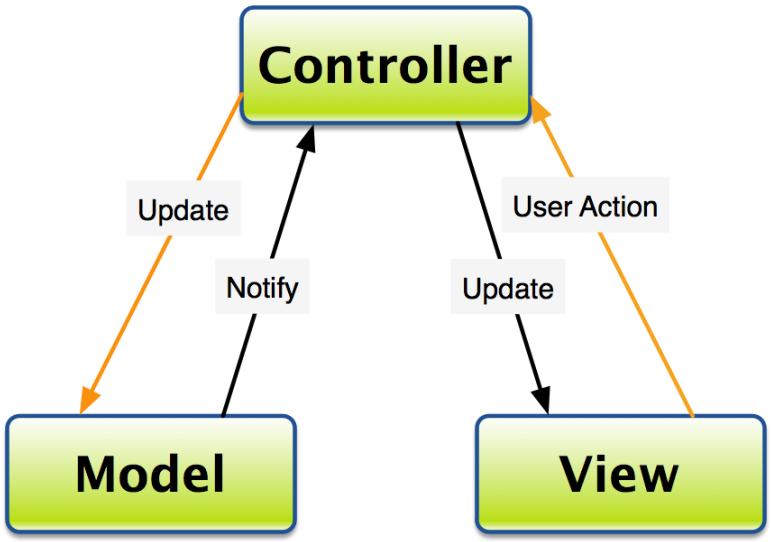
Arkitektur(Christopher)

Til ethvert projekt er det vigtigt at finde ud af hvilken arkitektur man ønsker at benytte til sit projekt. Dette bør gøres inden for mange klasser er skabt, så der ikke kan opstå tvivl om hvilke lag der må indeholde hvad.

Til dette projekt valgte vi at benytte en forholdsvist kendt model, MVC, til at holde styr på informations flow, i vores program.

MVC Teori

Valget af MVC, skete på baggrund af at vi ønskede at holde informations flowet så simpelt som muligt, men stadig ønskede noget kontrol over informations flow.



I MVC-modellen er der tre lag, hver med sit formål og sine begrænsninger, til hvad de må indeholde. De tre lag kaldes Model, View og Controller og vil i de følgende afsnit blive beskrevet.

Model

Model laget i MVC er det lag som blandt andet kan indeholde Data-Access Layers(DAL) klasser og via disse henter information uden for programmet og sender det videre til andre klasser, som kan behandle det eller vise det.

Model laget er selvstående og har egentlig ingen afhængigheder fra andre lag. Dette er en stor hjælp for programmørerne, i det de dermed kan teste modellaget separat og teste at dets funktioner virker som de skal, uden at skulle have fat i andre lag. Det betyder også at hvis man senere ønsker at udskifte funktionalitet i øvrige lag, kan det gøres uden at have en effekt på modellaget.

Controller

Controllerens rolle er den at mediere mellem view og model laget. Det er også controller, som tager imod input fra bruger og om nødvendigt, tjekker at de har de rigtige rettigheder til at foretage handlinger i programmet.

Såfremt en bruger har de rigtige rettigheder, skal den herefter sende et signal ned til modellaget, som den beder foretage eksempelvis en handling, en udregning eller en hentning af data.

Når modellaget har foretaget sit arbejde, kan der ske en af to ting:

- Modellaget sender dataene direkte til view
- Modellaget sender dataene tilbage til controller, som så sender det op til view.

De to scenarier dækker forskellige gruppers synsvinkel på MVC, den ene er dermed ikke mere korrekt end den anden. Man kan dog argumentere for at hvis data skal kontrolleres, bør modellaget sende informationen op til controlleren, hvorefter denne sender det til view.

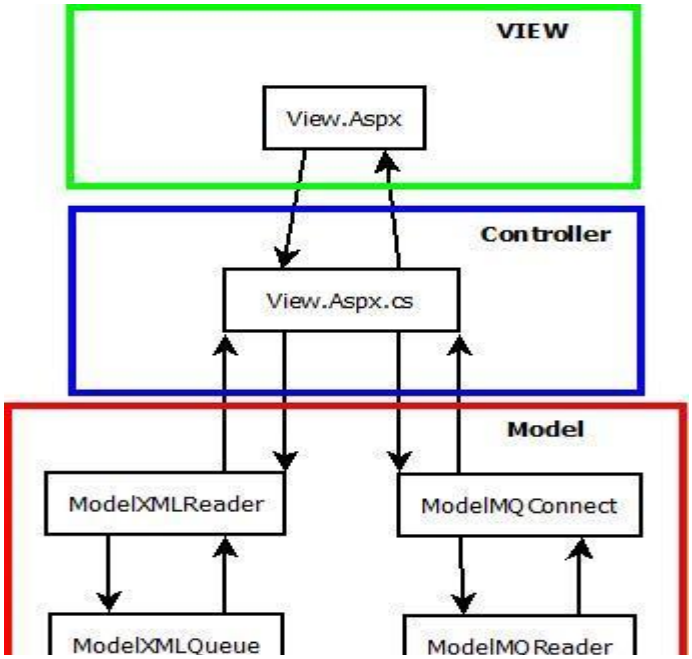
View

View er ofte i projekter tæt koblet med controller laget, da det tænkes at controlleren sender information til viewet. Det modsatte er også tilfældet. Når brugeren benytter input funktioner, eksempelvis button_click, skal der fra view laget kaldes en funktion som ligger i controlleren og tager imod input.

MVC i praksis

De nedenstående afsnit vil forklare hvordan vi har benyttet de tre lag i MVC-modellen.

Billedet ved siden af giver en repræsentation af hvordan MVC er brugt til klasserne i vores projekt.



MVC modellen som vi benyttede

Modellaget

I modellaget i vores projekt ligger 4 klasser, hvor af to af dem henter data udefra:

- ModelXMLReader som modtager information fra Web.Config og XML-filen tilkoblet projektet.
- ModelMQConnect, som opretter forbindelse til Websphere MQ og senere læser information fra dennes message queues.

Udover disse to klasser, ligger der yderligere to, som pakker informationen fra ModelXMLReader og ModelMQConnect ind som objekter.

Grunden til vi vælger at pakke informationen ind i objekter, frem for strings, er at det giver mindre arbejde i sidste ende. Med indpakning som objekter skal vi dermed ikke til at substring og konvertere mellem variabeltyper. Indpakningen som objekter giver dermed bedre overskuelighed, når informationen modellaget, skal benyttes i controlleren, da man så kan benytte enten en public variabel fra en af pakke-klasserne eller kan benytte en get-funktion sat i en af pakke klasserne.

Controlleren

I vores controller ligger metoderne, som starter funktionerne i programmet, når programmet køres. Der er her to overordnede metoder:

Page_Load

Page_Load henter først information fra Web.Config, omkring navn på Websphere MQ's Queuemanager, hvilken IP, som Websphere MQ programmet lytter på, hvilken port på IP'en og eventuelle kanaler, som der benyttes ved brug af client forbindelse.

Page_Load beder herefter modellaget om at give den en liste over message queues, en liste som modellaget indhenter fra den tilhørende XML-fil.

Når dette er gjort, kalder Page_Load en metode i modellaget, som går ind og læser informationen fra alle message queues; Hvor gammel ældste besked er og hvor mange beskeder der ligger og venter på at blive sendt videre.

Når al information fra message queues er indlæst, sender den via en metode informationen ud i view.

Timer1_Tick

Denne metode foretager næsten det samme som Page_Load, med den forskel at den ikke foretager en connect, da dette allerede er gjort via Page_Load. Derudover behøver Timer1_Tick heller ikke indlæse fra xml-filen, hvad message queue navne er, deres max queue alder og opdateringsinterval, da dette også allerede er foretaget i Page_Load.

Timer1_Tick benytter dog modsat Page_Load opdatering intervallet. Denne benyttes til at sikre at de individuelle message queues, kun opdaterer, efter de tider der er sat i XML-filen.

Viewet

Vores view er kun html og asp kode, viewet i vores projekt har ingen brugerinput, da der er tale om et program der skal køre selvstændigt og blot vises på en skærm.

Den eneste funktion der er viewet, er timeren, hvorfra tiderne til denne bliver hentet fra en dictionary der ligger i controlleren. Dermed styrer controlleren opdatering intervallet i viewet.

Arkitektur konklusion

MVC synes vi har givet os en funktionel model, som er nemt at finde rundt i og som giver en lav coupling mellem lagene. Det er nemt at finde rundt, der er ikke unødvendige ekstra lag og information er let at hente mellem lagene.

Valg af udviklingsmetode(Kim)

Da der fra starten af projektet var en del usikkerhed omkring opgavens omfang og sværhedsgrad Var der et ønske fra både regionen og vores side, for at udvikling skulle foregå agilt. Da dette vil gøre det nemmere at tilpasse projektet når der efterhånden kommer mere sikkerhed omkring opgaven.

Regionen er i gang med at indføre SCRUM som deres udviklingsmetode og da vi også føler os bedste tilpas med at køre SCRUM, er vi tilbøjelige til at køre det som udviklingsmetode, men vi vil først sikre os at dette også er tilrådeligt. Det har vi tænkt os at gøre via Alistair Cockburn's 'Methodology per project' og Todd Little's 'Complexity and uncertainty' de to metoder burde give os en god indsigt i hvilken udviklingsmetode dette projekt er bedste tjent med.

Vi vil starte med at lave Todd Little's 'Complexity and uncertainty'.

Table 1

Complexity attributes and their scores*

	Complexity score				
Attribute	**1**	**3**	**5**	**7**	**10**
Team size	1	5	15	40	100
Mission criticality	Speculative	Small user base	Established market	Mission-critical with large user base	Safety-critical with significant exposure
Team location	Same room	Same building	Within driving distance	Same time zone +/– 2 hrs.	Multisite, worldwide
Team capacity	Established team of experts	New team of experts	Mixed team of experts and novices	Team with limited experience and a few experts	New team of mostly novices
Domain knowledge gaps	Developers know the domain as well as expert users	Developers know the domain fairly well	Developers require some domain assistance	Developers have exposure to the domain	Developers have no idea about the domain
Dependencies	None	Limited, well insulated	Moderate	Significant	Tight integration with several projects

** Minimally complex = 1, highly complex = 10.*

Team size(3): Holdet består af 2 personer, og har derfor scoret en 3'er.

Mission criticality(5):Established market fordi systemet skal bruges af medarbejdere og brugere på daglig basis.

Team-Location(3): I den samme bygning. Folkene der arbejder på projektet kan snakke og hjælpe hinanden med eventuelle problemer der skulle kunne opstå.

Team-Capacity(7): Holdet og emnet er nyt, der er ikke megen viden eller erfaring omkring f.eks. Websphere MQ, hverken blandt udviklerne eller regionen.

Domæneviden(7): Udviklingerne har et kendskab til området. Udviklingerne vil kunne bruge den viden de har på området, selvom det vil blive nødvendigt for holdet at lære endnu mere om området inden projektet skal laves.

Afhængigheder(5): Systemet skal samarbejde med både Websphere MQ og Global monitor, begge disse har potentiale for at give problemer.

Table 2

Uncertainty attributes and their scores*

	Uncertainty score				
Attribute	**1**	**3**	**5**	**7**	**10**
Market uncertainty	Known deliverable, possibly defined contractual obligation	Minor changes in market target expected	Initial guess of market target likely to require steering	Significant market uncertainty	New, unknown, and untested market
Technical uncertainty	Enhancements to existing architecture	We think we know how to build it	We're not quite sure if we know how to build it	Some incremental research involved	New technology, new architecture; might be some exploratory research
Project duration	1–4 weeks	6 months	12 months	18 months	24 months
Dependencies, scope flexibility	Well-defined contractual obligations or infrastructure with published interfaces	Several interfaces Scope isn't very flexible	Scope has some flexibility	Some published interfaces Scope is highly flexible	No published interfaces

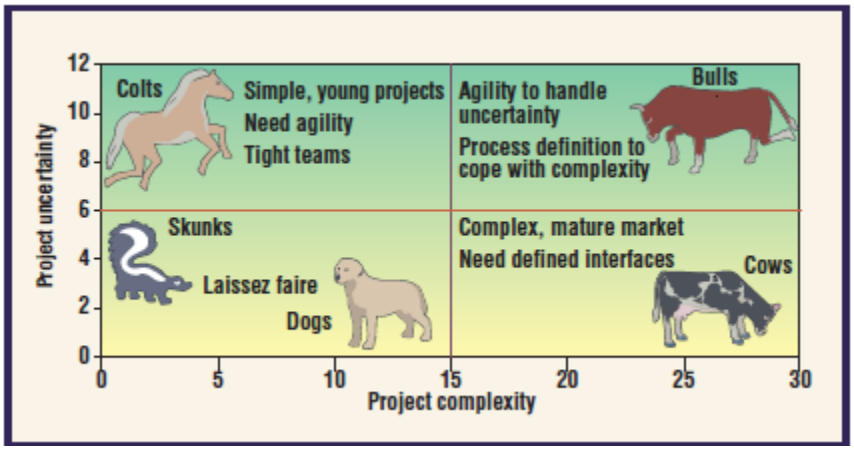
* *Minimally complex = 1, highly complex = 10.*

Market uncertainty(3): Markedet er rimelig sikkert, men der vil nok komme nogle ændringer.

Technical uncertainty(10): Ny teknologi, det vil blive nødvendigt at lave nogle undersøgelser om hvordan systemet skal laves.

Project duration(3): Vi har 10 uger til at lave et program der fungere og kan afleveres til regionen.

Dependencies scope flexibility(5): Scope har noget fleksibilitet da det kan være nødvendige at skære features vis deadline ikke kan nås.



Vi har nu scoret vores projekt nu skal vi så udregne vores scoren for de to.

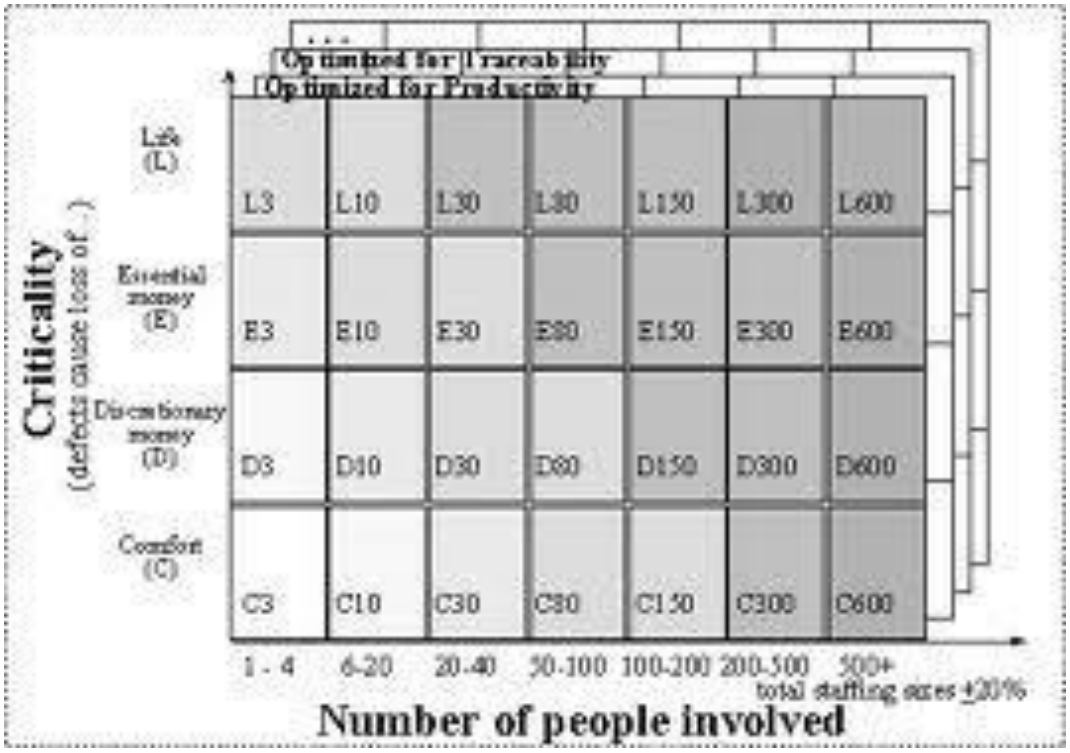
Formularen for complexity er: complexity = $2\sum \log 10x$, hvor x er lig med scoren vi fik fra complexity skemaet.

Formularen for uncertainty er: uncertainty = $2\sum \log 10y$, hvor y er lig med scoren vi fik fra uncertainty skemaet.

med disse formularer har vi scoret, 11,84 i Complexity og 6,29 i uncertainty.

Det vil sige at vi ligger på det øvre venstre felt i diagrammet herover, projektet er derfor en såkaldt 'Colt', Todd Little foreslår til denne type projekt at man bruger en agil udviklingsmetode, med små tætte grupper, hvilket passer godt med vores ønske om at køre SCRUM og vores gruppe størrelse. SCRUM kan også hjælpe os med at tackle den høje complexitet, specielt de daglige SCRUM-møder vil være behjælpelige med dette og vil samtidigt kunne afhjælpe, eventuelle problemer med uncertainty.

Som vi nu har vist med Todd Little's Complexity and uncertainty metode vil det være en god ide at køre SCRUM som vores udvikling metode, så vil gerne bekræfte dette via Alistair Cockburn's 'Methodology per project'.



Denne metode til at vælge udvikling model går meget enkelt ud på at alt efter for stort et hold og criticality projektet har, skal man vælge en passende metode dertil. Hvis man har en stor gruppe udviklere skal man bruge en mere omfattende udvikling metode. Hvis man har et højt criticality skal man sikre at udviklingsmetoden bliver fulgt korrekt af alle involvere.

Vi scorer med vores projekt lavt på begge parametre da vi kun er 2 mand på holdet og projektet ikke er kritisk for regionalt IT.

Det vil sige vi har scoret projektet som en C3, det vil sige at vi ikke behøver at have en omfattende udviklingsmetode, og at vi derfor godt kan køre en meget agil metode som f.eks. SCRUM.

Det vil sige at vi også her har fået bekræftet, at det vil passe godt for os at køre SCRUM som vores udviklingsmetode.

Konklusion

Både Todd Little og Alistair Cockburns analyser viste at det kunne være en god idé at køre SCRUM som vores procesmodel, det passer også godt med gruppens egne ønsker om at køre denne som vores udviklingsmetode. Det er specielt de daglige møde og små iterationen som

passer godt med vores ønsker til udviklingsforløbet der har påvirket vores endelig valg af SCRUM som vores udviklingsmodel.

Når vi nu har fået bestemt at SCRUM ville passe godt til vores projekt, skal det selvfølgelig siges at vi ikke kan køre rigtig SCRUM med alt hvad det indeholder, da vi kun er en to mands gruppe.

Men hvad vi kan gøre er at tage nogle af de rigtige gode værktøjer og metoder som SCRUM tilbyder, her tænker vi bl.a. på brugen af en produkt backlog, sprint backlogs, daglige uformelle møder hvor vi aftaler arbejdsopgaver med hinanden, og selvfølgelig tæt kontakt med kunden under udviklingen af projektet. Ved at bruge disse værktøjer og metoder kan vi sikre os at det vi får afleveret både lever op til kundens, men også vores egne ønsker til hvordan det endelige produkt skal se ud, derfor har vi valgt SCRUM som vores udviklingsmetode.

User Stories(Kim + Christopher)

Efter vi havde gennemført de to tests og valgt SCRUM som vores udviklings, valgte vi at bruge user stories til at repræsentere vores brugeres krav og ønsker til produktet. Vi valgte user stories over use cases fordi vi mente at use cases hurtigt kunne blive for formelle i forhold til hvad opgaven krævede og vores egne behov.

US1: Brugeren vil gerne kunne se opdaterede data fra MQ-queues.

Det er vigtigt for brugeren, at dataene der vises fra MQ-queues er opdateret, da brugerne skal kunne tage stilling til om de skal foretage handlinger på MQ-serveren. Såfremt dataene ikke er opdateret jævnligt, vil det kunne betyde store forsinkelser på løsning af problemer, med hensyn til igangsættelse af serverne igen, Hvis dette ikke bliver gjort hurtigt efter at problemet opstår, vil det kunne forsinke mange meddelelser mellem regionens sundhedssektor.

US2: Brugeren vil gerne kunne se om meddelelser overskrider max alder som den enkelte meddelelse må have.

Hvis meddelelser overskrider max alder, er det et tegn på at der er problemer ved modtagerens server, hvilket er en opgave som Regional IT skal servicere. Hvis meddelelser

lider af større forsinkelse, fordi Regional IT ikke kan se at serverne er nede, kan der gå længe inden at der er nogen der forsøger at sætte modtagerens server i gang. Dermed kan der ske større forsinkelser på meddelelser mellem regionens sundhedssektor, hvilket i sidste ende kan betyde at patienter ikke får den behandling eller medicin de skal have til tiden. I andre tilfælde kan det betyde at en læge eksempelvis skal bruge mere tid på at forklare et apotek, at Fru Hansen skal have et bestemt mærke af Insulin, da hun ikke kan tåle et andet præparat.

US3: Brugerne skal kunne se om en meddelelse er for gammel evt. via farvekode.

Farvekoder hjælper visuelt med at forklare Integrationsafdelingen om hvorvidt message queues kører som de skal. Tal kan fortælle mange ting, men hvis en queue har en tidsgrænse der er lavere end en anden og de ansatte ikke alle er klar over forskellene i tidsgrænser, kan det være der gribes ind før nødvendigt eller i andre tilfælde alt for sent. Farvekoderne mindsker dermed mængden som de ansatte i Integration skal huske ud over deres andre opgaver.

US4: Brugerne skal kunne se om meddelelser ophober sig i MQ evt. via farvekode.

Meddelelser der ophober sig kan være et symptom på at serveren har travlt, men kan også indikere et mere alvorligt problem, som f.eks. at serveren er gået ned. Farvekoderne vil også her hjælpe, da det vil give Integrationsafdelingen et hurtigt visuelt overblik, over meddelelsernes status.

US5: Bruger skal kunne se serverens ram, cpu, og I/O status

Det er vigtigt at bruger kan se status på serveren, da eventuelt forhøjet ressourceforbrug kan indikere et problem med serveren.

Et for højt ressourceforbrug kan for eksempel indikere at beskeder ikke bliver leveret hurtigt nok. Dette kan betyde at læger ikke får nødvendige beskeder i tide, derfor er det vigtigt at man fra regionen side kan se om serveren kører som de skal.

US6: Brugeren vil gerne modtage en e-mail hvis der opstår problemer med deres queues

Det er nødvendigt at sende en mail da brugerne ikke altid vil have netop overvågnings skærmen åben.

Product Backlog(Kim)

Efter vi havde udfærdiget vores user stories, havde vi et møde med Thorkild Friis, hvor vi fik sat værdier på de enkelte user stories og derefter satte vi dem i vores produkt backlog.

Eftersom vi kun er to mand i gruppen har vi ikke en product owner, derfor var vi begge med til at arrangere product backloggen. Vi har forsøgt at lægge de user stories med højest værdi øverst i backloggen, men har selvfølgelig taget højde for at nogle user stories ikke kan laves før vi f.eks. har fået oprettet forbindelse til Websphere MQ. Der var også en prioritering for regionens side, der hed at de ville have MQ monitoring færdig før vi begyndte at integrere Zabbix delen ind i projektet.

Efter mødet med Thorkild satte vi os sammen og udførte en omgang planning poker, for at sætte hvor lang tid de enkelte user stories vil tage. Selve planning poker processen foregår meget uformelt og vi har derfor ikke taget denne del med.

Efter mødet med Thorkild og efter vi havde gennemført vores planning poker, udformede vi som gruppe denne product backlog. Vi har via vores user stories lavet nogle backlog items, som vi mener, kan repræsentere de user stories vi har, og som i sidste ende burde udmønte i et funktionelt projekt. Vi valgt at bruge product backlog items **PBI** i stedet for task, da vi synes det var lidt uhåndgribeligt, at sætte et timetal på opgave som vi ikke har nogen praktisk erfaring med at lave på forhånd. Med produkt backlog items bruger man i stedet såkaldte story points **SP**, som er en lidt mere abstrakt måde at måle sværhedsgraden for den enkelte PBI. **SP** er en værdi som sættes gruppen og repræsentere den arbejdsindsats som gruppen estimere skal bruges til den enkelte opgave.

	Backlog Items	Værdi	SP	Test	Noter
1	Opret kontakt fra Java til Websphere MQ med nødvendig fejlhåndtering	90	80	check om fejlhåndtering fungerer efter hensigt	
2	Hent information fra serveren og vise det via	90	40	kan se korrekt og opdateret information	

	console			via console	
3	Tjek status på meddelelse	80	20	Test at status for meddelelser er korrekt	
4	udvid systemet så det kan vise informationen via JSP med farvekode	35	50	kan se korrekt og opdateret information via JSP	
5	Udvid systemet så det kan håndtere flere queues	60	50	man kan se flere queues med korrekt og opdateret information	
6	Vis information for flere queues	35	40	Test at information fra flere queues er korrekt	
7	Udvid systemet så det kan håndtere informationer fra Zabbix	85	80	Test om information modtages. Test information vises korrekt.	
8	Programmet skal kunne sende en mail hvis der er problemer med deres message queues	30	20	tjekke om mailen sender tidslig og relevant information ved MQ-fejl	

Tabellen ovenover viser vores product backlog som den så ud da vi begyndte vores første sprint, der kom selvfølgelig ændringer undervejs disse ændringer vil blive vist i vores sprint backlogs.

Estimering af opgaven

Vi har valg at lave en 3 punkts estimering for at få en bedre ide om hvor lang tid vi kan regne med at projektet vil tage. 3 punkts estimering er en rigtig god måde at udregne projekt længde, da man bruger de 3 mulige scenarier optimistisk, realistisk og pessimistisk. Ved at bruge disse 3 scenarier, får man en god middelværdi som man derefter kan bruge i selve planlægningen. Der er dog det problem at hvis personerne der udføre 3 punkts estimeringen, ikke er erfaren inden for området der arbejdes med, kan det være svært at få en realistisk estimering af projektet.

Den vigtigste værdi man får ud af denne estimering er middelværdien eller **m**, udregningen af denne værdi ligger mere vægt på den realistiske estimering end den gør de andre to estimeringer. Den anden vigtige værdi vi får ud af 3 punkts estimeringen er værdien varians **v**, denne værdi viser hvor stort udsvinget er mellem den pessimistiske og optimistiske estimering er og giver et godt indblik i hvor stor usikkerheden er på den enkelte opgave og som helhed.

De enkelte vurderinger er lavet ud fra vores product backlog items eller **PBI**.

	Optimistisk	Realistisk	Pessimistisk	middelværdi	standardafvigelse	varians
Tasks	a	b	c	m	s	v
PBI1	3	5	10	5,6	1,4	1,96
PBI2	2	5	10	5,4	1,6	2,56
PBI3	1	3	5	3	0,8	0,64
PBI4	1	3	5	3	0,8	0,64
PBI5	5	7	10	7,2	1	1
PBI6	1	3	5	3	0,8	0,64
PBI7	5	8	10	7,8	1	1
PBI8	1	2	3	0,4	0,4	0,16

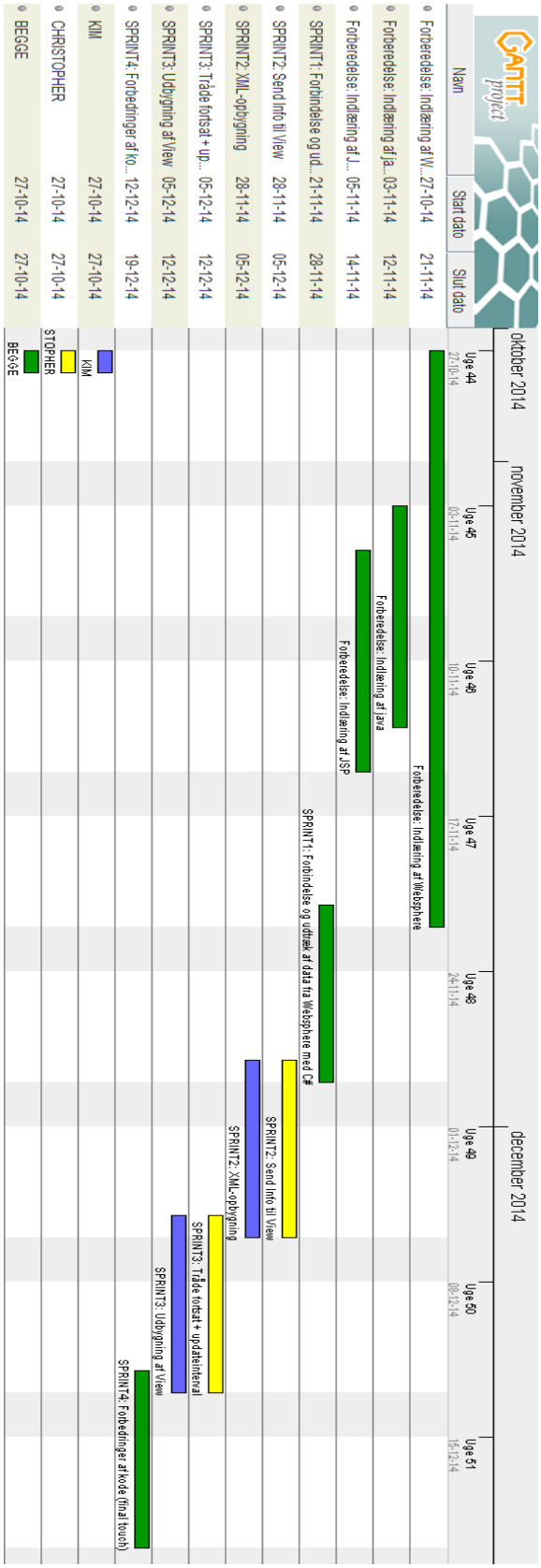
I alt				37		8,6

Som man kan se, estimerede vi at opgaven som helhed ville tage cirka 37 arbejdsdage at udføre, hvilket betyder at vi formentlig ikke vil kunne færdiggøre projektet indenfor den deadline.

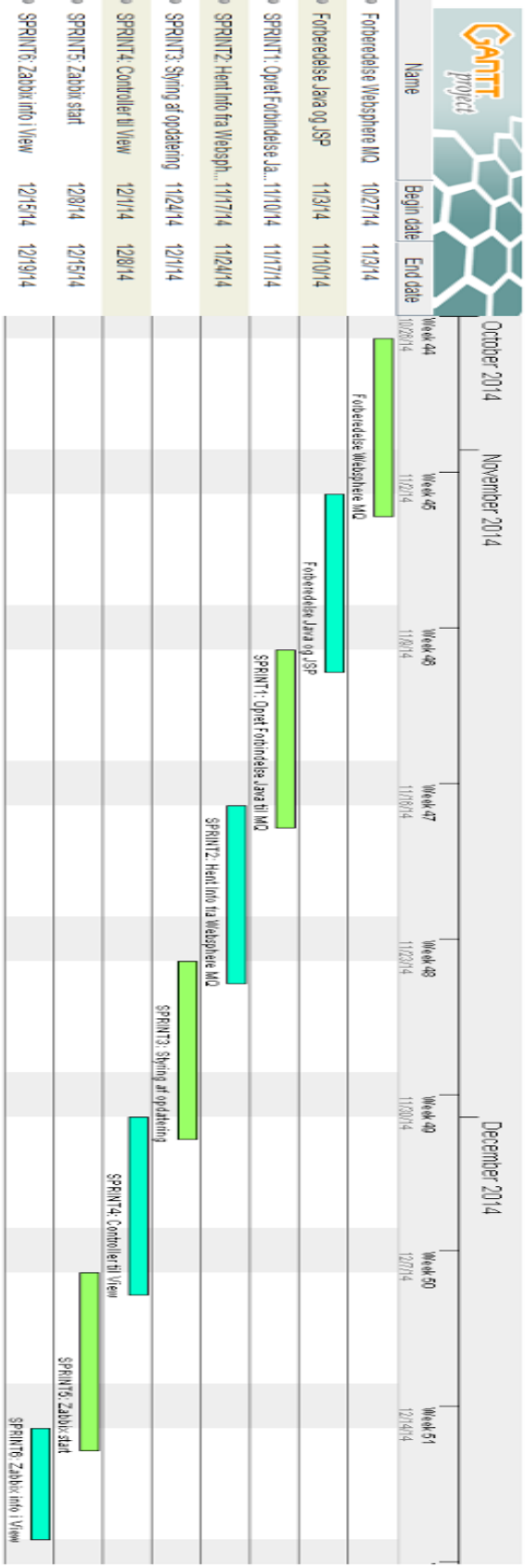
Dette har den effekt at vi skal passe på med at lave nye features hen mod slutning af projektet for at sikre at det vi levere er testet tilstrækkeligt.

selvom der er nogle problemer med denne estimerings metode, mener vi at den har givet os et godt indblik i hvor langt tid vi kan regne med at projektet vil tage og dermed givet os lidt mere klarhed om hvordan opgaven skal udføres, plus hvad vi skal have specielt fokus på.

Planlægning(Kim+ Christopher)



Faktisk Tidsplan



Originale Tidsplan

Vi har valgt at inkludere to billeder af tidsplaner, hvor det til højre viser den originale og tiltænkte tidsplan, som vi først gik ud fra, men senere måtte stoppe med at følge, på grund af forsinkelser. Billedet til venstre viser den tidsplan vi fulgte efter forsinkelserne.

Nu hvor vi har fået fastslået at det vil være en god ide at køre SCRUM, kan vi begynde at planlægge projektforløbet og vælge hvilke metoder vi vil forsøge at bruge til udførelsen af projektet.

Vi startede med at lave en indledende tidsplan for projektet, vi havde i starten en ide om at vi ville køre sprint af to ugers varighed, men grundet forsinkelser i løbet af opstarts perioden valgte vi at sætte sprint længden ned til en uges varighed. Grunden til at vi mente det var nødvendigt at sætte sprint længden ned, var til dels også at vi ønskede at få mindst tre gode sprints gennemført, men også at usikkerheden var stor i starten af projektet og at vi ved at holde sprint længden nede ville kunne sikre at vi ikke udviklede i en forkert retning længe uden kontakt med kunden. Korte sprints kan også hjælpe udviklere med at finde eventuelle fejl, mangler eller begrænsninger med projektets problemstilling.

Det var fra starten et ønske fra kunden at programmet skulle udføres i Java 1.6SE og vi havde planlagt at den første uge skulle gå med at sætte os ind i Java og udviklingsværktøjet Eclipse, da dette var hvad kunden foreslog at vi kunne bruge.

Vi skulle også selv stå for opsætningen af Websphere MQ, da vi ikke kunne få adgang til regionen testmiljø, da regionens egne testere skulle bruge det i løbet af hverdagen til diverse tests.

Vi havde derfor planlagt at det første sprint udelukkende skulle gå med at få oprettet forbindelsen til Websphere MQ.

Hvis vi mod forventning ikke har forbindelse til Websphere MQ efter det første sprint vil vi tage en snak med regionen, om hvorvidt det er muligt at komme i kontakt med en ekstern konsulent for at få gang i projektet.

Når vi har fået oprettet forbindelsen til Websphere MQ, har vi planlagt at køre normale sprints indtil projektets afslutningsdato som vi har sat til den 19.december.

Cost-Time-Scope analyse(Christopher)

Til dette projekt er, har vi en givet tid på 10 uger, til at planlægge, programmere og dokumentere et produkt til en virksomhed. Hvor meget man kan nå af et sådant produkt på 10 uger, er hvad der vil blive forsøgt beregnet i de følgende afsnit.

Størrelsen af et produkt leveret kan beregnes på flere måder, til dette projekt, vælger vi at estimere projektets endelige størrelse ved hjælp af tre faktorer: Cost, Time og Scope.

De tre faktorer hænger stærkt sammen og former en trekant, der angiver størrelsen af det endelige produkt. Såfremt der ændres på en faktor, så denne bliver større vil dette gøre trekantens rumfang og omkreds større, hvilket kan sættes i parallel med projektet, som i så fald også vil blive større.

Cost Time Scope Teori

De følgende afsnit beskriver teorien bag de tre punkter.

Cost

Omhandler prisen på projekter. Såfremt en virksomhed siger at et projekt max må koste en million kroner, har man dermed en max størrelse på den ene kant. Det er dermed op til projektgruppen at finde ud af, hvor meget der kan udvikles for en million kroner. Hvor mange arbejdstimer der er mulighed for at bruge på udvikling af programmet, hvor stor en projektgruppe der er råd til. Andre vigtige faktorer, som også kan komme ind under cost, er om der skal lejes eller købes udstyr til projektgruppen, for at de kan udvikle produktet, samt om der skal lejes plads til at huse projektgruppen. Dette er blot nogle af de ting der skal tages højde for, når prisen er fastsat.

Time

Omhandler tidsbegrænsninger på projekter. Der kan være sat en begrænsning på, hvor lang tid det må tage at udvikle et givet stykke software til en virksomhed. Sætter man eksempelvis en tidsbegrænsning på tre måneder, til levering på et stykke software, sætter dette en

barriere for udviklerne af softwaren. Når tiden er en barriere, handler det om, hvordan man bedst muligt anvender tiden og hvor meget man anskuer der kan nås at udvikles indenfor den givne tidsperiode. Hvor meget der kan udvikles på inden for tidsperioden kommer så an på gruppens erfaring med udvikling af typen af software, antallet af udviklere på projektet, antallet af arbejdstimer per udvikler og hvor meget der skal dokumenteres i forhold til produktet.

Scope

Er omfanget og kvaliteten af produktet, som skal leveres til kunden. Scope fortæller noget om hvor meget der fra kundens side er krævet af det endelige produkt, hvor mange funktioner det skal have, hvor veldokumenteret det ønskes og kvaliteten af produktet. Vælger man at justere på antallet af funktioner, som et program skal have i et projekt, uden at man vil give mere for det(cost), eller afsætte mere tid(time), kan udviklerne af produktet blive nødt til at gå på kompromis med kvaliteten. Det kan være at udviklerne bliver nødt til at spendere mindre tid på testning af koden, hvilket giver større sandsynlighed for at der opstår problemer, ved brug af programmet. Det kan også være at der ikke bliver nær så meget tid til at dokumentere programmets funktionalitet og det derfor er sværere for kunden at benytte programmet. I begge tilfælde med kompromis med kvaliteten, kan det betyde en nedsættelse i produktivitet for virksomheden der køber programmet.

Cost Time Scope Praksis

Følgende afsnit beskriver hvordan vi brugte Cost Time og Scope i praksis i forhold til dette projekt.

Time

I forhold til dette projekt er der sat nogle grænser, af både de studerende som udvikler og firmaet som skal modtage produktet, samt de studerendes akademi.

I forhold til opgaven er der sat en tidsgrænse, som hedder d.19. december for udvikling af programmet, dermed har vi i alt 8 uger at udvikle i. Derudover er det en tidsgrænse at der kun er mulighed for at arbejde på regionen 4 dage om ugen, hvilket giver et total på 32 arbejdsdage, til at udvikle produktet, som Region Syddanmark skal have.

Scope

Når der er sat en grænse på godt 8 arbejdsuger på regionen, sætter dette også nogle grænser for hvor meget der kan nås at udvikles, såfremt man vil opnå en hvis kvalitet på softwaren.

Kvaliteten der er aftalt mellem de studerende, som udvikler til regionen og de regionalt ansatte, er at der skal være et kørende program, som kan oprette forbindelse til MQ og herfra enten til en visuel brugerflade eller til konsolvindue vise status på MQ queues. Dette er første prioritet, anden prioritet er at få programmet til at læse fra Zabbix, hvad serverens status er, i forhold til CPU og Ram forbrug.

Med et total på godt 8 arbejdsuger, hvori der kan udvikles et produkt, er det ikke sikkert der bliver tid til at udføre den anden opgave, såfremt man ønsker at sikre at softwaren, som læser fra MQ har en arkitektur, hvor ved den er nem at videreudvikle på og softwaren er testet, så man er sikker på at det der er kodet virker som det skal og at der ikke opstår uventede fejl, ved kørsel af softwaren.

Cost

Prisen på udviklingen af dette software, er forholdsvist minimal, da der er tale om en opgave, som de studerende laver som en del af deres hovedopgave. De studerende modtager dermed ikke løn fra regionen, for at udvikle softwaren, men modtager blot deres Studie Understøttelse. Dermed er prisen for softwaren, kun de mandetimer, som regionen selv skal bruge på at konsultere med de studerende, omkring software udviklingen.

Prisen kan dog senere blive større, såfremt regionen ønsker at videreudvikle på softwaren, når de studerendes projekt tid er ovre.

Sprint backlog(Kim)

Sprint backlogs er et af de rigtige gode værktøjer man har, når man køre SCRUM. Brugen af sprint backlogs giver både projektledere, men også udviklerne selv en rigtigt godt overblik over hvor langt teamet er kommet og om hvorvidt sprintet køre som det skal, om det er bagud, eller om det bliver færdig før tid.

Vi har brugt sprint backlogs mest for at holde styr på hvor langt vi er noget i de enkelte sprints, det vil sige vi mest har brugt til at skønne hvor meget vi har noget på foregående sprint og hvor meget vi derfor kan regne med at nå på det næste. Da vi ikke har et stort team mente vi ikke det var nødvendigt at bruge burndown charts eller sprint velocity estimeringer da disse ville have meget lidt værdi, i forhold til kunden og os selv.

Når vi nu har valgt ikke at lave sprint velocity estimeringer, må vi skønne før hvert sprint hvor meget vi mener vi kan nå, dette bør dog ikke være et problem når vi kun er 2 personer der skal være enige.

Projektforløbet(Kim + Christopher)

Projektforløbet der varede fra 27/10 2014 til 19/12 2014, vil i de følgende afsnit blive beskrevet i detaljer, vi vælger at dele det op, så de første afsnit beskriver opstarten og afsnittene herefter beskrives lineært tidsmæssigt.

Valget af opgaven

Vi startede projektet mandag den.27/10 med et møde med Morten Lyhne Thuesen, hvor vi fik stillet den opgave som vi endte med at lave. Den opgave vi fik stillet omhandlede queue monitoring af Region Syddanmarks Websphere MQ queues. Det var dog ikke den opgave som vi originalt regnede med at vi skulle lave. Oprindeligt havde vi talt om at lave et SCRUM projekt for dem, men denne ide blev skrottet da den ville indeholde meget lidt programmering. Derefter var der tale om at lave en tilbygning til den stored procedure, som Christopher havde lavet i løbet af hans praktikperiode på regionen. Denne blev dog også fravalgt da regionen mente denne opgave ville indeholde for lidt programmering til en hovedopgave.

Opgaven vi blev enige om til det første møde, blev derfor i sidste ende queue monitoring programmet som tidligere nævnt. Opgaven bestod i at lave en Java Server Page, som kunne vise information om status på regionens MQ server. Programmet skulle kunne kontakte MQ serveren og her tjekke på hvor gammel den ældste meddelelse som ligger på den er. Programmet skal også kunne tjekke hvor mange meddelelser der ligger i queuen. Såfremt meddelelserne når en hvis alder eller der begynder at ophobe sig meddelelser, skal JSP siden skal signalere med farvekoder, hvad status er. Her tænkes i første omgang at der skal

indikeres med grøn, såfremt der næsten ingen meddelelser er og de ikke er særlig gamle, gult, hvis noget begynder at hobe sig op eller nærmer sig den maksimale tid, som de må ligge og vente og Rød, såfremt der ligger flere meddelelser end der maksimalt må ligge eller tiden på ældste meddelelse overskrider grænseværdien for alder.

Opstartsperioden

Det blev opfordret fra regionens side at vi selv søgte information omkring Websphere MQ, da integrationsafdelingen ikke selve havde den nødvendige viden omkring hvordan Websphere MQ virker. Vi fik indtrykket af der ville være rigeligt med information at finde på nettet omkring Websphere MQ, men også omkring Eclipse og Java.

Java og Eclipse var der også masser af information omkring, men i forhold til Websphere MQ, var informationen ofte forældet eller til en anden version, end den som regionen ønskede at vi programmerede op til. Vi gav dog ikke let op, i vores forsøg på at finde information omkring Websphere MQ og valgte at spørge ind til information omkring det ved de ressourcer vi havde. Dette inkluderede regionens egne ansatte, som linkede os til information, som dog også viste sig ikke at være brugbart.

Kontaktpersonerne vi havde på regionen havde dog meget travlt og der var ofte problemer med kommunikationen, det havde vi dog taget højde for via vores risikoanalyse. Når vi ikke kunne komme videre, brugte vi tiden på at forbedre vores evner inden for Java, brugen af Eclipse, samt skrivning af teori til projektet. Den langvarige mangel på kommunikation begyndte dog at blive et kritisk problem, og vi besluttede at vi ville insistere på at få en siddeplads på regionen. Dette gjorde vi for at øge kommunikationsmulighederne og for at øge vores synlighed til regionen. Dette viste sig at være en god ting, for i samme uge blev vi tilbudt den eksterne konsulent, dette var dog først fire uger inde i projektperioden.

Starten på sprint forløbet

Grunden til informationen viste sig ikke at være brugbar, fandt vi først ud af, da vi fik et møde med en ekstern konsulent for regionen. Det viste sig at for at benytte den kode der var fundet, var der brug for en Windows Professional Edition, da man i Professional Edition har mulighed for at sætte brugergrupper. Dette viste sig ikke at være mulig med de versioner af Windows, som er installeret på vores computere.

Grundet hvor langt inde i projektforløbet vi nu var, samt den eksterne konsulents manglende viden omkring opsætning af forbindelse mellem Java og Websphere MQ, blev løsningen at vi gik over til at programmere i C# og dermed lavede en ASP løsning af programmet, som den eksterne konsulent vidste noget mere om og derfor hurtigt kunne sætte os ind i.

Vi informerede regionen om, at det ikke var muligt at gå videre med at lave en JSP løsning grundet detaljerne nævnt ovenfor og de acceptere at koden blev i C# i stedet, med en ASP website.

Det problem vi havde med forbindelsen til Websphere MQ, fandt vi ud af i løbet af samtalen med den eksterne konsulent, var at vi havde forsøgt at skabe forbindelsen via en 'Client-connection', det skabte problemer da vi som før nævnt ikke kunne sætte brugergrupper i vores version af Windows. Denne forhindring blev overvundet i samarbejde med den eksterne konsulent. Løsningen kom via en såkaldt 'Bindings connection', som i modsætning til en client-connection er lavet til at fungere på samme maskine. Denne information havde vi ikke stødt på før vores samtale med den eksterne konsulent.

Sprint 1

Efter vi fik oprettet forbindelsen til Websphere MQ, med hjælp fra den eksterne konsulent, gik vi i gang med at løse de andre opgaver i vores product backlog. De to efterfølgende sprints gik fint og vi fik den første iteration af programmet til at køre forholdsvist hurtigt. Den første iteration af vores program bestod hovedsageligt af en forbindelse til vores Queue Manager og den test queue vi havde opsat. Derefter viste vi så den information vi fik tilbage fra Websphere MQ, med en konsol applikation. Vi valgt at gøre det på denne måde for at sikre os at informationen vi fik ud var korrekt. Ved at gøre det med en konsol Applikation sikrede vi os at, vi ikke spildte tiden på et view der var unødvendigt på dette stadie af udviklingen.

Sprint 2

Efter vi havde fået programmet til at virke via et konsol applikation projekt, overførte vi det til en ASP.NET webforms projekt, som vi havde tiltænkt skulle være den form projektet i sidste ende skulle afleveres i. Den anden iteration blev således at vi kunne hente information fra Websphere MQ og vise det via web forms. Samtidig havde vi arbejdet på en XML-klasse til læsning af diverse queue information, her menes for eksempel queue-navn og den

maksimale queue alder. Det vil sige ved afslutningen af vores andet sprint havde vi fået færdiggjort alle de opgaver vi havde taget ind i vores sprint backlog, men også løst problemet med at gemme data uden en database tilknyttet.

Sprint 3

Til vores tredje sprint havde vi valgt at fokusere på at få programmet til at fungere med flere queues, det havde vi tænkt kunne gøres med threading, men dette gav os problemer når siden skulle kunne loade dynamisk, alt efter hvor mange queues vi havde lagt i XML-filen. de første iterationer af denne funktion, havde den mangel at man ikke kunne sætte opdatering intervaller for de individuelle queues, hvilket selvfølgelig var et krav fra kunden at man skulle. Det var helt sikkert en lærings proces for os begge at arbejde med tråde igen da det var langt tid siden vi havde arbejdet med det sidst. Vi fik dog det ud af vores problemer med tråde, at vi lærte en del alternativer til brugen af tråde. Vi fik i løbet af vores udforskning af emnet anledning til arbejde med threading tasks da vi mente disse måske kunne hjælpe os med at sætte timerne individuelt som før nævnt for et krav, men også med denne metode havde vi problemer. VI havde derfor ikke held med at få den troede løsning til at virke inden for det tredje sprint, men vi fik optimeret den visuelle præsentation, ved brugen af farvekode når f.eks. beskeder i queuen har lagt for længe bliver queue navnet rødt.

Ved afslutningen af tredje sprint havde vi derfor den visuelle præsentation færdiggjort, men vi manglede stadig en god løsning på hvordan vi kunne vise information fra flere queues, vi besluttede derfor at prøve at finde alternative løsninger på hvordan man kunne takle dette problem.

Sprint 4

På grund af de store forsinkelser i starten af projektperioden, blev vores fjerde sprint det sidste sprint vi gennemførte. Det sidste sprint fokuserede vi på at fikse vores problemer med at læse information fra flere queues, da dette var det eneste vi manglede inden vi skulle til at arbejde med Zabbix.

På fjerde sprint fandt vi endelig, i samråd med Morten Lyhne Thuesen fra regionen, en løsning på problemet. I stedet for at bruge tråde, bruger vi nu en event som køre på selve ASP-siden som køre via en asp-timer, som er slags server kontrol. der kan sættes til det

interval man ønsker, denne timer køre så en metode som står for at opdatere queue data, denne proces er bedre beskrevet i afsnittet med kode forklaringerne.

Projektafslutningen

Efter vi blev færdige med denne løsning blev vi i samråd med regionen enige om at stoppe udviklingen, da der ikke var tid til at sætte sig ind i Zabbix hvilket ville være nødvendigt hvis vi skulle fortsætte projektet.

Sprints(Kim)

Vi vil nu beskrive vores sprint forløb og hvilke backlog items vi tog ind på de enkelte sprints.

Sprint 1 backlog

Vi valgt til vores første sprint kun at tage et backlog item ind, da der var stor usikkerhed omkring opgavens omfang og sværhed, samt det faktum at vi ikke kan komme videre med projektet hvis denne opgave ikke bliver gennemført. Denne usikkerhed skyldes hovedsageligt den manglende viden om emnet, fra både vores og regionens side. Vi var derfor fra regionens side blevet lovet, at vi ville have en ekstern konsulent til rådighed til opgavens udførsel, hvis vi ikke selv har fundet en løsning.

	Navn	Værdi	Tid	Test	Noter
1	Opret kontakt fra Java til Websphere MQ med nødvendig fejlhåndtering	90	80	check om fejlhåndtering fungerer efter hensigt	færdig ved sprint afslutning

Sprint 1 konklusion

Vi fik endelig kontakt til Websphere MQ, med hjælp fra den eksterne konsulent som vi var blevet lovet, dog blev det med C#. Overgangen til C# skete fordi konsulenten var C# programmør, ikke Java programmør og dermed manglede den nødvendige erfaring, for at sætte os i gang med at lave det som et Java program. Det var selvfølgelig et problem for regionen da det system det skal kører på er Java Eclipse baseret. Men på grund af de store forsinkelser vi havde med at komme i gang med projektet, blev vi sammen med regionen

enige om at skifte til C#. Dette af den ene grund, at vi blev nødt til at komme videre med projektet for at få noget nyttigt ud af det. Ved slutning af sprintet havde vi endelig kontakt til Websphere MQ og kunne derfor komme videre med projektet. Vi havde lært en masse om hvordan Websphere MQ fungerede, og følte os nu i stand til at bruge dette i det videre forløb.

Sprint 2 backlog

Eftersom vi endelig fik oprettet en forbindelse til serveren i første sprint, er vi i sprint 2 fri til at tage nogle flere backlog item ind, vi har derfor valgt følgende user stories til sprint 2.

Som man kan se har vi nu ændret vores backlog items for at vise vi har skiftet til et ASP.NET projekt. Vi fandt også en ekstra opgave, i at programmet skulle kunne kører uden en database, denne blev også sat ind i sprint backloggen.

	Navn	Værdi	Tid	Test	Noter
2	Hent information fra serveren og vise det via console	90	40	kan se korrekt og opdateret information via console	færdig ved sprint afslutning
3	Tjek status på meddelelse	80	20	Test at status for meddelelser er korrekt	færdig ved sprint afslutning
4	udvid systemet så det kan vise informationen via ASP.NET med farvekode	35	50	kan se korrekt og opdateret information via ASP.NET	Kan vise info via asp side dog ikke med farvekode endnu.
ekstra opgave	Systemet skal fungere uden en database		30	information kan hentes og bruges af programmet	færdig ved sprint afslutningen

Sprint 2 Konklusion

Der skete meget i sprint 2, efter at vi i sprint 1 endelig havde fået forbindelse til Websphere MQ. Efter den første dag kunne vi vise information fra Websphere MQ via Console, og efter anden dag kunne vi vise det via en ASP.NET webform. Samtidig arbejde vi på ekstra opgaven, så programmet kunne køre uden en database, det valgt vi at gøre med XML. Denne opgave blev også færdiggjort inden for sprint perioden. Det eneste vi ikke nåede at lave var visningen af dataene via farvekode, grunden var at vi ikke var sikre på hvordan man bedste gjorde dette dynamisk. Programmet skulle være dynamisk fordi det var et krav, at man skulle kunne tilføje queues til deres queue manager, XML-filen og programmet derefter automatisk ville vise den nye queue.

Vi nåede meget på dette sprint og fik afsluttet næsten alle de backlog items vi havde taget til dette sprint. Vi vil forsøge at finde en løsning på problemet med farve boksende i næste sprint. Ved afslutningen af sprint 2 var vi derfor næsten klar til at finde en løsning på hvordan vi kunne vise informationen fra flere queues.

Sprint 3 backlog

Vi vil fokusere på at få viewet og trådende til at fungere på dette sprint. Vi vil prøve at iterere på hvordan vi viser dataene.

	Navn	Værdi	Tid	Test	Noter
4	udvid systemet så det kan vise informationen via ASP.NET med farvekode	35	50	kan se korrekt og opdateret information via JSP	færdig inden for sprintet
5	Udvid systemet så det kan håndtere flere queues	60	50	man kan se flere køer med korrekt og opdateret information	færdig inden for sprintet dog med mangler
6	Vis information for flere	35	40	Test at information fra	problemer med den

	queues			flere queues virker	trådede løsning

Sprint 3 konklusion

Vi fik i det tredje sprint fundet ud af, hvordan vi kunne vise flere bokse med queues i vores View. Løsningen kom som en indkapslet tabel i et view, som så i controlleren udfyldes i et foreach loop, der sørger for at der for hver queue oprettes et tabel, hvori informationerne bliver vist.

Løsningen er et skridt i den rigtige retning og gør at vi nu kan fokusere mere på at sørge for at opdateringsintervallerne overholdes, så at disse kører individuelt frem for alle på samme tid.

Vi havde dog stadig en del problemer med at programmerne havde en tendens til kun at ville køre på de computere, hvorpå de var blevet lavet originalt. Et af problemerne ligger i at dll-filerne ikke virker til at ville køres fra Dropbox.

Den midlertidige løsning er blevet at man opretter et nyt projekt og kopierer tingene over, hvorefter man kan køre programmet.

Sprint 4

Vi havde nogle problemer med at få udvidet system i forrige sprint, så vi har taget disse to backlog items med igen da vi ikke mente det var blevet færdiggjort. Vi tog forhindringerne vi havde med trådene med som et bug-fix, da vi mente problemet vi havde med det var tæt på en løsning.

	Navn	Værdi	Tid	Test	Noter
5	Udvid systemet så det kan håndtere flere queues	60	50	man kan se flere queues med korrekt og opdateret information	færdig inden for sprintet

bug-fix	find en løsning på problemerne med tråde	50	50	Test at information fra flere queues bliver hentet korrekt	blev færdig fandt en løsning uden tråde

Sprint 4 konklusion

Vores store problem med tråde var at opdateringsintervallet ikke blev overholdt, under kørsel. Vi prøvede en del forskellige løsninger til de forhindringer vi havde med trådene, blandt andet Threading tasks som vi havde læst kunne hjælpe, men heller ikke det virkede for os da opdateringsintervallet stadig ikke blev overholdt. Vi fandt senere ud af at problemet var i Page_Load. Den !ispostback vi havde sat, blev simpelthen ikke kaldt når siden blev genindlæst, af årsager vi ikke helt forstod. Senere viste det sig, at være fordi vi havde givet Page_Load separat updater i viewet, som valgte at ignorere den !ispostback condition vi havde sat i controlleren.

Den updater vi havde sat var i form af et '<meta http-equiv="refresh" content="10" >'

Da vi fandt ud af det var updateren vi havde sat i viewet, var vi allerede gået over til en ny løsning, som involverede en asp:timer.

Vi fik dog efter et møde med regionen et godt råd til hvordan dette kunne løses, løsning gav os muligheden for helt at droppe trådene. Da vi snart ikke havde mere tid flere sprints valgt vi at bruge denne løsning, løsning er beskrevet nærmere i programmering afsnittet i denne rapport. Ved afslutning af sprint 4 havde vi dermed, et funktionelt proof of concept, til regionen.

Konklusion på sprint forløbet

Vi kom som før nævnt forholdsvis sent i gang med at sprinte i projektperioden, det skyldtes hovedsageligt at vi skulle sætte os ind i meget nyt materiale. Men også til tider ringe kommunikation med Regionen, som hovedsageligt udsprang af travlhed blandt de kontaktpersoner vi havde fået tildelt af regionen. Det gav det udslag at vi desværre ikke kunne nå lige så mange sprints som vi havde ønsket fra starten. Vi kunne heller ikke køre med det sprint længde på to uger som vi ellers havde planlagt fra starten, da vi på den måde

kun ville kunne have gennemført mere end to hele sprints. Dette medførte selvfølgelig at vi ikke kunne uddybe de enkelte Backlog items, så meget som vi måske kunne have ønsket da vi startede af projektet.

Vi nåede på grund af forsinkelserne i starten af projektet, ikke at komme i gang med at integrere Zabbix overvågningen i projektet, men dette var også en anden prioritet for regionen. Dette var selvfølgelig ikke optimalt, men som før nævnt var prioriteten fra regionens side at vi fik selv Websphere MQ monitoringen til at virke.

Vi skiftede efter første sprint programmerings sproget over til C#, dette skete hovedsageligt på grund af tidspres, og vi mente det ville være optimalt at kunne aflevere noget funktionelt til regionen ved projektets afslutning. Grunden til vi valgt at skifte til C# var todelt, den første grund var at vi som programmører havde mere erfaring med at programmer i dette sprog, men også at den konsulent som i sidste ende hjalp os med at få oprettet en forbindelse også var bedst kendt med C#. Skiftet til C# blev selvfølgelig diskuteret med regionen inden den endelige valg blev truffet, men også regionen mente at det var nødvendigt at skifte til c#, hvis vi skulle have noget ud af det i sidste ende. Dette var selvfølgelig ikke optimalt i forhold til regionens ønsker, og vi blev til et møde med regionen hen mod slutningen af projektperioden, fortalt at vores projekt formentlig vil bliver brugt som et såkaldt 'proof of concept'.

Vi synes begge at vi fik meget ud af vores sprints, og ved vores møder med regionen har de også udvist tilfredshed med det leverede produkt.

Vi mener at ved at bruge SCRUM, og dermed sprints gav os de værktøjer vi havde behov for, for at takle den høje usikkerhed der var i starten af projektet. Det viste sig også at være en god ting med sprints af en uge varighed i starten, da det afgrænsede den tid vi havde til de enkelte opgaver. På den måde blev vi på en naturlig måde nød til at fokusere meget på at få de enkelte opgaver afsluttet.

Vi har helt bestemt lært en masse omkring brugen af Websphere MQ og hvordan man tilgår denne. Vi mener at vi med den viden vi har indsamlet ii løbet af projektet, vil kunne hjælpe regionen meget i eventuelle fremtidige projekter inden for samme område, og at vi med det produkt vi aflevere til dem, kan videre give dem denne viden.

Vi synes vores sprints har forløbet rigtig godt og at vi i sidste ende har fået det bedste produkt ud af det, som der var muligt indenfor de rammer vi endte med at arbejde med.

Sekvensdiagrammer(Christopher)

Sekvensdiagrammer benyttes til at vise hvad funktionalitet i systemet gør, ved at give et indblik i hvad der sker, når en bruger benytter systemet eller en automatisk proces startes. Hertil viser sekvensdiagrammer hvilke metodekald der foretages i hvilke klasser, ved brug af systemets funktioner. Sekvensdiagrammer kan også vise om visse handlinger i systemet foretages flere gange (loop) eller kun foretages under visse omstændigheder (condition/if).

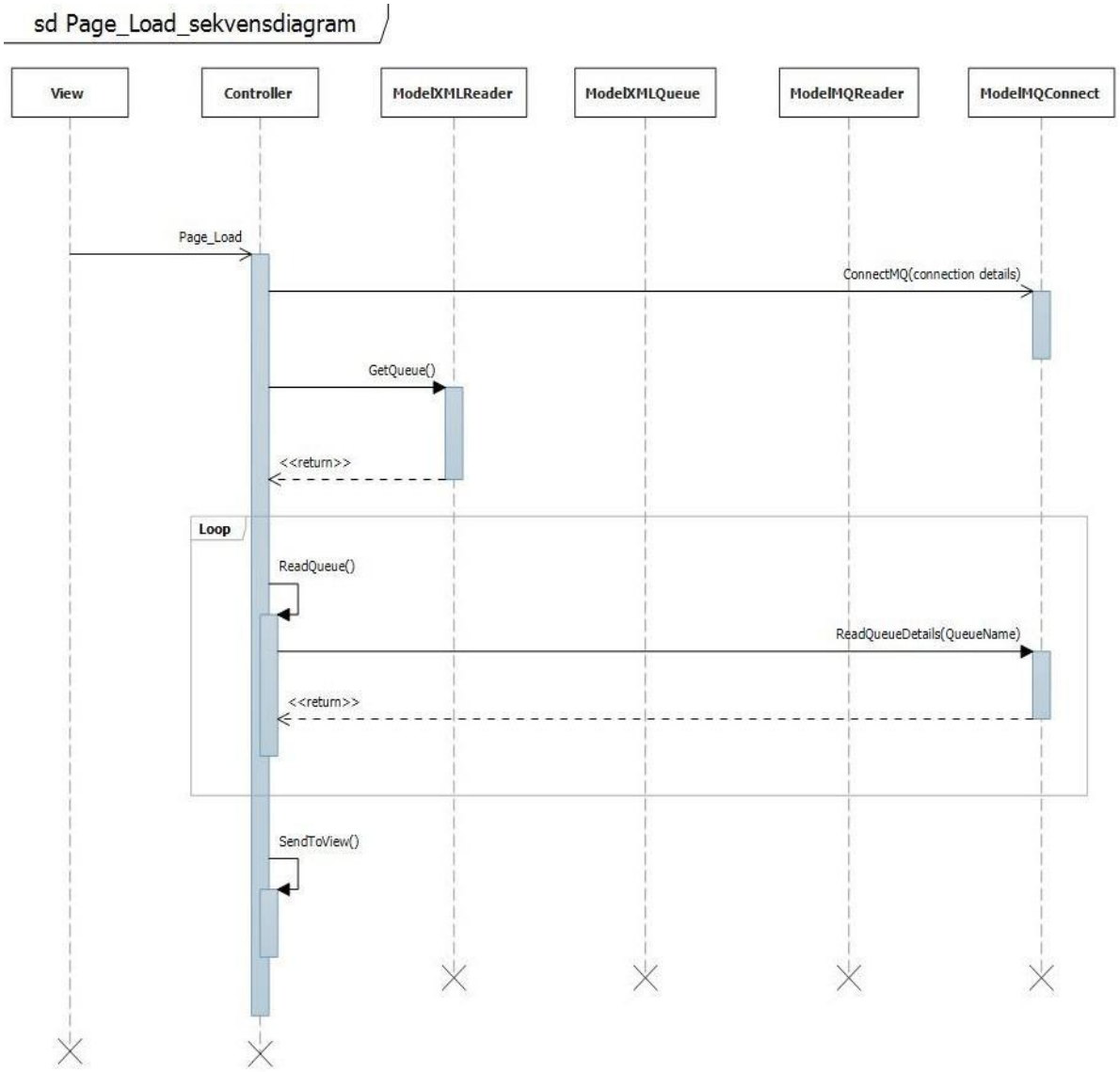
Til vores system har vi valgt at udarbejde to sekvensdiagrammer, hvor det ene viser hvad der sker når Page_Load() metoden køres, når programmet starter og Timer1_Tick(), som køres umiddelbart efter Page_load().

I sekvensdiagrammene tager vi udgangspunkt i kørsel af sekvensdiagrammerne fra asp-siden View, hvor metoderne som sagt automatisk startes efter hinanden, når programmet køres.

Vi har valgt at vise loops og conditionals i diagrammene, men undlader at vise asp-elementer, som starter metoderne og i Timer1_Ticks tilfælde gentager dens kørsel. Vi har heller ikke valgt at vise hvad der vil ske, såfremt der opstår en fejl i forsøg på at oprette forbindelse eller læse queues, men har dog sikret metoderne med exception handling i form af try-catch, som vil fange fejl og vise problemer frem, som måtte være opstået, frem for at programmet crasher.

I de to afsnit herunder, vil vi med to sekvensdiagrammer hvad der sker i metoderne Page_Load() og Timer1_Tick(). Vi vil til begge diagrammer også beskrive hvad metoderne i diagrammerne foretager sig.

Sekvensdiagram til Page_Load()



I dette diagram viser vi hvordan Page_Load() kaldes i controller klassen fra viewet.

Efter Viewet har kaldt Page_Load(), kører Page_Load() ConnectMQ() metoden, som ligger i vores ModelMQConnect klasse, som er et data Access layer. ConnectMQ() benytter de parametre den er udstyret med, til at oprette en forbindelse til Websphere MQ.

Når ConnectMQ har oprettet forbindelse, starter GetQueue() metoden, som kaldes i ModelXMLReader. GetQueue er en dictionary metode, dictionary er en key-value store tabel, det betyder at den er en sammensætning af en nøgle og en værdi, hvor nøglen bruges til at henvise til værdien, uden nøglen kan man dermed ikke få fat i den henviste værdi. Nøglen og værdien i dictionary kan være af næsten en hvilken som helst art, blot man definerer det ved

oprettelsen af dictionarien. I dette tilfælde benyttes string med queuename som key og et objekt af typen modelxmlqueue.

Når dictionarien er blevet fyldt med alt information fra xml filen, sender vi dictionarien til vores controller klasse. Informationen fra dictionarien bliver herefter brugt i et foreach loop, hvori informationen sendes ind i metoden ReadQueue().

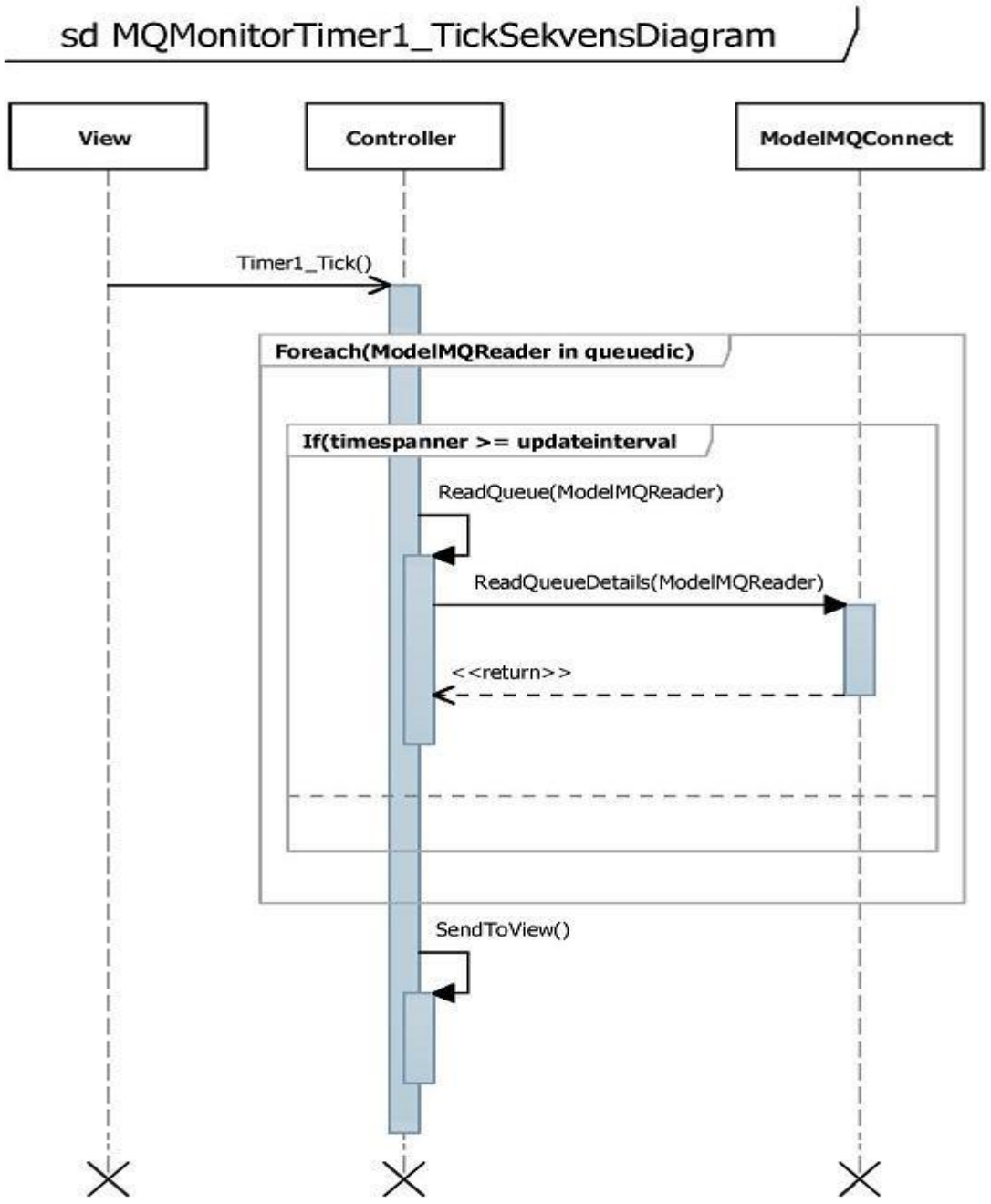
ReadQueue() bliver efterfølgende brugt til at behandle informationen, der er i dictionary. Metoden består af to dele: En læsning af information fra en queue, ved hjælp af ReadQueueDetails(), som ligger i ModelMQConnect klassen.

Herefter kommer en Conditional**,** der benytter den info til at sammenligne en tidsværdi fra xml filen, med en tidsværdi fra den queue der lige er blevet læst.

Er værdien fra queue mindre end den sat i xml filen, vil vores dictionary blive opdateret med en værdi og hvis den er højere eller lig med den værdi sat i xml filen, vil den opdatere med en anden værdi.

Efter at ReadQueue er færdig med at opdatere info i dictionary, kører vi SendToView() metoden, som så looper igennem dictionarien og værdien er sat i dictionary, benytter den en conditional til at opdatere viewets bokse, med info og den ene eller den anden farve.

Sekvensdiagram til Timer1_Tick()



Timer1_Tick()s og Page_Loads sekvensdiagram har et par fællestræk, de benytter begge ReadQueue(), som bruges til at hente data fra queue ved hjælp af ReadQueueDetails() i ModelMQConnect klassen. Begge metoder sætter også en bool værdi på et ModelMQReader objekt i dictionaryen og begge benytter SendToView til at sende dataene op til viewet.

Forskellen på Page_Load() og Timer1_Tick er, at sidstnævnte ikke har brug for at hente en liste med queue information ved hjælp af GetQueue() metoden, som er beskrevet i sidste afsnit, da denne liste allerede er hentet. Timer1_Tick() har heller ikke brug for at oprette en forbindelse til Websphere MQ, da denne blev oprettet i Page_Load() metoden.

Timer1_Tick() har dog en conditional statement omkring ReadQueue(), som ikke findes i Page_Load() metoden. Denne conditional sørger for at medmindre dens betingelse bliver opfyldt, køres ReadQueue() ikke. Dermed foretages der ingen hentning af data med ReadQueueDetails og ingen ændring af ModelMQReader objektets bool-værdi, medmindre betingelsen i vores conditional statement bliver opfyldt.

Programmering(Kim + Christopher)

Vi har i de tidligere afsnit omkring arkitektur og sekvensdiagrammer snakket om hvordan systemet er opbygget. I arkitektur afsnittet forklarede vi sammenhængen mellem klasserne, deres formål og grunden til vi har valgt at benytte en MVC-arkitektur. I sekvensdiagrammerne viste vi, hvordan de forskellige metoder i klasserne kaldes fra controller klassen og i dette afsnit går vi skridtet videre og forklarer hvordan koden i de forskellige metoder i de forskellige klasser virker. Vi har delt det hele op i afsnit, hvor hovedafsnittet vil være klassen. I disse afsnit forklarer vi klassens hovedformål og i underafsnittene forklarer vi metoderne der ligger i de individuelle klasser. Metodernes afsnit vil både dække deres nuværende kode og tidligere iterationer. Vi håber at dette vil give et bedre billede af processen, som hver klasse og metode har gennemgået.

Test

Sideløbende med vores programmering er der foretaget utallige tests for at se om ting virker som de skulle. Test af om der bliver sendt den rigtige information til Websphere MQ, test af om informationen modtages rigtig fra Websphere MQ. En af de ting vi nok har brugt mest tid på at teste var behandlingen af den information der blev sendt til view. Her er der gjort utrolig mange tests for at finde ud af hvorfor information vi ønskede ikke kom frem, ikke blev vist i rigtig rækkefølge eller blev opdateret i det rigtige interval.

Vores tests har været af forskellige karakterer, for nogle områder har det været nødvendig at kopiere kode over i separate projekter for at teste det, i andre tilfælde er koden startet som

et separat projekt og efter at vi havde testet at det virkede, overførte vi det til hovedprojektet. Et eksempel her er hvordan vi først testede vores XML læsning og skrivning i et projekt for sig, hvor vi sikrede os at der kom den rette information ud af XML-filen og at der ved skrivning til XML-filen blev oprettet de rette elementer de rette steder.

I selve projektet er der også foretaget step-by-step tests, hvor koden er kørt linje for linje, for at finde ud af præcis hvor noget kode ikke virkede som ønsket. Til dette er der i Visual Studio en indbygget funktion, der lader programmører se hvordan de forskellige variabler bliver opdateret og får tildelt værdier.

Et eksempel på hvor vi benyttede dette, var ved indlæsning af elementer fra XML-filen, hvor vi i controlleren skulle sikre os at elementerne blev overført til vores dictionary, som ønsket.

Mange fejl blev fundet, hvor en variabel ikke lige havde fået det rigtige navn eller ikke fik tildelt den data de var tiltænkt på grund af en manglende parentes sæt eller metoder der ikke blev kaldt i den rigtige rækkefølge.

I disse tilfælde har både step-by-step tests og slavisk gennemgang af kode, efter forsøg på kørsel af hovedprojektet, hjulpet os med at finde fejl, som var af afgørende betydning for at programmet virkede.

Controlleren: View.Aspx.cs

```
ModelMQConnect _mqc = new ModelMQConnect();
ModelXMLReader qr = new ModelXMLReader();
public static Dictionary<string, ModelMQReader> queuedic = new Dictionary<string, ModelMQReader>();
string htmlstring;
private readonly string _queuemanagername = ConfigurationManager.AppSettings["QueueManagerName"];
private readonly string _serverip = ConfigurationManager.AppSettings["QueueManagerIP"];
private readonly int _queuemanagerport = Convert.ToInt32(ConfigurationManager.AppSettings["QueueManagerPort"]);
private readonly string _channelname = ConfigurationManager.AppSettings["QueueManagerChannel"];
```

Controlleren er kontrolcenteret for hele systemet, det er fra denne hvor hele systemet skal køre automatisk. I toppen af klassen har vi en del variabler, de to første er erklæringer af objekter af klasserne ModelMQConnect og ModelXMLReader. Som navnene siger, er der tale om klasser i modellaget, disse to klasser er hvad der bruges til at foretage de fleste af de nødvendige handlinger der skal ske via controlleren.

Efter disse to, kommer fire variabler, som indeholder information fra vores config fil, de fire variabler henter information der er sat i config filen. Grunden til de skal hente informationen fra config-filen, er at vi i vores controller ikke ønsker statisk information, vi ønsker at controllerklassen skal være så dynamisk som muligt og dermed give brugeren af programmet nemmere mulighed for selv at styre ting, så som navnet på Queuemanager, dennes IP, port og eventuel channel. På den måde vil det for brugeren ikke være nødvendig at ændre i den egentlige programkode, Ved kørsel er et C# projekt kompileret, sådan at koden ikke længere er i et let læseligt sprog og dermed vil det ikke være let at ændre. Ved at sætte variablerne i dele af programmet, som ikke bliver kompileret, er det gjort lettere for brugeren at senere ændre denne information, så at det passer specifikt til deres system. Det betyder også at hvis brugeren senere vil benytte programmet fra en ny server, en ny queuemanager eller vælger at ligge xml filen på en anden lokation end den vi har sat, vil det ikke være svært for brugeren at ændre denne information.

Page_Load

```
protected void Page_Load(object sender, EventArgs e)
{

    try
    {
        _mqc.ConnectMQ(_queuemanagername, _serverip, _queuemanagerport, _channelname);
        if (!IsPostBack)
        {
            //queuedic gets information from getqueue about all queues in xml-file
            queuedic = qr.GetQueue();

            //first run of the program,it reads all the queues in queuedic
            foreach (ModelMQReader t in queuedic.Values.ToList())
            {
                ReadQueue(t);
            }
            //and then sends it all to the innerhtml
           Sendtoview();
        }
    }

    catch (Exception)
    {
        _mqc.DisconnectMQ();
    }

}
```

Som tidligere nævnt, er Page_Load metoden den første der køres når programmet startes. Metoden er omkranset af en try-catch funktion. Try-catch er en sikkerhedsfunktion, der gør at såfremt der opstår en fejl i koden, som metoden prøver at kører, vil der i stedet for et systemcrash, ske en tilbagemelding i form af en fejlmeddelelse eller som i tilfældet med denne metode, foretage en disconnect fra Websphere MQ.

Det metoden her er sat til at afprøve, er først at forbinde til Websphere MQ. Dette gør den med de tidligere nævnte variabler, som indlæser information fra configuration-filen.

I en if(!postback) tildeler vi en dictionary kaldet queuedic informationen fra xml-filen.

Informationen vi henter i xml-filen, skal vi bruge i den næste del af Page_Load, hvor vi i et foreach loop sender alle ModelMQReader objekterne i vores dictionary ind i metoden ReadQueue.

ReadQueue

```
public void ReadQueue(ModelMQReader d)
{
    lock (this)
    {
        d = _mqc.ReadQueueDetails(d); //read from queue

        if (d.GetMaxQueueAge >= d.GetQueueAge) //is age higher or lower than maxage?
        {
            d.SetAgeChecker = true; //if it is less than maxage, set agechecker equal to true
            if (queuedic.ContainsKey(d.GetQueueName)) //if queue with queuename exists in dictionary do
            {
                queuedic[d.GetQueueName] = d; //if yes, then set its values to d
            }
            else
            {
                queuedic.Add(d.GetQueueName, d); //else add it to the dictionary.
            }

        }
        else //if age is higher than maxage do this
        {
            d.SetAgeChecker = false; //agechecker false
            if (queuedic.ContainsKey(d.GetQueueName)) //same as above
            {
                queuedic[d.GetQueueName] = d;
            }
            else
            {
                queuedic.Add(d.GetQueueName, d);
            }
        }
    }
}
```

Readqueue metoden skal benytte ModelMQReader objekterne til to ting:

Første ting den skal bruges fra objekterne er navne variablen tilknyttet hver af dem. Disse bruger den til at læse Queuedetails med metoden ReadQueueDetails, der ligger i ModelMQConnect klassen, ReadQueueDetails vil blive beskrevet nærmere i afsnittet til ModelMQConnect klassen.

Det næste metoden skal benyttes til, er at behandle den information, som sendes tilbage fra ReadQueueDetails. Dette gør den ved hjælp af den af de to parametre MaxQueueAge og GetQueueAge.

Den første af de to parametre beskriver en grænseværdi, som er sat til den hver enkelt queue, i XML dokumentet. Hvis værdien i GetQueueAge overskrider eller er lig med værdien i MaxQueueAge, vælger metoden at sætte bool variablen AgeChecker til false, ved hjælp af setter metoden SetAgeChecker.

Metoden vil herefter undersøge, hvorvidt der i vores dictionary eksister en key, med navn på den Queue vi lige har undersøgt tiderne for. Dette gøres ved at benytte en metode der er tilknyttet dictionary; dictionary.containskey(). Viser det sig at den netop læste queue, eksisterer i vores dictionary, vælger vi at ændre værdierne tilknyttet det ModelMQReader objekt, hvor informationen omkring denne queue bliver gemt. Dette gøres ved at sætte Objektet i dictionarien lig med den der blev indlæst via page_load og som netop har fået ændret værdien på sin bool parameter. På denne måde opdaterer vi informationen, som så kan benyttes af metoden SendToView().

SendToView

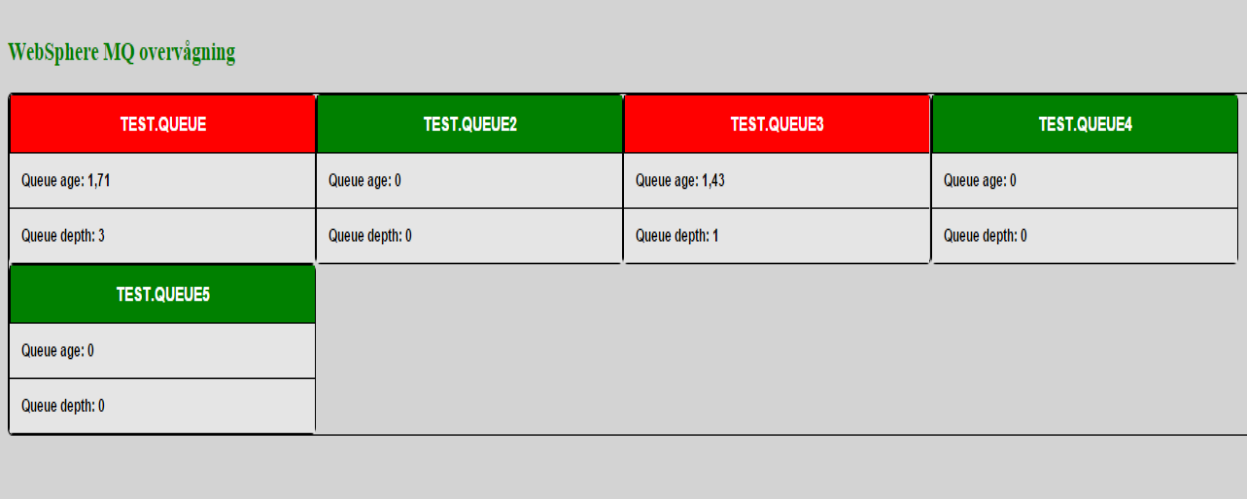
```
public void Sendtoview()
{
    htmlstring = "";
    foreach (ModelMQReader t in queuedic.Values.ToList())
    {
        htmlstring += "<div class='box'><table ><tr>";
        htmlstring += "<table ><tr>";
        if (t.GetAgeChecker)
        {
            htmlstring += "<td style='background-color:green'>" + t.GetQueueName + "</td>";
        }
        else
        {
            htmlstring += "<td style='background-color:red'>" + t.GetQueueName + "</td>";
        }
        htmlstring += "<tr><td> Queue age: " + t.GetQueueAge + "</td></tr>";
        htmlstring += "<tr><td>Queue depth: " + t.GetQueueDepth + " </td></tr>";
        htmlstring += "</table></div>";
    }
    div.InnerHtml = htmlstring;
}
```

SendToView er den metode, som vi benytter til at sende information omkring forskellige queues til view laget. Metoden benytter en htmlstring, som indsættes ind i en div class kaldet box. Dette foregår i et foreach loop, som sikrer at alle ModelMQReader objekternes parametre, bliver benyttet til at sende information til viewet. Inde i loopet ligger en conditional der undersøger tilstanden på bool parameteret på det specifikke ModelMQReader objekt, som loopet netop er i gang med at gennemgå. Såfremt bool værdien er sat til true, sættes tabels baggrundsfarve til at være grøn, i den linje hvor navnet på Queuen kommer til at stå. Når dette er sket fyldes de efterfølgende linjer med QueueAge og Queuedepth.

Dette loopes igennem, til der i dictionary ikke eksister flere queues, hvorefter vi ved at sætte div.innerhtml lig med den htmlstring, som vi netop har udfyldt.

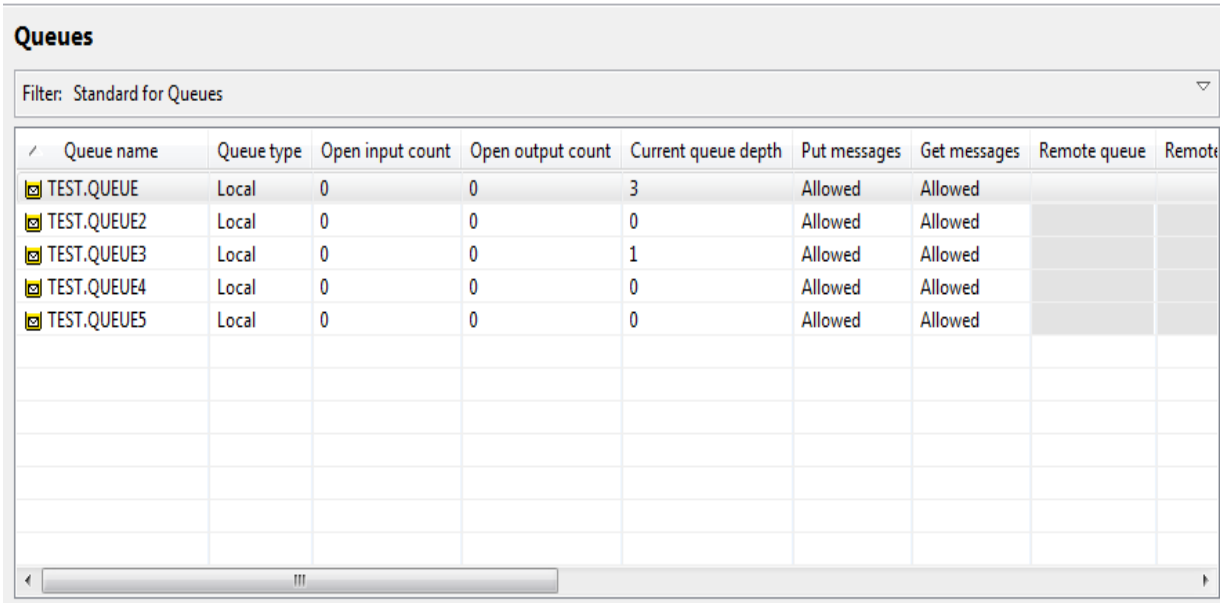
SendToView er det sidste der bliver kørt i både metoden page_load og metoden timer1_tick, da man ikke ønsker at opdatere view, før man har gennemgået al information fra queues i dictionary.

Outputtet fra SendToView



Sådan ser outputtet ud hvis vi har fem queues i XML-filen, programmet loader automatisk flere queues ind hvis man tilføjer flere queues til XML-filen og Websphere MQ.

Som man kan se i billedet ovenfor, viser vi queue age og queue depth. Vi valgte at holde os til disse to datapunkter, da vi mente de var det mest relevante på dette stadie af udviklingen. Men der er selvfølgelig mulighed for at udvide systemet, så det kan vise alle variabler i vores data holding klasse ModelMQReader. billedet ovenover har læst vores test queues i Websphere MQ queue manageren, som da billedet ovenover blev taget så ud som billedet herunder.



Som man kan se har vi det samme antal queues her, som bliver vist i vores view og den samme queue depth.

Disse queues er selvfølgelig nogle vi selv har oprettet, da vi ikke havde adgang til regionens Websphere MQ testmiljø.

Man kan gennem Websphere MQ manuelt sætte nye beskeder ind i vores queues, dette var nyttigt når vi skulle teste opdaterings intervallet og om dataene vi fik ud var korrekt.

Timer1_Tick

```
protected void Timer1_Tick(object sender, EventArgs e)
{
    DateTime timeNow = DateTime.Now;
    TimeSpan tstimer;

    foreach (ModelMQReader r in queuedic.Values.ToList())
    {
        tstimer = timeNow - r.GetLastUpdated;
        double timerspanner = tstimer.TotalSeconds;

        if (timerspanner >= r.GetUpdateInterval)
        {
            ReadQueue(r);
        }
    }
    Sendtoview();
}
```

Denne metode kom til efter at vi i flere iterationer, havde forsøgt med forskellige typer tråde, at få viewet til at opdatere de specifikke queues i intervaller. Vi forsøgte først at benytte et loop omkring tråde der blev overloaded via lambda metodikken. Lambda funktionen kan benyttes med tråde, når man ønsker at der skal parameter værdier med, i den metode der skal kaldes. Problemet her viste sig at være, at vi ikke kunne finde nogen nem måde at forhindre vores loopede tråd, i blot at starte en ny tråd, ved hver gennemgang af loopet, frem for hvad vi egentlig ønskede. Den ønskede funktionalitet, var at der ved hver iteration af loopet, skulle benyttes de tråde, som allerede var lavet i den første iteration af loopet. Dette var dog ikke hvad der skete, hvad der skete, var netop at den ved hver gennemkørsel af loopet, oprettede helt nye tråde og dermed ignorerede de tråde der allerede kørte, fra sidste gennemkørsel af loopet.

Et andet problem, som var med system.threadings tråde, var at vi ikke kunne vælge at benytte et while(true) loop omkring det loop med trådene. Såfremt vi forsøgte at gøre dette, ville hjemmesiden blot ikke vise sig, da den blev ved med at behandle det samme information om og om igen.

```
static void Main(string[] args)
{
    Testy test = new Testy();
    Random random = new Random();
    //øverste metode skal køre i et
    var tasks = new Task[3];
        for (int i = 0; i < 3; i++)
        {
            int randomnumber = random.Next(3, 10);
            TimeSpan t = TimeSpan.FromSeconds(randomnumber);
            int k = i;
            tasks[i] = Task.Factory.StartNew(() => test.Taller(k, t));
        }
    Task.WaitAll(tasks);
    Console.WriteLine("Færdig");
}
```

Det næste vi forsøgte, var at benytte tasks, fra system.threading.tasks. Denne model havde vi set kunne virke i et konsol program, hvor vi ved hjælp af en funktion tilknyttet tasks, kaldet WaitAll havde set at vi kunne få task trådene til at vente med at starte en ny instans af en tråd, til den tidligere var færdig med at køre.

Vi forsøgte derfor at overføre dette til vores asp.net projekt og se om vi kunne få det til at virke her. Dette gjorde vi på nogenlunde samme måde, som med almindelige system.threading tråde, blot med inkludering af funktionen waitall. Vi satte den altså til, i et loop i et while(true) loop, at oprette tråde med ModelMQReader objekterne i vores dictionary. Vi mente dermed at vi burde have fundet vores løsning, problemet var dog bare stadig, at når vi forsøgte at køre projektet, blev den aldrig færdig med at indlæste hjemmesiden og der kom ikke noget frem på skærmen. Vi gav dog ikke op og forsøgte i flere omgange at prøve at få både threading tråde og threading.task tråde, til at virke efter hensigten, men måtte i sidste ende opgive begge måder at styre tråde på. Vi var blevet klogere på hvordan forskellige former for tråde virkede, men havde ingen resultater der kunne bruges til vores projekt.

Gennembruddet kom, efter en snak med Morten Lyhne Thuesen fra regionen. Morten spurgte ind til, hvorvidt det ikke ville være muligt at benytte en timer, til at styre opdateringsintervallet og vi valgte herefter at undersøge, hvorvidt det ville virke.

Det viste sig at vi godt kunne styre opdateringen af alle tråde ved hjælp af en timer, men dog stadig skulle benytte en conditional inde i loopet, til at sikre at den kun læste de specifikke queues i de ønskede tidsintervaller. Vi benytter derfor i timer1_tick metoden en conditional der sammenligner to tidsintervaller. Det ene tidsinterval, ville vi få fra vores XML fil og det andet ville vi ved at benytte en timespan der tog forskellen på tiden, hvorpå en queue sidst var blevet læst, med hvad tiden nu er. Såfremt tiden i vores timespan er lig med eller højere end tiden sat i vores XML fil, fik metoden lov til at foretage en ny læsning på en queue. ved at foretage en ny læsning på en queue, satte den også en ny tid til, hvornår denne queue sidst var læst og vores conditional vil på ny være false indtil en hvis tid er gået.

ModelMQConnect klassen

```
public class ModelMQConnect
{

    MQQueueManager _mqManager;
    private const int OpenInputOptions = MQC.MQOO_INPUT_SHARED | MQC.MQOO_BROWSE | MQC.MQOO_INQUIRE | MQC.MQOO_FAIL_IF_QUIESCING;
```

Som navnet på klassen siger, er dette en modellags klasse og denne klasse skal bruges til at forbinde til Websphere MQ.

Klassen har derfor også metoder til at oprette forbindelse til Websphere MQ, men benyttes desuden også til de metoder der skal frakoble programmet fra Websphere MQ, såfremt der opstår fejl og indeholder metoden, som bruges til at bringe data tilbage fra Websphere MQ's queuemanagers queues.

Som det kan ses af billedet herover starter klassen med at oprette et objekt af typen MQQueuemanager, dette er objektet der bruges til at oprette forbindelse til Websphere MQ. Objektet ligger i starten af klassen, sådan at forbindelsen ikke kun eksisterer inde i metoden, som opretter forbindelsen.

Det næste der ses er en konstantieret int, med en samling værdier. Grunden til der bruges en konstant, er at man ikke ønsker at værdierne i variablen ændrer sig senere, samt at de

værdier der tildeles den er af en konstant int type, dette ses også i navnet på de efterfølgende attributter, da MQC står for Message Queue Constant.

Den første attribut sat er MQOO_INPUT_SHARED. Denne attribut forklarer Websphere MQ at der med forbindelsen der bliver oprettet er åbnet for at benytte MQget kald, såfremt der er oprettet forbindelse fra dette eller et andet program der benytter samme kald.

Den næste Attribut er MQOO_BROWSE. Med denne i vores forbindelse, forklarer vi Websphere MQ at programmet der benyttes har tilladelse til at hente information om meddelelser der ligger i diverse queues.

Herefter er der MQOO_INQUIRE, denne attribut forklarer Websphere MQ at programmet har tilladelse til at læse attributter på queues. Der kan her være tale om hvor mange meddelelser der er på en queue, hvor gamle meddelelser, som ligger i queues er og om den blot skal læse hvor gamle meddelelserne er eller om den skal hente dem fra Websphere MQ og derefter slette dem i deres queues.

MQOO_FAIL_IF_QUIESCING, fortæller Websphere MQ, at såfremt der opstår et problem i forsøget på at oprette en forbindelse, skal den lukke ned. Quiescing er en måde at få et system til at lukke ordentligt ned, såfremt en fejl opstår og eksisterer også i visse andre programmer.

Efter disse parametre er sat, kan vi gå i gang med at oprette forbindelsen til Websphere MQ, hvilket vi gør i void metoden ConnectMQ.

ConnectMQ metoden

```
public void ConnectMQ(string mqManagerName, string mqServerIP, int mqServerPort, string mqChanelName)
{
    string transport = "";
    try
    {
        Hashtable connectionProperties = new Hashtable();

        //server
        if (mqServerIP == "" || mqChanelName == "")
        {
            transport = MQC.TRANSPORT_MQSERIES_BINDINGS;
            connectionProperties.Add(MQC.TRANSPORT_PROPERTY, transport);
        }
        else //client
        {
            transport = MQC.TRANSPORT_MQSERIES_CLIENT;
            //connect as client
            connectionProperties.Add(MQC.TRANSPORT_PROPERTY, transport);
            connectionProperties.Add(MQC.HOST_NAME_PROPERTY, mqServerIP);
            if (mqServerPort != -1)
            {
                connectionProperties.Add(MQC.PORT_PROPERTY, mqServerPort);
            }
            connectionProperties.Add(MQC.CHANNEL_PROPERTY, mqChanelName);
        }
        _mqManager = new MQQueueManager(mqManagerName, connectionProperties);
    }
```

ConnectMQ er metoden, som bruges til at oprette forbindelse til Websphere MQ. Den er nu en metode af typen void, men har tidligere været både string og bool. Metoden startede som en string metode, for at vise til slut, at der nu var oprettet forbindelse til serveren. Dette droppede vi dog, da vi ikke synes der var nogen grund til at vise denne linje i viewet. Nu er det bekræftelse nok, at boksene med message queues viser sig, for at vise at der er forbindelse. såfremt der ikke var forbindelse ville der ikke komme nogen af de tabeller frem, hvor message queue status står i.

Af samme grund valgte vi også at gå væk fra at have den som en bool metode, vi får bekræftelse på forbindelse ved at tabeller viser sig.

Som metoden er nu, er den af typen void, med 4 parametre: MQManager navn, server ip, port og channel. Disse sikrer at metoden kan køres fra mere end en lokation og at hvis Region Syddanmark senere ønsker det, kan de via config filen blot ændre værdierne der bliver sat ind.

Metodens indhold er indkapslet i en try catch og giver i catch parten mulighed for at en udvikler kan sørge for at der bliver vist en fejlmeddelelse, såfremt der måtte opstå en fejl, når man forsøger at oprette en forbindelse.

I vores try, starter vi ud med at oprette et hashtable, til at opsamle informationen omkring vores forbindelses detaljer, derfor også navnet connection properties til den.

Efterfølgende starter vi en conditional, der skelner mellem vores forbindelsestype, såfremt vi ikke angiver nogen værdier til serverip og channelname, betyder det at vi vil benytte en bindings forbindelse, hvilket betyder at det er en lokal forbindelse og ikke en forbindelse som client til MQManager.

I en bindings forbindelse skal vi som vist på billedet af metoden benytte MQC.TRANSPORT_SERIES_BINDINGS, hvilket selvsagt forklarer MQManageren at vi opretter forbindelse til den lokalt, med en bindings forbindelse.

I vores else condition forsøges der at oprette forbindelse til MQManageren, som client. Dette skal bruges i server/client løsninger og giver mulighed for en fjernforbindelse, hvor man eventuel kan benytte en TCP-løsning, hvor modellaget lægges på serveren og klienten besidder view og controlleren. Af selvsamme grund er modellaget også bygget i dette program, som en del der kan stå selvstændigt, hvilket vil gøre det nemmere, senere at konvertere indholdet af projektet til en server/client løsning.

Efter vi har tilføjet data til vores hashtable omkring forbindelsestypen vi ønsker at oprette, tilføjer vi det til MQQueueManager objektet, som opretter forbindelsen til queue manageren i Websphere MQ.

Såfremt ingen problemer opstår i forsøget på at oprette forbindelse, vil der nu være oprettet forbindelse til queue manageren indtil programmet lukkes ned eller disconnect metoden køres.

DisconnectMQ metoden

```
public void DisconnectMQ()
{
    if (_mqManager != null)
    {
        _mqManager.Commit();
        _mqManager.Disconnect();
    }
}
```

Disconnect metoden er forholdsvis simpel, den starter ud med at undersøge hvorvidt der i vores MQQueueManager objekt er nogen detaljer og såfremt der er, foretager den først et commit, for at sikre at data der måtte være ved at blive sendt, bliver sendt og afbryder derefter forbindelsen til queue manageren i Websphere MQ.

ReadQueueDetails metoden

```
public ModelMQReader ReadQueueDetails(ModelMQReader mqr)
{
    MQQueue queue = _mqManager.AccessQueue(mqr.GetQueueName, OpenInputOptions);
    DateTime currentTime = DateTime.Now;
    mqr.SetLastUpdated = currentTime;

    try
    {
        MQMessage message = new MQMessage();
        MQGetMessageOptions gmo = new MQGetMessageOptions();

        gmo.Options = MQC.MQGMO_BROWSE_FIRST;

        queue.Get(message, gmo);

        int depth = queue.CurrentDepth;
        DateTime msgage = message.PutDateTime;
        DateTime now = DateTime.UtcNow;
        TimeSpan difference = now - msgage;
        double age = Math.Round(difference.TotalMinutes, 2);

        mqr.SetQueueAge = age;
        mqr.SetQueueDepth = depth;
    }
```

ReadQueueDetails har som ConnectMQ også været gennem en udviklingsfase. Denne startede ligesom ConnectMQ med at være en string metode, og blev senere til både typen List og senere igen til Dictionary, før den som nu blev til en ModelMQReader metode.

Da ReadQueueDetails metoden var af typen string, var det hensigten, at den blot skulle sende en string tilbage til hvor den bliver kaldt. Denne string ville på daværende tidspunkt indeholde information omkring alderen på ældste meddelelse på den Queue der blev angivet med et navn, i en string i parameteret til metoden.

Senere valgte vi at gå væk fra dette, da den information vi ønskede tilbage kunne have forskellige længder og vi mente der måtte være bedre måder at sende information tilbage end en string. Dette var specielt gældende, da de ting der skulle ligges i den string, var et datetime objekt med alderen på ældste meddelelse og en integer der angiver dybden på den givne queue.

Vi forsøgte først med list og dictionary, hvor vi i List ville benytte strings, så at informationen var separeret og i dictionary valgte vi at benytte int og datetime som key og value.

I begge tilfælde kunne vi dog hurtigt se at mens det var muligt for de to at holde på informationen og muligt for os at hente informationen fra metoden i vores kald fra controlleren, var de ikke optimale. List var ikke optimal, da der stadig skulle foretages tilbagekonvertering af indholdet til int og datetime. Dictionary var ikke optimal, da den kun kunne indeholde to stykker information per key-value par.

Den tredje information vi godt ville have med, fandt vi ud af, da vi begyndte at have styr på hvordan vi ville have vores view til at se ud. Informationen vi ville have med var navnet på den queue der lige var blevet læst.

ModelMQReader

Den mest optimale måde at overføre tre stykker information af forskellige variabeltyper, var at benytte en skabelon klasse, denne skabelon klasse blev ModelMQReader.

```
public class ModelMQReader
{
    private string queueName;
    private double queueAge;
    private double queueDepth;
    private double maxQueueAge;
    private double updateInterval;
    private DateTime lastUpdated;
    private bool agechecker;
```

```
public double GetQueueDepth
{
    get { return queueDepth; }
}
public double SetQueueDepth
{
    set { queueDepth = value; }
}
public double GetQueueAge
{
    get { return queueAge; }
}
public double SetQueueAge
{
    set { queueAge = value; }
}
```

Med en skabelon klasse til at indeholde informationen fra vores queue, havde vi nu mulighed for at sende tre stykker information med et objekt. Vi konverterede derfor metoden til typen ModelMQReader og sikrede at informationen vi hentede fra queue, ville blive koblet med information vi sendte ind i metoden, ved at sætte vores parameter til at være af typen ModelMQReader også.

På dette tidspunkt havde vi også indset at der egentlig ingen grund var til at sende en datetime til controlleren. Det vi havde brug for, var trods alt hvor gammel meddelelsen var

og ikke den præcise tid den var blevet sendt til denne queue. Af denne grund valgte vi også at ændre i metoden, så at denne udregnede hvor gammel meddelelsen er.

Dette gør vi ved at indlæse tiden fra queue i en datetime og tiden nu i en anden variabel. Vi tog så forskellen på tiderne med en Timespan.

Da den ønskede maximal alder, før en meddelelse var for gammel på queues alle var i minutter, konverterer vi derfor tiden til en double, hvor vi begrænser antallet af decimaler til to. Vi valgte to decimaler af den grund at tusindedele af et minut ikke ville give mere mening for brugeren, end hundrededele.

ModelXMLReader

Til kravet om at programmet skulle kunne køre uden en database tilknyttet, valgt vi at gøre bruge af en XML-fil. Valget faldt på XML på grund af hvor nemt det er for selv programmerings nybegyndere at redigere XML-filer, samt det fakta at xml ligge lokalt på maskinen programmet kører, så der er ingen unødvendige opkald til en database. Desuden skal queue dataene som skal bruges af programmet, kun hentes en gang ved programmets opstart, var det vores opfattelse at brugen af en database ikke gav meget mening. Da regionen samtidig havde udtrykt en holdning om at de helst ikke ville bruge en database, var valget nemt og faldt som før nævnt på XML.

Til dette formål har vi lavet en XMLreader klasse, med en metode kaldet GetQueue. I denne metode gør vi brug af XML-serializer, StreamReader, StreamWriter, samt en data holding klasse som bruges til både læsning og skrivning af XML-filen.

```
if (File.Exists(filepath))//If the files exist it will be read via Deserializer
{

    XmlSerializer reader = new XmlSerializer(returnlist.GetType());

    StreamReader sr = new StreamReader(filepath);

    returnlist = (List<ModelXMLQueue>)reader.Deserialize(sr);

}
```

Som man kan se ovenover foregår læsning meget simpelt, vi har foroven erklæret returnlist som værende en liste af klassen ModelXmlQueue hvilket er den data holding klasse som er nævnt ovenover, denne klasse vil blive beskrevet senere i dette afsnit. Man bruger denne

returnlist til at fortælle XmlSerializeren hvilket objekt type den skal forvente at få fra StreamReaderen. Derefter typecaster vi den data vi får fra Deserializeren til en list som bliver sat i returnlist. På den måde har vi fået alt data fra XML-filen til at ligge i en liste.

filepath variablen bliver hentet fra vores WebConfig fil, vi valgte at gøre det på den måde for at undgå såkaldt 'hard coded' data i vores program.

```
else// if the file doesn't exist a new file is made via serializer
{

    List<ModelXMLQueue> listQueues = new List<ModelXMLQueue>();
    listQueues.Add(new ModelXMLQueue("Configurate XML File", 5, 5));

    XmlSerializer serializer = new XmlSerializer(listQueues.GetType());
    StreamWriter writer = new StreamWriter(filepath);

    serializer.Serialize(writer.BaseStream, listQueues);

    writer.Close();
    GetQueue();

}
```

Vi valgt for at det skulle være lidt nemmere at få vores program til at køre, at det skulle kunne skrive en dummy XML-fil hvis den rigtige XML-filen ved et uheld blev slettet. Vi syntes dette var en god ide fordi at man så nemmere ville kunne se hvordan XML-filen burde se ud, når man skal tilføje nye queues igen. Herunder er et eksempel på hvordan den autogenererede fil ville se ud hvis programmet ikke kunne finde XML-filen ved opstart.

```
<?xml version="1.0"?>

<ArrayOfModelXMLQueue xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <ModelXMLQueue>

  <queueName>Configure XML File</queueName>

  <maxQueueAge>5</maxQueueAge>

  <updateInterval>5</updateInterval>

 </ModelXMLQueue>
```

</ArrayOfModelXMLQueue>

Der er dog det problem med XMLSerializer at den skal bruge en public klasse med public variabler, hvis man skal bruge en data holding klasse, dette gør selvfølgelig denne klasse hensigtsmæssig at bruge, da man så vidt muligt skal undgå at bruge public variabler, på grund af sikkerhedshensyn. Det er selvfølge ikke nødvendigt i forhold til de krav regionen har stillet til programmet, men vi mente stadig at det ville være god kodeskik ikke at tilgå public variabler mere end nødvendigt.

Derfor valgt vi at konvertere den returnliste med ModelXMLQueue vi har fået fra XML-filen, over til en Dictionary der indeholder en string og ModelMQReader som er det objekt vi bruger i resten af programmet da dens variabler netop ikke er public.

```
foreach (ModelXMLQueue xmlq in returnlist)
{
    ModelMQReader mqr = new ModelMQReader(xmlq.GetQueueName, xmlq.GetMaxQueueAge, xmlq.GetUpdateInterval);
    queuedic.Add(mqr.GetQueueName , mqr);
}
```

Vi valgt at bruge en Dictionary da denne tilbyder type sikkerhed hvilket for eksempel list<>, og hashtable ikke gør. Type sikkerhed er ikke et stort problem da vi er et lille hold, men hvis en udefrakommende skal læse vores kode er det nemt at identificere hvad vi havde tænkt denne variable skal bruges til. Efter det returnerer vi vores dictionary, hvis der opstår fejl i løbet af læsningen returnerer vi en tom dictionary.

Vi havde også kigget på at bruge en almindelig XMLWriter/Reader, men mente at denne vil gøre det unødvendigt besværligt at vedligeholde. Med XMLSerializer metoden skal man blot tilføje en public variable mere i data holding klassen, samt dennes constructor hvis man altså ønsker at skrive en ny default fil, hvis ikke kan man nøjes med at tilføje en public variable.

Den nok største fordel XMLSerializer og grunden til vi valgt at bruge den er at dataene der kommer ud er i form af et objekt, og på den måde kan undgår man unødvendigt forvirring, da alt data ligger i et objekt, så man ikke behøver diverse arrays for at transportere dataene. dette ville være tilfældet ved brug af XMLReader.

ModelXMLQueue

ModelXMLQueue er som før nævnt vores data holding klasse, Den bliver udelukkende brugt a ModelXMLReader klassen.

```
[Serializable, XmlRoot("queues")]
public class ModelXMLQueue
{
    public string queueName;
    public double maxQueueAge;
    public double updateInterval;

    public ModelXMLQueue() { }
    public ModelXMLQueue(string queueName, double maxQueueAge, double updateInterval)
    {
        this.queueName = queueName;
        this.maxQueueAge = maxQueueAge;
        this.updateInterval = updateInterval;
    }
```

Det er som sådan en helt standard klasse med variabler og en constructor, det eneste man skal være opmærksom på er at variablerne skal være public for at XMLSerializer kan bruge dem. Den annotation lige over klassen fortæller XMLSerializeren at den skal sætte queue dataene ind under XML root elementet 'queues', vi forsøgte en del andre løsninger før denne men blev enige om at dette var den bedste måde at gøre det på. Så længe at denne klasse ikke bliver brugt af andre klasser en ModelXMLReader, bør det ikke være et problem at variablerne er public.

Løsning på problemet med Tråde

Vi havde i det foregående sprint haft en del problemer med at få en trådet version af programmet til at virke, men takket være et råd fra Morten, en af vores kontaktpersoner på regionen fandt vi en god måde at gøre det på. Vi fandt ud af at ved brugen af et såkaldt Updatepanel, contenttemplate, og en timer, alt sammen noget der er indbygget i ASP.NET. Kunne vi med dem skabe det event vi havde bruge for at køre vores program.

```
<head runat="server">
    <title></title>
    <link href="StyleSheet.css" rel="stylesheet" />
    <%--<meta http-equiv="refresh"  content="10"/>--%>
</head>

<body>
    <header><h3>WebSphere MQ overvågning</h3></header>
    <form id="form1" runat="server">



        <asp:ScriptManager ID="ScriptManager1" runat="server" EnablePageMethods="true">
        </asp:ScriptManager>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server" >
            <ContentTemplate >
                <asp:Timer ID="Timer1" runat="server" Interval="10000" OnTick="Timer1_Tick">
                </asp:Timer>
                <div id="div" runat="server" class ="box">

                </div>
            </ContentTemplate>
        </asp:UpdatePanel>

    </form>

</body>
</html>
```

En meget simpel løsning på et problem der havde plaget os i et stykke tid, man sætter meget simpelt asp:timer intervallet efter hvor ofte man vil have metoden i code behind skal køre. I dette tilfælde er den sat til 10 sekunder. Som man kan se ved det udkommenterede kode, havde vi før vi fandt på at bruge updatepanel, forsøgt os med en page refresh, men denne måde at gøre det på triggerede, ikke 'ispostback' på page_load hvilket førte til en del problemer med den måde programmet blev udført på, for eksempel lukkede den ikke trådene/forbindelserne til Websphere MQ ordentlig efter hvert refresh, hvilket førte til at der kunne være rigtig mange åbne forbindelser til de enkelte Websphere MQ test queues vi havde sat op. Dette problem blev dog løst med den nye tilgangsmåde. Det største problem med den nye løsning er den ikke skalere lige så godt som løsningen med trådene ville have gjort, det er selvfølgelig et problem hvis regionen mod forventning tilføje mange nye queues. Hvis dette sker, vil det være nødvendigt at forsøge med en trådet løsning igen.

Diskussion(Kim + Christopher)

Hvad er gået godt

I løbet af projektperioden har der været et godt samarbejde mellem projektgruppens medlemmer. Dette har helt klart været en fordel da vi som oftest har været på samme side med hensyn til opgaveløsninger og fordeling. Dermed ikke sagt at der ikke har været gnister, men de blev heldigvis aldrig til flammer. Det gav sig også til udtryk når vi var til møder ved regionen, at vi havde en fælles holdning for det meste. Når det enkelte gruppemedlem havde problemer, kunne den anden overtage og som oftest finde en løsning eller en måde at komme videre mod en løsning af problemet. På den måde komplimenterede vi hinanden godt, i arbejdsforløbet. For et eksempel kunne den ene koncentrere sig om at få ModelMQConnect til at virke mens den anden tog sig af XML-delen. Denne dynamik har fungeret rigtig godt, og da vi endelig kom i gang med at sprint, kom der hurtigt noget ud af det.

Samarbejdet med vores kontaktpersoner fra regionen, har været meget svingende men har i sidste ende givet et godt udkast. Efter vi fik muligheden for at sidde inde på regionen, forsvandt de fleste af de kommunikationsproblemer der havde plaget projektet i starten.

Det gav os en meget bedre mulighed for at få kontakt til Morten og Thorkild når vi fysisk var til stede på arbejdspladsen.

Vores møde med den eksterne konsulent Johnni Blaaberg Jensen, gav os lige netop den indsigt i hvad vi havde gjort forkert, i vores forsøg på at oprette forbindelsen til Websphere MQ. Møderne med Johnni var meget produktive, og det var der vi fik det gennembrud, der lod os komme videre med projektet.

Vores Sprint gik for det meste rigtig godt, vi fik løst de problemer vi havde og gik for eksempel fra et udynamisk til en dynamisk løsning. Vi startede på den måde med statiske variabler til projektet og via vores iterationer fik vi så gjort programmet dynamisk, på den måde havde vi i vores sprints altid noget funktionelt at arbejde med.

Hvad er gået skidt

Vi skulle fra starten have været meget bedre til at opsøge vores kontaktpersoner på regionen, da de ikke altid var lige gode til at svare indenfor et acceptabelt tidsrum. Det var

den overvejende grund til at projektet blev forsinket i starten, når kommunikationen manglede. Både regionen og vi selv havde en fornemmelse af at indledende problemstilling med at få Websphere MQ til at fungere og kommunikere med Java/Eclipse.

Vi tog os for meget tid til selv prøve at finde en løsning på dette problem, og set i bakspejlet skulle vi have insisteret på at få hjælp meget tidligere i forløbet. Vores tøven og manglende opsøgning på hjælp fra regionen, kostede os desværre meget dyrebart tid, som kunne være brugt på noget mere konstruktivt. Det skal dog siges, at på trods af forhindringerne i starten af projektet nåede vi stadig at levere noget brugbart til regionen i sidste ende.

Det lykkedes os ikke at udføre projektet i Java/Eclipse, som regionen ellers havde ønsket, dette var en konsekvens af vores manglende viden om netop Java, men også som før nævnt den tid vi mistede i starten af projektet.

Vi kom alt for sent i gang med at sprinte i forløbet, det udsprang som før nævnt af, at vi simpelthen ikke var gode nok til at opsøge regionen, når vi stødte på forhindringer, som vi ikke selv hurtigt kunne overkomme. Dertil skal det også siges at regionen ikke altid var for hurtig, når vi så endelige havde spurgt om hjælp.

Hvad kunne vi have gjort bedre

Vi skulle have været meget mere opsøgende og insisterende på hjælp fra regionen, når vi gik i stå. Dette kunne have været allerede en uge inde i projektet, da vi synes vi havde undersøgt Websphere MQ tilstrækkeligt. Allersenest burde vi have taget kontakt til dem omkring to uger inde i projektet, hvor vi kunne se at vi ikke på egen hånd, ville kunne få oprettet forbindelse mellem Java og Websphere MQ. Her burde vi have gjort det klart over for regionen at selv med alle de guides vi havde fundet, kunne vi ikke se en løsning komme i nær fremtid på problemet. Problemet med at oprette forbindelsen fra Java til Websphere MQ, var at selv om der var mange guides var de ofte til forkerte versioner af Websphere MQ/Java og dette betød at de guides vi fandt, ikke ville virke til den version vi havde tilgængelig. Et andet problem med forbindelsen mellem Websphere MQ og Java, fandt vi senere ud af var på grund af to forskellige måde at opnå forbindelse med henholdsvis Binding og Client connection. Hvor vi havde gået ud fra at man skulle bruge en client connection. Vi fandt senere ud af at dette var umuligt, da man skal bruge Windows professional for at få netop denne måde til at fungere. Windows Professional er nødvendigt da man skal sætte

brugergrupper manuelt, hvilket man ikke kan med Windows Home Premium. Det var her den eksterne konsulent hjalp.

Det var hovedårsagen til at vi skulle have været hurtigere til at bede regionen om hjælp, da det formentlig ville have taget os meget længe at finde denne løsning selv.

Vi var ikke så gode som vi nok selv havde ønsket til at overholde deadlines, som vi havde satte os selv. Det resulterede i at vi specielt i starten af projektet manglede ting som vi egentlig mente at vi havde lavet. Det sagt havde vi også nogle problemer hen mod slutningen, da vi nok begge mente at vi var ved at være færdige og derfor slappede lidt for meget af. Vi blev enige om at for at rette dette, ville vi skrive det i vores dagbog, så vi bedre kunne huske hvad der skulle laves til hvornår. Det hjalp dog ikke altid, når et af punkterne vi glemte, var netop at skrive dagbog.

Et problem vi fandt ud af tidligt, var at medmindre vi angav mødetider i rette tid, var der god chance for at disse ikke blev overholdt, glemt eller misforstået. Dette gav sig eksempelvis til kende ved at der en dag havde været snak om at møde klokken 10 på erhvervsakademiet, mens den anden have troet at næste møde ville foregå på Magion.

Efter dette var sket et par gange, aftalte vi at vi for eftertiden ville sikre at mødetiden og stedet til næste møde, var kommunikeret videre ordentligt. Det hjalp også her, at vi efterfølgende valgte at skrive mødetider i vores dagbog.

Vi skulle have bedt om at få en plads inde på regionen meget tidligere i projektperioden, da dette ville have afhjulpet mange af de problemer vi stødte på tidligt i projektet, blandt andet havde kommunikations problemerne ikke været nær så problematiske som de viste sig at være.

Hvad kunne de have gjort bedre

Efter vores første samtale med konsulenten blev det os klart at denne heller ikke havde kompetencer indenfor opsætning af Websphere MQ med Java. Dette skulle dog ikke forhindre ham i at prøve at hjælpe os med at overkomme denne barriere. Det lykkedes ham dog at hjælpe os med opsætningen af forbindelse til Websphere MQ via C# udviklingsmiljø. Efter at vi forklarede dette til regionen og forklarede dem at det ikke ville være muligt at sætte det op som et Java projekt, på vores egne eller deres computere, inden for en rimelig

tid, blev vi enige om at udviklingen kunne fortsætte som et C# projekt kombineret med ASP.NET. Dette problem kunne have været undgået, hvis regionen havde bedt konsulentfirmaet specifikt om en Java Programmør.

Regionen skulle have afsat mere tid til os, da det stod os klart at den første kontaktperson vi havde fået tildelt, hverken havde nok tid eller kendskab til at hjælpe os tilstrækkeligt i gang med projektet. Det gav specielt udslag i at han ikke returnerede mails og at de guides han fandt til os ofte var utilstrækkelige.

En ting er tid, en anden er også plads. For kommunikationen med regionen led af meget envejskommunikation, som ikke blev besvaret eller ikke blev besvaret i tide. Derfor ville direkte dialog med dem have hjulpet en del. For at få denne direkte dialog med regionen, ville det være nødvendigt at vi sad på regionens kontorer og ikke andet steds. Dette var dog ikke muligt da der i et hvis omfang manglede siddepladser på regionen, hvilket gjorde det umuligt for vores kontaktpersoner at finde steder hvor vi kunne sidde. Vi burde derfor inden projektets start have snakket med regionen om, hvorvidt der ville være siddepladser til os derinde og såfremt de ikke kunne bekræfte at der alle arbejdsdage ville være plads, ikke have valgt at starte på dette projekt.

Afsluttende konklusion(Kim + Christopher)

Vi valgte i starten af projektet at vi gerne ville køre SCRUM som vores udviklingsmetode og da den analyse vi foretog via Todd Little's uncertainty and complexity metode, viste at SCRUM ville fungere godt til vores projekt faldt valget meget naturligt derpå.

Vi synes at tomands-SCRUM har fungeret rigtig godt til vores projekt, og vi har på ingen måde fortrudt dette valg.

Tiden brugt på at forstå Websphere MQ tog længere end vi oprindeligt havde tiltænkt, dette der var forvirring over de mange forskellige versioner der findes af Websphere MQ, samt manual til disse. De guides vi kunne finde var ofte tiltænkt en anden version end den vi skulle bruge. Der var også problemer med selve forbindelsen mellem programmet og Websphere MQ, da der findes to måder at forbinde til Websphere MQ. En af de måder man kan forbinde til Websphere MQ kræver f.eks. Windows Professional editionen, da man skal ind og sætte brugergrupper, hvilket man ikke kan med Windows Home Edition. Dette var

dog ikke nævnt nogen steder som vi kunne finde i vores søgninger efter dokumentation: Det var først efter vores samtale med den eksterne konsulent Johnni Blaaberg Jensen at vi fandt den rigtige måde at oprette forbindelsen.

Den største udfordring vi havde med projektet var uden tvivl den forbindelse vi skulle have oprettet til Websphere MQ. Det skyldes hovedsageligt vores og regionens manglende viden om produktet, de ansatte på regionen som oprindeligt havde stået for opsætningen af deres message queues, ikke var ansat længere. Regionen er derfor afhængig af eksterne konsulenter, og så derfor helst at vi selv kunne indhente den nødvendige viden omkring Websphere MQ.

I sidste ende lærte vi meget omkring opsætningen, kørsel og tilgangen programmatisk til Websphere MQ, dette er en færdighed vi kan tage med os ud i erhvervslivet.

Vi havde i gruppen, en del erfaring med at arbejde i ASP.NET og det gjorde bestemt opgaven nemmere for os, da vi skiftede til et C# udviklingsmiljø. Vores eneste forhindring var dermed, autoopdateringsmekanikken som vi roligt kan sige voldte os en del problemer. Dermed sagt vi ikke overkom problematikken omkring denne, det gav os nemlig muligheden for at udforske nye områder af ASP.NET. Det var netop ved denne udforskning, at vi fandt løsningen med updatepanel og asp:timer kontrolmekanikken.

Skiftet til C# skete efter det viste sig at det tidsmæssigt ikke længere var forsvarligt for os at fortsætte med Java. Det gav os god indsigt i logikken i systemudviklingsprocesser og specielt cost/time/scope management, at vi var nødt til at ændre programmeringssproget og skære ned på funktionaliteten tiltænkt programmet, da vi hverken kunne ændre på cost eller time aspekterne af denne treenighed.

Det at projektet skulle benytte varierende opdateringsintervaller, gav os også god mulighed for at undersøge forskellige måder, hvorpå man kan benytte tråde til at holde styr på de objekter, som vi ønskede vist i vores view. Både den almindelige form af tråde, fra System.Threading og den udvidede måde i System.Threading.Tasks har lært os en del omkring disse, som vi ikke havde kendskab til før projektet.

At det endelige valg til at styre vores objekter så i sidste ende ikke faldt på en trådet løsning, havde mere at gøre med tid og mekanik end manglende mulige løsninger, hvor de kunne være brugt.

Vi har lært hvor vigtigt det er at have åben og klar kommunikation, med samarbejdspartnere i et projekt. At dialog tager 2 sider, og hvis den ene part ikke svarer er det ikke en dialog. Meget tid blev ikke brugt optimalt i starten af projektperioden, på grund af manglende kommunikation fra begge sider. Ofte var problemet travlhed blandt vores kontaktpersoner, andre gange var vi ikke gode nok til kommunikere at vi havde svært ved at komme videre. Vi lærte at hvis der er mulighed for det, er det altid bedre at være fysisk tilstede på arbejdspladsen og opsøge en dialog mere direkte. Som også nævnt i diskussionsafsnittet havde vi også et problem med arbejdsmiljøet, når vi ikke kunne sidde på regionen, de eneste muligheder vi havde, var biblioteker, skolen eller arbejde hjemmefra, hvoraf ingen af dem er optimale hvis man ønsker en dialog med arbejdsgiver.

Vores erfaringer med Java/Eclipse som udviklingssprog og miljø, gik som sådan godt da der ikke er de store forskelle på Java og C#, problemet for os var at vi ofte ikke kunne finde de korrekte .jar filer der var nødvendige for at programmere en forbindelse til Websphere MQ. Derimod var problemet oftest at vi skulle bekendtgøre os selv med Eclipse som udviklingsmiljø, hvilket vi havde nogle problemer med da Eclipse er væsentlig forskelligt i forhold til Visual Studio.

Vi synes på trods af de opstarts vanskeligheder vi havde, at vi har fået rigtigt meget ud af denne projektperiode. Vi fik lavet et selvstående program, der kan benyttes til at overvåge message queues, der er både dynamisk og struktureret på en måde, hvorved vi synes det er let at gå til både for folk der vil benytte programmet, men også for andre programmører, som i fremtiden kunne ønske sig at bruge det.

Derudover har vi selvfølgelig lært en del omkring projektstyring og hvordan både andre og vi selv kan blive nødt til at begrænse funktionalitet i programmer på grund af de stopklodser der sættes, være de i form at tid, pris eller omfang.

Sidst har vi nok også lært en del omkring nødvendigheden af kommunikation i et projektforløb, hvor man med sin proces ønsker at få så meget ud af det som muligt.

Kildehenvisninger

http://www-01.ibm.com/support/knowledgecenter/SSFKSJ_7.5.0/com.ibm.mq.ref.dev.doc/q101870_.htm

http://alistair.cockburn.us/Methodology+per+project

Todd Little Complexity Uncertainty – fra SUM pensum 4. Semester