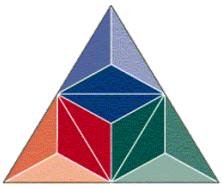
DB FACTORY 2

HIBU STUDENTPROSJEKT 2012
CYTRAX DB FACTORY



HØGSKOLEN
i Buskerud

cytrax

JARLE DIDRIKSEN

ERIK ALEXANDER EDLAND

TOR-CHRISTIAN H. ERIKSEN

JØRGEN GRØNDAL

VEGARD KAASIN

SIMEN SKOGLY RUSSNES

DOKUMENTER

1. Dokumentoversikt
2. Prosjektsammendrag
3. Prosjektplan
4. Teknologidokumenter
5. Analysedokument
6. Designdokument
7. Implementasjonsdokument
8. SCRUM – Metodikk og regler
9. Backlogger
10. Kvalitetssikring
11. Test-caser
12. Aktivitetsoversikt

REVISJONSHISTORIKK			
Versjon	Dato	Endringer	Tildelt
1	25.5.2012	Opprettet	SR

1 Introduksjon

Dokumentoversikten er en oversikt over hvilke dokumenter som inngår i den innbundne versjonen av prosjektet.

1.1 Forfatter(e)

Simen S. Russnes (SR)

2 Dokumentoversikt

2.1 Dokumentoversikt

Dokumentoversikten er en liste og beskrivelse av dokumentene som er levert.

2.2 Prosjektsammendrag

Prosjektsammendrag er en oppsummering av prosjektet. Dette er blant annet en beskrivelse av arbeidsprosessen, av prosjektet, utfordringer og personlige evalueringer.

2.3 Prosjektplan

Prosjektplanen beskriver planleggingen for prosjektet. Her inngår hva opgaven går ut på, hva som skal leveres og til hvilket tidspunkt, mål og begrensninger, organisering og mål gjennom prosjektet.

2.4 Teknologidokumenter

En rekke dokumenter som beskriver teknologier vi har vurdert å benytte oss av.

2.5 Analysedokument

Analysedokumentet viser en analyse av hvordan vi skal løse oppgaven. Dette går ut på hvilke deler som inngår og hva som inngår under de forskjellige delene.

2.6 DESIGNDOKUMENT

Designdokumentet er neste steg mot implementasjon som bygger på analysedokumentet. Designet brukes for å gjøre utviklingen av applikasjonen enklere og for å unngå problemer ved å oppdage de på forkant.

2.7 IMPLEMENTASJONSDOKUMENT

Implementasjonsdokumentet er en oversikt over hvem som har utviklet hva i applikasjonen.

2.8 SCRUM – METODIKK OG REGLER

Scrumdokumentet er en beskrivelse av vår prosjektmodell, Scum. Her gis en introduksjon til Scrum. Det utredes om ulemper og fordeler og definisjoner brukt i modellen, og hvordan vi har valgt å bruke Scrum.

2.9 BACKLOGGER

Backloggene består av en product backlog og en sluttrapport for hver sprint.

Product backlog inneholder foreslåtte oppgaver for videre utvikling.
Sluttrapportene består av et burndown chart, periode, godkjente og ikke godkjente oppgaver.

2.10 KVALITETSSIKRING

Testdokumentet er en beskrivelse av vår kvalitetssikring av produktet. Dette er delt inn i to deler, dokumentgjennomgang og testing. Testingen er i forbindelse med kode, og dokumentgjennomgang i forbindelse med dokumentene vi produserer i løpet av prosjektet.

2.11 TEST-CASER

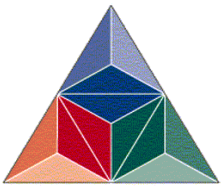
Test-caser er en oversikt over testene vi har gjennomført i prosjektet og resultatet på disse.

2.12 AKTIVITETSOVERSIKT

Aktivitetsoversikten er en oversikt over timefordeling brukt på forskjellige aktiviteter i prosjektet.

PROSJEKTSAMMENDRAG

HIBU STUDENTPROSJEKT 2012
CYTRAXDB FACTORY



HØGSKOLEN
i Buskerud

cytrax

/REVISJONSHISTORIKK		
Versjon	Dato	Endringer
1.0	24.5.2012	Første utgave

INNHOLDSFORTEGNELSE

1 INTRODUKSJON

Dette dokumentet skal beskrive gruppens oppfatninger og tanker om hvordan gjennomføringen av prosjektet har vært. Vi vil redegjøre for hvordan oppgaven utviklet seg fra visjon til produkt, hvordan vi har fått erfaring og utnyttet SCRUM sine sterke sider i prosjektsammenheng og hvilke utfordringer som har oppstått under prosessen. I tillegg vil alle gruppemedlemmene skrive refleksjoner om egen deltakelse i prosjektet.

1.1 FORFATTER(E)

Simen S. Russnes (SR)

Jørgen Grøndal (JG)

2 PROJECT REVIEW

2.1 PROSJEKTET GENERELT

Gruppen hadde på forhånd bestemt seg for å jobbe sammen som gruppe dersom det var mulig, selv om to av gruppemedlemmene hadde høstsemesteret i Australia. Med det kunne vi samtidig bevise at man i en tidlig planleggingsfase ikke var avhengig av å være på samme lokasjon, men at fjernkommunikasjon også fungerer bra i prosjektsammenhenger.

Vi ønsket å ha en oppgave der vi kunne designe produktet og ha et tett samarbeid med oppdragsgiver.En prosjektoppgave fra Cytrax AS virket spennende, og vi tok kontakt med Cytrax. Responsen vi fikk fra dem gjorde oss komfortable med å velge oppgaven fra dem. De svarte raskt og virket veldig interessert i gjennomføringen av prosjektet.

Oppgaven fra Cytrax gikk ut på at vi skulle utvikle en applikasjon for generering av kildekode til manipulering av eksisterende databaser. Cytrax hadde tidligere utviklet en applikasjon som gjorde akkurat dette. Den ble videreutviklet over lang tid og de ønsket å lage den fra bunnen av igjen for å få en mer fleksibel applikasjon. Den opprinnelige applikasjonen het DBFactory, vi valgte derfor å kalle applikasjonen vår for DBFactory 2.

Applikasjonen skulle kunne behandle flere typer databaser og generere kode for forskjellige språk, så det var derfor ønskelig at generering og innlesing ble utført via plugins. Man kan da enkelt utvikle nye moduler og kompilere disse til .dll filer, som applikasjonen gjenkjenner og kan ta i bruk.

2.2 BRUK AV SCRUM SOM PROSJEKTMODELL

Gjennom prosjektet vårt har vi brukt Scrum som prosjektmodell. Grunnen til at vi først tenkte på Scrum var at noen fra gruppa hadde vært borti det før. Vi syntes også at det var en verdifull metode å sette seg inn i og forstå da den har mange positive sider ved utvikling av prosjekter.

Siden vi hadde lite erfaring med bruk av Scrum som prosjektmodell var det også mye å sette seg inn i. Mye av tiden før første presentasjon ble brukt til å bli kjent med prosjektmodellen.

Da vi startet å bruke Scrum hadde vi skjønt at en av utfordringene ved å bruke Scrum er det å estimere timeantall på forskjellige oppgaver, spesielt når man ikke har erfaring med det fra før. Vi tok det til betraktning og prøvde å fokusere på at det var et problem som kunne oppstå. Det resulterte i at vi enkelt klarte å håndtere tilfeller der vi estimerte feil.

Vi er veldig fornøyde med hvor effektivt det har vært å bruke Scrum. Det å lage stories og så dele opp i mindre tasker har gjort det veldig forutsigbart for oss med tanke på å velge og utføre oppgaver gjennom prosjektet.

Underveis i utviklingen av prosjektet har vi hatt daglige scrum-møter på maks 15 minutter hvor vi har gitt en statusoppdatering fra hver av deltakerne. Dette har vært med på å styrke samarbeidet i gruppa da alle til enhver tid vet hva de andre holder på med og enkelt tillater alle å ha en oversikt over fremdriften i prosessen. Det har også vært lett å komme over utfordringer fra dag til dag da dette kommer frem i scrum-møtene, og en løsning har vært lett å passe inn.

Vi har også daglig kunnet se fremdriften på burndown-chart for sprinter etter hvert som vi har brent timer.

Hver gang en sprint ble fullført utførte vi et Sprint review møte der vi gikk gjennom alle stories fra utført sprint. Vi kontrollerte hver story hvor vi enten godkjente, eller tok den med i neste sprint. Audun var med på de fleste møtene men hadde desverre ikke anledning i sprint nr. 4 og sprint nr. 7. Da oppdragsgiver har vært så tett med og vært med på sprint review møter har det vært enkelt for oss å tilpasse neste sprint ut fra tilbakemeldingene vi fikk.

Selv om Scrum har vært utfordrende å sette seg inn i har det lønnet seg gjennom prosjektet. I tillegg til at det har gjort utviklingen lettere har vi fått erfaring i åbruke det som prosjektmodell.

2.3 TEKNOLOGIER

2.3.1 Qt

Sammen med Cytrax ble det bestemt at applikasjonen skulle utvikles i C++ med Qt. Qt er et rammeverk for utvikling i C++ som har latt oss fokusere på høy-nivå-programmering. I tillegg til basisfunksjonalitet som f.eks QString, har vi tatt i bruk modellen QDomDocument for å håndtere data fra database, noe som har gjort utviklingen mye mer forutsigbart. Qt er også i en stor grad plattformuavhengig, noe som gjør at applikasjonen vi har utviklet nå kan kompileres til å kjøre på både Windows, Mac og Linux.

Qt har i tillegg et IDE (Qt Creator) vi har tatt i bruk under utviklingen. IDEet i seg selv har fungert veldig bra og har vært intuitivt, i tillegg til at det har en innebygget GUI-designer vi har hatt god nytte av.

2.3.2 Doxygen

For enkelt å kunne dokumentere implementasjonen av applikasjonen har vi tatt i bruk doxygen. Dette er en teknologi som lar oss å generere en API-spesifikasjon i html-format. Det blir da veldig enkelt å navigere gjennom alle de forskjellige klassene og metodene for å se hva hver del gjør.

For å automatisk generere API-spesifikasjonen måtte vi kun skrive kommentarer på en spesiell måte i alle header-filene i prosjektet. Doxygen henter deretter ut og genererer html-dokumentene. Dette gir en oversikt over alle parametere og returverdier, samt en kommentar for hver metode i applikasjonen.

2.3.3 Git/Github

For å organisere utviklingen vår var vi nødt til å bruke et system for versjonskontrollering. Jørgen hadde erfaring med bruk av Git og Github tidligere og etter å ha introdusert gruppen for Git, besluttet vi at vi skulle bruke Git og Github for versjonskontroll.

Når vi begynte med implementasjonen fant vi raskt ut at Git fungerte veldig godt. Den har god innebygget funksjonalitet for å lime sammen filer som er utviklet forskjellig, og når det er uklart hvilken utgave av en metode den skal bruke lager den en konflikt som brukeren enkelt kan redegjøre. Git har gjort det veldig enkelt for oss å samarbeide og drive med parallell utvikling. Dersom en person har hatt behov for kode som en annen person har ansvar for kan dette enkelt utvikles og lastes opp. Deretter kan man lime inn disse endringene isolert fra resten av utviklerne. Dette passer utmerket sammen med en smidig utviklingsmetode som Scrum.

2.3.4 Kunagi

Siden vi har brukt Scrum som prosjektmodell synes vi det var en god ide å bruke et verktøy for å hjelpe oss gjennom prosessen. Et slikt verktøy gjør det enkelt å dele opp og

velge arbeidsoppgaver, samtidig som man automatisk kan se fremgangen i prosjektet fortløpende.

Jørgen oppdaget tidlig Kunagi, som viste seg å være akkurat det vi trengte. Grensesnittet er i nettleseren og gjorde det enkelt å jobbe sammen fra flere pc-er, da vi kunne legge applikasjonen på en server og få tilgang fra hvor som helst.

Når man har delt opp oppgaver for en sprint velger man ganske enkelt oppgaver fra en liste og brenner timer etter hvert som man har jobbet på de. Man vil da få en automatisk oppdatert graf som viser hvordan vi ligger an i forhold til hvor lang tid vi har igjen.

2.4 UTFORDRINGER

Da vi startet prosjektet var vi klare over at to av gruppemedlemmene (Vegard og Simen) skulle tilbringe det første semesteret av hovedprosjektet på et utvekslingssemester i Australia. Vi så for oss at dette ville bli en utfordring fordi vi utelukkende måtte kommunisere over Skype, og ikke fikk jobbe så tett sammen som man vanligvis ville gjort.

I planleggingsfasen hvor vi hadde forskjellige lokasjoner fikk vi likevel gjennomført planlagte oppgaver på en god måte. Vi hadde møter via Skype og fikk fordelt arbeidsoppgaver.

Da Vegard og Simen kom tilbake like før jul 2011 var det veldig greit å fortsette samarbeidet og utviklingsprosessen videre med hele gruppen samlet. Vi synes det var gøy at det gikk så bra som det gikk, da vi i forkant fikk advarsler om at det kunne bli utfordrende.

I tillegg ble vi advart om at Scrum kunne være vanskelig å sette seg inn i dersom man ikke hadde rørt det før. Vi merket fort at det var mye teori å sette seg inn i for å kunne bruke metodikken på best mulig måte, men klarte å løse det uten store problemer.
Vi synes at en av de største utfordringene var å estimere timeantall på forskjellige arbeidsoppgaver. Etter hvert som vi fikk mer erfaring med estimering så vi også at estimatene stemte bedre. Siden vi forutså at det kunne bli vanskelig møtte vi ingen store overraskelser og klarte å håndtere bruken av Scrum på en bra måte, som gjorde utviklingsprosessen mer effektiv.

2.5 OPPNÅDDE MÅL

Vi føler oss fornøyde med produktet og har nådd de kravene som var satt da vi planla applikasjonen. Vi fokuserte mye på å designe applikasjonen i forkant av kodeskrivning slik at det ble lett å implementere og videreutvikle i fremtiden, noe vi har fått til og er fornøyde med.

På den måten har vi tatt i bruk den kunnskapen vi har opparbeidet oss gjennom årene ved HiBu, og laget noe som er til nytte for vår oppdragsgiver.

Vi har bevist at det ikke nødvendigvis er et problem å samarbeide over kontinenter ved hjelp av Skype over lengre tid. Selv om to av gruppemedlemmene var på utveksling i utlandet under første semester av prosjektet har det latt seg gjøre å samarbeide på en effektiv måte.

I tillegg har vi fått erfaring med, og dratt nytte av å bruke Scrum som prosjektmodell i hovedprosjektet. Dette har gitt oss nyttig kompetanse og har hjulpet oss med utviklingen.

2.6 PERSONLIGE EVALUERINGER

Som en del av oppsummeringen skriver hver av oss en egen personlig evaluering av prosjektprosessen.

2.6.1 Tor-Christian H. Eriksen

Som prosjektleder og pådriver for å velge Scrum som prosjektmodell har det vært spennende å jobbe med en modell som er ulik fra andre modeller normalt brukt ved HiBu.
Det har helt klart vært en fordel at prosjektet var et rent softwareprosjekt. Selv om vi har vært seks personer fordelt på prosjektet har vi klart å fordele arbeidsoppgaver gjennom sprintene slik at alle har hatt sitt å bidra med. Jeg er godt fornøyd med hvordan Scrum har fungert.

Prosjektet har gitt meg mange gode erfaringer, spesielt i forbindelse med planlegging og strukturering, og jeg har fått brukt mye av det jeg har lært gjennom årene på HiBu. Mitt inntrykk er at vi har alle vært flinke til å hjelpe hverandre og aktivt søk hjelp hos hverandre ved behov.

Det har også vært nødvendig å snakke ut om problemer som har oppstått i gruppa. Som prosjektleder har jeg forsøkt å være en person alle kan snakke med om eventuelle utfordringer og komplikasjoner. Det har vært noen problemer, og de har vi klart å løse på en ryddig og ansvarlig måte.

For oss har design og implementering gått veldig hånd-i-hånd. Under implementering gikk vi flere ganger tilbake til å redesigne deler av en løsning, eller gjorde mindre endringer i designet basert på oppdagelser og erfaringer gjort under implementering. Dette mener jeg er en viktig erfaring å ta med seg videre.

Jeg er godt fornøyd med produktet vi har utviklet for Cytrax, og det er godt dokumentert slik at videre utvikling av applikasjonen kan skje så smertefritt som mulig.

2.6.2 Jørgen Grøndal

Gjennomføringen av dette prosjektet har vært veldig givende. Jeg har fått veldig mye nyttig erfaring når det kommer til å samarbeide i gruppe over lengre tid. Selve oppgaven har vært spennende å arbeide med. Prosessen fra konsept / visjon til ferdig produkt har

vært fylt med mange utfordringer som jeg synes gruppen sammen har taklet på en god måte.

Rollene jeg har hatt i prosjektet er dokumentansvar og versjonskontroll. Dokumentstandard ble tidlig etablert og dokumenter har derfor fått en standardisert form. Siden vi har benyttet scrum som prosjektmodell har rollene vært av mindre betydning og vi har alle bidratt på alle områder.

Versjonskontroll ble tidlig bestemt til å være Git via Github.com. Jeg har hatt erfaring med bruk av git tidligere og synes det er et godt verktøy for å drive med utvikling i et team. Når vi begynte med implementasjon har jeg vært den personen gruppen har henvendt seg til dersom det var spørsmål rundt bruk av Git.

Utover dette har jeg arbeidet mye med design og analyse og har bidratt mye med kritiske spørsmål til tekniske løsninger. De delene av design og analyse jeg har hatt hovedansvar for er: GUI, sammenligning av dokumentmodeller, overordnet arkitektur, kontroller og generator API. I forbindelse med implementasjon har jeg hovedsakelig arbeidet med generator plugin for VB.net og sammenligning (Inkl. GUI) og visning av data i tabeller (GUI).

Jeg synes vi har samarbeidet godt i gruppen og utfordringer vi har støtt på har blitt løst. Det har vært veldig morsomt å arbeide i hovedprosjekt og jeg har fått brukt mye av det jeg har lært i løpet av disse tre årene. Dette er alt fra tekniske ferdigheter som programmering og design, til mer administrative i form av dokumentskriving, samarbeid og presentasjoner.

Jeg er veldig fornøyd med produktet vi har klart å levere, både dokumentasjon og prosjektgjennomføring ovenfor skolen og utviklet produkt ovenfor Cytrax.

2.6.3 Jarle Didriksen

Prosjektet har vært veldig spennende å gjennomføre, og har bydd på utfordringer og mye moro. Oppgaven vår har vært spennende å jobbe med, og jeg er stolt av å ha vært med i utviklingen av programmet vårt, helt fra den tidligste planleggingen fram til siste kodelinje ble skrevet. Vi har hatt en arbeidsgiver som har vært engasjert, hjelpsom og responsiv. Noe som har vært veldig til nytte gjennom prosjektet.

Ansvarsområdet mitt gjennom prosjektet har vært testing og logistikk, og jeg har i den sammenheng lest mye om det å teste i softwaresammenhenger. Dette er et område jeg ikke hadde mye erfaring i på forhånd, og det var derfor interessant å få mer innsikt i det. Utover dette har jeg i analysefasen holdt mest på med bruken av plugins for å lese inn databaser, noe jeg arbeidet videre med i designfasen. Videre i designfasen jobbet jeg med filbehandling og kontrollerne i systemet, noe jeg også arbeidet på i implementasjonsfasen. Dette innebærer åpning og lagring av filer og håndtering av andre komponenter i systemet.

Gjennom dette året har jeg tilegnet meg masse erfaring i prosjektarbeid, og hvordan det er å jobbe i et større prosjekt over lengre tid. Jeg har fått brukt kunnskapen jeg har lært i

fagene på skolen, og fått mer erfaring i å bruke de, noe som har vært veldig gøy. Jeg har også fått mye ny kunnskap med på kjøpet, for eksempel programmering med Qt og testing.

Som gruppe valgte vi jo å bruke scrum, noe som ikke har blitt brukt så ofte på skolen før. Vi måtte derfor sette oss inn i mye av det på egen hånd, og det syns jeg vi klarte bra. Det har vært spennende å jobbe på denne måten, og jeg synes at scrum har gitt oss mye tilbake. Jeg er fornøyd med samarbeidet vi har hatt på gruppa vår, og alle har stått på og hjulpet hverandre. Jeg sitter igjen med et inntrykk om at vi har vært en sammensveiset gjeng hele prosjektet, og at kjemien har vært god.

Jeg synes det har vært en utfordring med at alle virkelig må vise initiativ og ta ansvar, noe som er viktig når man jobber med scrum. Men jeg syns at vi alle har vært dedikerte til prosjektet og at vi har håndtert dette bra.

Alt i alt syns jeg prosjektet har vært kjempebra, og jeg er fornøyd med hvordan vi har gjennomført det. Jeg syns også at vi har hatt et bra samarbeid, og jeg er stolt av produktet vi har laget sammen.

2.6.4 Alexander Edland

Dette har vært en spennende mulighet til å prøve seg i et større samarbeidsmiljø, og selv med utfordringer fra første stund så har vi taklet det meste. Gruppen har fungert veldig bra sammen, og det er nok en del av grunnen til at vi ikke har hatt flere problemer.

Jeg har vært teknologiansvarlig, og har ofte bistått med råd og bistand med tekniske problemstillinger underveis. Utover det har jeg designet og implementert factories og grunnmuren for dokumentsammenlikning, samt noe prototyping for å kartlegge ytelse, spesielt i overgangen til *QDomDokument*. Som teknologiansvarlig har jeg også utført en del administrativt som produksjon og vedlikehold av nettside og andre verktøy (scrum, dokumentvalidering).

Valget av SCRUM som prosjektmodell har vist seg å være smart, da vi flere ganger har gjort mindre endringer på «ferdigstilte» løsninger. Overgangen fra klassehierarki var en spennende vri som ikke var fullt så liten, men som også har betydd mye for hvor prosjektet er idag. En annen fordel med scrum er hvordan rollene ikke har vært like fremtredende, slik at oppgavefordeling har vært problemfri av natur.Samtidig som alle bisto hverandre på teamet ved å fordele kunnskap og ferdigheter, så har det vært spennende å være sin egen sjef (til en viss grad).

Prosjektet har for meg vært muligheten til å leke med mange nye teknologier, og både Qt og GitHub vil garantert komme til nytte senere. Videre har det vært en gylden sjansen til å sette teori og kunnskap igjennom skolegangen på prøve. Mye har falt meg naturlig igjennom det siste halvåret, noe som neppe hadde vært tilfelle uten all forkunnskapen. Det har vært spennende å samarbeide om et softwareprosjekt og oppleve de spesielle problemstillingene man kommer opp i derav, og gøy å se at vi fikk løst alt sammen raskt og radig takket være prosjektmodellen og gode verktøy (Git), og ikke minst en velfungerende gruppe.

Totalt sett har vi gjort en knall jobb alle som én, og sluttproduktet har blitt både bra utformet og dokumentert for videre forbedring.

2.6.5 Vegard Kaasin

Mine hovedansvarsområder i prosjektet har vært Scrum og design.Siden jeg har hatt noe erfaring med Scrum fra før var det naturlig for meg å ta rollen som Scrum-master. Det var en utfordring for alle å sette seg inn i Scrum som prosjektmodell, men etter som vi fikk prøvd oss på å bruke Scrum har det gått veldig bra. Det jeg synes har fungert best med Scrum er den løpende dialogen og møtene vi har hatt med Cytrax underveis i prosjektet.

Design og analyse-fasen er noe jeg synes vi har fått til veldig bra. Vi brukte god tid på å designe applikasjonen, noe som vi fikk igjen for når vi begynte å implementere.

Videre i prosjektet har jeg hatt mye ansvar for GUI. Qt og Qt Creator(IDE) var nytt for meg, men prototyping underveis i designfasen hjalp veldig når vi skulle begynne å implementere. Jeg er utrolig fornøyd med valget om å bruke Qt og jeg synes det har fungert utrolig bra.

En annen ting jeg synes har vært positivt er hvordan gruppa har taklet at ikke alle var samlet når prosjektet startet. Jeg var i Australia når vi startet prosjektet, men gruppa var veldig flinke på å integrere meg i arbeidet med prosjektet og det var ikke noen vanskelig overgang å komme tilbake til Norge.

Jeg synes gruppa har jobbet bra sammen, og jeg er veldig fornøyd med produktet vi har klart å levere både til Cytrax og til Hibu.

2.6.6 Simen S. Russnes

Det har vært en veldig positiv opplevelse å ta del i dette prosjektet. Jeg har lært mye om det å jobbe sammen i en gruppe på et så omfattende prosjekt. I tillegg har det å kunne ta i bruk den kunnskapen jeg har bygget opp i løpet av studietiden ved HiBu har vært veldig gøy, da jeg har sett et fullverdig produkt utvikle seg fra start til slutt.

Jeg var en av de to som tilbrakte semesteret før jul i utlandet og var noe bekymret for at det kunne bli en utfordring. Det viste seg at det ikke var noe problem da gruppa var flink til å opprettholde kontakten mens medlemmene var splittet over landegrensene ved hjelp av blant annet Skype. Det var trolig fordi vi var så tidlig ute med å sette sammen gruppa og at vi var bevisste på at det kunne bli utfordrende, at det gikk så bra som det gikk.

Å bruke Scrum som prosjektmodell har vært en spennende utfordring som vi har dratt nytte av i prosjektet. Selv om det var mye å sette seg inn i til å begynne med gikk det veldig greit, og vi har blant annet dratt nytte av daglige Scrum-møter, sprint review og spesielt måten man fordeler arbeidsoppgaver.

Mine hovedansvar var implementasjon og budsjett. Da budsjettet var veldig lite omfattende har jeg fokusert mer på design og implementasjon av applikasjonen, samt å skrive dokumenter for gruppa.

Alle på gruppa har jobbet og slitt og gjort sitt beste for å utvikle vårt endelige produkt både i form av dokumenter og applikasjon. Vi har alle fått erfaring fra nye teknologier og arbeidsmetodikker og har håndtert utfordringer sammen gjennom prosjektet.

Jeg er stolt av å kunne levere vår endelige oppgave og kunne si at jeg er del av DBFactory fra HiBu i Kongsberg. Det har vært veldig givende å jobbe som en selvstendig gruppe på denne måten fra ide til ferdig produkt, og jeg ville ikke gjort det annerledes om jeg fikk sjansen til å prøve igjen.

2.7 KONKLUSJON

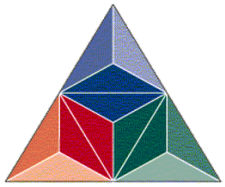
Prosjektet har vært utfordrende, men har gitt oss mye relevant erfaring for å samarbeide. Vi er alle fornøyde med det vi har produsert, både applikasjonen og dokumentasjonen tilhørende prosjektet.

Vi har fått ta i bruk kunnskapen vi har tilegnet oss i løpet av disse tre årene ved HiBu og er stolte av det vi har utviklet for Cytrax As.

Det har vært en positiv opplevelse å samarbeide i en større gruppe på et så omfattende prosjekt. Vi har jobbet hardtog synes vi har fått vist at vi klarer å gjennomføre prosjektarbeid på en profesjonell måte.

PROSJEKTPLAN

HIBU STUDENTPROSJEKT 2012
CYTRAXDB FACTORY



HØGSKOLEN
i Buskerud

cytrax

INNHOLDSFORTEGNELSE

REVISJONSHISTORIKK			
Versjon	Dato	Endringer	Tildelt
1.0	2.1.2012	Første versjon	VK/JG
1.1	8.2.2012	Oppdatert milepæler, datoer i sprint og prosjektstyringsverktøy	JG
2.0	26.3.2012	Oppdatert for 2. presentasjon.	JG
3.0	28.5.2012	Oppdatert for endelig rapport	JG

1 PROSJEKTPLAN

1.1 INNLEDNING

I forbindelse med vår bacheloroppgave ved Høgskolen i Buskerud(HiBu) har vi fått i oppdrag fra Cytrax AS å lage en kodegenerator. Den skal generere kode som opptrer som et interface mellom applikasjon og database. Cytrax er en bedrift som jobber med programutvikling.

De fleste applikasjonene som Cytrax utvikler benytter seg av databaser. Aksess og manipulering av data i disse databasene er standardisert og forskjellene er strukturen på databasen. Cytrax har i dag en applikasjon som de bruker for å generere koden som snakker med databasen fra applikasjonen, men denne applikasjonen er lite modulær og er vanskelig å utvide med nye kodespråk / databasetyper.

Oppgaven vår blir derfor å lage en applikasjon som gjør det samme, men som er mer fremtidsrettet ved at den kan utvides med nye plugins som støtter generering til flere kodespråk eller innlesing fra andre databasetyper.

1.2 HENSIKT MED DOKUMENTET

Hensikten med dette dokumentet er å beskrive planleggingen for prosjektet. Dette dokumentet skal inneholde hva oppgaven går ut på, hva som skal leveres når, mål og begrensninger med prosjektet, hva vi må ha på plass før vi starter, organisering av prosjektet, prosjektkontroll, estimering av hva vi kommer til å gjøre når og overordnede mål gjennom prosjektet.

1.3 FORKORTELSER

Forkortelse	Beskrivelse
Qt	Rammeverk for blant annet GUI i C++

1.4 FORFATTER(E)

Jørgen Grøndal (JG)

Vegard Kaasin (VK)

2 GRUPPESAMMENSETNING

Navn	Tor-Christian Eriksen
Alder	27 år
Ansvarsområder	Gruppeleder og analyse/krav
E-post	trofast@gmail.com
Telefon	915 41 517
Linje	Data - Embedded Systems

Navn	Jørgen Grøndal
Alder	23 år
Ansvarsområder	Dokumenter og versjonskontroll
E-post	joergen.g@gmail.com
Telefon	977 17 235
Linje	Data – Embedded Systems

Navn	Jarle Didriksen
Alder	22 år
Ansvarsområder	Logistikk og test
E-post	d_con89@hotmail.com
Telefon	926 05 868
Linje	Data – Embedded Systems

Navn	Alexander Edland
Alder	22 år
Ansvarsområder	Web og teknologi
E-post	tripflag@gmail.com
Telefon	922 28 336
Linje	Data – Embedded Systems

Navn	Simen Russnes
Alder	21 år
Ansvarsområder	Implementasjon og økonomi/budsjett
E-post	simen_russnes@hotmail.com
Telefon	474 17 939
Linje	Data – Embedded Systems

Navn	Vegard Kaasin
Alder	21 år
Ansvarsområder	ScrumMaster og design
E-post	vegard_k_@hotmail.com
Telefon	900 72 126
Linje	Data – Embedded Systems

3 OPPGAVEN

Vi skal utvikle en applikasjon m/ GUI som skal lese en databasemodell og produsere kildekode for å lese/skrive til databasen. Innlest databasemodell skal representeres i programmet som en dokumentmodell. Denne modellen skal kunne lagres i tekstformat(for eksempel XML), og representeres grafisk i applikasjonen.

Koden som applikasjonen genererer for VB.Net skal være lik koden den gamle generatoren genererer. Applikasjonen skal utvikles i C++ med QT. Applikasjonen skal ha støtte for å utvides med flere typer databaser, og generere kode for flere språk, deriblant VB og C++. For å lese inn og generere kode modulært skal dette skje via plugins. Det skal være mulig å endre innholdet i dokumentmodellen, for eksempel endre navn på tabeller og verdier.

3.1 HVILKE KRAV STILLES TIL APPLIKASJONEN?

I forbindelse med utvikling av applikasjonen for Cytrax har vi fått noen initielle beskrivelser av hva applikasjonen skal kunne gjøre. Disse listes her opp som krav, men siden vi bruker SCRUM som prosjektmodell vil alle krav komme som userstories i productbacklog med tilhørende test case.

- Applikasjonen skal lese inn en databasemodell
- Innlest modell skal generaliseres som en dokumentmodell med informasjon om hvilken database den ble generert fra.
- Dokumentmodellen skal kunne lagres som en prosjektfil
- Prosjektfilen skal kunne åpnes i applikasjonen
- Endringer som er utført på dokumentmodellen skal følge prosjektfil
- Dokumentmodell fra en prosjektfil skal kunne sammenlignes med andre databasemodeller for å se forskjeller / endringer
- Samme dokumentmodell skal alltid generere lik kode og prosjektfil
    - Data som eksporteres til prosjektfil må derfor sorteres på forhånd.
- Generering av kildekode og innlesing av databasemodell gjøres via en plugin for å ha god støtte for utvidelser
- Applikasjonen bør ha et overordnet fokus på å være plattformuavhengig

Utover dette er det blitt avtalt at applikasjonen skal utvikles i C++ med Qt som rammeverk.

3.2 KRAV TIL DOKUMENTASJON

Høgskolen har en del krav til dokumentasjon som skal følges igjennom prosjektprosessen. Gruppen har bestemt at vi skal bruke SCRUM som prosjektmodell og navn på dokumenter vil kunne avvike fra hva man vanligvis ser. Alle de obligatoriske dokumentene vil likevel bli dekket selv om vi bruker en annen prosjektmodell.

Kravspesifikasjon er et dokument som skal inneholde alle krav med tilhørende oversikt over testspesifikasjon som hører til. Siden vi bruker SCRUM vil alle krav være laget som userstories i produkt backlog. I tillegg vil hver user story i productbacklog ha en tilhørende acceptance test som beskriver hva som må være på plass før den blir godkjent.

I SCRUM er productbacklog veldig dynamisk og det er derfor viktig at vi kan spore endringer i productbacklog gjennom prosjektets gang.

Nedenfor ser man en oversikt over planlagte dokumenter og om det inngår i innlevering før en presentasjon.

Dokument	Klar til presentasjon #
Visjonsdokument	1
Prosjektplan	1, 2, 3
Product backlog	1, 2, 3
Test cases (Ligger i product backlog)	1, 2, 3
Sprint rapporter	2, 3
Dokumentstandard	1, 3
Kvalitetssikring for prosjektet	1, 3
Analysedokument	2, 3
Designdokument	2, 3
Teknologidokumenter	1, 2, 3
Beskrivelse av SCRUM prosedyrer	1, 2
Møteinnkallinger	1, 2, 3
Møtereferater	1, 2, 3
Budsjett	1, 3
Kontrakt	1
Timelister	1, 2, 3
Kodestandard	1
Endelig prosjektrapport	3

4 MÅL MED PROSJEKTET

Bakgrunnen for prosjektet er at vi skal tilegne oss kunnskap om og demonstrere ferdigheter i grunnleggende prosjektarbeid. Dette innebærer at vi skal kunne samarbeide, planlegge, dokumentere, organisere, utvikle og teste i prosjektet. Vi skal utvise lederskap og benytte tilgjengelige verktøy som er hensiktsmessige for gruppen.

Vi skal utvise besluttsomhet knyttet til valg av teknologier.

Vi skal skaffe oss erfaring i bruk av SCRUM som prosjektmodell.

Vi skal få erfaring med kommunikasjon mot kunde.

Vi skal produsere et produkt som skal bidra til verdiskapning for kunden vår.

Vi skal gjennomføre et prosjekt på en profesjonell måte og være konsis i måten vi dokumenterer på.

5 FORUTSETNINGER FOR GJENNOMFØRING AV PROSJEKTET

5.1 KUNNSKAP OG EGENSKAPER

Det er en forutsetning for prosjektet at alle på gruppen tilegner seg den nødvendige kunnskapen som trengs for å gjennomføre utvikling av en applikasjon. Dette innebærer å sette seg inn i alle rutiner vi har laget, samt søke hjelp fra de som innehar nødvendig kunnskap om de forskjellige områdene.

Alle på gruppen bør ha generell kunnskap om C++ og ha en god evne til raskt sette seg inn i ny teknologi som vi skal bruke underveis.

Det kreves nøye planlegging og mye innsats fra alle for å få gjennomført prosjektet. Alle må være innstilt på å jobbe ekstra om vi har utestående arbeid.

5.2 SOFTWARE

Gruppen må installere og ha fungerende software i henhold til det som er avtalt for kontroll av prosjektet. Dette innebærer en fungerende IDE, versjonskontroll, kompilator, officepakke og nettleser.

5.3 BRUKERKONTOER

Grunnet gruppens valg for versjonskontroll må alle på gruppen opprette konto på Github og bli tilknyttet prosjektet der.

Alle på gruppen må ha bruker på Dropbox og bli tilknyttet prosjektets mappestruktur.

5.4 LOGISTIKK

For å kunne gjennomføre prosjektet må gruppen ha tilgang på et arbeidsrom / kontormiljø med plasser for alle gruppemedlemmene. Gruppen fikk tildelt rom D302 i slutten av November 2011.

6 FREMDRIFTSPLAN

6.1 SPRINTER

Sprint #	Dato (Fra og til)	Hovedmål med sprint	Estimert antall timer
1	24.11.2011 – 15.12.2011	Få på plass nødvendig dokumentasjon og planer	80
2	10.1.2012 – 31.1.2012	Lage design for dokumentmodell Lage API for: Innlesing av databasemodell og generering av kildekode	400
3	8.2.2012 – 1.3.2012	Lage design for: Innlesing av database, redigering av dokumentmodell, generering av kode. Analyse og design for GUI, med eksempel på layout	400
4	14.3.2012 – 29.3.2012	Lage og holde 2. presentasjon Ferdigstille design, lage skjelettet til applikasjonen fra design.	400
5	19.4.2012 – 3.5.2012	Utvikling: Innlesing av databasemodell via plugin og dokumentmodell.	280
6	5.5.2012 – 16.5.2012	Utvikling: GUI, generering av kode, endring av dokumentmodell og plugins.	280
7	18.5.2012 – 28.5.2012	Lage ferdig dokumentasjon for innlevering. Klargjøre applikasjon for leveranse	280
Sum antall timer i sprint(Ikke inkl. timer utenom):			2120

6.2 PRESENTASJONER

Presentasjon	Dato	Tid	Beskrivelse
1	4.1.2012	12:30	Vise hva vi skal gjøre, når vi skal gjøre det, og hvordan vi skal utføre prosjektet.
2	29.3.2012	12:30	Gå igjennom den valgte tekniske løsningen. Forklare endringer i prosjektplan. Videre fremdrift.
3	12.6.2012	09:00	2 deler, salgsdel og teknisk del.

Tabellen over viser plan for når vi skal ha presentasjoner og hovedmål for hvert prosjekt.

7 MILEPÆLER

I løpet av prosjektet har vi satt oss noen milepæler. Dette er tydelige mål i utviklingsprosessen og produktets tilstand.

7.1 SPRINT #2 UTFØRT

Denne milepælen markerer den første ordentlig gjennomførte sprinten for vår del samt at designprosessen er satt i gang. Vi synes dette er en god milepæl som indikerer at vi er på riktig spor.

Vi anslår å nå denne milepælen den 1. Februar.

7.2 ARKITEKTUR OG TEKNISK LØSNING GODKJENT AV OPPDRAGSGIVER

Den andre milepælen vår er når design av applikasjon er ferdig og godkjent av oppdragsgiver. Denne milepælen betyr at vi begynner på implementasjon og at vår tolkning av løsning er god nok for oppdragsgiver.

Vi antar at denne milepælen blir nådd den 9. Mars.

Samtidig som denne milepælen nærmer seg slutten vil vi forberede 2. presentasjon.

7.3 APPLIKASJON MED EN FUNKSJONELL PLUGIN FOR GEN. OG INNLESNING

Den tredje milepælen er når applikasjonen er ferdig utviklet med støtte for viktigste funksjonalitet for oppdragsgiver. Dette vil si en plugin for å generere til VB.net samt innlesing fra SQLAnywhere 12. Når dette er på plass så har vi nådd det viktigste kravet fra arbeidsgiver, nemlig at applikasjonen skal produsere kode lik det applikasjonen produserer i dag. I tillegg er jo applikasjonen modulær og har god støtte for utvidelser med flere plugins.

7.4 ALL DOKUMENTASJON FERDIG FOR INNLEVERING

Siste milepælen vår er å få ferdig all dokumentasjon som skal leveres til skole og oppdragsgiver.

Vi har satt som mål å være ferdig med all dokumentasjon som skal leveres til 29.5.2012. (En dag før frist)

8 UTVIKLINGSFASER

8.1.1 Analyse

I denne fasen utredes hva som skal gjøres og hvorfor. Siden prosjektet fra Cytrax til dels er utredet med tanke på behov og funksjonalitet vil denne fasen i hovedsak bestå av valg og begrunnelser for teknologi og metodikk.

Det er estimert at denne fasen vil pågå til midten av januar.

8.1.2 Design

I designfasen kommer gruppen til å arbeide med den overordnede strukturen og arkitekturen i programmet. Det vil bli laget UML diagrammer som skal hjelpe oss videre i implementasjonsfasen og det danner grunnlaget for en modulær og moderne applikasjon.

Det er estimert at denne fasen vil pågå til begynnelsen av mars. Når designfasen begynner å nærme seg slutten vil materiell som vi har kommet frem til bli presentert. Denne presentasjonen vil skje i siste halvdel av februar.

8.1.3 Implementasjon

Denne fasen består av programmering. Her vil testing og utvikling av kode stå sentralt. Det er også viktig at dokumentasjon oppdateres når opprinnelige planer endres.

9 ANSVARSOMRÅDER OG ROLLER

Roller	Tildelt
Oppdragsgiver	Cytrax
Prosjektleder	Tor-Christian
Analyse/Krav	Tor-Christian
Dokumenter	Jørgen
Versjonskontroller	Jørgen
Økonomi/Budsjett	Simen
Implementasjon	Simen
Design	Vegard
Scrum	Vegard
Teknologi	Alexander
Web	Alexander
Logistikk	Jarle
Test	Jarle

10 PROSJEKTKONTROLL

10.1 UTVIKLINGSMILJØ

10.1.1 IDE + Kompilator

Som IDE skal gruppen bruke QtCreator. Dette er et utviklingsverktøy som har veldig god støtte for Qt biblioteket vi skal bruke. Innad i gruppen brukes både Windows og Linux som operativsystem og det var derfor viktig at vi fant en IDE som fungerer kryssplattform.

Vi kommer til å benytte den medfølgende kompilatoren til Qt Creator, som er GCC.

10.1.2 Versjonskontroll

Siden vi skal utvikle en ferdig applikasjon er det helt essensielt at vi benytter programvare for versjonskontrollering. Ved bruk av versjonskontroll vil man enkelt kunne spore endringer som er utført, samt gå tilbake til fungerende versjoner dersom det har skjedd feil.

Til versjonskontroll skal gruppen benytte seg av Git via Github (1). Her finnes det et ferdiglaget verktøy man bruker for å kommunisere med versjonskontrollsystemet som gruppen skal benytte.

10.2 TESTSPESIFIKASJON

Testing i prosjektet gjøres på flere nivåer og er i hovedsak beskrevet i dokument for kvalitetssikring (2). Aksepteringstesting for userstories er beskrevet i produktbackloggen med testscenario for alle userstories.

10.3 PROSJEKTMODELL

10.3.1 SCRUM

For gjennomføring av prosjektet har gruppen bestemt seg for å bruke SCRUM. Årsaken til valget er at det virker som en spennende utviklingsmodell som gruppen ønsket å lære mer om ved å ta den i bruk. Ut i fra det vi kjente til angående SCRUM virker det som en effektiv utviklingsmodell dersom den blir tatt i bruk på en god måte.

10.4 PROSJEKTSTYRINGSVERKTØY

Vi har sett på flere muligheter når det kommer til et verktøy som gjør prosjektstyring enklere. I utgangspunktet ønsket vi å bruke en videreutviklet versjon av AMEO sine excel-ark for å administrere prosjektet. Etter å ha brukt dette på den første sprinten vår

for dokumentasjon og lignende hadde vi blandede følelser. Det var upraktisk å måtte oppdatere excel-arkene hele tiden.

I Scrum kan det være vanskelig å få oversikt over hvem som gjør hva. Det er derfor viktig for oss å kunne vise at alle har arbeidet og bidratt til prosjektet. Vi synes at bruk av excel-dokumenter til prosjektstyring gjør det vanskelig å vise dette. Vi har derfor sett på andre alternativer, blant annet ScrumWorks og Kunagi.

10.4.1 ScrumWorks

Scrumworks var et verktøy vi ble tipset om av ekstern sensor Emil. Dette er et veldig kraftig verktøy som hadde mulighet for gratis lisens dersom det var færre enn ti brukere. Vi prøvde derfor å få tak i en webserver å kjøre dette på. Vi fikk låne en server av Cytrax og fikk ScrumWorks opp å kjøre.

ScrumWorks er veldig kraftig og har mye funksjonalitet som er knyttet til effektivitet og beregning av kostnader. Dette var funksjonalitet som skapte mye overhead for oss, og vi valgte å se etter andre alternativer.

10.4.2 Kunagi

Kunagi er et mer begrenset system enn ScrumWorks er når det gjelder funksjonalitet, men inneholdt det vi trenger av funksjonalitet for å drive en Scrum-prosess. Se teknologidokument (3) for utredning om kunagi.

Gruppen besluttet å bruke Kunagi som prosjektstyringsverktøy.

10.5 ANDRE DOKUMENTER

10.5.1 Budsjett

For å holde orden på hvilke utgifter vi har i prosjektet må vi føre et budsjett over planlagte utgifter samt påløpte utgifter underveis.

10.5.2 Timelister

Timelistene (4) våre beskriver antall timer vi har brukt på forskjellige aktiviteter og datoer. På forsiden er alle aktiviteter beskrevet med en oppsummering sånn at man enkelt kan se totalt timefordeling mot en aktivitet.

10.5.3 Kodestandard

Kodestandard er noe vi som en utviklingsgruppe må ha på plass. Kodestandarden vi skal bruke er definert i (5).

11 RISIKOANALYSE

Når man arbeider med prosjektarbeid i gruppe er det essensielt å ta høyde for at ting ikke går som planlagt. I de fleste prosjektmodeller vil man lage en oversikt over hvilke risikoer som kan påvirke prosjektet og deretter skrive en handlingsplan. Siden gruppen vår benytter seg av SCRUM bruker man mindre ressurser på planlegging for risikoer. Hele filosofien bak SCRUM er at man danner sprinter med bakgrunn i en viss hastighet og gjennomføringsevne. Dersom det forekommer uventede hendelser er det relativt enkelt å tilpasse sprinter og forventet hastighet. Dette gjør at man reduserer bruk av ressurser på å planlegge for eventuelle uforutsette hendelser og kan fokusere på gjennomføring av prosjektet.

12 REFERANSER

1. **DB Factory.** *Teknologidokument - Git og Github.* 2011.

2. —. *Kvalitetssikring.* 2011.

3. —. *Teknologidokument - Kunagi.* 2012.

4. —. Timeliste. 2012.

5. —. *Kodestandarder.* 2011.

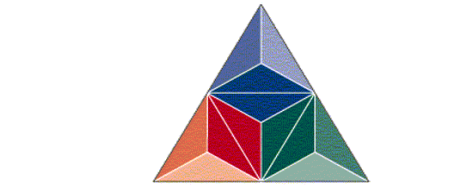
6. —. *SCRUM - Metodikk og regler.* 2011.

7. —. *Teknologidokument - Doxygen.* 2011.

8. —. *Teknologidokument - Qt.*

TEKNOLOGIDOKUMENT FOR CPPCHECK

HIBU STUDENTPROSJEKT 2012
CYTRAXDB FACTORY



REVISJONSHISTORIKK			
Versjon	Dato	Endringer	Tildelt
1	8.2.2012	-Opprettet	JD

INNHOLDSFORTEGNELSE

1 INTRODUKSJON

Vi ønsker å undersøke denne teknologien fordi vi ønsker å bruke et verktøy for å kunne utføre statisk testing av våre kildekodefiler.

1.1 FORKORTELSER OG FORKLARINGER

Synonym	Beskrivelse
QT	Et kryss-platform rammeverk for C++
Bugs	Uventede feil i en applikasjon

1.2 FORFATTER(E)

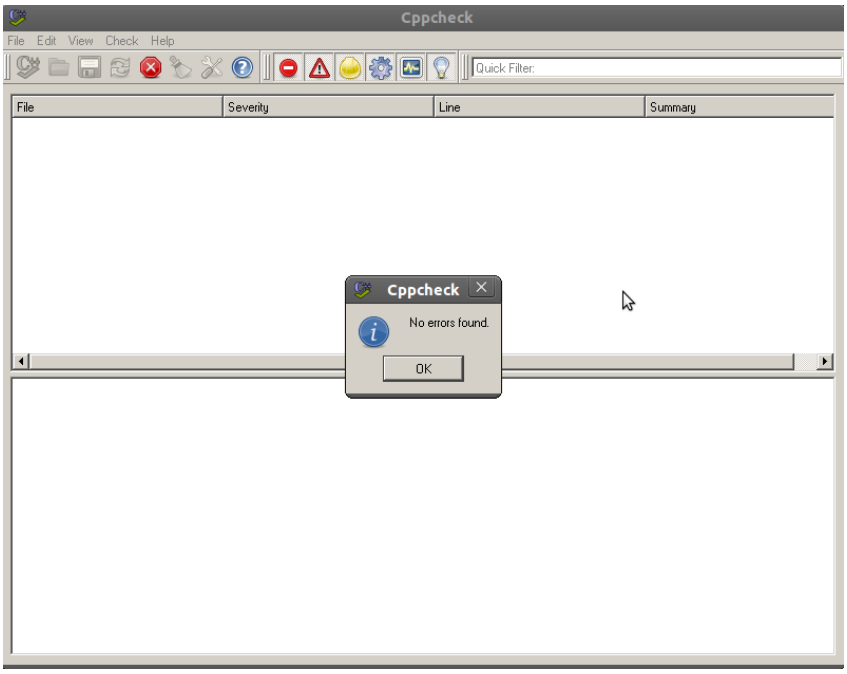
Jarle Didriksen (JD)

2 HVA ER CPPCHECK

Cppcheck er et analyseverktøy for C/C++-kode. Verktøyet er GUI-basert for Windows og terminalbasert for Linux. Cppcheck finner ikke syntaksfeil slik som kompilatorer gjør, men detekterer heller de typene bugs som kompilatorer vanligvis ikke gjør.

2.1 EKSEMPEL PÅ VANLIG BRUK

La oss si at vi har en kildefil som heter *main.cpp* som vi ønsker å sjekke med Cppcheck. I Windows-versjonen vil vi da velge menyen Check -> Files -> main.cpp, og deretter vil Cppcheck undersøke kildefilen. Om ingen feil er funnet vil det bli seende ut som i *figur 1* på neste side. Tilsvarende vil terminalvarianten bli seende ut som i *figur 2*.



Figur 1 – Sjekking av en fil i Cppcheck

```
snorre@snorre-3000-N200:~/QtProjects/arealsirkel$ cppcheck main.cpp
Checking main.cpp...
snorre@snorre-3000-N200:~/QtProjects/arealsirkel$
```

Figur 2 – Sjekking av en fil i Cppcheck (terminal)

2.2 Andre ting som er verdt å nevne?

Cppcheck er et gratis program lisensiert under GNU General Public License (GPL), og er således gratis for oss å bruke. Programmet er testet og ser ut til å fungere fint med Qt, dog har det ikke blitt testet med mer avanserte Qt-funksjoner på nåværende tidspunkt.

3 Hvorfor bruke Cppcheck?

3.1 Fordeler

Cppcheck finnes både til Windows og Linux, det krever lite ressurser og det er dessuten gratis. Det er også svært lite å sette seg inn i, og man kan begynne å sjekke sine egne filer så og si umiddelbart etter installasjon.

3.2 God funksjonalitet

Man kan selv velge hva slags feil man skal lete etter, alt fra én feil til alle feil som er mulige for Cppcheck å finne. I tillegg kan man velge å ta bort filer man ikke vil sjekke,

dersom man skal sjekke en hel mappe. Noen ganger vil man kanskje ønske å lagre resultatene fra analysen, og det kan Cppcheck lagre i en tekstfil. Det kan også genereres output i XML-format. Man kan også formattere outputen så den ser annerledes ut vha. templates, f.eks å få output som er kompatibelt med Visual Studio eller gcc, eller man kan lage sitt eget design.

Man kan skjule bestemte typer feil om man vil, f.eks om man vet om en feil, men ønsker å se etter andre. Dette kan spesifiseres ytterligere ved å skrive kommentarer i selve kildefilen slik som i følgende eksempel:

```
char arr[5];
// cppcheck-suppress arrayIndexOutOfBounds
arr[10]=0;
```

Denne koden ville vanligvis gitt feilmelding, men hvis man har med kommentaren i midten samtidig som man setter et flagg vil denne feilen overses.

3.3 YTELSE

Dette er et lettvektig program som krever lite.

4 ULEMPER

Ettersom at Cppcheck først og fremst er for C++ kode kan det være at den ikke greier å tolke mer avanserte Qt-funksjoner.

5 VIDERE LESING

5.1 EKS. BRUKERVEILEDNING

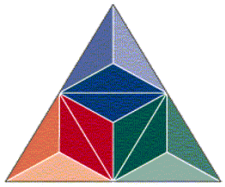
Hjemmesiden til Cppcheck inneholder en god brukerveiledning som er listet under referanser.

6 REFERANSER

1. Cppcheck. *Cppcheck.*[Sist besøkt: 8 2 2012.] [Internett] http://cppcheck.sourceforge.net/
2. Cppcheck. *Brukermanual.* [Sist besøkt: 8 2 2012.] [Internett] http://cppcheck.sourceforge.net/manual.pdf

TEKNOLOGIDOKUMENT FOR DOXYGEN

HIBU STUDENT PROSJEKT 2012
CYTRAX DB FACTORY



HØGSKOLEN
i Buskerud

cytrax

INNHOLDSFORTEGNELSE

REVISJONSHISTORIKK			
Versjon	Dato	Endringer	Tildelt
1	7.12.2011	Første versjon	JD
2	14.12.2011	Andre versjon	JD
2.1	15.12.2011	Revidert andre	JD
2.1.1	2.1.2012	Layout	VK

1 INTRODUKSJON

Vi ønsker å se nærmere på Doxygen fordi det høres ut som et godt program for å hjelpe oss med å dokumentere kode. Vi hadde hørt om Doxygen tidligere, og vurderte dette som et verktøy, og under første møte med Cytrax kom Doxygen opp som forslag fra Audun.

Vi ønsker å avdekke hva slags dokumentasjon vi kan bruke Doxygen for, og til hvilken grad dette lar seg gjøre.

1.1 FORKORTELSER OG FORKLARINGER

Synonym	Beskrivelse
QT	Et kryss-platform rammeverk for C++

1.2 FORFATTER(E)

Jarle Didriksen (JD)

1.3 TILDELT

Jarle Didriksen (JD)

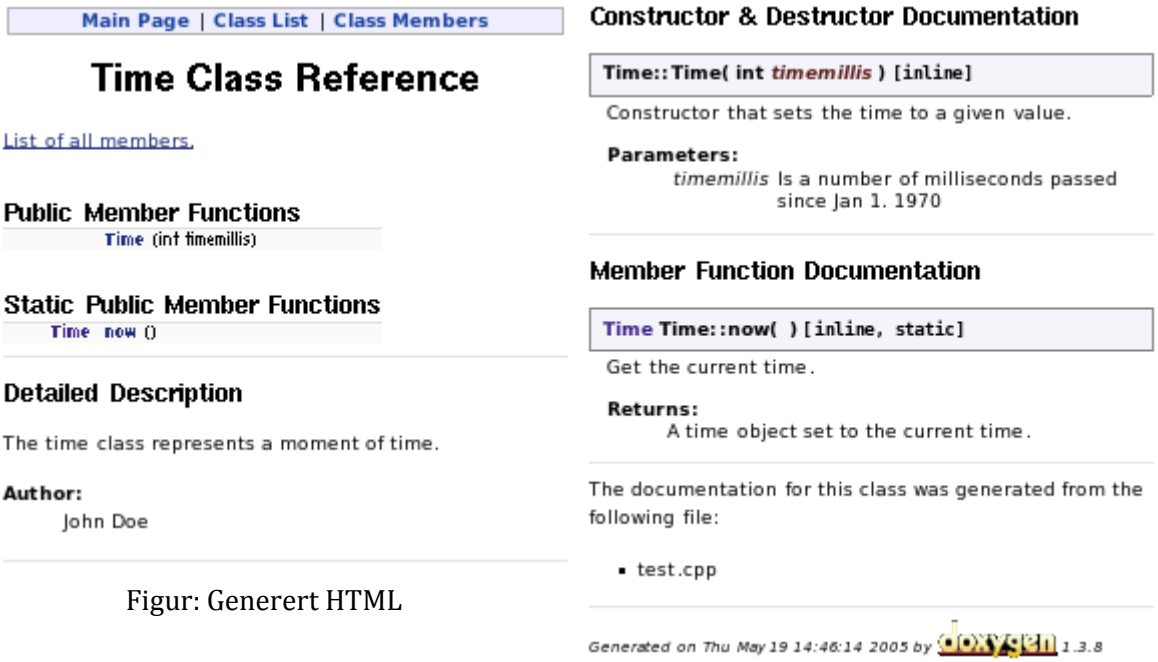
2 HVA ER DOXYGEN

Doxygen er et dokumentasjonssystem for et bredt utvalg programmeringsspråk, deriblant C++ som er det vi bruker for vårt prosjekt. Doxygen er kryss-plattform og kjører på de plattformene som vi bruker.

2.1 EKSEMPEL PÅ VANLIG BRUK

Doxygen bruker kommentarene vi skriver i koden. Et eksempel på C++-kode med dokumentasjonskommentarer er:

```
class Time {
    public:
        /**
         * Constructor that sets the time to a given value.
         *
         * @param timemillis Number of milliseconds
         *        passed since Jan 1, 1970.
         */
        Time (int timemillis) {
            // the code
        }
        /**
         * Get the current time.
         *
         * @return A time object set to the current time.
         */
        static Time now () {
            // the code
        }
};
class NameOfClass
```



Main Page | Class List | Class Members

Time Class Reference

List of all members.

Public Member Functions

Time (int timemillis)

Static Public Member Functions

Time now ()

Detailed Description

The time class represents a moment of time.

Author:
John Doe

Constructor & Destructor Documentation

Time::Time(int *timemillis*) [inline]

Constructor that sets the time to a given value.

Parameters:
timemillis Is a number of milliseconds passed since Jan 1. 1970

Member Function Documentation

Time Time::now( ) [inline, static]

Get the current time.

Returns:
A time object set to the current time.

The documentation for this class was generated from the following file:

- test.cpp

Generated on Thu May 19 14:46:14 2005 by doxygen *1.3.8*

Figur: Generert HTML

2.2 Andre ting som er verdt å nevne?

Doxygen er lisensiert under GNU General Public License, og er således gratis for oss å bruke. Dokumenter vi produserer med Doxygen er avledede arbeider og påvirkes derfor ikke av lisensen.

3 Hva er Doxygen

3.1 Sammenligne med annen / tilsvarende teknologi

Mange tilsvarende teknologier er proprietære. DOC++ er et alternativ, som den første versjonen av Doxygen lånte kode fra, men Doxygen ble senere omskrivd.

3.2 Fordeler

Med Doxygen kan vi generere en online dokumentasjonsutforsker, lignende Javadoc. Doxygen støtter dokumentasjon-tags som brukes i QT, noe som er et stort pluss for oss. Doxygen kan generere output for Word, PostScript, PDF, HTML og «man pages», så det er mange valgmuligheter.

3.3 God funksjonalitet

Dokumentasjonen trekkes ut direkte fra kildene, noe som gjør det enklere mtp. å holde dokumentasjonen konsistent med kildekoden. Doxygen kan konfigureres til å trekke ut kodestrukturen fra udokumenterte kildefiler, som gjør det enklere å navigere i større kildedistribusjoner. Man kan også «utnytte» programmet for å lage vanlig dokumentasjon om det er ønskelig. Doxygen kan kryssreferere dokumentasjon og kode, så det er enkelt for leseren å se den faktiske koden.

3.4 Ytelse

Ytelsen er bra.

4 Ulemper

Ingen ulemper som er signifikante for oss.

5 Eks. Brukerveiledning

Det er en utfyllende brukerveiledning på hjemmesiden til Doxygen, henvist til i referanse nr. 3.

6 REFERNASER

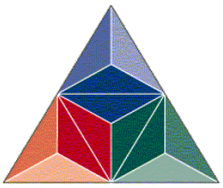
1. Doxygen. *Doxygen.* [Internett] [Sitert: 7 12 2011.] http://www.stack.nl/~dimitri/doxygen/.

2. Wikipedia. *Wikipedia.* [Internett] [Sitert: 7 12 2011.] http://en.wikipedia.org/wiki/Doxygen.

3. Doxygen. *Doxygen.* [Internett] [Sitert: 7 12 2011.] http://www.stack.nl/~dimitri/doxygen/manual.html.

Teknologidokument for Git og Github

HIBU STUDENTPROSJEKT 2012
CYTRAX DB FACTORY



HØGSKOLEN
i Buskerud

cytrax

REVISJONSHISTORIKK			
Versjon	Dato	Endringer	Tildelt
1.0	21.12.2011	Første versjon	JG

INNHOLDSFORTEGNELSE

1 INTRODUKSJON

Versjonskontroll er et system som lagrer endringer i filer over tid og gjør deg i stand til å hente tilbake tidligere versjoner av filene. Det finnes mange forskjellige systemer som håndterer dette, ett av dem er Git.

Git er hovedsaklig designet for å bruke veldig lite ressurser sammenlignet med andre systemer.

For å ta i bruk Git har man flere alternativer, blant annet kan man installere git på en dedikert server å bruke kommandoer direkte mot den, alternativt kan man bruke for eksempel Github som vi skal se på i dette dokumentet. Der finnes det endel ferdig funksjonalitet som er veldig nyttig for utvikling, blant annet kodereview, automatisk sammensveising av endringer og fjernlager for filer.

1.1 FORKORTELSER OG FORKLARINGER

Synonym	Beskrivelse
QT	Et kryss-platform rammeverk for C++

1.2 FORFATTER

Jørgen Grøndal (JG)

1.3 TILDELT

Jørgen Grøndal (JG)

2 HVA ER GIT OG GITHUB

Git er teknologien som håndterer selve versjonskontrolleringen. Git ble initielt utviklet og designet av Linus Torvalds siden han ikke fant versjonskontrollsystemer som tilfredsstile kravene han hadde når han jobbet med utvikling av kernels til Linux.

Git er designet for å være veldig effektivt og ha god støtte for distribuert og ikke-lineær utvikling. Git passer veldig godt sammen med en smidig prosjektmodell som SCRUM fordi utviklingen ofte går i mange retninger samtidig under en sprint.

I git er en versjon representert ved en commit. Det som skjer når man lagrer en commit er at man tar et bilde av alle filene og lagrer dette. For å spare plass og unngå dobbellagring av filer så vil alle filer uten endringer kun inneholde en peker til forrige versjon. Dette gjør at man ikke sitter med kopier av store mengder data som ofte skjer i andre versjonskontrollsystemer.

2.1 BRANCHING

Branching er en av de viktigste funksjonene i Git dersom man tar det i bruk på en god måte. Man kan se på branching som muligheten til å lage pekere til en gitt versjon. Man kan videre benytte en slik branch til å utvikle systemet i en egen retning, og deretter smelte det sammen igjen med en annen branch. Det er laget støtte for automatisk sammensveising av forskjellige brancher, men dersom endringere avviker for mye må man manuelt korrigere feilene, det finnes mange gode verktøy som gir deg en enkel oversikt over hvor avvikene er sånn at man kan korrigere dette. Denne funksjonen støtter oppunder tanken om ikke-lineær utvikling.

Hvis man ser på dette i sammenheng med for eksempel SCRUM kan man benytte en slik branch til å representere en oppgave i en sprint. Når oppgaven er utført og godkjent smelter man den sammen med hovedbranchen til applikasjonen, og man sitter med en ny versjon.

Github igjen tar dette et steg videre og tilbyr funksjonalitet som slår sammen kodereview og sammensveising av brancher. Dette kalles en "Pull request" og vil gi mulighet for å diskutere kode ned på linjenivå direkte i nettleseren.

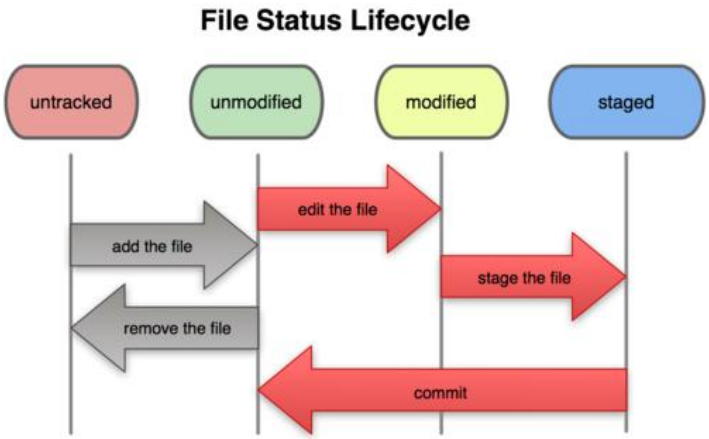
2.2 LOKALE OPERASJONER

Git tilbyr muligheten for å dytte data man har lokalt til et fjernlager. Likevel er de aller fleste operasjoner lokale, samt at de aller fleste versjoner blir lagret lokalt. Det er kun når man er ferdig med en utviklingsbranch at man burde dytte den til fjernlageret sånn at det kan sveises sammen med resten av prosjektet.

En vanlig arbeidsflyt i Git foregår som følger:

1. Man modifiserer filer i mappestrukturen
2. Filene som er endret blir "staged", det betyr at det taes et bilde av de modifiserte filene.
3. Man utfører en commit. Nå vil de filene som tidligere var "staged" blir permanent lagret til git.

Det som er viktig å huske på er at alle disse operasjonene kun er lokale. For å få operasjonene og endringene publisert må man dytte endringene i den branchen man ønsker til fjernlageret.



Figur 1: Livssyklus til en fil som er i Git lageret (1)

Figuren ovenfor viser livssyklusen til filer fra før de kommer inn i git lageret til de blir fjernet.

3 HVORFOR BRUKE GIT OG GITHUB

Git er et relativt nytt versjonskontrollsystem og flyter veldig godt sammen med smidige prosjektmodeller da det gir veldig god støtte for ikke-lineær utvikling. Samtidig vil bruk av Git via Github gi oss et fjernlager der alle filene våre blir lagret, med tilhørende backup fra leverandøren.

Vi vil få ferdig funksjonalitet for kodereview og diskusjon rundt koden, og alt legger til rette distribuert arbeid. Dette gir oss en mye mer robust samarbeidsplatform når det kommer til utvikling, og man kan tenke på hva man utvikler i sin oppgave istedenfor å bruke tid på å informere andre om endringer.

3.1 SVN

SVN er et annet versjonskontrollsystem som fungerer på en helt annen måte. Her har man endringslogg på hver enkelt fil. Når man skal gjøre endringer i en fil sjekker man ut filen som gjør filen låst for andre. Dette er en helt ok måte å samarbeide på, men krever mye synkronisering og er tungvindt ved ikke-lineær utvikling.

For vår del ville bruk av SVN betydd at utføring av utviklingsoppgaver i en sprint ville krevd mye samkjøring og kommunikasjon som hadde redusert effektiviteten i utviklingen.

Når det kommer til å arbeide med SCRUM som prosjektmodell ser vi en klar fordel å bruke Git sammenlignet med SVN.

3.2 FORDELER

Som nevnt tidligere er en av de største fordelene støtte for ikke-lineær utvikling. Det er en veldig effektiv håndtering av store prosjekter på grunn av at filer uten endringer ikke dupliseres men lagres som en peker til forrige versjon.

En annen klar fordel er at man får et fjernlager ved bruk av Github som gjør distribuert utvikling mye enklere. Man kan på få minutter benytte en ny PC og direkte klone hele fillageret til datamaskinen.

3.3 ULEMPER

Kan være komplisert å sette seg inn i en god arbeidsflyt som inkorporerer Git.

4 TILLEGG TIL GIT OG GITHUB

Det er et ferdiglaget verktøy som både har GUI og er kommandolinjebasert. De mest vanlige operasjonene finner man i verktøyet med GUI, men dersom man skal utføre spesielle operasjoner har man kommandolinjeverktøyet.

5 VIDERE LESING

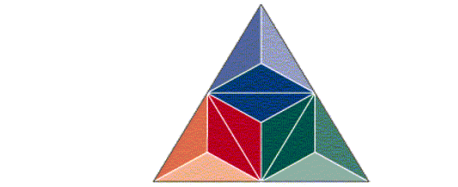
Det er skrevet en bok om Git som kan lastes ned gratis på nett. (1)

6 REFERANSER

1. **Chacon, Scott.** *Pro Git.* [Internett] [Sitert: 12 December 2011.] http://progit.org/ebook/progit.pdf.

TEKNOLOGIDOKUMENT FOR KUNAGI

HIBU STUDENTPROSJEKT 2012
CYTRAX DB FACTORY



REVISJONSHISTORIKK			
Versjon	Dato	Endringer	Tildelt
1	25.1.2012	Første utgave	JG

INNHOLDSFORTEGNELSE

1 INTRODUKSJON

Når vi bruker SCRUM som prosjektmodell, er vi avhengig av å kunne dokumentere fremdrift og administrere sprinter, produkt backlog og oppgaver. For å gjøre dette benyttet vi oss først av en videreutviklet versjon av AMEO (1) sine excel dokumenter. Vi at det ble veldig mye endringer og oppdateringer i disse dokumentene. Som et resultat av dette ønsket vi å finne et verktøy som var bedre egnet. Kunagi er et slikt verktøy som vi skal se litt nærmere på.

1.1 FORFATTER(E)

Jørgen Grøndal (JG)

2 HVA ER KUNAGI

Kunagi (2) er et verktøy som administrerer scrum prosessen. Det kjøres fra en webserver og gir brukerne et webgrensesnitt der man kan administrere alt fra user stories i product backlog til timeforbruk på en spesifikk task. I tillegg gir den mulighet til å eksportere data i form av pdf-filer.

2.1 BEGRENSNINGER

Funksjonelt sett har man noen begrensninger i Kunagi. Det er ingen tilpasningsmuligheter når det kommer til å generere rapporter. Det vil si at når data er eksportert må vi antakeligvis manuelt formatere og modifisere dataene når vi skal presentere det.

Å spore review av dokumenter og kode var noe vi synes var vanskelig å administrere via Kunagi. Dette har nok med at vi har begrenset erfaring med bruk av SCRUM som prosjektmodell og ikke er godt kjent med "best practice" innenfor Q/A i scrum.

3 HVORFOR BRUKE KUNAGI?

Siden vi har liten erfaring med bruk av Scrum vil et verktøy som kan administrere deler av prosessen legge bedre til rette for at vi fokuserer på de viktigste områdene innenfor scrum. Kunagi er et verktøy som har den basisfunksjonalitet vi trengte og inneholder lite overflødig funksjonalitet som man ville benyttet seg av i en mer profesjonell setting der lønnsomhet og effektivitet er viktige nøkkelord.

3.1 SCRUMWORKS

Scrumworks er et annet verktøy som tilbyr mye av den samme funksjonaliteten som Kunagi. Vi har også sett på Scrumworks som et alternativ men applikasjonen krever mer av en server. I tillegg synes vi det var tungvindt at man måtte administrere produktet via en egen applikasjon og at dette ikke var mulig via webgrensesnittet.

Scrumworks har også veldig mye funksjonalitet som har med det økonomiske aspektet av prosjektstyring som vi ikke har behov for til vårt prosjekt. Vi synes derfor at Kunagi er et enklere verktøy som passer best til de behovene vi har.

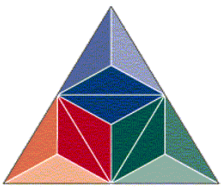
4 REFERANSER

1. **AMEO.** *Project Report for AMEO.* Kongsberg : HiBu, 2011.

2. Kunagi. *Kunagi.* [Internett] [Sitert: 10 January 2012.] http://kunagi.org/.

Teknologidokument for Qt

HIBU STUDENT PROSJEKT 2012
CYTRAX DB FACTORY



HØGSKOLEN
i Buskerud

cytrax

1 INNHOLDSFORTEGNELSE

REVISJONSHISTORIKK			
Versjon	Dato	Endringer	Tildelt
1.0	01.01.2012	Første versjon	VK / AE
1.1	02.01.2012	Skrevet mer utfyllende	VK
1.2	02.01.2012	Lagt til mer stoff	AE

1. INTRODUKSJON

Ved valg av C++ som utviklingsplattform kan det endelige produktet i utgangspunktet kjøre uavhengig av miljø (operativsystem). Ulempen er at C++ ble utviklet for kommandolinjen, og står dermed uten innebygget rammeverk for produsering av grafiske brukergrensesnitt. Det finnes flere utvidelser for å løse dette, blant flere MFC (Windows), GTK (linux) og QT (plattformuavhengig). Vi ønsker å benytte QT da rammeverket har en lang rekke frynsegoder.

1.1 FORKORTELSER OG FORKLARINGER

Synonym	Beskrivelse
Qt	Et kryss-platform rammeverk for C++
QtestLib	Testbibliotek for Qt
QtCreator	Qts IDE

1.2 FORFATTER(E)

Vegard Kaasin (VK)
Alexander Edland (AE)

1.3 TILDELT

Vegard Kaasin (VK)
Alexander Edland (AE)

2 HVA ER QT?

QT er et sett generiske klasser, et «rammeverk», som tilbyr en stor mengde funksjonalitet for å lette utvikling av programmer i C++. Qt er et kryssplattform rammeverk, noe som gir kode som kjører på mange andre plattformer, som for eksempel Windows, Linux, Mac OS x og Symbian. Qt har god funksjonalitet for utvikling av

applikasjoner med GUI, noe vi verdsetter høyt. Det er hovedsaklig beregnet å bruke C++ i Qt, men har i senere versjoner fått støtte for java. Qt har også en egen IDE, Qt Creator, som vi skal benytte oss av.

2.1 EKSEMPEL PÅ VANLIG BRUK

QT er ofte førstevalget ved utvikling av applikasjoner som bygger på nettverkskommunikasjon, og/eller som bør kunne kjøre på Windows, Mac og linux. Rammeverket forener også miljøer ved å abstrahere vekk ulikheter som biblioteks forskjeller.

Eksempel på et hello world program i Qt følger under.

```
1 #include <QApplication>
2 #include <QLabel>

3 int main(int argc, char *argv[])
4 {
5     QApplication app(argc, argv);
6     QLabel *label = new QLabel("Hello Qt!");
7     label->show();
8     return app.exec();
9 }
```

Som vil gi utskriften:

Figur1. Utskrift i Qt

2.2 ANDRE TING SOM ER VERDT Å NEVNE?

Qt er ett gratis rammeverk. Det har mange ferdiglagde biblioteker og moduler som gjør det enklere og raskere for utviklere å kode. Her finner du blant annet submoduler for GUI, multimedia, nettverk, OpenGL, Sql, XML, test og mye mer. Qts testmodul QtestLib gir også mulighet for enkelt lage tester.

3 HVORFOR BRUKE QT?

Qt er ett av de beste gratis rammeverkene for utvikling av C++ kode som er tilgjenlig idag. I møte med Cytrax ble det også foreslått at vi skulle bruke Qt.

3.1 SAMMENLIGNE MED ANNEN / TILSVARENDE TEKNOLOGI

MFC var et av de første rammeverkene for grafikk i C++ og kjører kun på Windows. Løsningen har mange problemer i tillegg til denne begrensningen, og anses som ubrukelig. GTK er kryss-plattform, men følger linux-filosofien, noe som gjør at løsningen

passer dårlig inn i et monolittisk system som Windows[6]. GTK+ og WxWidgets er to andre alternativer til Qt. Vi har ikke funnet andre noen store forskjeller på disse rammeverkene, men det virker som det er litt smak og behag. Noen mindre forskjeller finnes, som at Qts biblioteker gjør at du kan kode på færre linjer, men har litt tregere ytelse enn WxWidgets. De andre rammeverkene kan også utvikles i flere kodespråk som C i GTK+, noe som ikke har noe å si for vårt prosjekt.

3.2 FORDELER

QT stiller sterkt i forhold til andre løsninger. Rammeverket er høyst dynamisk og kan innbakes i enhver plattform, med få ulemper. I tillegg tilbys en rekke frynsegoder som nettverks-API og egne variabeltyper som forenkler utviklingen vesentlig. Qt sin GUI-del er ikke minst meget godt uttenkt, og gir utvikleren frie tøyler med få begrensinger.
Vi har også noen i Gruppa som har brukt Qt før, samt det kom forslag fra Cytrax om å bruke det på møte. Ved å bruke samme rammeverk, IDE og testbibliotek fra samme utgiver minimerer vi risikoen for feil eller utforutsette hendelser.

3.3 YTELSE

Qt gir god ytelse og god støtte for trådprogrammering. Kan få høyt minneforbruk i forhold til andre rammeverk ved bruk av mange submoduler[7]. QT-rammeverket er alltid implementert på den mest effektive metoden i ethvert system, enten modulært (linux) eller monolittisk (Windows) slik at operativsystemets egen utforming fører til optimal ytelse. MFC har noe høyere generell ytelse enn QT, men QT stiller langt sterkere med en standardisert logisk utforming og solide oppbyggelse (få muligheter for unexpected behavior).

4 VIDERE LESING

Referanser Qt har god støtte på hjemmesidene(*se referanse 1*), der Online Support, Wiki, forum og Doc ligger tilgjengelig. I tillegg har vi fått tak i boken *C++ GUI Programming with Qt 4, Second Edition*. Her finnes en detaljert beskrivelse om Qt, som tar utgangspunkt i at man ikke har brukt Qt før.

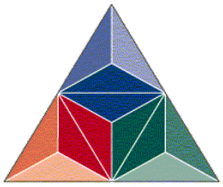
5 REFERANSER

1. http://qt.nokia.com (sist besøkt 01.01.2012)
2. http://en.wikipedia.org/wiki/Qt_(framework) (sist besøkt 01.01.2012)
3. http://wiki.wxwidgets.org/WxWidgets_Compared_To_Other_Toolkits (sist besøkt 01.01.2012)
4. http://www.gtk.org/ (sist besøkt 01.01.2012)
5. C++ GUI Programming with Qt, Second Edition(2008), Jasmine Blanchette, Mark Summerfield
   ISBN: 0-13-235416-0

6. http://gnomejournal.org/article/100/php-gtk-widgets-gadgets-an-interview-with-elizabeth-smith (sist besøkt 02.01.2012)
7. http://stackoverflow.com/questions/1346207/qt-application-performance-vs-winapi-mfc-wtl (sist besøkt 02.01.2012)

TEKNOLOGIDOKUMENT FOR QTESTLIB

HIBU STUDENTPROSJEKT 2012
CYTRAX DB FACTORY



HØGSKOLEN
i Buskerud

cytrax

INNHOLDSFORTEGNELSE

REVISJONSHISTORIKK			
Versjon	Dato	Endringer	Tildelt
1.0	19.12.2011	Første versjon	SR
1.1	02.01.2012	Revidert, Fiksa layout Mindre endringer	VK JD

1. INTRODUKSJON

Vi har valgt å bruke Qt som rammeverk for utvikling av applikasjonen vår. QTestLib tilbyr en enkel måte å teste applikasjoner på og vi har valgt å bruke dette som hovedtestprogram.

1.1 FORKORTELSER OG FORKLARINGER

Synonym	Beskrivelse
QT	Et kryss-plattform rammeverk for C++
QTestLib	Rammeverk for å teste applikasjoner laget i C++

1.2 FORFATTER(E)

Simen S. Russnes (SR)

1.3 TILDELT

Simen S. Russnes (SR)

2 HVA ER QTESTLIB

QTestLib er et rammeverk laget for å teste Qt-baserte applikasjoner og biblioteker. Rammeverket inkluderer all vanlig funksjonalitet i enhetstestingsrammeverk. Det finnes også utvidelser for å teste GUI.

2.1 EKSEMPEL PÅ VANLIG BRUK

QTestLib brukes til å lette på kodeskrivingen for enhetstesting i Qt-baserte applikasjoner og biblioteker.

Man kan for eksempel teste en metode "toUpper" i en QString (som skal omgjøre en tekststreng til store bokstaver) på følgende måte:

```
void TestQString::toUpper()
 {
    QString str = "Hello";
    QCOMPARE(str.toUpper(), QString("HELLO"));
}
```

Når man da kjører testen vil man få følgende output:

```
********* Start testing of TestQString *********
Config: Using QTest library 4.8.0, Qt 4.8.0
PASS   : TestQString::initTestCase()
PASS   : TestQString::toUpper()
PASS   : TestQString::cleanupTestCase()
Totals: 3 passed, 0 failed, 0 skipped
********* Finished testing of TestQString *********
```

Dette kan selvfølgelig utvides til å inneholde mange fler tester.

initTestCase og cleanupTestCase er funksjoner som automatisk blir kalt før og etter.

2.2 Andre ting som er verdt å nevne?

QTestLib inngår i standard Qt, og det vil da ikke være noen problemer med lisensiering for oss.

3 Hvorfor bruke QTestLib?

3.1 Sammenligne med annen / tilsvarende teknologi

QTestLib tilbyr et grensesnitt for å teste applikasjonen, hvor vi ellers måtte ha skrevet alt fra bunnen av dersom det ble valgt å ikke bruke QTestLib.

3.2 Fordeler

Rammeverket er et lettvektig, og er nesten uavhengig av annen software. Det tilbyr god funksjonalitet for kjapp testing, og enkel GUI-testing. Videre har det tråd-sikkerhet under outputs i testing og man kan enkelt utvide med egne selvlagde typer.

Når man tester en klasse og diverse funksjoner lager den en logg over resultater og evt. grunner til at en test feilet.

3.3 God funksjonalitet

GUI-hendelser: Man kan generere inn falske GUI-inputs for å kunne teste grensesnittet vha. testbiblioteket.

God funksjonalitet for å kjøre samme test mange ganger med forskjellig data. Dette kan gjøres ved å legge inn data i et array istedenfor å hardkode testene om og om igjen med forskjellig data inputs.

3.4 Ytelse

Ytelsen er bra.

4 TILLEGG TIL QTESTLIB

QTestLib har en god dokumentasjon som gjør det enkelt for oss å sette oss inn i og bruke det effektivt.

5 VIDERE LESING

5.1 EKS. BRUKERVEILEDNING

QTestLib Manual, som ligger på Qts hjemmesider har en strålende gjennomgang av hvordan man bruker deres grensesnitt. De går gjennom fem kapitler hvor de starter enkelt i kapittel en, og går mer inn i detalj utover kapitlene. Dette ligger tilgjengelig på:

http://developer.qt.nokia.com/doc/qt-4.8/qtestlib-tutorial1.html

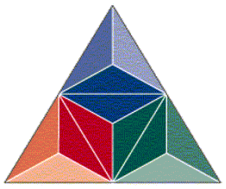
6 REFERANSER

1. Nokia. *Nokia.* [Internett] [Sitert: 19 12 2011.] http://developer.qt.nokia.com/doc/qt-4.8/qtestlib-manual.html.

2. Nokia. *Nokia.* [Internett] [Sitert: 19 12 2011.] http://developer.qt.nokia.com/doc/qt-4.8/qtestlib-tutorial1.html.

ANALYSEDOKUMENT

HIBU STUDENTPROSJEKT 2012
CYTRAX DB FACTORY



HØGSKOLEN
i Buskerud

cytrax

REVISJONSHISTORIKK		
Versjon	Dato	Endringer
1.0	8.2.2012	Første utgave
2.0	21.3.2012	Oppdatert med nye og/eller oppdaterte analysedokumenter
2.1	28.5.2012	Siste utgave Mindre endringer Korrekturlest

INNHOLDSFORTEGNELSE

1 INTRODUKSJON

I forbindelse med oppgaven fra Cytrax AS er det behov for å analysere og se på en rekke aspekter ved oppgaven. Applikasjonen som skal utvikles vil bli utviklet i flere stadier, hvor grunnfunksjonalitet og høyst prioriterte ønsker fra Cytrax AS blir analysert og designet først.

1.1 FORKORTELSER OG SYNONYMER

Forkortelse	Beskrivelse
Tabell	Table(database) – engelsk
QT	Et kryss-plattform rammeverk for C++
API	Application Programming Interface – Kildekodebasert spesifikasjon (wikip.)
Dokumentmodell	Representerer all informasjon applikasjonen benytter, blant annet databasemodellen, informasjon om generering, osv.

2 ANALYSE AV DOKUMENTMODELLEN

I applikasjonen blir innlest database lagret i en dokumentmodell. For at modellen skal være funksjonell må de essensielle egenskapene til databasen analyseres og dokumenteres. Egenskapene blir i dette dokumentet analysert og kommentert med tanke på applikasjonen.

2.1 FORFATTER(E)

- Vegard Kaasin(VK)
- Simen Russnes(SR)
- Jørgen Grøndal(JG)

2.2 DOKUMENTMODELL

Vi deler opp egenskapene i dokumentmodellen til 5 forskjellige nivåer for å ha oversikt over hvor egenskapene skal ligge. Ved tilkobling til database vil plugin evt. hente ut alle egenskaper fra databasen og lagre dette. Under følger nivåer med eksempelparametere.

Project-level	Beskrivelse
ProjectName	Prosjekt navn
CompanyName	Firma navn
DeveloperName	Utviklerens navn
Description/Comment	Beskrivelse av prosjekt
LastImport	Dato for forrige import

LastWritten	Dato for forrige generering
Date	Dato

Plugin-level	Beskrivelse
PluginName	Plugin navn
DatabaseType	Hviklen database(eks. «SQL ANYWHERE»)
Export Language	Hvilke plugins som ble brukt for å generere kode
Import Language	Hvilken plugin som ble brukt for å lese inn databasen

Database-level	Beskrivelse
Type	Sql, oracle osv.
Name	Databasens navn
Size	Størrelse
Driver	Driver
Server	Server
Username	Brukernavn
Password	Passord
Port	Port
TableSorting	Sortering av Tabeller/Views
Connection string	String for å connecte mot databasen
Connection timeout	Timeout for tilkoblingen
ConnectionType	Tilkoblings type
AutoCommit	
CharacterEncoding	Encoding på characters i databasen
Table	Beskrivelse
ColumnSort	Sorteringsmetode
Indexes	ID osv
Triggers	Oppdatere i en tabell trigger en oppdatering i en annen.
Column Sort	Sortering av kolonne
TableName	Navnet på tabell
TableIncluded	True/False
ExportGenerator	Navn på generator
Column	Beskrivelse
ColumnName	Kolonnenavn
Constraints	Eks: Mellom 1 og 10, eller må være true/false
Datatype	Int, varchar osv.
DatatypePrecision	Presisjon på datatypen
ColumnIncluded	Bool; True/False
DatatypeLenght	Varchar(60)

Characterencoding	Utf-8 etc
AutoIncrement	
Key	Beskrivelse
Primarykeyname	Navnet på primarykey
Foreign keyname	Navnet på foreign key
ReferencedTable	Foreignkey: hvilke tabell er det referert til
ReferencedColumn	Foreignkey: hvilke kolonne er det referert til
AllowNullValue	Tillat nullverdi
Match type	Simple/full
UpdateRule	Not permitted, Cascade values, SetValues to null/default
DeleteRule	Sammesom over
CheckOnlyOnCommit	

Template-level	Beskrivelse
PropertyName	
PropertyValue	
XMLTag	
DefaultValue	
DocumentPropertyList	

Typemap-level	Beskrivelse
Typemap	Standard typemap
CustomTypemap	Egendefinert typemap

2.3 GENERALISERING AV EGENSKAPER I DOKUMENTMODELLEN

Når vi representerer data og egenskaper i dokumentmodellen ønsker vi å ha en generell måte å gjøre det på. Vi har derfor utviklet to template-klasser til det formålet. Den ene klassen kalles DocumentProperty, og gir mulighet til å lagre verdien, navnet og en tilhørende XML-tag til en hvilken som helst egenskap. Siden det er templatebasert kan man lagre hvilken som helst type variabel, så det kan være alt fra en enkel verdi, eller en hel tabell.
Det skal også være mulig å gjenopprette den originale verdien til alle lagrede verdier i systemet, så DocumentProperty vil også ha en variabel for lagring av dette.

Den andre klassen vil fungere som en liste hvor man kan lagre et sett med slike egenskaper. Dette vil blant annet gi oss mulighet til å legge til uforutsette egenskaper i en generell liste, for eksempel for database-egenskaper.
Vi enkapsulerer en QList og definerer et sett tillatte funksjoner for håndtering av listen. Vi ser for oss at dette er en klasse som blir arvet fra, og brukt til å for eksempel representere et sett med tabeller, kolonner, egenskaper eller en annen type data.

Vi har også en klasse vi kaller "Additional Properties" som inneholder en slik liste hvor man kan legge inn n antall uforutsette egenskaper. Dette kan være nyttig når man leser fra forskjellige databaser og man ikke vet nøyaktig hvilke egenskaper hver forskjellig databasetype har. Klasser som inneholder en slik "Additional Property" vil for eksempel være "Databaseinformasjons-klassen".

2.4 Endre egenskaper i dokumentmodell

Vi vil gjøre det mulig å kunne endre egenskapene i dokumentmodellen. Dette gjøres ved at vi innfører en default verdi i template-klassen som settes når prosjektet starter og når databasen blir lest inn. Denne verdien kan ikke endres etter den først er satt.

2.5 Egenskaper ved forrige generering

Når vi skal generere kode fra en dokumentmodell det nyttig å vite detaljer om siste generering av kode og innlesing av database. Disse egenskapene blir hentet frem fra den lagrede dokumentmodellen og vist frem når prosjektet blir lest inn.

2.6 Flere generatorer pr tabell

Det skal være mulig å ha flere generatorer pr tabell, det vil si det skal være mulig å generere kode til C++ for 5 tabeller, og generere kode til vb.net til de samme 5 i tillegg til 5 andre. Dette kan gjøres ved at vi innfører en egenskap i databasemodellen, under table, som har en referanse til alle generatorer som skal generere kode.

3 Analyse av innlesing via plugin

Vi ønsker å analysere bruken av plugins i forbindelse med å lese inn en databasemodell, og i den sammenheng vil vi undersøke hva en plugin skal gjøre.

3.1 Forfatter(e)

- Jarle Didriksen (JD)
- Tor-Christian H. Eriksen (TCE)

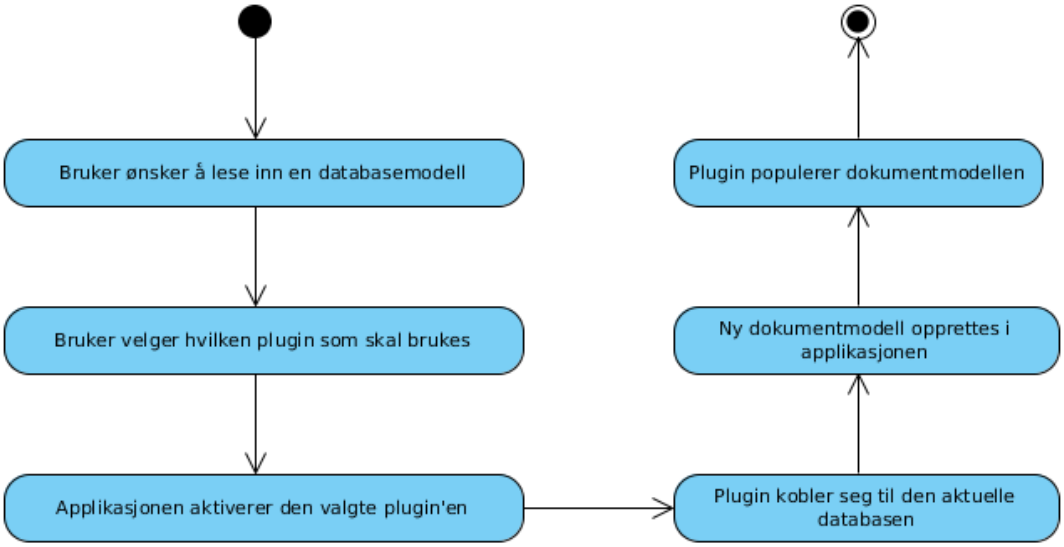
3.2 Diagrammer og utredninger

En plugin er et sett med softwarekomponenter som tilfører funksjonalitet i et større program. Plugins kan som regel ikke kjøre på egenhånd siden de vanligvis avhenger av tjenester i hovedprogrammet.

Vi ønsker å bruke plugins for å lese inn en databasemodell for programmet vårt. Programmet skal støtte flere SQL-dialekter, og det er derfor viktig at plugins kan skille mellom de forskjellige typene. Det blir utviklet en plugin for hver dialekt.

Plugin har som rolle å koble til databasen og hente ut nødvendig informasjon. Dette tilbyr plugin-interfacet som virtuelle metoder, og det er opptil utvikleren av plugin å tilpasse disse metodene for databasetypen den skal brukes mot. I tillegg skal plugin populere dokumentmodellen for denne databasen.

Et aktivitetsdiagram viser sekvensen for dette slik vi ser det for oss i figur 1.



Figur 1: Aktivitetsdiagram

4 Analyse av API for innlesing

For å kunne utvide applikasjonen ved et senere tidspunkt skjer innlesing av databasestrukturen via en plugin. For hver databasedialekt har man behov for en egen plugin. Et API gjør det lettere å skrive en utvidelse ved et senere tidspunkt, slik at samme prosedyre blir fulgt. En ekstern person, for eksempel en bruker av applikasjonen, skal ha mulighet til å lage en plugin for en spesiell databasedialekt. I dette tilfellet er det en ansatt hos Cytrax AS.

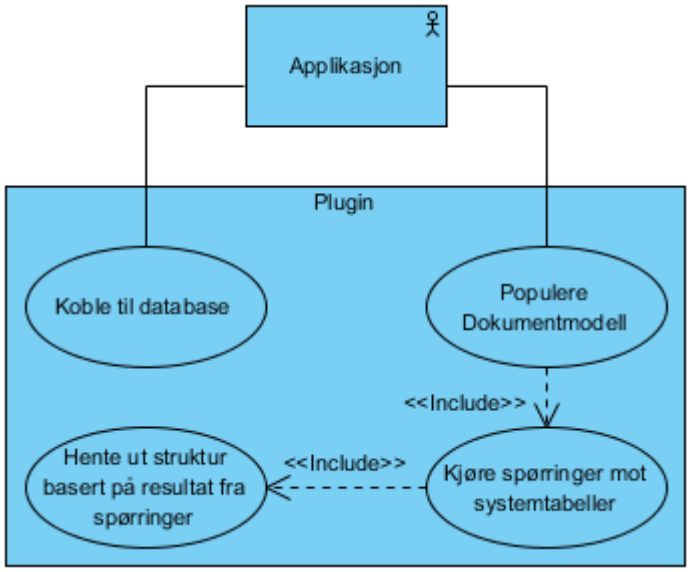
4.1 Forfatter(e)

- Tor-Christian H. Eriksen (TCE)

4.2 Diagrammer og utredninger

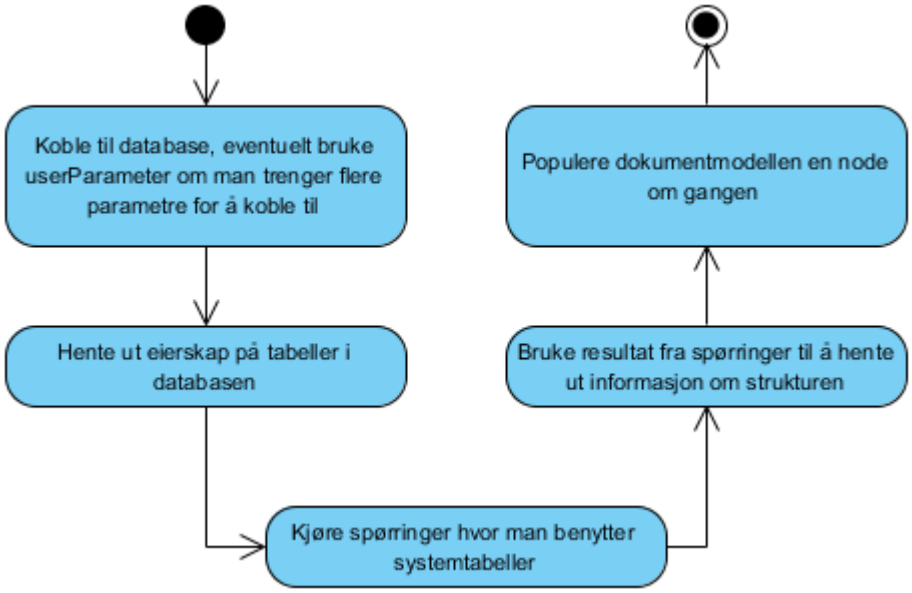
API'et skal gi en beskrivelse av funksjonalitet og oppbygning av pluginen. For å unngå å måtte oppdatere og rekompilere plugins ved en eventuell oppdatering av dokumentmodellen er detforsøkt å holde API'et så enkelt som mulig. Ved større

endringer i dokumentmodellen (endringer i definisjonsfiler) må man rekompilere plugins.



Figur 2: Use Case diagram

Når man benytter API'et til å skrive en plugin er det ment at pluginen skal benyttes i en prosess ved innlesing av en databasestruktur. Denne starter ved å opprette en tilkobling til databasen. Deretter lager man en eller flere spørringer hvor man benytter systemtabellene i databasen. Til slutt hentes ut strukturen basert på resultatet fra foregående spørringer. Det er tenkt at alt en bruker av applikasjonen behøver å gjøre er å velge korrekt plugin, og oppgi nødvendig informasjon for å koble til databasen.

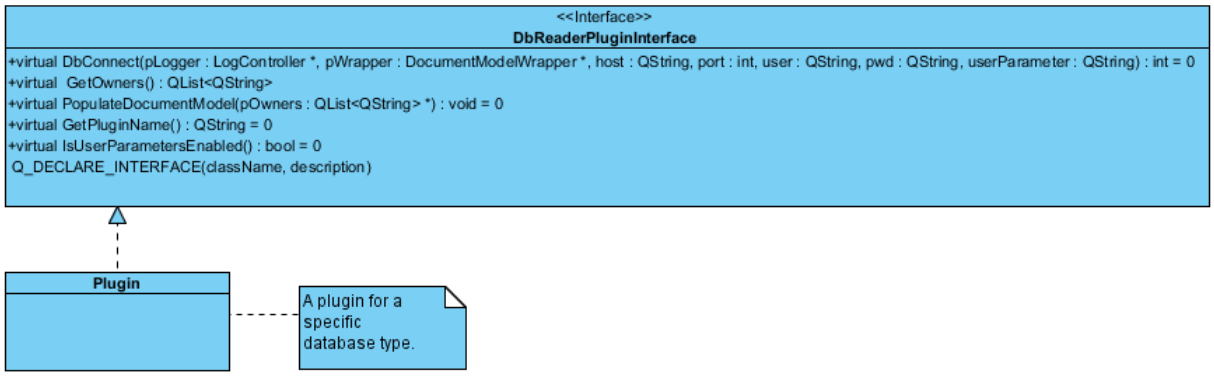


Figur 3: Aktivitetsdiagram

Man er nødt til å implementere funksjonene i interfacet, og benytte disse for å sette opp tilkobling, populere dokumentmodellen og sette et navn på pluginen. Utover dette så står utvikleren fritt til å implementere egne metoder og funksjonalitet.

Det er opp til utvikleren å tilføre nødvendige operasjoner for å hente ut databasestrukturen og eventuelt annen informasjon.

4.3 BESKRIVELSE AV METODER



Figur 4: Klassediagram

Man har muligheten til å håndtere feil ved åthrowe et objekt av typen ErrorHandler. For mer informasjon om feilhåndtering kan man se i dokument om feilhåndtering (1).

Q_DECLARE_INTERFACE er nødvendig for at QT skal kunne identifisere header filen som et interface.

4.3.1 DbConnect()

Metoden benyttes for å koble opp mot databasen, og må kalles før `GetOwners()` og `PopulateDocumentModel()`. Den inneholder parametere for å sette adresse, port, brukernavn og passord.

Om man trenger flere parametere når man kobler seg opp mot databasen kan man benytte parameteren `QStringuserParameter`, og parse den selv i plugin. Man må huske på å returnere `true` i `IsUserParametersEnabled()`.

4.3.2 GetOwners()

Brukes for å hente ut eierskap på tabeller i databasen og skal returnere en liste over alle eiere i form av en `QList<QString>*`.

4.3.3 PopulateDocumentModel()

PopulateDocumentModel benyttes for å lese ut nødvendige tabeller og legge inn data i dokumentmodellen. All funksjonalitet bør lages i denne metoden, men om ønskelig står utvikleren fritt til å lage flere metoder. Alle ekstra metoder må kalles på gjennom `PopulateDocumentModel()`. Applikasjonen kaller kun på de virtuelle metodene i interfacet:

- DbConnect()

- GetOwners()
- PopulateDocumentModel()
- GetPluginName()
- IsUserParametersEnabled()

4.3.4 GetPluginName()

Returnerer navnet på pluginen i form av en QString.

4.3.5 IsUserParametersEnabled()

Applikasjonen bruker denne parameteren for å sjekke om man kan fylle inn UserParameters i GUI før DbConnect() kalles.

5 ANALYSE AV PLUGIN VED KODEGENERERING

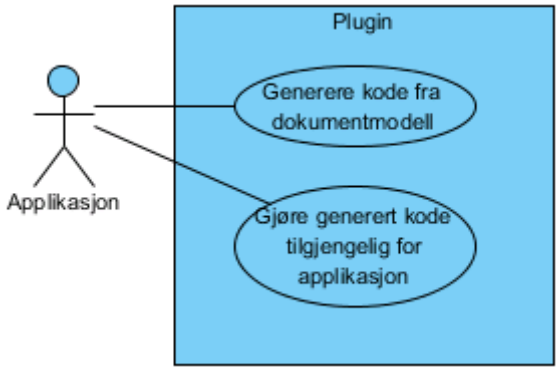
For å få støtte for plugin ved generering av kode er det ønskelig å få definert et interface som beskriver hvilken oppførsel alle plugin som genererer kode skal ha. Dette henger tett sammen med prosessen som beskriver hva som skal utføres ved generering av kode. Som utgangspunkt for å definere API'et ser man for seg hvordan prosessen for generering av kode foregår.

5.1 FORFATTER(E)

- Jørgen Grøndal

5.2 DIAGRAMMER OG UTREDNINGER

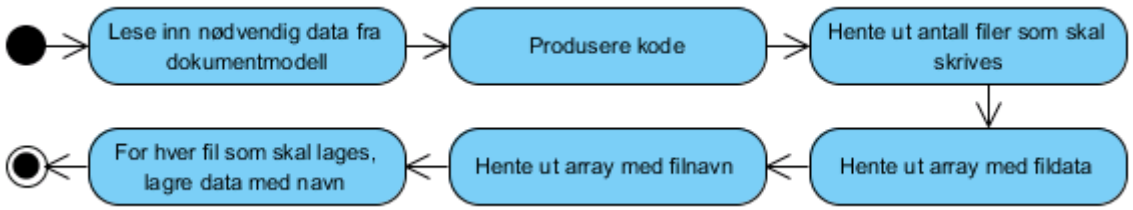
For å kunne se for oss hva en plugin for generering skal gjøre for applikasjonen har vi valgt å bruke use-case diagram.



Figur 5: Use-case diagram som viser hva plugin gjør for applikasjonen

Plugininterfacet må definere hvilken oppførsel en implementert plugin skal ha for å utføre funksjonaliteten beskrevet i figur 5.

Etter analyse via use-case har vi laget et aktivitetsdiagram som beskriver en tenkt rutine for generering av kode. Det vil si hvilke operasjoner vi ønsker at applikasjonen skal be plugin utføre.



Figur 6: Aktivitetsdiagram som viser hvilke operasjoner applikasjonen ønsker at plugin utfører

6 ANALYSE AV TYPEMAPPING

Generering av databasekode til Vb.net eller C++ må støtte typemapping mellom databasetype og kodespråk. Siden det er ønsket at denne prosessen skal være automatisk, samt ønske om å endre spesifikke datatyper må vi analysere hvordan et slikt system skal se ut.

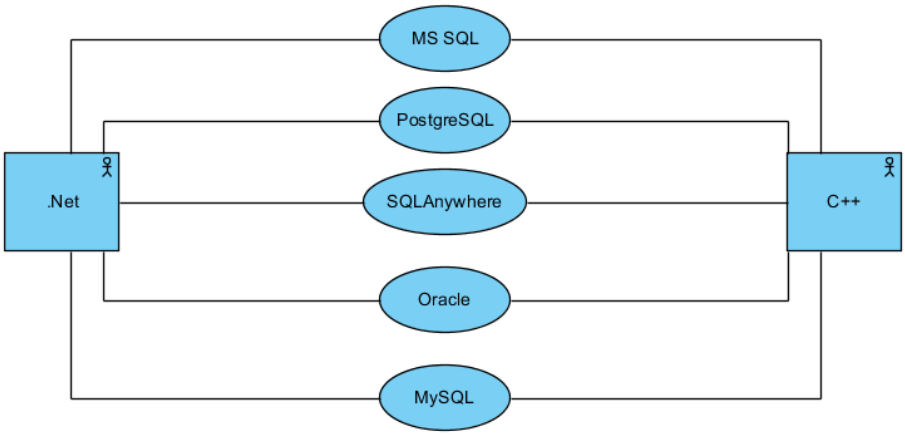
6.1 FORFATTER(E)

- Vegard Kaasin (VK)

6.2 DIAGRAMMER OG UTREDNINGER

Typemapping-systemet vi skal bruke lages på bakgrunn av at det benyttes ulike datatyper i en database og i andre kodespråk(VB.net, C++). For eksempel benytter en SQL-Anywhere database seg datatypen bit, mens Visual Basic og C++ vil bruke boolean/bool. Prosessen med å mappe datatyper mot hverandre vil vi at skal gå automatisk, dermed må sett med mapping mellom databasetyper og kodespråk defineres. I tillegg til automatisk mapping av datatyper vil vi gjøre det mulig å endre manuelt datatypen for en spesifikk kolonne.

Typemapping-systemet skal bestå av flere typer språk som hver støtter mapping mot alle ønskelige databasetyper.



Figur 7: Modell av tenkt system

6.3 TYPEMAPPING-SYSTEM

Typemapping-systemet vil bestå av en forhåndsdefinert XML-fil. Når det genereres kode vil programmet hente all informasjon om typemapping og lagre den. Det skal også være mulig å endre typemappingen for generering. Dette vil være aktuelt for to caser:

- Endre Typemapping for alle kolonner og tabeller/views.
- Endre Typemapping for spesifike kolonner.

6.4 DOKUMENTMODELL

For å endre typemapping for alle kolonner blir endrede datyper lagret i dokumentmodellen under «typemapping» deretter under hva slags generator som skal brukes og hvilke språk det skal genereres kode til. Programmet vil sjekke om det finnes unntak i typemapping ved generering.

For å endre typemapping spesifikt for en kolonne vil en egenskap «mMappedDatatype» bli satt til å vise hvilken datatype som skal endres og hva den skal mappes til samt hvilke database og hvilke språk datatypen hører til.

7 ANALYSE AV GUI

Dette avsnittet skal avdekke hvordan vi har gått frem for å analysere hva det grafiske brukergrensesnittet i applikasjonen skal gjøre.

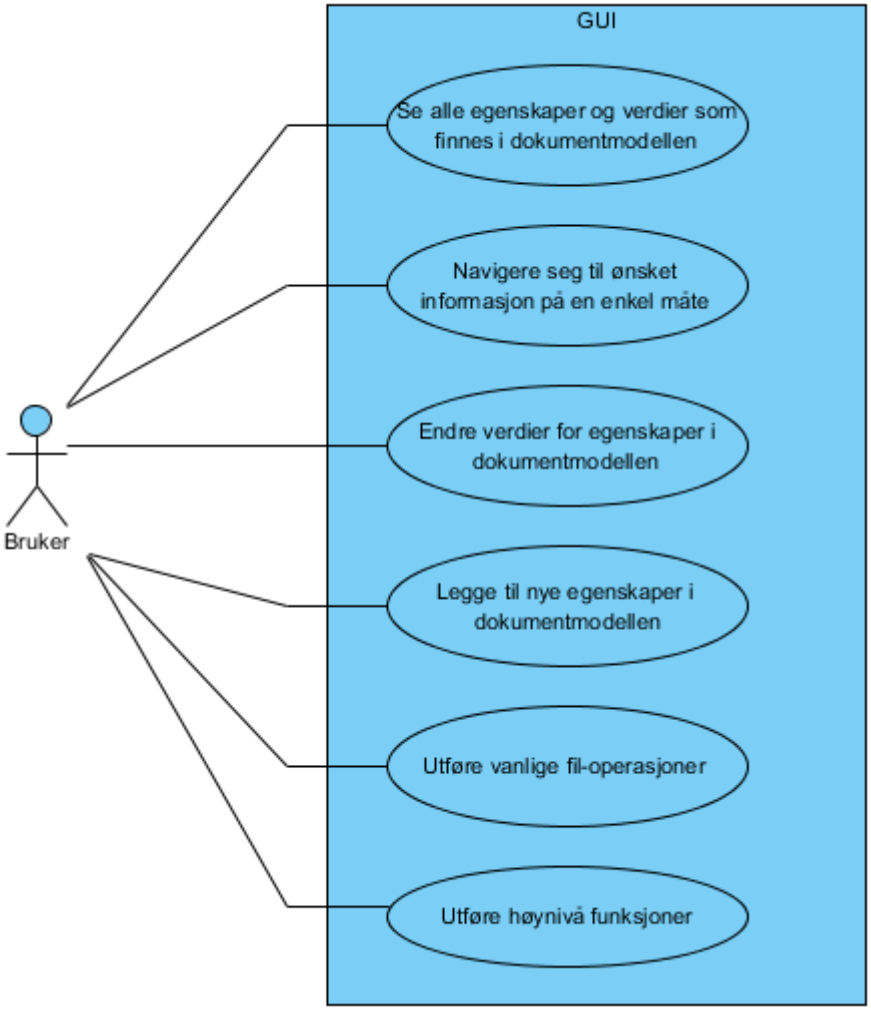
7.1 FORFATTER(E)

- Jørgen Grøndal

7.2 UTREDNINGER OG DIAGRAMMER

Hensikten med å ha et grafisk brukergrensesnitt (GUI) er å tilby brukeren av applikasjonen en intuitiv og effektiv metode å manipulere og få presentert relevant data på.

Å bestemme hvilke elementer som skal vises i GUI, og senere også på hvilken måte er en vanskelig prosess. Viser man for mange kan det bli vanskelig å se relevant informasjon og vice versa.



Figur 8: Use-case diagram som viser hva GUI skal gjøre for brukeren

Fra figur 1 kan vi utlede flere viktige hovedpunkter vi må tenke på. Disse er:

- Hvilke data vi skal vise for bruker
- Logisk gruppering av data sånn at navigering blir enkelt og intuitivt
- Hvordan vi skal la bruker endre verdier / egenskaper
- Hvordan vi skal la bruker legge til egenskaper
- Hvordan skal bruker kontrollere applikasjonen (Fil-operasjoner)

- Hvor bruker skal utføre høynivå funksjoner fra

7.2.1 Hvorfor lage et bra GUI

For å lage et bra grafisk brukergrensesnitt er det mange elementer man må vurdere. De vanligste feilene man gjør når man utvikler GUI (2) er som følger:

- Glemme brukeren
- Legger restriksjoner for brukeren ved å aktivere og deaktivere elementer
- For mye funksjonalitet og informasjon på høyeste nivå

Dersom man klarer å unngå disse er man allerede på god vei mot et godt GUI.

Vi skal liste opp design prinsipper vi ønsker å følge når vi utarbeider design og layout for GUI i applikasjonen.

- Være konsis og tydelig i hele applikasjonen. Dersom noe kan misforstås, iterer om elementet igjen
- Visuelle tilbakemeldinger for brukeren. For eksempel en bruker venter på en operasjon. Da kan man gi brukeren status i form av % eller lignende.
- Tekst skal være kort og konsis. Tenk enkelt.
- Sporbarhet. En bruker skal vite hvordan man havnet et sted. F.eks. om man er i en dialog burde man vite hvor man trykket for å få den frem.
- Ha støtte for tastatursnarveier. F.eks. Bruke CTRL + Z for å angre på en handling o.l.
- Være konsise i bruk av kontroll- og menyelementer.
- Ikke ha for mange elementer i menyer eller undermenyer.

7.2.2 Data som skal vises for bruker

Det viktigste er å finne ut hvilke data i dokumentmodellen vi skal vise frem og på hvilken måte. Stort sett all data skal være tilgjengelig, alle elementene må derfor prioriteres og grupperes. Deler av informasjon er detaljert og trenger ikke være synlig til enhver tid.

Ut i fra kunnskap vi har på nåværende tidspunkt har vi gjort følgende vurdering av data i dokumentmodellen:

7.2.2.1 Primærinformasjon

De elementene som vi vurderer som viktigs å være enkelt tilgjengelig er som følger:

- Tabeller
- Prosjektinformasjon
- Databaseinformasjon

Utover dette er det også to elementer til som er viktig, men som vi antar vil bli mindre brukt. Disse elementene er:

- Generatorinnstillinger

- Hvordan databasetyper skal mappes til variabler i forskjellige kodespråk (Typemapping)

Man kan tenke seg at de viktigste elementene vil være i form av "hele" sider som kan navigeres til ved hjelp av en hovedmeny. Det betyr at det i hovedsak at det skal være ét klikk i GUI før man får tilgang på informasjonen man søker.

7.2.2.2 Sekundærinformasjon

Mye av dataene som finnes i dokumentmodellen vil man ikke kunne finne uten å vite hvor det er lokalisert. På grunn av dette definerer vi det som sekundærinformasjon. I sammenheng med GUI betyr dette at det vil gi lite mening å presentere sekundærinformasjon uten at man spesifikt velger dette.

Den informasjonen vi har definert som sekundærinformasjon er som følger:

- Tabellegenskaper
- Kolonner
    - Kolonneegenskaper
- Generatorinnstillinger for en spesifikk generator
- Utvidet typemapping informasjon

For å få tilgang til disse dataene er man altså nødt til å navigere til korrekt primærinformasjon.

7.2.2.3 Hvilke operasjoner en bruker skal utføre

Som vist i figur 6 er operasjoner en bruker ønsker å utføre knyttet til modifisering av dokumentmodellen.

Operasjoner på dokumentmodell:

- Legge til tabell med kolonner og egenskaper
- Endre tabell, kolonner og egenskaper
- Endre prosjektinnstillinger
- Endre databaseinnstillinger
- Legge til databaseinnstillinger
- Endre genereringsinnstillinger

Operasjoner på prosjektfil:

- Ny
- Åpne
- Lagre
- Lagre som
- Sammenligne med annen prosjektfil

Operasjoner for importering: (Innlesing av databasemodell)

- Les inn database (Med tilhørende dialog for å sette innstillinger)

- Les inn database for sammenligning (Sammenligne med db i dokumentmodell)

Operasjoner for eksportering: (Generering av kode)

- Generer kode (Med tilhørende dialog for standard valg, som destinasjonsfolder)
- Velge generator(er)

8 REFERANSER

1. **DB Factory.***Feilhåndtering.* s.l. : DB Factory, 2012.

2. **Hobart, James.** Principles of good GUI design. *ClassicSys.* [Internett] [Sitert: 9 Februar 2012.] http://www.classicsys.com/css06/cfm/article.cfm?articleid=20.

3. Nokia. [Internett] [Sitert: 26 Januar 2012.] http://developer.qt.nokia.com/doc/qt-4.8/sql-types.html.

4. **briggs, Kim.** MySQLdatatypes . *kim briggs website.* [Internett] [Sitert: 14 03 2012.] • http://kimbriggs.com/computers/computer-notes/mysql-notes/mysql-data-types.odt .

5. **Oracle.** Datatypes. *Oracle.com.* [Internett] Oracle. [Sitert: 14 03 2012.] http://docs.oracle.com/cd/B19306_01/server.102/b14200/sql_elements001.htm.

6. **Sybase.** Data types. *sybase.com.* [Internett] Sybase. [Sitert: 14 03 2012.] • http://manuals.sybase.com/onlinebooks/group-pbarc/conn5/sqlug/@Generic__BookTextView/33273;pt=33273;uf=0.

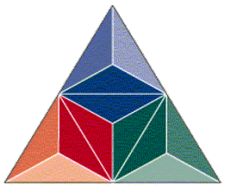
7. **Sybase15.** Sybase datatypes. *Sybase15.* [Internett] [Sitert: 14 03 2012.] http://www.sybase15.com/datatypes.

8. **Lalonde, Brian.** MsSQL datatypes. *webcoder.* [Internett] Webcoder, 23 06 2004. [Sitert: 14 03 2012.] http://webcoder.info/reference/MSSQLDataTypes.html.

9. **Postgresql.** Postgresql. *postgresql.* [Internett] postgresql. [Sitert: 14 03 2012.] http://www.postgresql.org/docs/7.4/static/datatype.html.

DESIGNDOKUMENT

HIBU STUDENTPROSJEKT 2012
CYTRAX DB FACTORY



HØGSKOLEN
i Buskerud

cytrax

REVISJONSHISTORIKK		
Versjon	Dato	Endringer
1.0	8.2.2012	Første utgave
2.0	24.3.2012	Andre utgave, oppdaterte/nye dokumenter
2.1	28.5.2012	Siste utgave Mindre endringer Korrekturlest

INNHOLDSFORTEGNELSE_Toc326075876

1 INTRODUKSJON

Designdokumentet er neste steg mot implementasjon. Punkter og oppdagelser i analysedokumentet er basis for designet. Designet brukes for å gjøre utviklingen av applikasjonen enklere og for å unngå problemer som kan oppdages på forkant. Ut ifra analysen designes løsninger på problemer slik som vi oppfatter dem.

Avsnittene som er kalt **Forfatter(e)** forklarer hvem som har vært med på designet. Personer med fet skrift er hovedforfatter, og har hatt hovedansvaret. Personer med normal skrift har vært med på diskusjoner, gitt innspill, og/eller vært delvis med på designet.

1.1 FORKORTELSER OG SYNONYMER

Forkortelse	Beskrivelse
Dokument	All informasjon om databaseprosjektet lagres her.
QT	Et kryss-platform rammeverk for C++
API	Application Programing Interface – Kildekodebasert spesifikasjon (wikip.)
XML	eXtensibleMarkup Language
XML Element	Alt fra (og med) elementets start-tag, til (og med) elementets slutt-tag <mytag> this is my tag </mytag>
Klassediagram	Et diagram for oppbygging av et sett av klasser
MVC	Model View Controller

2 OVERORDNET ARKITEKTUR

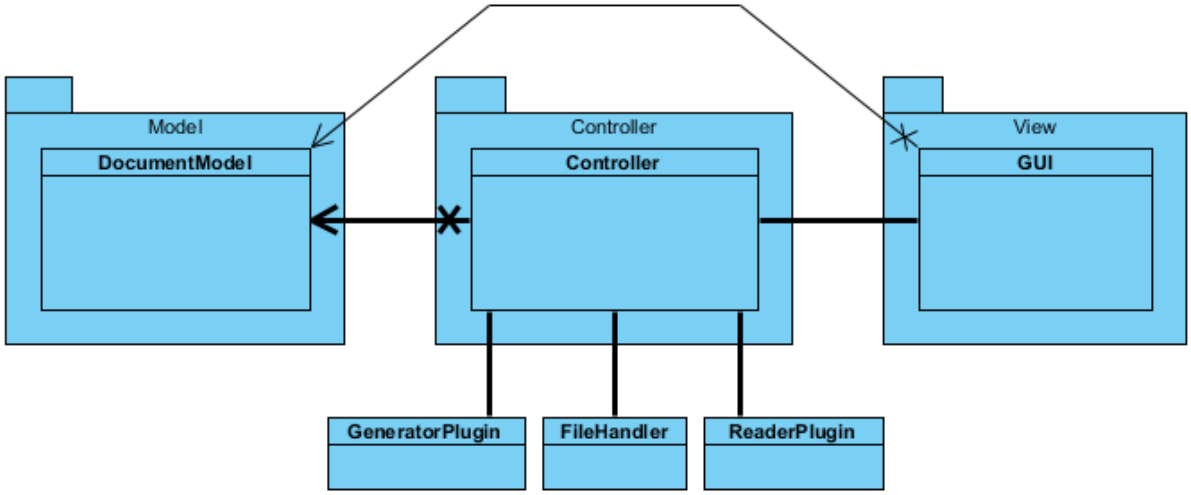
2.1 FORFATTER(E)

- **Jørgen Grøndal (JG)**

2.2 INTRODUKSJON

For enklere å kunne sette seg inn i hvordan applikasjonen er bygget opp har vi laget et diagram som viser overordnet arkitektur. Dette diagrammet generaliserer klassene til større og mer forståelige klasser, for eksempel består kontrolleren vår av mange kontrollere, men i dette diagrammet er det beskrevet som kun en klasse.

Som en del av designprosessen har vi bestemt oss for å følge Model-View-Controller-mønsteret. (1)

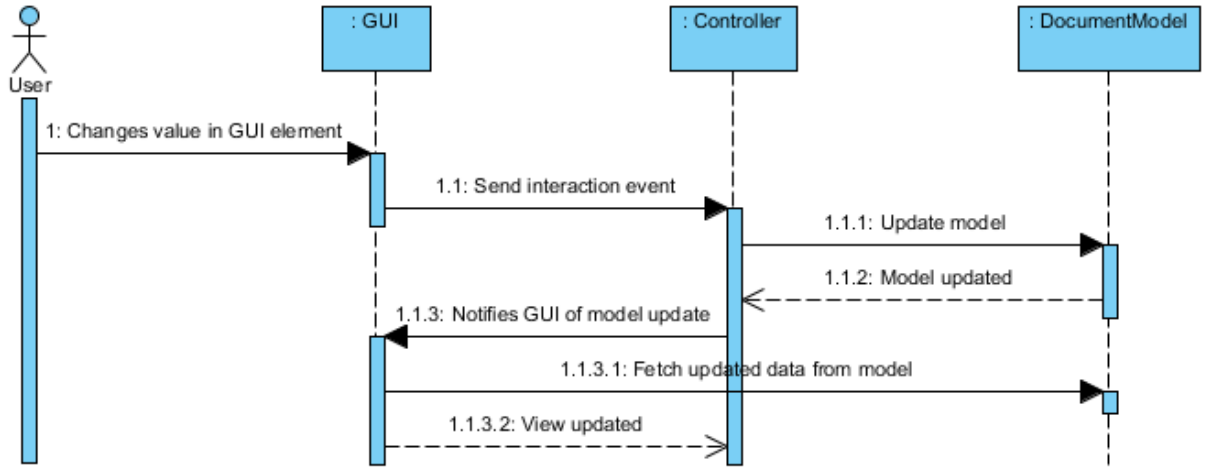


Figur 1: Klassediagram for overordnet arkitektur

Som man ser av klassediagrammet over er det identifisert nøkkelklasser som inngår i MVC-mønsteret. Dette har vi gjort for å kunne separere logikk fra datastruktur, og la GUI håndtere det den skal, nemlig hvordan data skal vises. Det ble derfor et naturlig valg å bruke MVC-mønsteret.

De viktigste oppgavene til systemet er å lagre data i dokumentmodellen, implementere korrekt logikk i kontrollerne og presentere relevant data på en god måte via GUI. Utover dette trenger applikasjonen funksjonalitet via plugins for både innlesing av database og generering av kode. Den siste funksjonaliteten er å kunne eksportere og importere til og fra prosjektfiler. Det vil si at man skal kunne lagre en komplett dokumentmodell som en xml-fil.

2.3 KOMMUNIKASJON I VÅR IMPLEMENTASJON AV MVC-MØNSTERET



Figur 2: Sekvensdiagram som viser hvordan vi skal implementere MVC-mønsteret

Som figuren over viser ser man at implementasjon av MVC-mønsteret er gjort på en mer passiv måte. Det mest vanlige er at modellen sender beskjed til viewet når en endring har

skjedd. Vi har kommet frem til at vi ikke har behov for dette, da alle endringer i modellen kun skjer via kontrolleren. Beskjed til viewet blir derfor sendt fra kontrolleren etter at endring i modellen har skjedd.

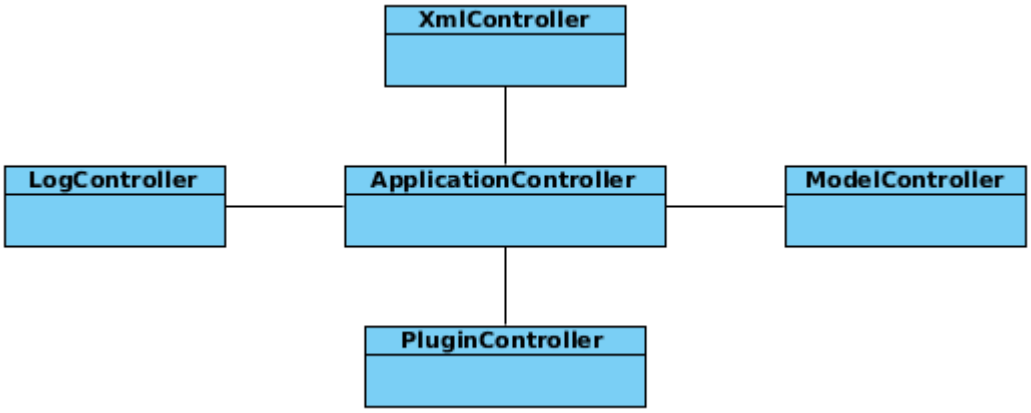
3 DESIGN AV KONTROLLER

Kontrolleren i MVC-mønsteret skal fungere som en form for navigatør. Det vil si at den tar imot handlinger som forekommer i GUI og utfører korrekt logikk i henhold til utført handling. Det er mye funksjonalitet som GUI skal ha tilgang på via kontrolleren. Oppdeling av kontrolleren ble derfor naturlig, med en applikasjonskontroller på toppnivå. Fra applikasjonskontrolleren får man tilgang på de mer spesialiserte kontrollerne som utfører de ønskede handlingene.

3.1 FORFATTER(E)

- **Jarle Didriksen (JD)**
- **Jørgen Grøndal (JG)**

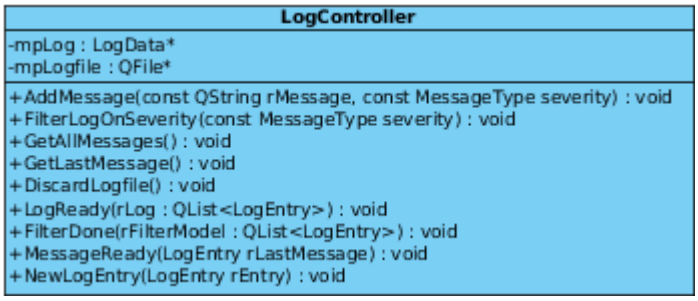
3.2 DIAGRAMMER OG UTREDNINGER



Figur 3: Design av kontroller – Overblikk av kontrollere

I programmet vårt har vi totalt fem kontrollere som har forskjellige ansvarsområder. Dette kan være alt ifra filbehandling til databasetilkobling.

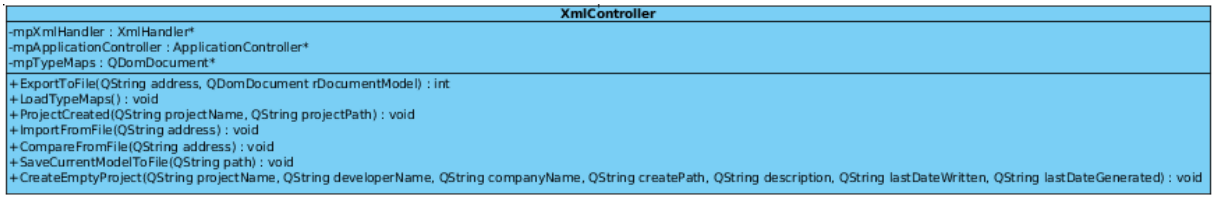
3.2.1 LogController



Figur 4: Design av kontroller – Klassediagram for LogController

LogController tar imot alle beskjeder fra applikasjonen som skal loggføres og signaliserer GUI-et om at det skal hente meldingen. Videre utredning om logging og design rundt dette finnes i kapittel 13.

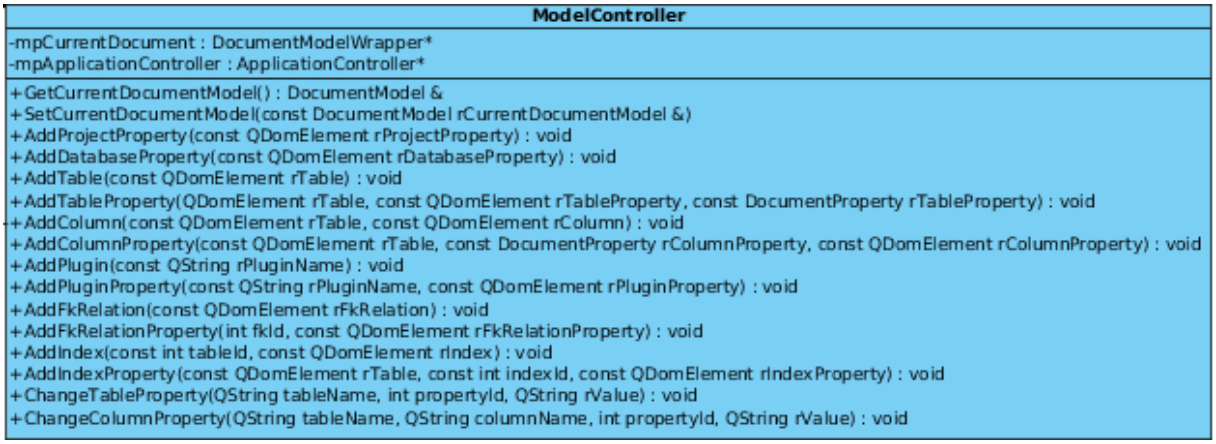
3.2.2 XmlController



Figur 5: Design av kontroller – Klassediagram for XmlController

XmlControllermottar signal om at det ønskes å lese inn eller skrive til en XML-fil, og overfører signalet videre til en klasse som håndterer dette. Videre utredning av håndtering av XML og design av dette finnes i kapittel 10.

3.2.3 ModelController

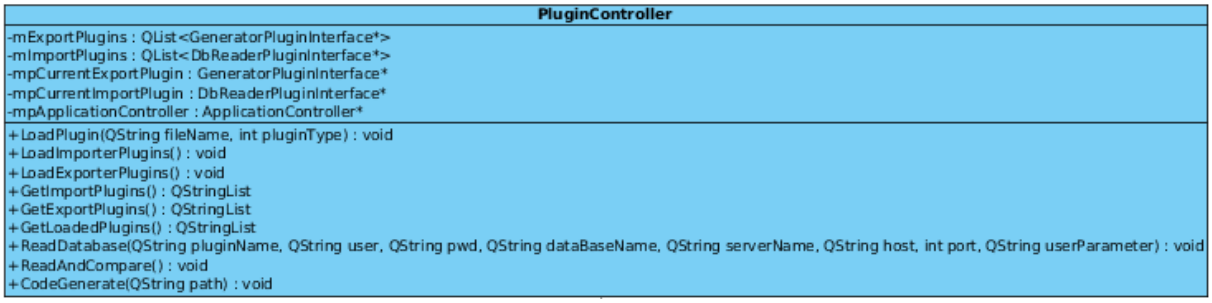


Figur 6: Design av kontroller – Klassediagram for ModelController

ModelController har ansvar for å holde dokumentmodellen oppdatert etter handlinger fra GUI-et. Det denne klassen gjør er i bunn og grunn å route signalet videre til

wrapperen vår som utfører de ønskede handlingene. I tillegg kan man endre egenskaper i denne kontrolleren.

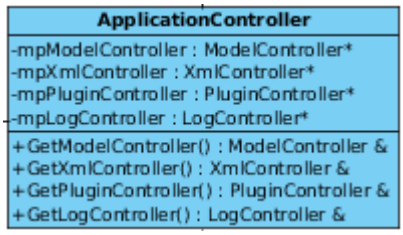
3.2.4 PluginController



Figur 7: Design av kontroller – Klassediagram for PluginController

PluginController har ansvar for å laste inn og initiere plugins, samt kjøre metodene til plugins. Dette innebærer å bruke metoder for å koble til og lese en database, og populere en dokumentmodell med informasjonen.

3.2.5 ApplicationController



Figur 8: Design av kontroller – Klassediagram for ApplicationController

ApplicationController er den kontrolleren som resten av programmet kommuniserer med for å snakke med de forskjellige kontrollerene. Hver kontroller kan aksesseres gjennom denne kontrolleren.

4 DESIGN AV DOKUMENTMODELL

For å representere data i et åpent prosjekt i applikasjonen har vi utviklet et design for et dokument (Dokumentmodell), og en enkapsulasjon for å håndtere dokumentet (DocumentModelWrapper).

Dokumentet lagrer og representerer data om prosjekt og om eventuelt innlest database for bruk i applikasjonen.

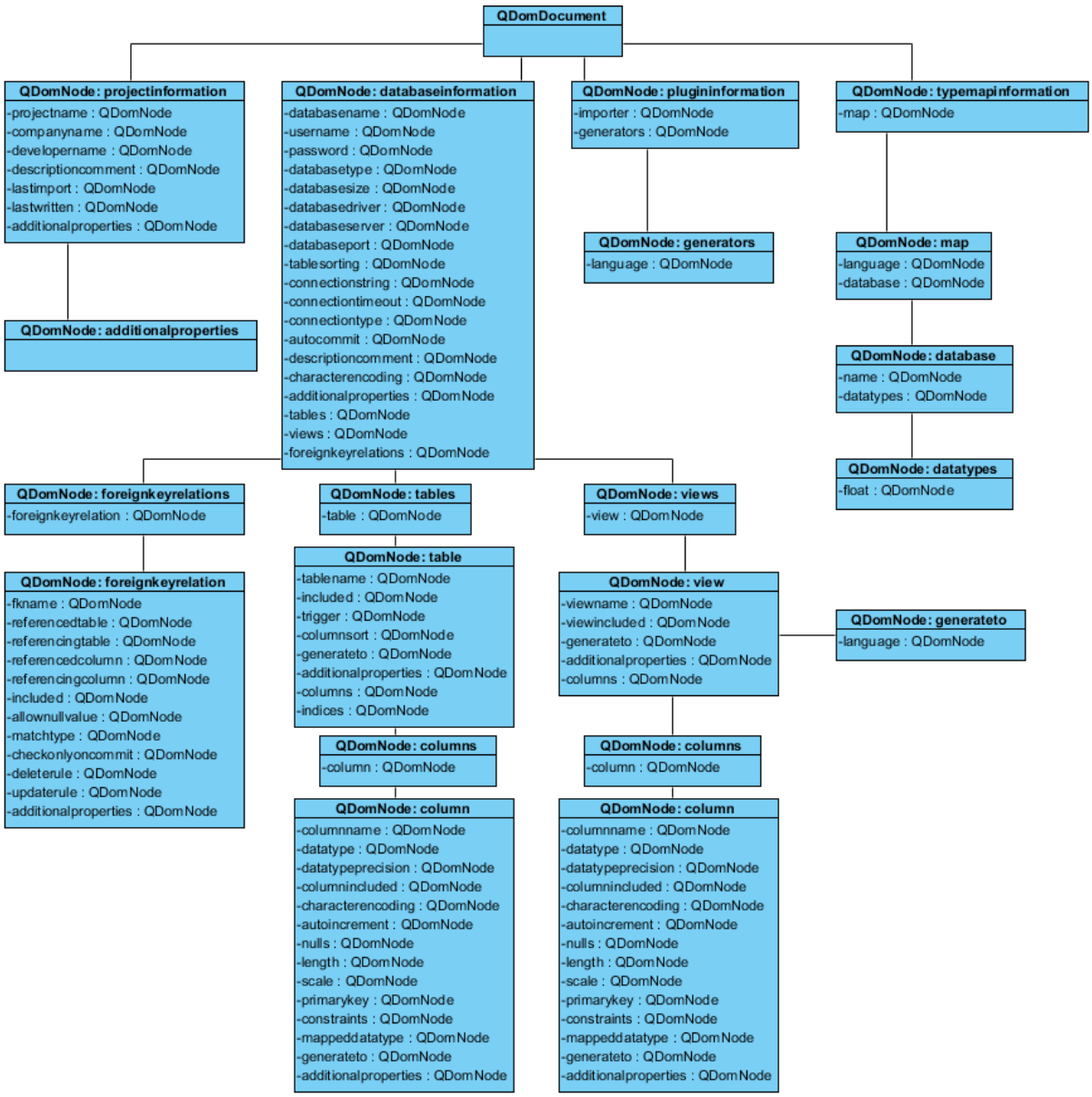
Under utviklingen av dokumentmodellen har vi designet selve strukturen, og implementert en enkapsulasjon i Qt for håndtering av data (DocumentModelWrapper). Siden vi lagrer prosjekter i XML var det naturlig for oss å ta i bruk en ferdig implementert klasse for håndtering av XML fra Qt (QDomDocument). Dette tillater oss å enkelt lese og skrive til XML, så lenge vi opprettholder den riktige strukturen underveis.

Wrapperen er derfor skreddersydd til vårt behov for å opprettholde nettopp det, og for å gjøre manipulasjon enklest mulig utover i applikasjonen uten å måtte tenke for mye over hierarkiet mens man behandler data.

4.1 Forfatter(e)

- **Simen Russnes(SR)**
- **Vegard Kaasin(VK)**
- Jørgen Grøndal (JG)

4.2 Diagrammer



Figur 9: Design av dokumentmodell-struktur

4.3 BESKRIVELSE AV DOKUMENTMODELL

Dokumentmodellen er der all data for prosjektet blir lagret. Dette er en struktur som gjør det enklest mulig å lagre, endre og å legge til nødvendig informasjon for en database i applikasjonen.

Dokumentmodellen består av fire deler:

1. **Informasjon om prosjektet**
   Dette er generell informasjon som beskriver et prosjekt. Navn på prosjektet, firmaets navn, utviklerens navn og en beskrivelse av prosjektet er hoveddelene, i tillegg til at man kan legge til ekstra informasjon dersom det skulle være ønskelig.

2. **Informasjon om Databasen**
   Her blir all informasjon om eventuell innlest database lagret.
   Informasjon om databasen blir delt inn i flere nivåer for å gjøre det hele mer oversiktlig.
   I det øverste nivået finnes det en generell beskrivelse av databasen, med informasjon som f.eks databasenavn, type, størrelse mm. Under dette nivået finnes det igjen tre nivåer som beskriver strukturen til innlest database. Dette er en liste over alle tabeller, views og foreignkeys.
   Hver tabell har generell informasjon om sine egenskaper, i tillegg til en liste over kolonner. Kolonnene består også av den samme type informasjon (se figur 9 for en komplett strukturbeskrivelse).
   Strukturen for views likner på den for tabeller ved at den også har en liste av kolonner og informasjon om sine egenskaper, men er noe mindre da den eneste generelle informasjonen inkludert er navn på view, og hvilke kolonner som inngår.
   Nivået for foreign keys inneholder en liste over alle foreign key relations for innlest database, hvor det vil bli lagret hvilken tabell og kolonne hver nøkkel refererer til, og hvor den blir referert fra. Det er også mulig å lagre informasjon om oppdateringsregler, hva som skjer når en nøkkel blir slettet samt annen generell informasjon for en fremmednøkkel.

3. **Informasjon om Plugins**
   I informasjon om plugins blir det lagret hvilke plugins som blir brukt i prosjektet. Man vil kunne lagre en plugin som den nåværende plugin for innlesning, og et sett med plugins for hvilke generatorer som har blitt lest inn og tatt i bruk.

4. **Informasjon om Typemapping**
   Dersom man skulle ønske å endre typemappingen for en gitt datatype for en database vil dette bli lagret her. Merk at dette vil kun bli tatt i bruk om man vil endre et typemap generelt for hele prosjektet, og ikke for en spesifikk kolonne (det vil bli lagret på kolonnenivå i informasjon om databasen). Om en bruker av applikasjonen skulle ønske å endre typemapping for en datatype for C++ i

SQLAnywhere12 lagres dette her ved å endre typemappinginstillinger i applikasjonen.

I figur 9 vises en komplett beskrivelse av strukturen for dokumentmodellen, hvor et QDomDocument her er rotnoden, og lister med QDomNoder videre representerer hierarkiet i modellen.

5 DocumentModelWrapper

For å kunne håndtere data i dokumentmodellen har vi utviklet en enkapsulasjon av et QDomDocument med støtte for all ønsket funksjonalitet i applikasjonen. Da QDomDocumentet er en veldig generell klasse fant vi ut at det var nødvendig å gi et sett med metoder spesielt for vårt tilfelle for å gjøre den videre utviklingen enklere.

5.1 Forfatter(e)

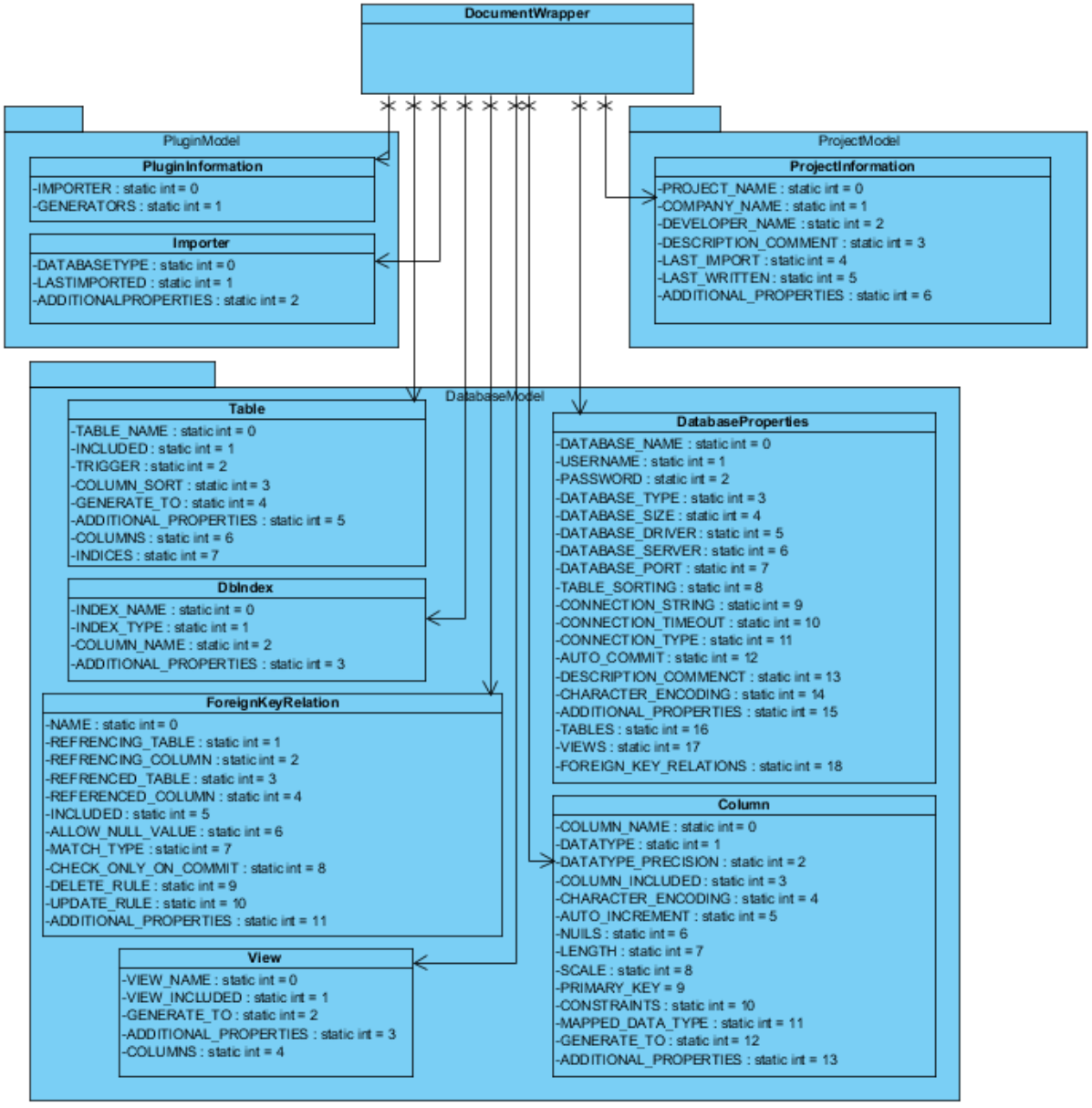
- **Jørgen Grøndal (JG)**
- **Simen Russnes (SR)**

5.2 DIAGRAMMER

Figur 10: Klassedigram for DocumentModelWrapper (del 1/2)

+ClearGenerateTo(QDomElement rParent)
+SetImporter(QString databaseType)
+AddGenerator(QString language)
+RemoveGenerator(QString language)
+AddView(QDomElement rView)
+AddViewProperty(QDomElement rView, QDomElement rProperty)
+AddMappedDatatype(QDomElement rColumn, QString language, QString datatype)
+GetViews()
+GetViewElement(QString viewName)
+Verify()
+SetStandardTypemap(QDomDocument pTypemaps)
+GetTypemap(QString codingLanguage, QString databaseName)
+ChangeStandardTypemap(QString codingLanguage, QString databaseName, QString databaseDatatype, QString newMappedDatatype)

Figur 11: Klassedigram for DocumentModelWrapper (del 2/2)



Figur 12: Indexer for å aksessere egenskaper i dokumentmodellen

5.3 BESKRIVELSE AV DOCUMENTMODELWRAPPER

Da vi har tatt i bruk den generelle datamodellen QDomDocument fra Qt for å enkelt kunne skrive og lese fra XML var det nødvendig å definere støtte for å utføre de handlingene vi ønsket å gjøre på dataen.

For å gjøre nettopp det har vi utviklet en enkapsulering av dokumentmodellen som vi har valgt å kalle "DocumentModelWrapper". Her tilbys en brukervennlig funksjonalitet for blant annet å legge til tabeller, kolonner eller views, endre datatype, eller annen informasjon. En liste over alle tilgjengelige funksjoner er vist i figur 10 og 11.

I tillegg finnes det en omfattende API-spesifikasjon i html-format som beskriver hver funksjon.

Da alle nivåer i dokumentet (f.eks kolonne-nivå eller tabell-nivå) vil bestå av et bestemt antall noder har vi i tillegg utviklet et sett med indexer for enkelt å aksessere de forskjellige elementene i prosjektet. Disse lar oss hente ut akkurat den noden vi ønsker uten å måtte søke gjennom en hel liste. En oversikt over alle indexene er vist i figur 12.

6 DESIGN FOR SAMMENLIGNING AV DOKUMENTMODELLER

6.1 FORFATTER(E)

- **Jørgen Grøndal (JG)**

6.2 INTRODUKSJON

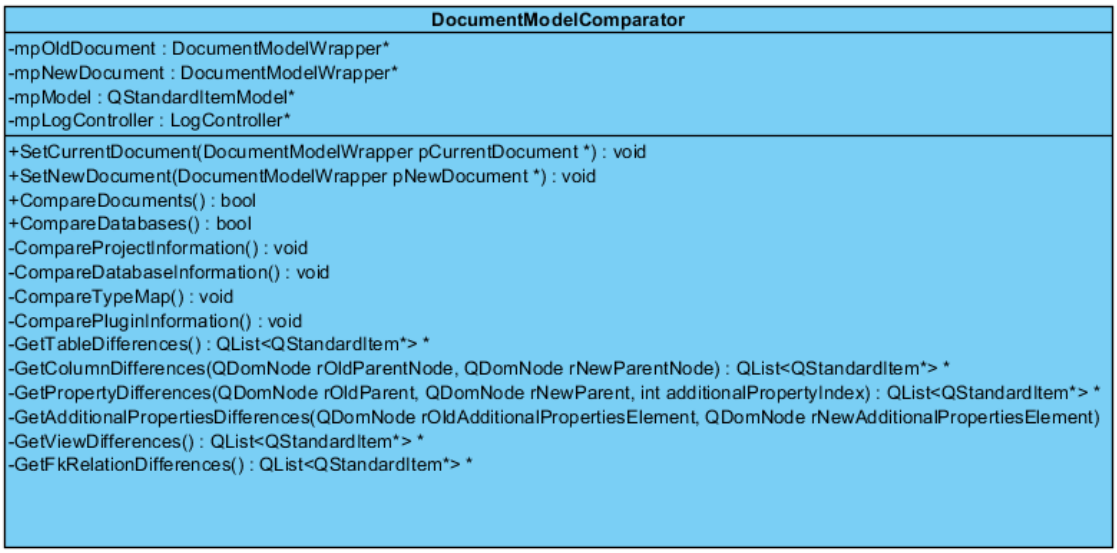
Løsningen vi har kommet frem til er å lage en ny dokumentmodell som vi populerer med differanser. Det vil si at når vi henter ut data fra ny og gjeldende dokumentmodell for å vise frem dette sammenligner vi med differansemodellen for å se om det er registrert en differanse. Verdien i differansemodellen skal inneholde informasjon om hvilken type differanse det er.

Differansetypene er som følger:

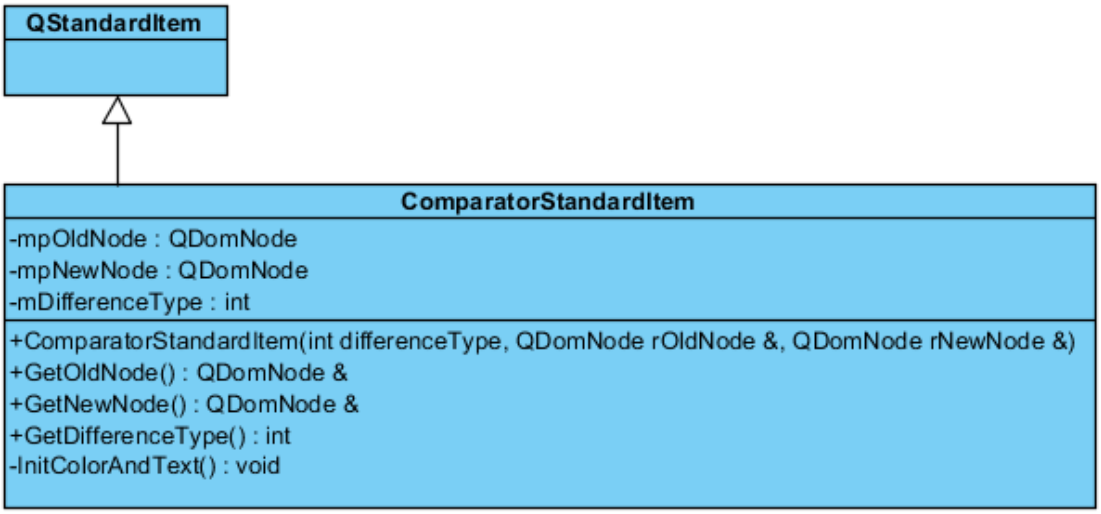
- Verdi er forskjellig
- Finnes kun i gjeldende dokumentmodell
- Finnes kun i ny dokumentmodell

Under implementasjon viste det seg at å populere en differansemodell ikke ville fungere når vi gikk over til å bruke QDomDocument. Vi måtte derfor lage en egen subklasse til QStandardItem og populere en QStandardModel som brukes av GUI for visning.

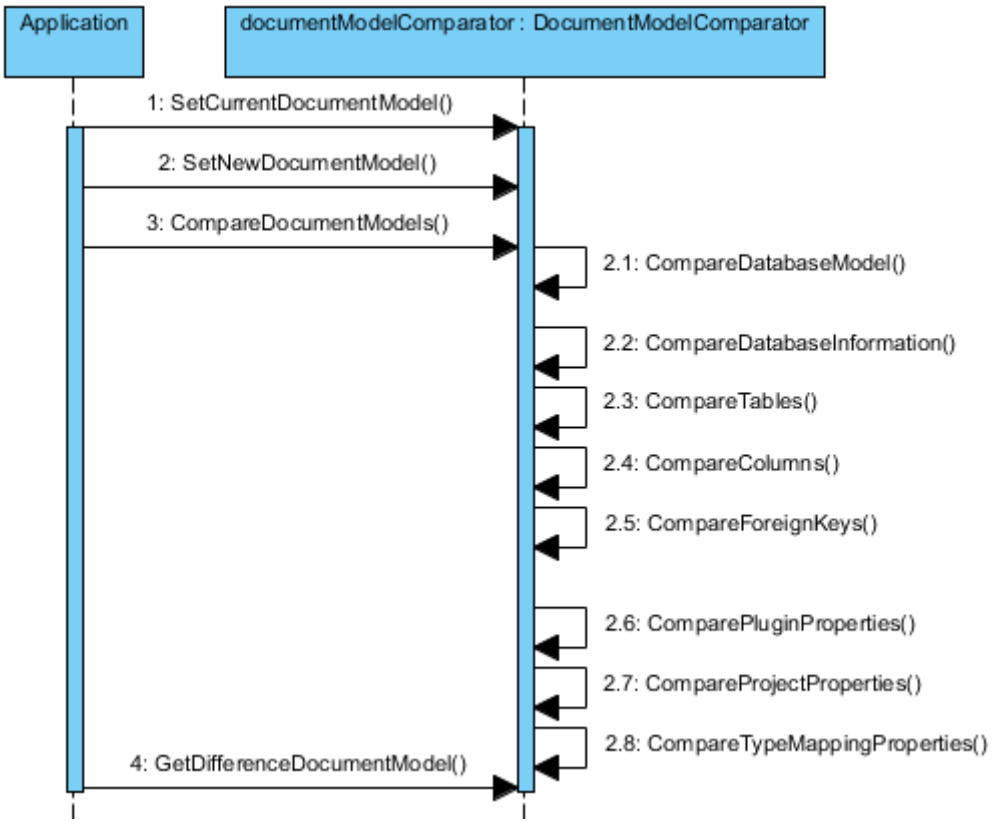
6.3 DIAGRAMMER



Figur 13: Klassediagram for klassen DocumentModelComparator



Figur 9: Klassediagram for egendefinert QStandardItem klasse



Figur 10: Sekvensdiagram som viser den tenkte sekvensen for sammenligning av dokumentmodeller

6.4 BRUK AV SAMMENLIGNINGSMODELL I APPLIKASJON

Det er hovedsakelig to måter man kan bruke sammenligningsmodellen på. Dette er å sammenligne en hel dokumentmodell eller bare databasemodellen med strukturen fra en innlest database.

6.4.1 Sammenligning av hele dokumentmodellen

Man ønsker å utføre en full sammenligning av dokumentmodellen når man for eksempel vil se på hva som er endret i forhold til en tidligere versjon. Når man ønsker å initiere en sammenligning åpner man et prosjekt og klikker på "Compare". Man velger videre hvilken prosjektfil man ønsker å sammenligne med, og vil deretter få opp en dialog som viser forskjellene.

6.4.2 Sammenligning av databasemodell

Dette brukes vanligvis dersom man ønsker å se hvilke endringer som er utført i forhold til gjeldende database. Denne differansen vises vanligvis når man skal oppdatere dokumentmodellen med nyeste versjon av databasen. Da skal man få opp et dialogvindu som viser forskjellene. Brukeren kan videre velge å forkaste eller godta oppdateringen.

7 DESIGN FOR PRESENTERING AV DOKUMENTMODELLEN

I applikasjonen skal man få presentert en visuell fremstilling av dokumentmodellen. Det skal være mulig å se alle data som ligger i dokumentmodellen, og gjøre endringer på disse om ønskelig. I første omgang er det aktuelt å presentere en dokumentmodell i applikasjonen. På et senere tidspunkt kan det være aktuelt å utvide dette til å presentere flere dokumentmodeller i applikasjonen.

Dokumentmodellen er beskrevet i et egne analyse- (2) og designdokument (3).

7.1 FORFATTER(E)

- **Tor-Christian H. Eriksen (TCE)**

7.2 HVORDAN PRESENTERE INFORMASJON

Dokumentmodellen er bygd opp hierarkisk, for eksempel:

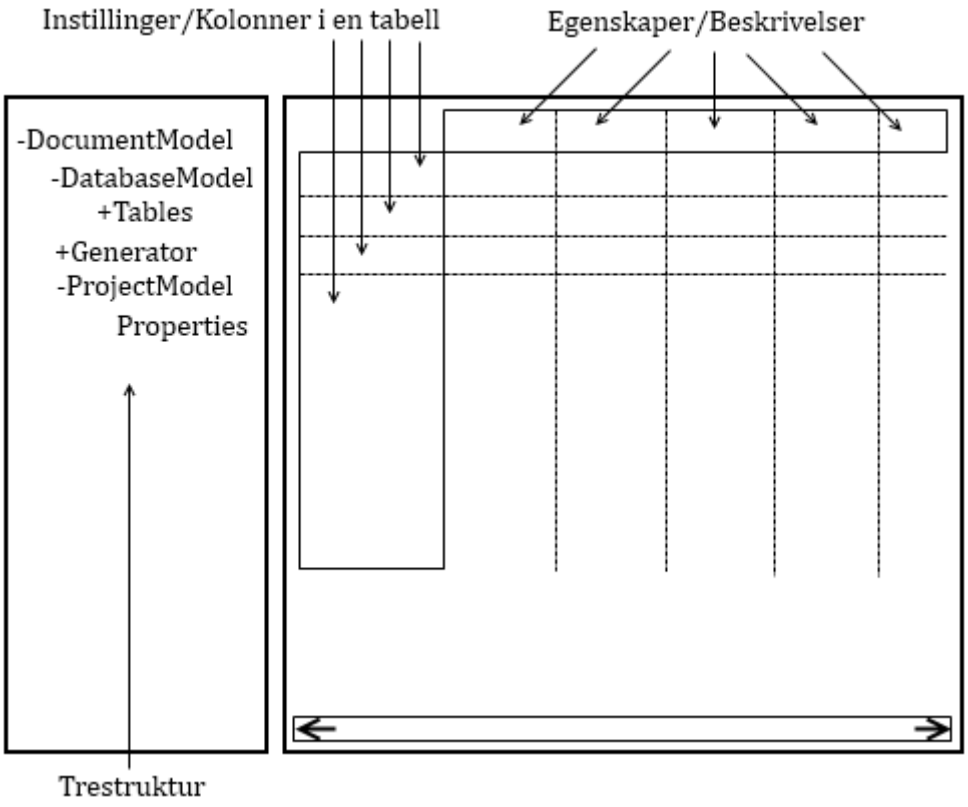
- Dokumentmodell
  - Prosjektmodell
    - Prosjektinformasjon
  - Databasemodell
    - Databasemodell
      - Databaseegenskaper
    - Tabeller
      - Tabell

Vi kan benytte oppbygningen som en basis for den visuelle presentasjonen av dokumentmodellen.

Vi ser for oss å benytte to vinduer i GUI for presentasjonen, det vil bestå av vindu A og vindu B. Vindu A vil presentere en trestruktur basert på den hierarkiske oppbygningen av dokumentmodellen. Vindu B vil vise informasjon for det valgte nivået.

Om brukeren for eksempel velger Tabeller fra vindu A så vil vindu B vise alle tabeller for databasemodellen. Om brukeren så går videre og markerer en spesifikk tabell vil vindu B vise kolonner i tabellen som rader og kolonneegenskaper som kolonner.

Figuren illustrerer hvordan vi kan benytte den hierarkiske strukturen til å representere dokumentmodellen:



Figur 11: Illustrasjon av tenkt utseende og navigering

7.3 MODELLER SOM SKAL PRESENTERES

Man kan dele opp modellene som skal presenteres i følgende hovedklasser:

- Prosjektmodell
- Pluginmodell
- Databasemodell

Modellene skiller seg fra hverandre ved at de alle inneholder forskjellig informasjon. Det er viktig at brukeren kan navigere modellene på en fornuftig og enkel måte. All informasjon skal presenteres for brukeren så det er nødvendig med en egen design for hver klasse.

7.3.1 Prosjektmodell

Prosjektinformasjonen består kun av et nivå. Dette nivået presenteres direkte til brukeren.

7.3.2 Pluginmodell

Pluginmodellen er noe mer komplisert og er først og fremst designet for å inneholde informasjon om plugins for kodegenerering. Det er en tanke at man også kan ha informasjon om innlesning om ønskelig.

7.3.3 Databasemodell

Databasemodellen er den mest kompliserte av modellene og har all informasjon lest ut av databasen. Egenskaper for databasen, tabeller, views, foreign keys, eventuelt systemtabeller osv. skal kunne presenteres og endres av brukeren.

8 DESIGN AV INNLESING VIA PLUGIN

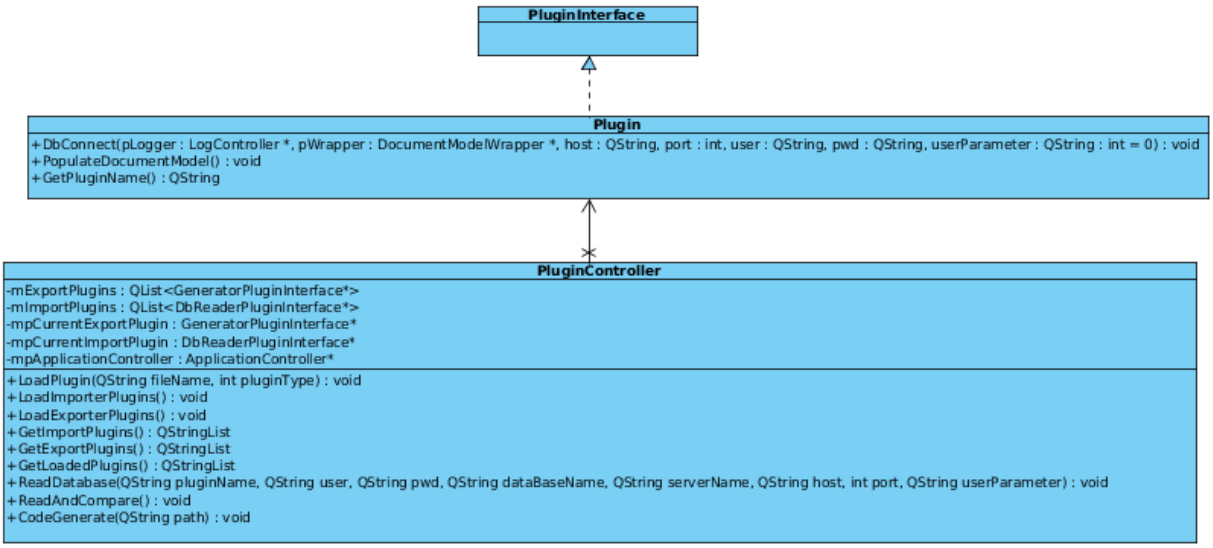
For å forstå hvordan vi bruker plugins for innlesing av en databasemodell. Vi illustrert dette ved hjelp av klasse- og sekvens- diagram som viser funksjonalitet og rekkefølgen på kall.

8.1 FORFATTER(E)

- **Jarle Didriksen (JD)**
- Tor-Christian H. Eriksen (TCE)

8.2 DIAGRAMMER OG UTREDNINGER

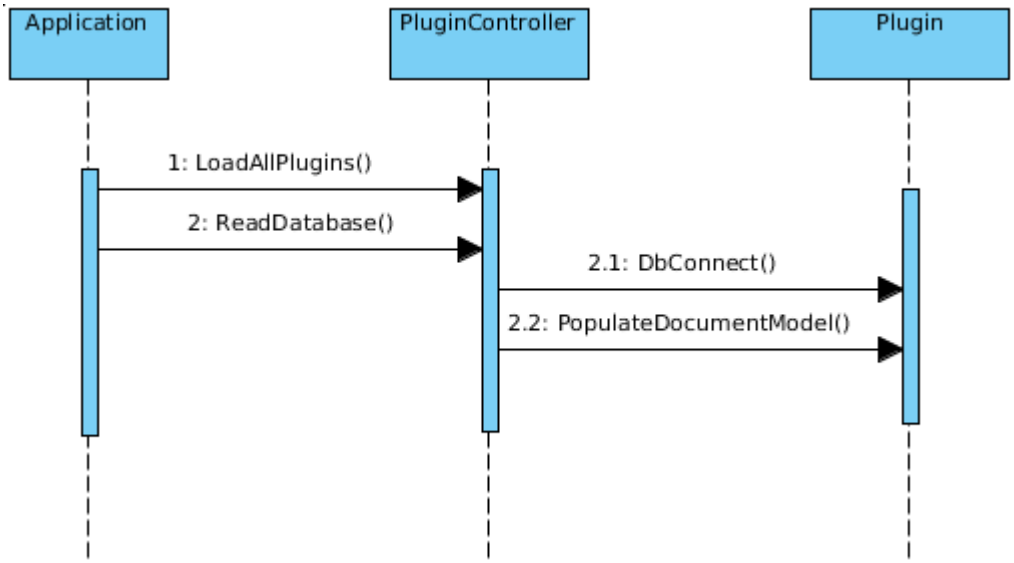
Vi har utledet et klassediagram som tar for seg sekvensen for innlesing utledet i analysen.



Figur 12: Innlesing via plugin – Klassediagram

Plugin benytter seg av metodene i plugin-interfacet og tilpasser metodene for databasetypen den brukes mot. PluginController har ansvar for å initiere plugin og kjøre dens metoder, og avslutte den etter at den er ferdig.

For å vise den tenkte sekvensen videre har vi utledet et sekvensdiagram som tar for seg en normal utførelse basert på klassediagrammet.



Figur 13: Innlesing via plugin – Sekvensdiagram

Når en plugin er valgt vil PluginController starte riktig plugin, og kjøre plugin sine metoder via ReadDatabase().Etter ferdig innlesing og populering av dokumentmodellen vil kontrolleren avslutte plugin'en.

9 DESIGN AV FACTORIES FOR INNLESNING

9.1 FORFATTER(E)

- **Alexander Edland (AE)**

9.2 INTRODUKSJON

I overgangen til *QDomDocument* utviklet vi et mellomledd mellom import-plugins og controlleren, for å danne en platform for dokumentpopulering. Hver av klassene i den gamle klassestrukturen ble et eget nivå i QDomDokumentet, og hvert nivå fikk en egen factory for å opprettes ifølge standarden vi definerte. Hvert nivå blir opprettet av en spesifikasjonsklasse som arver fra *GenericFactory*.

9.3 ANVENDELSE

For hvert element som skal settes inn i dokumentmodellen, så vil plugin opprette et nytt factory-objekt. Plugin vil kalle factoriens set-metode for hver verdi som skal settes inn, og verditypen indikeres ved bruk av dokumentmodellens enum for hver respektive factory. Dette sikrer korrekt rekkefølge internt i applikasjonen uten at plugin-forfatteren trenger å ta hensyn til dette, og forfatteren får samtidig godt overblikk over hvilken informasjon som forventes hvor.

Hver factory har også regler for hvilke noder som skal ha forhåndssatte verdier, og hvilke elementer som skal forbli blank. For eksempel vil Table- og Column-factories sette included-for-generation til TRUE.

9.4 ADDITIONAL PROPERTIES

I de tilfeller hvor plugin ønsker å sette inn data som ikke er forventet av applikasjonen (additional properties) vil set-metoden kunne kalles med tag-navn istedenfor applikasjonens alias for datatypen. Dette er en utvidelse man helst benytter ved midliertidige ekspansjoner, da alle funksjoner innad i applikasjonen må anvende lookups istedenfor indekser.

9.5 INNSETTING I DOKUMENTMODELLEN

Når et element er ferdigpopulert vil plugin kalle finalize-metoden som returnerer et QDomElement som sendes videre til controlleren for sortering og innsetting i dokumentmodellen.

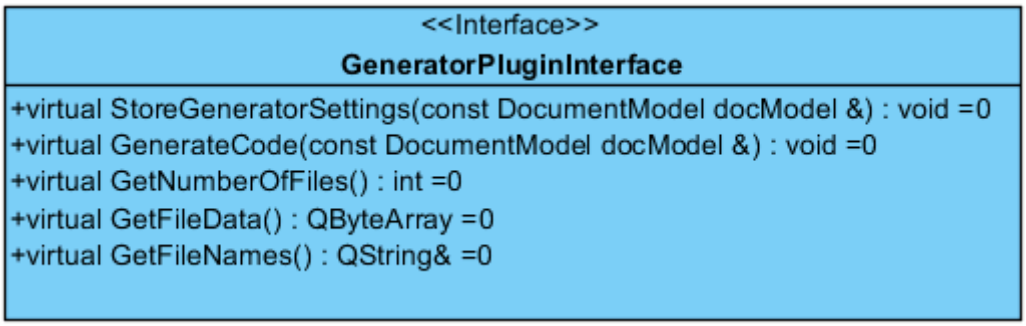
10 DESIGN AV PLUGIN FOR GENERERING

10.1 FORFATTER(E)

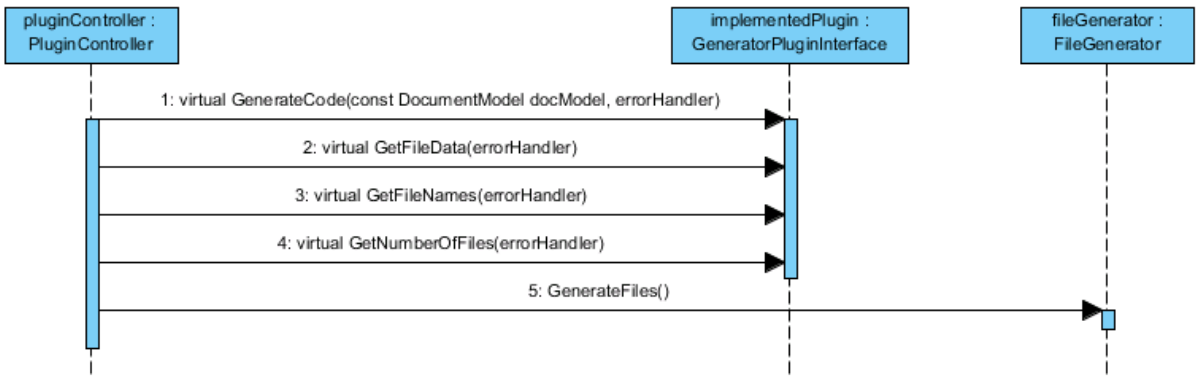
- **Jørgen Grøndal (JG)**

10.2 INTRODUKSJON

Ved utarbeidelse av interfacet er det lagt vekt på en tenkt sekvens fra analysen. Etter flere iterasjoner har vi kommet frem til at å sende inn selve dokumentmodellen er den beste måten. Da har generatoren tilgang på de nødvendige data, og det legges lite restriksjoner på hva en plugin kan gjøre.



Figur 14: Plugin for generering - Klassediagram for plugininterface



Figur 20: Plugin for generering - Sekvensdiagram for metodekall mot en generatorplugin

10.3 BESKRIVELSE AV METODER

Når en bruker initierer generering av kode vil metodene i interfacet over bli eksekvert i sekvens som beskrevet i figuren over. Unntaket er metoden StoreGeneratorSettings som blir kalt første gang en generator blir lagt til i et prosjekt.

10.3.1 StoreGeneratorSettings

Kjøres når en generator velges for et prosjekt. Da vil generatoren populere dokumentmodellen med generatorens egenskaper sånn at brukeren kan endre på disse før generering om ønskelig.

10.3.2 GenerateCode

Initierer kodegenereringen. Sender med dokumentmodellen som parameter. Plugin er selv ansvarlig for å hente ut informasjon fra dokumentmodellen.

10.3.3 GetNumberOfFiles

Returnerer antall filer som ble generert.

10.3.4 GetFileData

Returnerer et array med QByteArray. Index N tilsvarer filnavn i index N. Hver index inneholder all data som skal skrives til fil.

10.3.5 GetFileNames

Returnerer et array av QString der indexene korresponderer med index i fildata.

11 DESIGN AV XML-FIL FOR Å LAGRE DOKUMENTMODELL

For lagring av dokumentmodell har vi valgt å lagre data i XML-filer. Før vi utvikler applikasjonen vil vi designe XML-filen slik at vi kan designe skrive- og leserutiner basert på dette. XML-filen vil inneholde et komplett tilstandsbilde av dokumentmodellen.

11.1 FORFATTER(E)

- **Simen Russnes (SR)**

11.2 BESKRIVELSE AV TAGS

XML-dokumenter bygges opp av noder. Tags navngir informasjonen i hver node. En node åpnes og lukkes av et tag, og imellom tags finnes en enkeltverdi eller child-tags. "<server>localhost</server>" viser at «server» (tag) er «localhost» (enkeltverdi).

Videre er alle tags brukt i XML-designet vårt beskrevet i underkapitlene av 2.1.

11.3 DOCUMENTMODEL

<documentmodel> vil fungere som en rot-node for hele XML-filen. Alle de fire hoved-nodene vil være under denne i hierarkiet. Det vil si <project>, <database>,<typemap> og <plugin>.

11.4 «PROJECTINFORMATION»

Denne noden vil inneholde informasjon om prosjektet.

Under dette nivået finner vi:

```
<projectname>
<companyname>
<developername>
<description>
<lastimport>
<lastwritten>
<additionalproperties>
```

11.5 «DATABASEINFORMATION»

Denne noden vil inneholde informasjon om eventuelt innlest database i dokumentet.

I tillegg til generell informasjon om databasen som f.eks navn, databasetype og innloggingsinformasjon, vil det også være en representasjon av selve databasen som har blitt lest inn (og evt. modifisert) i form av <table> noder.

<databaseinformation> har følgende child nodes:

```
<databasename>
<username>
<password>
<databasetype>
<databasesize>
<databasedriver>
<databaseserver>
<port>
<tablesorting>
<connectionstring>
<connectiontimeout>
<connectiontype>
<autocommit>
<description>
<characterencoding>
<additionalproperties>
<tables>
<views>
<foreignkeyrelations>
```

11.5.1 «table»

Hver tabell vil lagres under <tables> i en <table> tag med informasjon om den gitte tabellen. Her vil det lagres informasjon om egenskaper for tabellen, samt kolonner og keys.

Tags som inngår i table er:

```
<name>
<included>
<trigger>
<columnsort>
<generateto>
<columns>
<index>
```

11.5.1.1 «column»

Kolonnerrepresenteresi<column>tags under <columns>.

Hver<column>harfølgende child nodes:

```
<name>
<datatype>
<datatypeprecision>
<columnincluded>
<characterencoding>
<autoincrement>
<null>
<length>
<scale>
<primarykey>
<constraints>
<generateto>
```

Dersom en column er en primarykey vil dette indikeres med <primarykey>true</primarykey>.

11.5.2 «view»

Hver view vil lagres under <views> i en <view> tag med informasjon om det gitte viewet. Her vil blant annet navn, og hvilke kolonner som inngår bli lagret.

Tags som inngår i view er:

```
<name>
<included>
<generateto>
<additionalproperties>
<columns>
```

11.5.3 «foreignkeyrelation»

Node i <databaseinformation> som representerer relasjoner mellom tabeller.

Tags som inngår i foreignkeyrelation er:

```
<fkname>
<referencedtable>
<referencedcolumn>
<referencingtable>
<referencingcolumn>
<included>
<allownullvalue>
<matchtype>
<checkonlyoncommit>
<deleterule>
<updaterule>
<additionalproperties>
```

Et eksempel på en lagret database er gitt under.

```
<?xml version='1.0' encoding='UTF-8'?>
<documentmodel>
    <projectinformation>
        <projectname>My Project</projectname>
        <companyname>DBFactory</companyname>
        <developername>Simen</developername>
        <description>This is my project</description>
        <lastimport>11-02-1990;13:32:42</lastimport>
        <lastwritten>11-02-1990;07:45:14</lastwritten>
        <additionalproperties/>
    </projectinformation>
    <databaseinformation>
        <databasename>Library</databasename>
        <username>superuser</username>
        <password>gxy324</password>
        <databasetype>SQLANYWHERE</databasetype>
        <databasesize>501</databasesize>
        <databasedriver>value</databasedriver>
        <databaseserver>value</databaseserver>
        <port>81</port>
        <tablesorting>sortbyname</tablesorting>
        <connectionstring>value</connectionstring>
        <connectiontimeout>value</connectiontimeout>
        <connectiontype>value</connectiontype>
        <autocommit>value</autocommit>
        <description>Library information</description>
        <characterencoding>UTF-8</characterencoding>
        <additionalproperties/>
        <tables>
            <table>
                <name>Books</name>
                <included>TRUE</included>
                <triggers>NONE</triggers>
                <columnsort>DEFAULT</columnsort>
                <generateto>
                    <plugin>vbdotnet10</plugin>
                </generateto>
                <additionalproperties/>
                <columns>
                    <column>
                        <name>Title</name>
                        <datatype>char</datatype>
                        <datatypeprecision>100</datatypeprecision>
                        <columnincluded>TRUE</columnincluded>
                        <characterencoding>UTF-DEFAULT99</characterencoding>
                        <autoincrement>value</autoincrement>
                        <nulls>value</nulls>
                        <length>1000</length>
                        <scale>2</scale>
                        <primarykey>TRUE</primarykey>
                        <constraints>None</constraints>
                        <mappeddatatype>
                            <cpp>string</cpp>
                        </mappeddatatype>
                        <generateto>
                            <plugin>vbdotnet10</plugin>
                        </generateto>
                        <additionalproperties/>
                    </column>
                </columns>
                <indices>
                    <index>
                        <name>value</name>
                        <type>value</type>
                        <column>value</column>
                        <additionalproperties/>
                    </index>
                </indices>
            </table>
        </tables>
        <views>
            <view>
                <name>Awesome view</name>
                <included>TRUE</included>
```

```
                <generateto>
                    <plugin>vbdotnet</plugin>
                </generateto>
                <additionalproperties/>
                <columns></columns>
            </view>
        </views>
        <foreignkeyrelations>
            <foreignkeyrelation>
                <fkname>value</fkname>
                <referencedtable>value</referencedtable>
                <referencedcolumn>value</referencedcolumn>
                            <referencingtable>value</referencingtable>
                <referencingcolumn>value</referencingcolumn>
                <included>true</included>
                <allownullvalue>value</allownullvalue>
                <matchtype>value</matchtype>
                <checkonlyoncommit>value</checkonlyoncommit>
                <deleterule>value</deleterule>
                <updaterule>value</updaterule>
                <additionalproperties/>
            </foreignkeyrelation>
        </foreignkeyrelations>
    </databaseinformation>
    <plugininformation>
        <importer>sqlanywhere12</importer>
        <generators>
            <language>cplusplus</language>
            <language>vbdotnet</language>
        </generators>
    </plugininformation>
    <typemapinformation>
        <map>
            <language>vbdotnet</language>
            <database>
                <name>SQLANYWHERE</name>
                <datatypes>
                    <float>Double</float>
                </datatypes>
            </database>
        </map>
    </typemapinformation>
</documentmodel>
```

Figur 21: XML-fil – Eksempel på en lagret database

12 Design for generering og eksportering av XML-filer

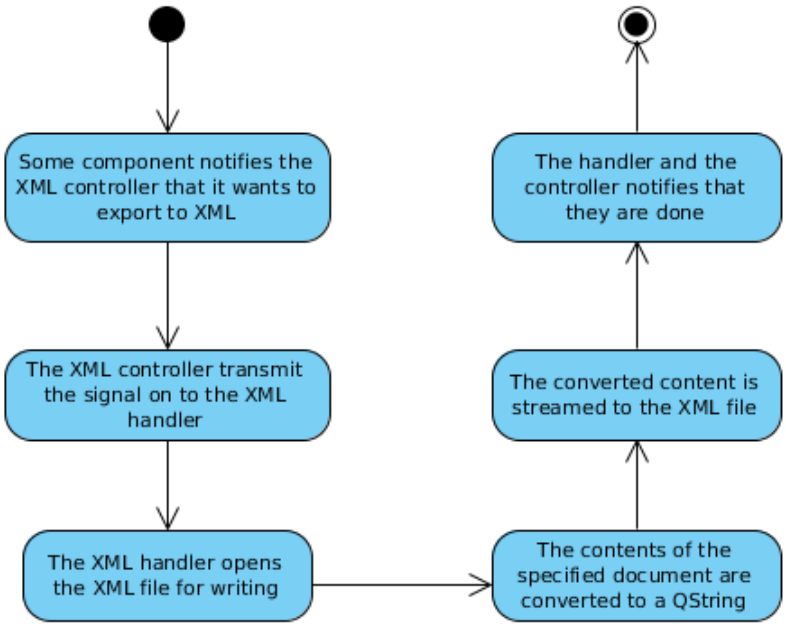
12.1 Forfatter(e)

- **Jarle Didriksen (JD)**

12.2 Introduksjon

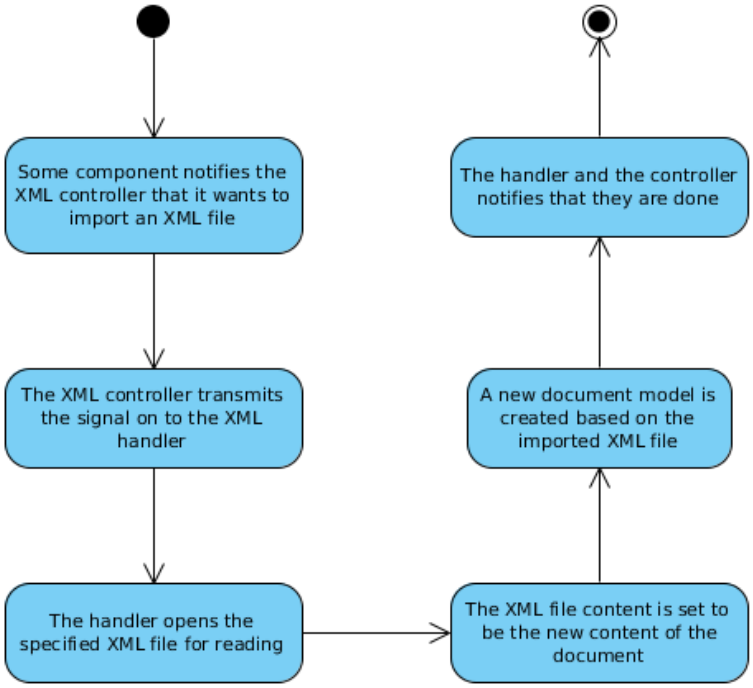
For generering og eksportering av XML-filer for lagring og innhenting av dokumentmodell vil vi lage klassediagram, sekvensdiagram og aktivitetsdiagrammer.

For å forklare en tenkt måte XML-håndteringen vil gjøres på har vi laget et aktivitetsdiagram for generering og et for eksportering i figurene under.



Figur 22: Design for importering og eksportering av XML-filer – Aktivitetsdiagram for eksportering

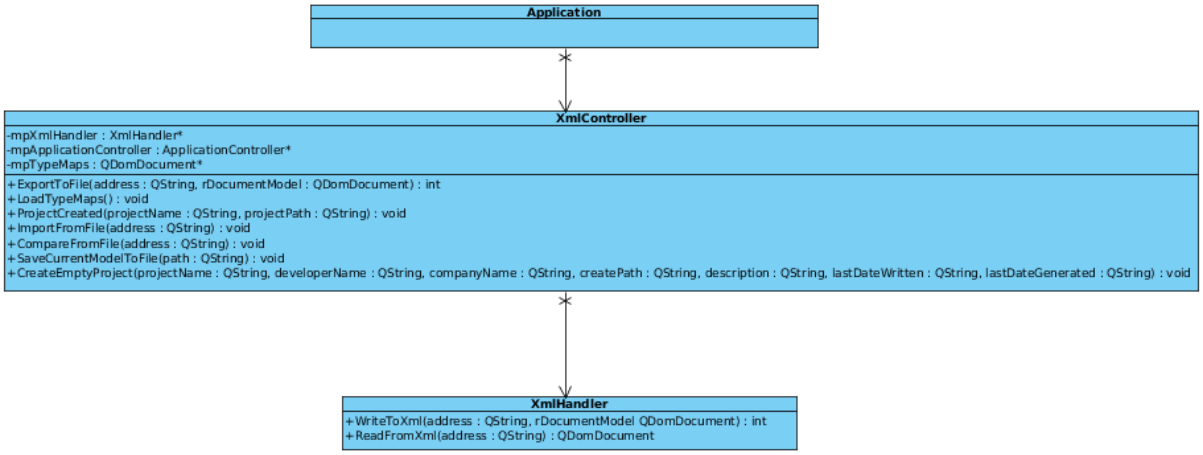
Som figuren viser vil vi ta alt innholdet i dokumentet og legge det i en spesifisert XML-fil. Dersom XML-filen ikke eksisterer vil det opprettes en. Hvis filen allerede eksisterer vil den bli overskrevet og erstattet med det nye innholdet.



Figur 15: Design for importering og eksportering av XML-filer – Aktivitetsdiagram for importering

Som figuren viser vil vi legge innholdet i XML-fila over i det spesifiserte dokumentet. XML-fila som skal leses fra må eksistere på forhånd for å gjøre dette.

12.3 KLASSEDIAGRAM OG SEKVENSDIAGRAM



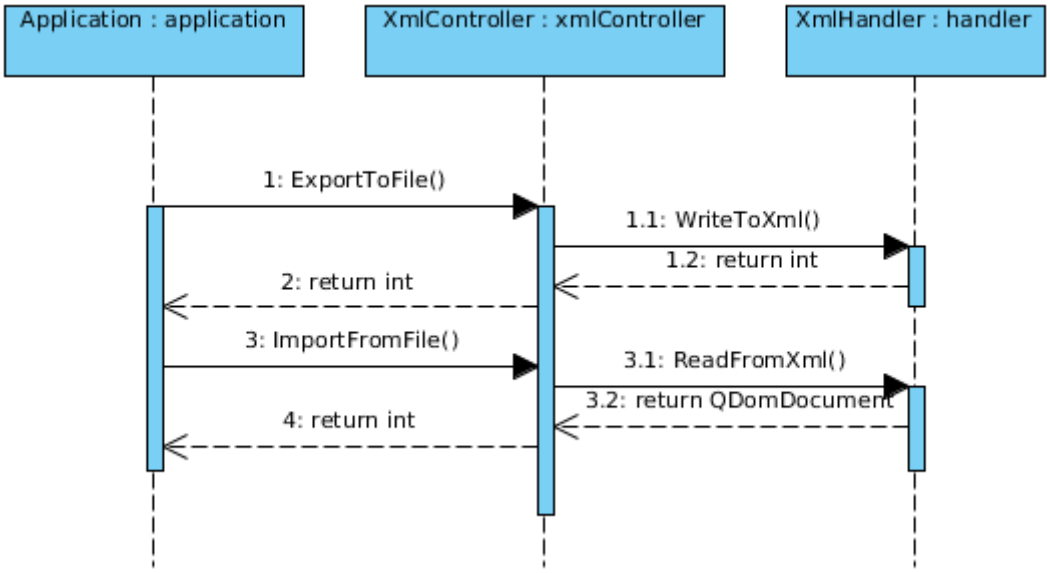
Figur 16: Design for importering og eksportering av XML-filer – Klassediagram

XmlController er klassen som applikasjonen bruker når det ønskes å skrive til eller lese fra en XML-fil. Når man importerer en XML-fil så opprettes det en ny dokumentmodell som får innholdet til XML-filen. Når man eksporterer til en XML-fil så tar man den nåværende dokumentmodellen og legger over alt innholdet i en XML-fil. Begge disse metodene sender signalet videre til *XmlHandler*.Det er klassen som utfører selve filbehandlingen, den som har direkte kommunikasjon mot eksterne filer.

I tillegg har XmlController metode for å opprette et nytt, blankt prosjekt. Denne brukes når man oppretter et nytt prosjekt i applikasjonen, og vil sette en ny dokumentmodell med initiell informasjon om prosjektet.

Dersom applikasjonen ikke greier å åpne en fil for lesing eller skriving vil det komme opp et dialogvindu som informerer brukeren om dette.

Under viser vi slik vi ser for oss at kommunikasjonen mellom de forskjellige klassene kan være.



Figur 17: Design for importering og eksportering av XML-filer – Sekvensdiagram

Applikasjonen kaller metodene til kontrolleren for å enten eksportere eller generere. XmlController vil utfra dette kalle metodene til XmlHandler, som deretter utfører de nødvendige handlingene.

13 DESIGN AV TYPEMAPPING

De fleste språk skiller mellom datatyper, men det er ofte stor variasjon i navngivning og begrensninger. Det er derfor nødvendig å finne de beste tilnærmelsene på kryss av språk, for minst mulig presisjonstap. Dette ønsker vi å løse med forhåndsbestemte konverteringsregler, «typemapping».

13.1 FORFATTER(E)

- **Vegard Kaasin(VK)**

13.2 XML-FIL

Vi ser for oss at typemapping er forhåndsdefinert og lagret i en XML-fil. Denne filen blir lest under generering av kode, og datatyper blir slått opp og mappet automatisk. Cytrax har gitt oss ett eksempel på hvordan dette kan se ut:

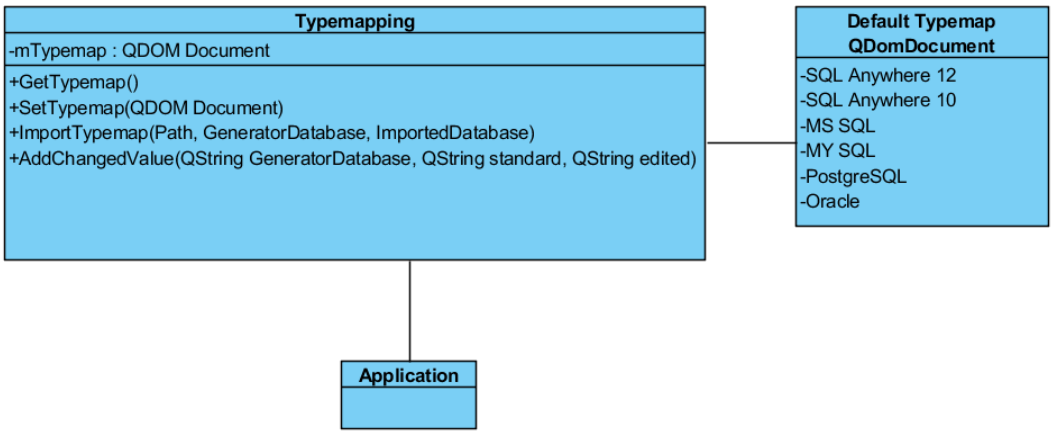
```
<datatypemapping>
      <languagename="c++">
            <databasetype="SQLanywhere">
                  <varcharlength="true">char*</varchar>
                  <numericlength="true">double</numeric>
                  <charlength="true">char*</char>
                  <bit>bool</bit>
                  <bigint>int64</bigint>
            </database>
            <databasetype="MSSQL">
                  (...)
            </database>
      </language>
      <languagename=".net">
            <databasetype="SQLanywhere">
                  <varchar>string</varchar>
                  <numericlength="true">decimal</numeric>
                  <charlength="true">string</char>
                  <bit>boolean</bit>
                  <bigint>long</bigint>
            </database>
            <databasetype="MSSQL">
                  <varchar>string</varchar>
                  <numericlength="true">decimal</numeric>
                  <bit>boolean</bit>
                  <long>long</long>
            </database>
            <databasetype="Oracle">
                  (...)
            </database>
      </language>
</datatypemapping>
```

I eksempelet kan vi se at vi først definerer en tag for hvilke språk det skal genereres kode til, etterfulgt av en tagget liste over alle SQL-dialekter. For hver dialekt finnes typemapinformasjonen der databasetypen er en tag, og datatypen den skal mappes til ligger som informasjon i hver tag. Vi kan også se at det er foreslått at length skal være med, men siden vi har en egenskap i dokumentmodellen som heter «Length» blir ikke denne tatt med.

Basert på datatypene definert i analysedokumentet for typemapping vil vi lage en XML-fil for hvert språk vi skal generere kode til. Dette gjøres fordi det skal være så enkelt som mulig å utvide applikasjonen, og forhindre at det jobbes kun med en stor XML-fil.

13.3 TYPEMAPPING I APPLIKASJONEN

XML-filen vil bli lest inn til applikasjonen som et «QDOM Document» i likhet med lagring og lesing av dokumentmodellen. Dette gjøres fordi det er enkelt å jobbe med og gir oss mulighet til å lese inn, hente verdier, endre verdier og lagre nye XML-dokumenter. En representasjon av hvordan dette skal implementeres er gitt i figuren under:



Figur 18: Typemapping – Klassediagram

13.4 HVILKE DATATYPER SOM SKAL MAPPES

Vi må også definere datatypene for databasen og hvilke datatyper de vil få i språkene vi skal generere til. Tabeller som inneholder denne informasjonen ligger i design-typemapping (4).

14 DESIGN AV FLERE GENERATORER PER TABELL

Det er ønskelig å kunne generere kode på et utvalg tabeller eller views, eller å generere kode til flere språk på samme tabell/view, eller et utvalg tabeller/views. Dette avsnittet skal dokumentere hvordan dette skal løses, og komme med forslag til design.

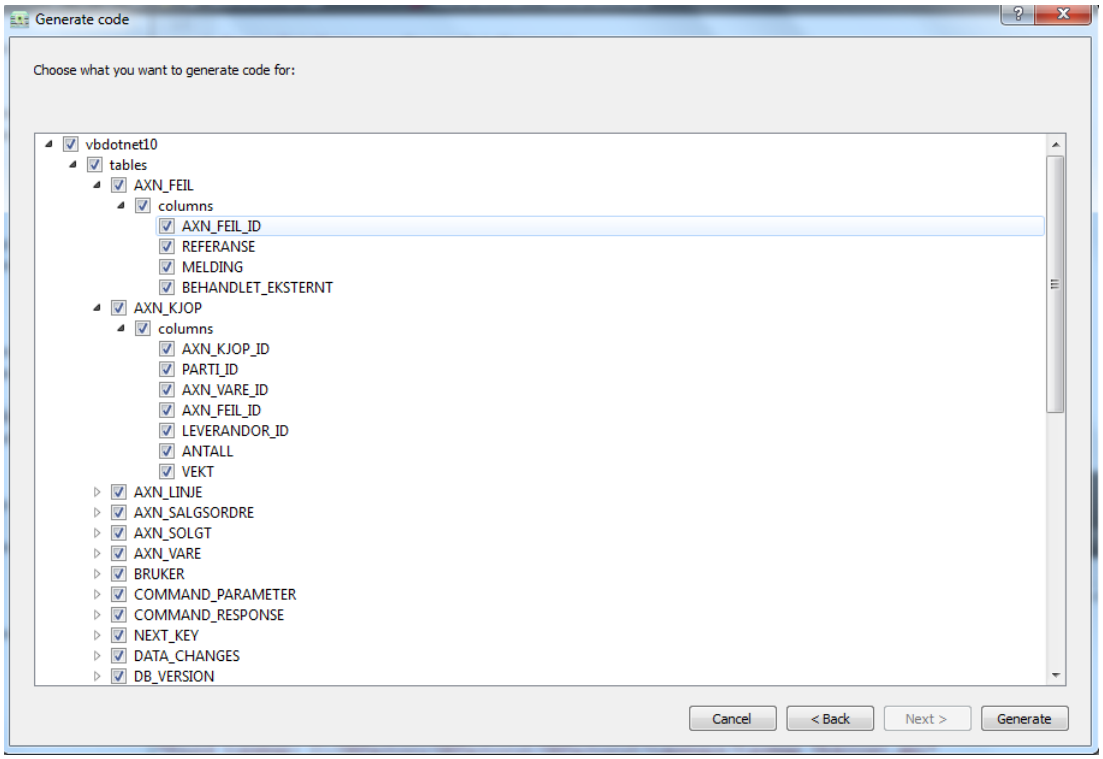
I dokumentmodellen (2)ligger det egenskaper i hver tabell/view om det skal genereres kode for tabellen/view, og hvilke generator det skal genereres kode for. Plugin for generering bruker dette til å finne hvilke tabeller/view den skal generere kode til.

14.1 FORFATTER(E)

- **Vegard Kaasin(VK)**

14.2 GUI

Etter å ha prototypet for gui fant vi ut at den beste måten å representere tabeller og views er å vise generatorene i en treeview og vise alle tabeller, views og deres columns som child-nodes.



Figur 19: GUI for å representere views/tabeller og generatorer

Som vi ser i figuren over er alle tabeller og views listet opp i en treeview. Hvis det er flere generatorer blir det satt opp på øverste nivå. Hvis du ikke vil ha med en generator huker du av checkboxen og alle childnodes blir også huket av.

15 DESIGN AV SYSTEM FOR LOGGING

15.1 FORFATTER(E)

- **Tor-Christian H. Eriksen (TCE)**

15.2 INTRODUKSJON

Vi har valgt å lage vårt eget system for logging. Det finnes allerede mange løsninger for logging, disse er ofte komplekse og har mye funksjonalitet som ikke er nødvendig for applikasjonen vår. Derfor har vi valgt å lage et system som passer til vår applikasjon med den funksjonaliteten vi ønsker.

Loggeren er ment å bli brukt hver gang man ønsker å gi brukeren av applikasjonen en tilbakemelding om en hendelse. Dette kan for eksempel være at man har brukt feil brukernavn for å koble til en database, endret konfigurasjon til en generator, at en innlest tabell er tom, en kritisk feil har oppstått, osv.

For hver melding som blir rapportert legges det til tidspunkt og klassifisering.. Det finnes flere typer meldinger, bla. **warning, config, info,** osv. Disse er definert i et enum.

I likhet med systemet for feilhåndtering har vi laget et design mtp. MVC, så det inneholder kontroller, datamodell og view. Mekanismen for feilhåndtering benytter loggeren for å lagre informasjon om eventuelle feil.

Man kan også bruke loggeren for debugging om ønskelig.

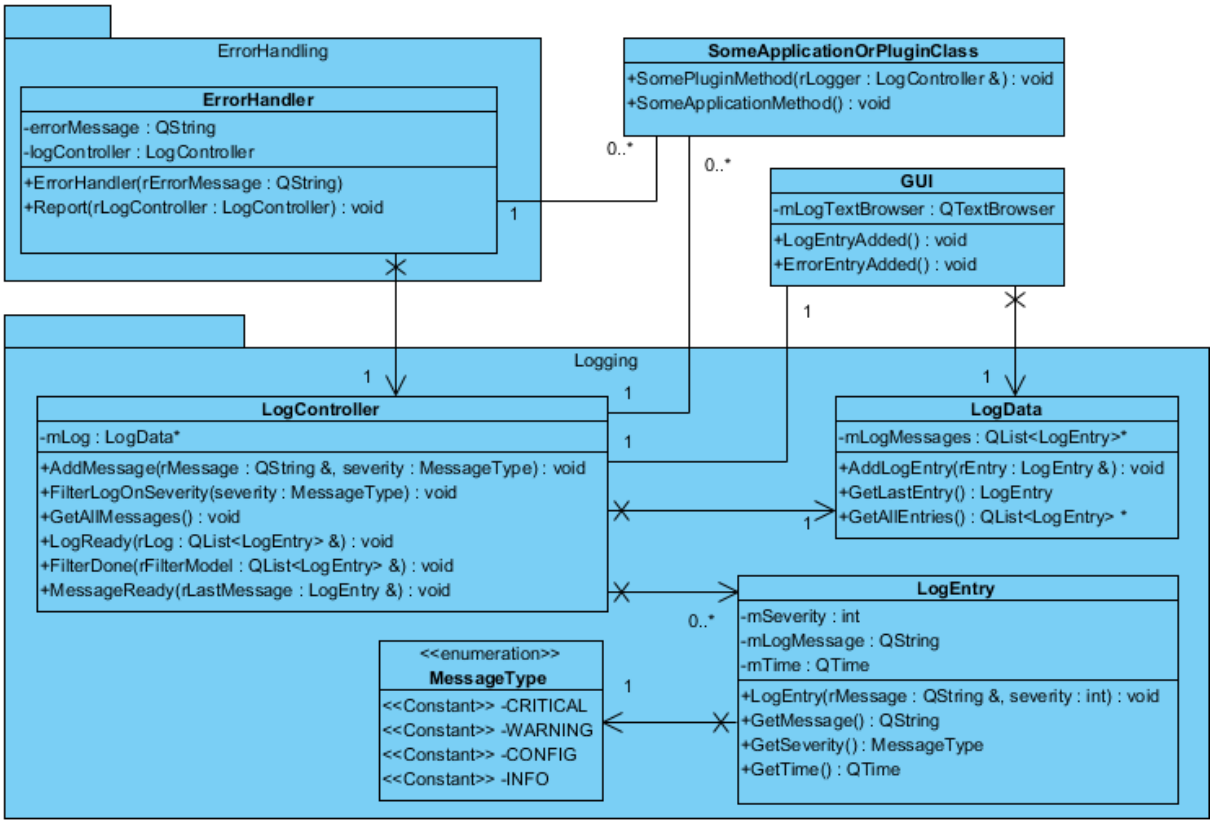
15.3 UML-BESKRIVELSE AV SYSTEMET

15.3.1 Hva systemet skal gjøre

Loggeren tar imot en melding med tilhørende klassifisering, definert i et `enum`. Meldinger blir lagret i en log som er tilgjengelig for brukere. Meldinger skal også kunne skrives ut fortløpende til et loggvindu.

Loggeren er ment for å loggføre alle interessante hendelser i applikasjonen. Det kan være ønskelig ved et senere tidspunkt å utvide systemet til å la brukeren konfigurere hvilke hendelser som skal fanges opp. Systemet som det er nå vil ha mulighet til å kunne filtrere meldinger som ligger i loggen.

15.3.2 Involverte klasser



Figur 20: Logging – Klassediagram

15.3.2.1 LogController

Alle beskjeder fra applikasjonen som skal loggføres sendes til kontrolleren. Kontrolleren har ansvar for å legge inn et nytt objekt av `LogEntry` i `LogData` og signalisere GUI'et om å hente meldingen.

15.3.2.2 ErrorHandler

Tar inn en feilmelding og sender denne videre til logging. Diagrammet tar kun for seg behandling av meldingen og ikke feilen i seg selv. Dette overlates til avsnitt om feilhåndtering.

15.3.2.3 SomeApplicationOrPluginClass

Objektet som sender en melding inn til kontrolleren, dette kan være av flere typer. Noen av typene er **warning, config**og**info**.

15.3.2.4 LogData

En klasse som inneholder metoder for å legge til og hente ut meldinger fra listen over meldinger, og beskriver datamodellen. Listen er representert som en `QList` av

`LogEntry` objekter. Metoden `prepend(const T &value)` legger *value* på starten av lista, slik at man ikke trenger å søke gjennom hele lista for å skrive ut siste melding i GUI.

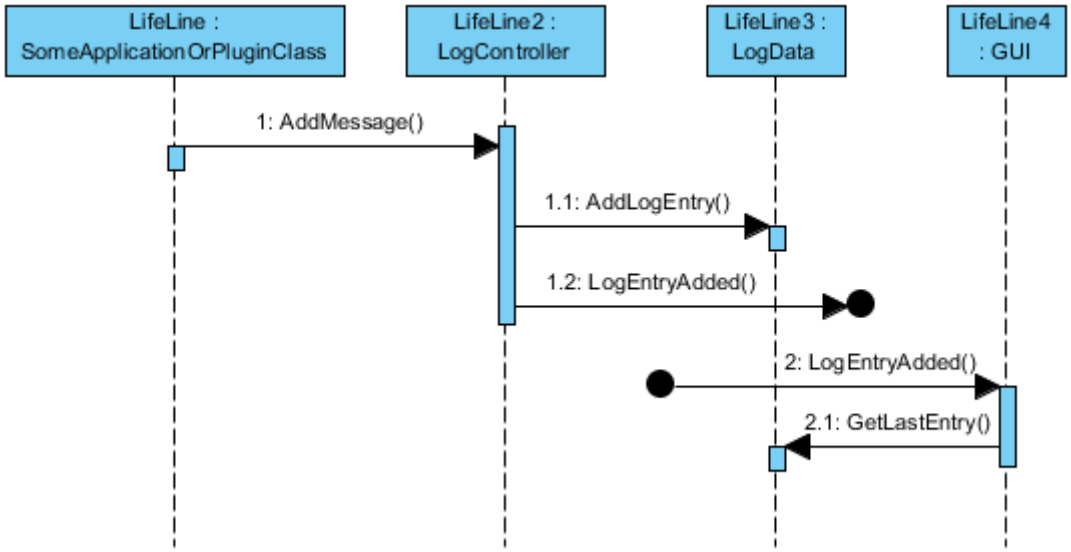
15.3.2.5 LogEntry

Inneholder all informasjon om en melding, blir opprettet hver gang kontrolleren mottar en ny melding. Konstruktøren setter `mTime`.

15.3.3 Kommunikasjon mellom objektene

I likhet med systemet for feilhåndtering signaliseres GUI'et når en ny melding kan hentes ut fra datamodellen. All kommunikasjon mellom view og datamodell går via kontrolleren.

I QT kan man representere meldinger mellom objekter som SIGNALS. Et signal kan festes til en eller flere metoder som defineres i en header ved å bruke nøkkelordet `slot`. Meldinger som skal sendes mellom objekter er `public slots` og innad i objektet er `private slots`.
Et signal kobles opp mot en eller flere metoder ved å definere dem med `connect()` i for eksempel en konstruktør der signalet har sin opprinnelse.



Figur 21: Logging – Sekvensdiagram

16 DESIGN AV FEILHÅNDTERING

I en applikasjon med GUI og mange funksjoner er det nødvendig å håndtere kritiske feil og loggføre dem. Det er også ønskelig at applikasjonen ikke skal stoppe opp om en feil oppstår, så lenge det er mulig.

Vi ønsker å følge MVC standard, derfor benytter vi en kontroller mellom datamodell og view. Oppbygningen av systemet gjør det enklere å tilføre ny funksjonalitet på et senere tidspunkt.

16.1 FORFATTER(E)

- **Tor-Christian H. Eriksen (TCE)**

16.2 UML-BESKRIVELSE AV SYSTEMET

Systemet er designet i UML, og er beskrevet ved hjelp av use case-, klasse-, aktivitets-, og sekvensdiagrammer. Diagrammene gir en oversikt over systemets funksjonalitet og hendelsesforløp.

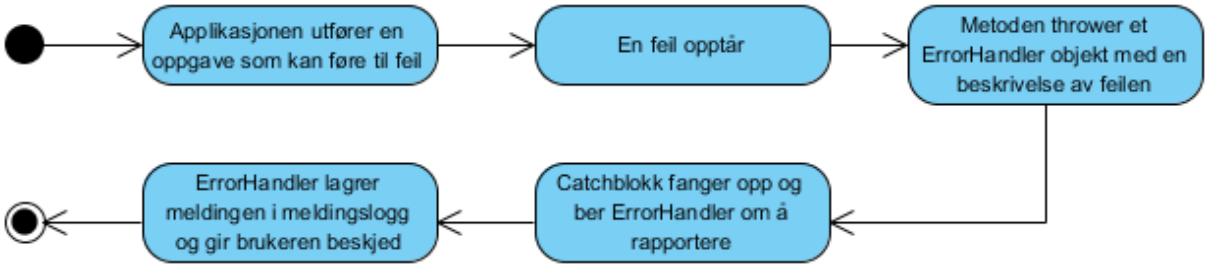
16.2.1 Hva systemet skal gjøre

En klasse kalt ErrorHandler fungerer som kontroller. Den er ansvarlig for å loggføre nye meldinger i datamodellen og gir beskjed til GUI når den kan gå inn i datamodell og hente ut data. Et objekt av klassen ErrorHandler vil være tilgjengelig for alle objekter som ønsker å rapportere feil.

16.2.2 Et typisk hendelsesforløp

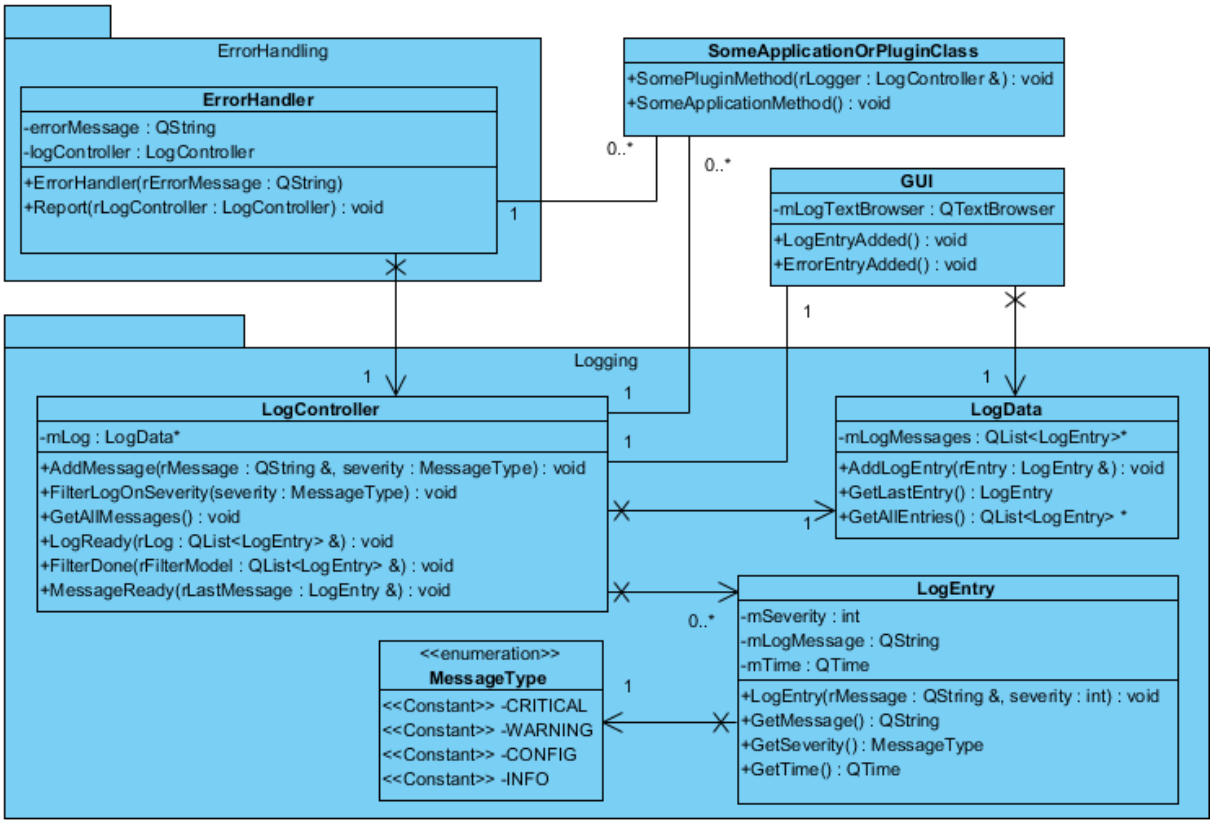
En feil vil normalt sett oppstå når en bruker av applikasjonen utfører en handling. Dette kan være fordi vi som utviklere ikke har forutsett mulige feilsituasjoner, eller om brukeren gjør en handling som fører til en feil som det er tatt hensyn til i koden.

I begge tilfellene så skal systemet for å fange opp og rapportere feil fungere optimalt.



Figur 30: Aktivitetsdiagram som viser et typisk hendelsesforløp

16.2.3 Involverte klasser



Figur 31: Feilhåndtering – Klassediagram

16.2.3.1 ErrorHandler

Håndterer meldingen fra objektet der feilen har oppstått og underretter brukeren av applikasjonen, samt ber `LogController` om å loggføre feilen. En feil kan fanges opp ved at en metode *thrower* et ErrorHandler objekt som inneholder en beskrivelse av feilen.

16.2.3.2 SomeApplicationOrPluginClass

Før det utføres kode som kan feile så har ErrorHandler blitt varslet om neste mulige feil som kan oppstå (beskrivelse av denne).

Om koden feiler vil en try-catch blokk fange opp et throw og varsle ErrorHandler objektetmed`void Report()`.

16.3 Eksempelkode: Om en feil oppstår

ErrorHandler klassen har ingen direkte innvirkning på håndtering av feilen; den fungerer kun som en felles klasse for å rapportere og lagre feil.
Korrekt håndtering av en feil må gjøres der hvor feilen oppstår. Dette kan for eksempel gjøres slik:

```
// example ”somefile.cpp”
SomeClass someObject = new SomeClass();
try
{
      someObject->someMethod();
}
catch ( ErrorHandler errorHandler )
{
errorHandler.Report();
}

//.h file
#include ”errorhandler.h”
class SomeClass
{
public:
      SomeClass();
      void someMethod();
};

// .cpp file
SomeClass::SomeClass()
{
}
void someMethod()
{
      // Entering code that may fail
      if(x == 0) throw ErrorHandler(“Division by zero”);
}
```

17 REFERANSER

1. **Burbeck, Steve.** How to use MVC. [Internett] [Sitert: 26 3 2012.] http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html.

2. **DB Factory.** *Analyse - Dokumentmodellen.* s.l. : DB Factory, 2012.

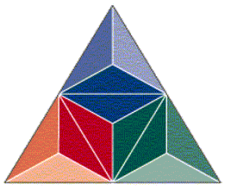
3. —. *Design - Dokumentmodellen.* s.l. : DB Factory, 2012.

4. **Factory, DB.** *Typemapping dokument.* s.l. : DB Factory, 2012.

5. Nokia. [Internett] [Sitert: 26 Januar 2012.] http://developer.qt.nokia.com/doc/qt-4.8/sql-types.html.

IMPLEMENTASJONSDOKUMENT

HIBU STUDENTPROSJEKT 2012
CYTRAX DB FACTORY



HØGSKOLEN
i Buskerud

cytrax

REVISJONSHISTORIKK		
Versjon	Dato	Endringer
1.0	28.5.2012	Første revisjon

INNHOLDSFORTEGNELSE

1 INTRODUKSJON

Dokumentet beskriver implementering og aspekter rundt vårt arbeid med å implementere designet i applikasjonen. Det beskrives hvordan vi har jobbet med å sette sammen applikasjonen, videreutviklet designet og benyttet oss av prototypene våre. Dokumentasjonen skal også gi sporbarhet tilbake til analyse- og design.

Det er forsøkt å dele opp i avsnitt, hvor hvert avsnitt tar for seg et tema i applikasjonen vår. Under avsnittet for prototyper kan dette for eksempel være Qt GUI, mens i avsnittet for implementering kan dette være Generator for VB.NET.

Avsnittene som er kalt **Utvikler(e),** forklarer hvem som har implementert den aktuelle funksjonaliteten. Utviklere markert med fet skrift har hatt hovedansvaret for implementeringen.

Alle som har vært ansvarlige for en implementering har skrevet om denne implementeringen. Derfor er dokumentet forfattet av oss alle.

1.1 FORFATTER

- **Tor-Christian H. Eriksen (TCE)**
- Alexander Edland (AE)
- Jarle Didriksen (JD)
- Jørgen Grøndal (JG)
- Simen Russnes (SR)
- Vegard Kaasin (VK)

2 PROTOTYPER

For å kunne komme frem til en god design som inneholder gjennomtenkte løsninger og funksjonalitet som vi vet fungerer har vi utviklet noen små prototyper. Prototypene skulle avdekke funksjonalitet og eventuelle utfordringer/begrensninger. Derfor var det ikke meningen å implementere dem direkte inn i applikasjonen.

2.1 QT PLUGINS

2.1.1 Hensikt

Cytrax ønsket modularitet i applikasjonen, og ville ha en mulighet for å kunne utvide applikasjonen ved et senere tidspunkt. Disse utvidelsene burde komme som ”tilleggspakker” for applikasjonen og en person i Cytrax skal kunne utvide applikasjonen uten å måtte endre kildekoden i selve programmet.

Vi var klar over muligheten for å benytte plugins, og at Qt hadde sitt eget API for utvikling og bruk av plugins. Det var derimot ingen av oss som hadde erfaring med utvikling av plugins.

2.1.2 Utvikler(e)

- **Tor-Christian H. Eriksen (TCE)**

2.1.3 Resultat og verdi av prototypen

Prototypen gav oss en forståelse og innføring i hvordan man utvikler, laster inn og benytter seg av plugins i en Qt applikasjon. Applikasjonen brukte en plugin for å

importere tekst fra fil, og en annen plugin for å eksportere data til en fil. Pluginene ble lastet inn dynamisk.

Pluginene hadde i forkant blitt laget i to individuelle prosjekter og bygget som dll filer.

Prototypen var vellykket og vi har brukt metodene for å laste inn og klargjøre plugins i den ferdige applikasjonen vår.

2.2 QT GUI

2.2.1 Hensikt

Vi hadde liten erfaring med å lage GUI i Qt Creator. Så hensikten til prototypen ble å hjelpe oss med å avdekke hvilke komponenter vi kunne benytte oss av i applikasjonen og hvordan de fungerte sammen.

2.2.2 Utvikler(e)

- **Vegard Kaasin (VK)**
- Tor-Christian H. Eriksen (TCE)

2.2.3 Resultat og verdi av prototypen

Vi kasserte noen av prototypene, mens andre ble tatt med videre og inn i applikasjonen. Prototypen ble påbegynt på av TCE, og kontinuert av VK (som senere har hatt ansvar om å implementere GUI).

Prototypene gav oss en god forståelse av hvordan Qt Creator sin GUI designer fungerer, og vi fikk praktisert en god del. Om man ser bort ifra utforsking av GUI designer og testing av komponenter, hadde prototypen ingen klar målsetting. Videoer på youtube fra brukeren *Voidrealms* (1) ble aktivt brukt for å bygge kunnskap og forståelse.

2.3 IMPORTERING FRA SQL ANYWHERE 12

2.3.1 Hensikt

Importering fra SQL Anywhere 12 skjer via en plugin. Det kan være komplisert og tungtvint å teste output fra pluginen med for eksempel qDebug() (2).

Derfor var det ønskelig å lage en applikasjon lik pluginen i funksjonalitet, men med output og input i konsoll. Prototypen skulle altså gi svar på hvordan man skal utføre innlesing fra SQL Anywhere 12.

2.3.2 Utvikler(e)

- **Tor-Christian H. Eriksen (TCE)**

2.3.3 Resultat og verdi av prototypen

Den første utfordringen kom i form av et krav fra Cytrax, det var ikke ønskelig å benytte seg av ODBC drivere for tilkobling mot database. Cytrax leverte en testdatabase som kunne brukes for testing.

Det ble benyttet native bibliotek, levert av Sybase (3), for å koble opp mot databasen. Biblioteket er skrevet i C, men det er fullt mulig å benytte seg av det i C++.

Det ble testet at man fikk koblet seg opp mot databasen, og hentet ut tabeller i databasen basert på spørringer mot systemtabeller. Prototypen gav gode svar på hvordan å koble seg opp, hente ut feilkoder, kjøre spørringer og behandle resultatsett fra spørringer i pluginen.

Tilkobling mot database, og spørring for å hente ut tabeller ble tatt videre inn i pluginen som leveres med applikasjonen vår.

3 DOKUMENTERING AV KODE

All kode er dokumentert med kommentarer i kildekoden. Kommentarene er formattert slik at de kan benyttes av Doxygen (4), og dokumentasjon av kode ligger på CD-en under mappen Doxygen. Dokumentasjonen kan leses av alle moderne nettlesere.

4 IMPLEMENTERING

Dokumenter og arbeid i analyse- og designfasen ble tatt videre og la grunnlag for implementeringen. Implementeringen foregikk gjennom flere sprinter, og ved flere tilfeller har vi begynt på implementeringen før designet har vært ferdig. I noen få tilfeller har vi gått tilbake og utbedret eller redesignet.

4.1 IMPORTERING FRA SQL ANYWHERE 12

Implementeringen var tidkrevende og til tider komplisert, mye tid ble spart ved å benytte prototypen for å teste kode og spørringer, før de ble tatt inn i prosjektet for pluginen.

4.1.1 Foreliggende dokumentasjon og ressurser

- Analyse (5) og design (6) av API for innlesing.
- Spørringer fra Cytrax ble brukt for å hente ut nødvendig informasjon, disse var tilgjengelige via github (7).

4.1.2 Utvikler(e)

- **Tor-Christian H. Eriksen (TCE)**

4.2 GENERATOR FOR VB.NET

Ved utvikling av generatoren for VB.Net var det viktig at den skulle produsere lik kode som den eksisterende generatoren gjorde. Vi fikk tilgang på kildekoden til generatoren som Cytrax har i dag, og dette gjorde prosessen mye enklere.

Siden dokumentmodellen i applikasjonen vår er representert på en annen måte måtte mye av koden skrives om. Omskrivingen var ganske rett frem, og etter hvert som vi hadde behov for å få tilgang på forskjellig informasjon ble wrapperen utvidet med ønsket funksjonalitet.

Når vi begynte på testing av applikasjon fant vi ut at vi ikke hadde noen måte å finne ut om generatoregenskaper allerede var lagt til eller ei. Vi valgte derfor å utføre en endring i API for generator, dette var at StoreGeneratorSettings() skal returnere en bool. Denne bool skal representere om det ble oppdatert noen verdi i dokumentmodellen, sånn at bruker kan endre egenskaper før generering dersom ønskelig.

Avsluttende for implementering var å få den til å laste i programmet, generere kode og gjøre den genererte koden tilgjengelig i applikasjonen.

4.2.1 Foreliggende dokumentasjon og ressurser

- Analyse (5) og design (6) av API for generering.
- Tidligere generert kode fra Cytrax sin eksisterende kodegenerator er brukt for å verifisere output fra generator.

4.2.2 Utvikler(e)

- **Jørgen Grøndal (JG)**

4.3 DOKUMENTMODELL

Dokumentmodellen beskriver hvor vi ønsker å lagre all informasjonen som er knyttet til et prosjekt. Dette inkluderer prosjektinformasjon, databaseinformasjon (Med tabeller, kolonner, foreignkeys og views), i tillegg til informasjon om plugins og typemapping.

Den opprinnelige dokumentmodellen besto av en rekke klasser som til sammen dannet applikasjonens modell av blant annet databasen. Denne modellen skulle eksporteres til XML, og det ville da vært nødvendig og konvertere all data til QDom format. Under et planleggingsmøte i slutten av april kom det et forslag fra Jørgen om å kunne representere hele dokumentmodellen som et QDomDocument. Et QDomDocument er en abstraksjon av et xml-dokument. Siden modellen vår hadde en naturlig hierarkisk oppbygning ble gruppen enige om at dette kunne fungere bra.

Klassene ble altså tatt bort fra designet, men selve strukturen de dannet ble beholdt. For å kunne bruke et QDomDocument på en effektiv måte måtte vi bygge funksjonalitet for å aksessere, endre, legge til og fjerne informasjon. Dette er bygget inn i en klasse vi har kalt DocumentModelWrapper. I tillegg må vi ha funksjonalitet for å la plugin som håndterer innlesing få generert nodene til dokumentet på en god måte. Dette har vi laget factory-klasser for.

Det nye designet kan leses om i designdokumentet (6).

4.3.1 Foreliggende dokumentasjon og ressurser

- Analyse (5) og design (6) av dokumentmodell.

4.3.2 Utvikler(e)

- **Simen Russnes (SR)**
- Jørgen Grøndal (JG)
- Alexander Edland (AE)

4.4 WRAPPER

Wrapperen sin oppgave er å gi applikasjonen vår tilgang til dokumentmodellen med metoder for oppretting, endring og henting av informasjon. Klassen ble først laget basert utelukkende på designdokumentet. Etter hvert som de andre delene av applikasjonen ble ferdig fant vi fort ut at det var behov for utvidet funksjonalitet. Wrapperen har derfor igjennom implementering gått igjennom mange endringer.

4.4.1 Utvikler(e)

- **Simen Russnes (SR)**

4.5 FACTORIES

En factory inneholder funksjonalitet for å kunne opprette et *QDomElement* (8). Hver definisjonsfil i dokumentmodellen har sin tilhørende factory. En factory blir brukt når man oppretter dokumentmodellen i en importplugin.

Factories har vært i kontinuerlig utvikling gjennom implementeringsfasen. Behovet for factories oppstod når vi forflyttet oss fra den klassebaserte dokumentmodellen til *QDomDocument*. Hver factory sørget for at nødvendige noder ble opprettet og satt inn i korrekt rekkefølge.

I utgangspunktet ville factoriene også ta seg av plassering av nodene i dokumentmodellen, noe som ble overlatt til kontrolleren for å tillate mer fleksibel sortering av nodene. Senere har factories blitt utvidet med et utvalg standardverdier.

Utbedringer har som alt annet fungert på et on-demand basis, hvor mangler har blitt korrigert ved behov.

4.5.1 Utvikler(e)

- **Alexander Edland (AE)**

4.6 KONTROLLERE

4.6.1 Foreliggende dokumentasjon og ressurser

Analyse (5) og design (6) av kontroller.

4.6.2 Applikasjonskontroller

MVC-Mønsteret deler opp logikk, visning og datamodell. All funksjonalitet som GUI ønsker å utføre mot datamodellen eller andre deler av applikasjonen skal ligge i en egen kontroller. For enklere å kunne organisere de forskjellige typene av kontrollere vi har valgte vi å ha ett toppnivå som inneholder de andre kontrollerne.

Når applikasjonskontrolleren blir laget vil opprette alle de andre kontrollerne, og tilby applikasjonen Get metoder for å hente ut disse.

Vi synes at dette gir en mer ryddig måte å få tak i logikk for GUI på, da man kun trenger å ha en referanse til applikasjonskontrolleren.

4.6.2.1 Utvikler(e)

- **Jørgen Grøndal (JG)**
- Jarle Didriksen (JD)

4.6.3 Modellkontroller

Modellkontrolleren er den komponenten som er ansvarlig for å holde dokumentmodellen oppdatert etter at det har skjedd handlinger i GUI-et.

Disse handlingene er for det meste at man legger til informasjon eller at man endrer eksisterende informasjon. Denne informasjonen er gjerne prosjekt/databaseegenskaper, og man kan også legge til eller endre tabeller, kolonner, plugins og foreign key relations.

I de fleste tilfellene er det wrapperen som utfører selve handlingene som endrer informasjonen, og dette vil i praksis si at modellkontrolleren viderefører kallene til wrapperen. Dette er fordi vi følger MVC-mønsteret. Dette betyr også at kontrolleren var under stadig oppdatering etter at det skjedde endringer i wrapperen.

4.6.3.1 Utvikler(e)

- **Jarle Didriksen (JD)**
- Jørgen Grøndal (JG)

4.6.4 Pluginkontroller

Pluginkontrolleren er ansvarlig for å håndtere plugins som skal brukes i applikasjonen.

Kontrolleren skal laste inn alle plugins i applikasjonen slik at de faktisk kan brukes. Disse blir lagret i hver sin liste basert på om de er for innlesing eller for generering.

Den skal også kunne kjøre metodene til en plugin, og har derfor metoder for å lese fra en database og for å generere kode. Det er også mulig å sammenligne en eksisterende dokumentmodell med den innleste informasjonen dersom det er ønskelig.

4.6.4.1 Utvikler(e)

- **Jarle Didriksen (JD)**
- Tor-Christian H. Eriksen (TCE)
- Vegard Kaasin (VK)

4.6.5 XMLkontroller

XMLkontrolleren sitt ansvar er å håndtere importering og eksportering av eksterne XML-filer. Noen enkle prototyper ble laget i forkant av implementeringen for å teste I/O-operasjoner i Qt, og videoer av *Voidrealms* (1) ble brukt for å bygge kunnskap på området.

Håndtering av XML-filer er ønskelig når vi lager et nytt prosjekt, lagrer et prosjekt, åpner et eksisterende prosjekt eller når vi skal laste inn typemapping-filer. Selve I/O-operasjonene utføres av en annen klasse som heter *XmlHandler*.

Når man importerer en XML-fil så skjer selve importeringen ved å kopiere alt innholdet i XML-fila over i et *QDomDocument* (8). Ved eksportering så strømmes alt innholdet i et *QDomDocument* over til en XML-fil. Det er også mulig å sammenligne to dokumentmodeller etter innlesing dersom det er ønskelig.

Oppretting av nytt prosjekt blir også håndtert i XMLkontrolleren, som da vil sette en ny dokumentmodell og generere en adresse til der prosjektet eventuelt skal lagres.

4.6.5.1 Utvikler(e)

- **Jarle Didriksen (JD)**
- Vegard Kaasin (VK)

4.7 SAMMENLIGNE ET DOKUMENT MED ET ANNET

Muligheten til å kunne sammenligne et dokument med et annet var noe Cytrax ønsket. Dette skulle hovedsakelig bli brukt for å lese inn en oppdatert versjon av en database, kunne se hvilke endringer som var foretatt, og godta eller droppe endringene.

For å løse dette var planen at vi skulle fylle opp et QDomDocument med endringene samt en beskrivelse av typen differanse. Etter at Alexander hadde implementert en versjon som gjorde dette fant vi fort ut at det ble veldig vanskelig å vise endringer, la bruker bestemme hva som var ønskelig, og deretter laste dette inn igjen. Årsaken til dette var at en differanse på et nivå ikke hadde annen informasjon knyttet til seg enn verdien og typen endring.

Jørgen arbeidet videre med en løsning som knyttet GUI og sammenligning tettere sammen. Denne løsningen baserer seg på å vise differansene i et QTreeView. Et slikt view populeres av en modell som består av QStandardItem.

Løsningen vi kom frem til var å lage en subclass av QStandardItem som vi kaller for ComparatorStandardItem. Den avviker ikke mye fra QStandardItem men inneholder metainformasjon som ikke kunne lagres i QStandardItem. Denne informasjonen var en peker til gammel og ny QDomNode som inngår i en differanse, samt en integer som definerer typen differanse. Basert på typen differansen blir visningstekst, farge på elementet og oppdatering forskjellig.

Når dette var på plass ble sammenligningen mye enklere. Nå vil den sammenligne hvert hierarkisk nivå og finne verdier som er forandret og tabeller og kolonner som er lagt til eller fjernet.

4.7.1 Foreliggende dokumentasjon og ressurser

- Designdokumentet (6)

4.7.2 Utvikler(e)

- **Jørgen Grøndal (JG)**
- Alexander Edland (AE)

4.8 HELP FUNKSJONALITET

Implementeringen begynte med en rask undersøkelse av hvordan vi kunne presentere, i applikasjonen, brukerveiledningene vi hadde produsert i forkant. Vi ønsket at en bruker av applikasjonen skulle kunne navigere gjennom de forskjellige dokumentene ut ifra en kategorisering.

Applikasjonen leter gjennom en help mappe, og lager en katalog basert på mappestrukturen og html filer.

Om man i etterkant ønsker å legge inn flere hjelpefiler, kan man gjøre dette ved å lage en passende mappe under help mappen og legge inn html filen der.

4.8.1 Foreliggende dokumentasjon og ressurser

Ingen analyse eller design dokumenter forelå i forkant av implementeringen.

Innholdet i hjelpefilene ble generert ut ifra eksisterende dokumenter. De genererte filene er *html* filer.

4.8.2 Utvikler(e)

- **Tor-Christian H. Eriksen (TCE)**

4.9 ERRORHANDLER

Implementeringen startet med å konstruere klassene som skulle inngå i håndteringen av feil. Deretter ble det testet at ErrorHandleren (6) fungerte som tiltenkt.

Implementeringen gikk ganske raskt, og bøy på få problemer.

4.9.1 Foreliggende dokumentasjon og ressurser

- Designdokument for feilhåndtering (6).

4.9.2 Utvikler(e)

- **Tor-Christian H. Eriksen (TCE)**

4.10 LOGGER

Loggeren ble implementert før rutinene for feilhåndtering, dette fordi feilhåndteringen var avhengig av en ferdig implementert logger.

Det første designet tok ikke hensyn til Model-View-Controller (MVC) (9), og ble derfor redesignet til å bestå av en egen kontroller som styrer kommunikasjon mellom GUI (view), og loggen (modell).

Det ble også avdekket behov for noen ekstra signaler og metoder, som ble implementert fortløpende. Dette for å spare antall kall mot kontrolleren fra GUI'et.

4.10.1 Foreliggende dokumentasjon og ressurser

- Designdokument for logging (6).

4.10.2 Utvikler(e)

- **Tor-Christian H. Eriksen (TCE)**

4.11 GENERERING VIA KOMMANDOLINJE

Cytrax ønsket å generere kode via kommandolinje for å slippe å starte opp applikasjonen om en prosjektfil allerede eksisterte for genereringen.

Det ble implementert støtte for å generere fra en angitt XML fil, og å angi en mappe å generere til. XML filen som angis sjekkes, slik at man vet at filen er en gyldig XML fil.

4.11.1 Foreliggende dokumentasjon og ressurser

Ingen analyse- eller designdokumenter forelå før implementering ble påbegynt.

4.11.2 Utvikler(e)

- **Tor-Christian H. Eriksen (TCE)**

4.12 GUI

Gui var det første som ble påbegynt av koding i applikasjonen. Etter å ha prototypet selve skallet på applikasjonen gikk dette ganske raskt å utvikle. Dette består av alt i hovedvinduet på applikasjonen. Etter å ha utviklet hovedvinduet fant Vegard en god måte for å integrere dokumentmodellen i Qt's eksisterende biblioteker slik at det ble enkelt å populere widgets/views med modellen vi allerede hadde.

Det ble tidlig satt fokus på implementering av importering og generering for å kunne starte å teste funksjonalitet opp mot GUI'et. Videre ble det laget funksjonalitet som endring av typemapping, oppretting av prosjekter, lagring og åpning av filer, help og about funksjon.

Etter at hovedfunksjonaliteten har kommet på plass har det vært en pågående jobb å utbedre GUI etter testing og behov. Redesigning av GUI er dokumentert i Designdokumentet (6).

4.12.1 Foreliggende dokumentasjon og ressurser

Analyse (5) og design (6) av for GUI.

4.12.2 Utvikler(e)

- **Vegard Kaasin(VK)**
- Simen Russnes (SR)

5 REFERANSER

1. **Cairns, Bryan.** Voidrealms. [Internett] [Sitert: 24 5 2012.] http://www.voidrealms.com.

2. **Nokia.** Qt Developer Network. [Internett] [Sitert: 24 5 2012.] http://qt-project.org/doc/qt-4.8/qdebug.html.

3. **Sybase.** Sybase. [Internett] [Sitert: 24 5 2012.] http://www.sybase.com/.

4. Doxygen. *Doxygen.* [Internett] [Sitert: 7 12 2011.] http://www.stack.nl/~dimitri/doxygen/.

5. **DBFactory.** *Analysedokument.* s.l. : DB Factory, 2012.

6. **DB Factory.** *Designdokument.* s.l. : DB Factory, 2012.

7. **Cytrax AS.** Github. [Internett] [Sitert: 26 5 2012.] https://github.com/audunhystad/DBFactory2/tree/OldCodeGeneratorFiles.

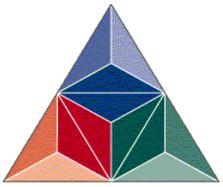
8. **Nokia.** Qt Developer Network. [Internett] [Sitert: 25 5 2012.] http://qt-project.org/doc/qt-4.8/qdomdocument.html.

9. **Steve Burbeck, Ph.D.** Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller(MVC). [Internett] [Sitert: 25 5 2012.] http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html.

SCRUM

METODIKK OG REGLER

HIBU STUDENTPROSJEKT 2012
CYTRAX DB FACTORY



HØGSKOLEN
i Buskerud

cytrax

INNHOLDSFORTEGNELSE

REVISJONSHISTORIKK			
Versjon	Dato	Endringer	Tildelt
1	2.1.2012	Første utgave	TCE / VK
1.1	8.2.2012	Lagt til utfyllende beskrivelse av story points og tidsestimat	TCE
2.0	26.3.2012	Oppdatert til 2. presentasjon	JG

1 INTRODUKSJON

Vår hovedoppgave er gitt av Cytrax AS som er et konsulent- og produktutviklingsselskap. De ønsker en applikasjon som genererer kildekode for blant annet C++ og VB basert på en databasemodell. Denne kildekoden skal benyttes i produkter som Cytrax AS utvikler for sine kunder.

1.1 DOKUMENTETS FUNKSJON

Dokumentet skal gi en introduksjon til Scrum, forklare definisjoner, samt å nevne fordeler og ulemper ved prosjektmodellen. Scrum har fokus på både kunde og utvikler, kunden blir involvert i prosjektet fra start til slutt. Hele utviklingsmiljøet får lov til å ta del i planlegging av prosjektet. Et prosjekt blir delt inn i flere små perioder som kalles sprinter. Scrum er en ”agile software development methodology” som på norsk står for smidig prosjektmodell. Detaljert planlegging gjøres før hver sprint påbegynnes, og kun den neste sprinten planlegges i detalj.

Det er fastsatt egne Scrumregler for prosjektgruppen som har blitt godkjent av alle gruppemedlemmer i DB Factory.

1.2 SYNONYMER, FORKORTELSER OG DEFINISJONER

Synonym	Beskrivelse
Scrum Master	En coach som hjelper teamet å bruke Scrum.
Produkteier	Ansvarlig for *Product Backlog* og leder utviklingen prosjektet.
Team	Medlemmer av utviklingsmiljøet.
Stakeholder	En person som ikke har direkte tilknytning til prosjektet, men har interesse i produktet som utvikles.
Sprint	En avgrenset periode (14-21 ukedager).
Product Backlog	En liste der alle "krav" fra kunden/eier registreres som *user stories.*
User Story	En beskrivelse av ønsket funksjonalitet formulert fra eier/kunde sitt standpunkt.
Task	En oppgave som gjennomføres i løpet av en sprint.

1.3 FORFATTER

Tor-Christian H. Eriksen (TCE)
Vegard Kaasin (VK)

1.4 TILDELT

Tor-Christian H. Eriksen (TCE)
Vegard Kaasin (VK)

2 METODIKK

2.1 ROLLER

I Scrum defineres 3 roller:

- **ScrumMaster**: en veileder som hjelper hele teamet å bruke Scrum mest mulig effektivt, men tar ikke beslutninger.
- **Produkteier**: ansvarlig for *Product Backlog*(se definisjoner) og leder prosjektet. Ikke nødvendigvis fysisk eier av produktet, men skal representere eiers interesser så vel som Scrum-teamets.
- **Team**: medlemmene i utviklingsmiljøet som har en tilknytning til sprinten.

2.2 DEFINISJONER

2.2.1 Product Backlog

Kravliste der alle krav for prosjektet er definert. Kravene blir beskrevet i *Product Backlog* som *User Stories*.

2.2.2 User Story

Forteller en historie(story) om hvordan en kunde eller bruker opplever produktet. En User Story inneholder et navn, kort hva den inneholder, og *akseptansekriterie*(tilstander som må være sanne for at en *User Story* skal være ferdig). Hver Story blir estimert etter prioritet og hvor viktig det er for prosjektet.

De vektlegges også med tanke på kompleksitet.
Estimatet som blir brukt kalles **Story Points**. Det finnes to estimater for *User stories*, *Coarse-Grained* og *Fine-Grained.* Kun stories som er *Fine-Grained* kan bli lagt til i en sprint.

- Coarse-Grained Stories er grove beskrivelser av krav. Coarse-Grained Stories blir ofte kalt *Epics.* En Coarse-Grained Story kan bli omgjort til flere små og enklere Stories.
- Fine-Grained Stories er de aller minste og mest detaljerte Coarse-Grained User Stories som blir brutt ned til mindre Fine-Grained Stories. Fine-Grained Stories har høyest prioritet i en Product Backlog.

2.2.3 Sprint Backlog

User Stories som skal gjennomføres i en sprint tas ut av Product Backlog og legges inn i Sprint Backlog som tasks, dette er *Fine-Grained user stories*.

Sprint Backloggen inneholder et estimat over hvor mange timer som er satt av til hver task, en kort beskrivelse av tasken, status, antall timer brukt og estimert tid igjen, samt ID slik at man kan spore tasken tilbake til Product Backlog.

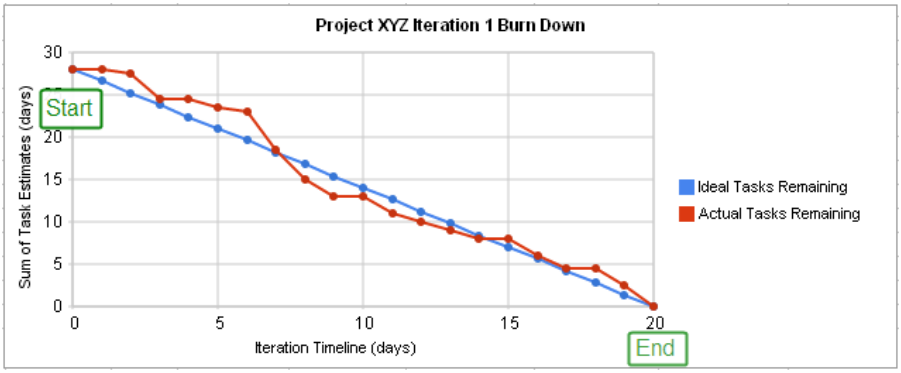
Hele teamet er ansvarlig for at Sprint Backloggen blir oppdatert. Dette er vanlig å gjøre daglig, eller en gang i uken.

2.2.4 Scrum Ban

Modifikasjon av Scrum som brukes som hjelpemiddel, basert på Scrum og Kanban(Rammeverk), for å se hvordan arbeidet i sprinten går. User Stories blir satt opp på en tavle der de enten er; Not started, Ongoing eller Completed.
«Regel»: ingen bør jobbe på mer enn to krav av gangen.

2.2.5 Burn down chart

Grafisk hjelpemiddel for å måle fremdriften i en sprint. Grafen består av Story Points og gjenstående tid. I daglige møter «brennes» det ned på hvor mye arbeid i timer som har blitt gjort. En indikator for hvor mye Teamet kan gjøre i en sprint(Story Points som blir brent ned) blir kalt ***Sprint Velocity****(hastighet).* Hvis Teamet brenner ned 20 Sprint Points, er hastigheten 20.



Figur 1: Scrum Burn Down Chart.

2.2.6 Sprint

Et prosjekt som benytter Scrum rammeverk gjennomføres i **Sprinter**(iterasjoner), på maks èn måned(vanlig er 2 uker). Før en sprint holder man et møte der Produkteier informerer om hvilke User Stories fra Product Backlog han vil ha gjort i denne sprinten. Teamet velger selv User Stories man mener er fornuftige og mulige å gjennomføre for sprinten. User Stories som blir valgt tas ut av Product Backlog og legges i Sprint Backlog. Stories som blir lagt inn i Sprint Backloggen er låst til denne sprinten og kan ikke tas ut eller endres. Hvis man ikke blir ferdig med en Story blir den lagt tilbake i Product Backlog.

2.2.6.1 Sprint Planlegging

Før man kan begynne med å lage Sprint Backlog må man planlegge hver sprint. I sprintplanlegging blir et sprintmål laget. Dette målet hjelper Teamet å vite hva de jobber mot i hver sprint. Det hjelper også Teamet å ta ut like User Stories til hver sprint så man kan lettere hjelpe hverandre å jobbe sammen. Det gjør det også lettere å kommunisere med utenforstående så de vet hva som foregår i hver sprint.

I sprintplanleggingen blir også relevante User Stories redefinert til mindre Fine-Grained Stories.

I noen tilfeller er ikke Story Points et godt nok estimat for en User Story. **Planning Poker** er da et hjelpemiddel man kan bruke for å estimere et backlog-krav. Relevante User Stories/Backlog-krav blir rangert individuelt av Teamet. Hvis det til slutt er uenighet i estimater må de som har estimert høyest og lavest argumentere for sine avgjørelser, før det blir rangert på nytt.

2.2.6.2 Under hver Sprint

For at Scrum skal virke mest mulig effektivt må man ha korte(15 min)daglige møterder alle medlemmene er tilstede. Det er opp til teamet å bestemme hva man ønsker å benytte møtet til, det er viktig at disse reglene følges. Mer om dette møtet finnes under avsnitt for regler.

Scrum Master har som oppgave å innkalle til og sørge for at møtene blir holdt hver dag.

Hvis man jobber i flere Team har man også daglige møter(Scrum of Scrums) der min 1 person fra hvert Team deltar. Her skal de samme tingene som i det daglige Team-møtet diskuteres, dette er ikke aktuelt i vårt tilfelle.

2.2.6.3 Etter hver Sprint

Etter en sprint har man et møte der man oppsummerer sprinten(**Sprint Review**):

- Hva ble gjort/ikke gjort?
- Presentasjon av hva som har blitt gjort
- Product backlog blir oppdatert

Det blir også holdt en **Sprint Retrospective** som kan sammenliknes med(eller inkluderes i) Sprint Review der alle i Teamet sier noe om hva som gikk bra/ikke bra gjennom sprinten.

Etter en sprint er «Definition of Done» at hver User Story skal være:

- Ferdig kodet
- Ferdig testet
- Kan implementeres i systemet.

3 REGLER

For at en prosjektgruppe som benytter Scrum skal fungere optimalt er det viktig å ha egendefinerte regler for hvordan man skal benytte Scrum. Det er ikke nok å kun følge ”maler” som man kan finne i bøker og litteratur på internett. Alle reglene definert i dette dokumentet er Scrum-relaterte regler.

3.1 HVOR GJELDER REGLENE

Reglene i dette dokumentet gjelder for alle medlemmer av prosjektgruppen DB Factory. Det er viktig at reglene blir fulgt for å optimalisere arbeid/samarbeid.

3.2 ENDRINGER I REGLER

Det er tillatt å endre på reglene. På slutten av hver sprint holdes det vi kaller et *Sprint debriefing* møte. Her diskuteres den foregående sprinten, og alle forslag for endringer i regler skal diskuteres her. Regler skal IKKE endres i en Sprint.

For at en regel skal endres, evt. en ny regel introduseres, skal Scrum Master være enig i endringene. Det er også viktig at hele prosjektgruppa forstår regelen og hvorfor den var endret/introdusert.

3.3 SPRINT-PLANLEGGINGSMØTER

Et sprintplanleggingsmøte har en tidsramme på 8 timer. Møtet skal konsumere en virkedag. Møtet består av to deler, begge på 4 timer hver. De første 4 timene er avsatt til å plukke ut user stories i Product Backlog, og dele disse opp i mindre user stories. De siste 4 er avsatt til å bestemme hvilke *Fine-Grained User Stories* som tas med i neste Sprint Backlog. Om det ikke er estimert story points og/eller tidsestimat for tasker må dette gjøres i henholdsvis del 1 og del 2 (1).

Vi har valgt å definere at én story point er en optimal arbeidsdag. Dette fører til at en user story med vekt 1 ikke nødvendigvis tar en dag å utføre, men for eksempel to dager om timene spres over dagene.

3.3.1 Generelt om møtet

Deltakere i møtet er Produkteier, Scrum Master og teamet. I vårt tilfelle betyr dette alle medlemmene av prosjektgruppa.
Andre personer kan inviteres ved behov, men skal ikke være tilstede lengre enn nødvendig.

Produkteier må klargjøre og populere Product Backlog i forkant av møtet. Om Product Backlog ikke er klar eller ikke tilgjengelig på møtet, er det Scrum Master sitt ansvar å legge frem en midlertidig Product Backlog som kan fungere som substitutt.
Om Produkteier ikke er tilstede skal Scrum Master tre inn dens fravær.

3.3.2 Første del

I den første delen av møtet skal teamet velge ut deler av Product Backlog som de mener at bør inngå i neste sprint. Etter sprinten er ferdig skal disse delene kunne presenteres til Produkteier som en fungerende komponent.

Teamet er ansvarlig for å gi et godt estimat over hvor mye av Product Backlog de kan fullføre i den neste sprinten.

Man skal ikke gå over de avsatte timene for den første delen av møtet. Videre diskusjon og analyse av Product Backlog må gjøres under sprinten. En upresis og kompleks oppgave i Product Backlog kan føre til at teamet ikke klarer å gi et godt estimat. Om en

oppgave ikke blir fullført i en sprint skal den føres tilbake i Product Backlog og tas med i neste sprint.

Teamet kan komme med forslag med hva de ønsker å gjennomføre i neste sprint. Produkteier bestemmer dog hvilke deler av Product Backlog som skal med.

3.3.3 Andre del

Produkteier må være tilgjengelig for teamet under denne delen for å besvare eventuelle spørsmål.

Det kun opp til teamet å avgjøre hvordan de ønsker å bygge opp den neste Sprinten. Teamet har ansvar for å bygge den opp slik at på slutten av Sprinten kan man presentere en rekke funksjonelle og testbare deler.

Resultatet av møtet skal være en Sprint Backlog. Dette er en liste over oppgaver, tidsestimater, beskrivelse av oppgavene og tildeling. Sprint Backloggen er grunnlaget for teamets arbeid, alle oppgaver i Sprint Backloggen føres opp på egne post-it lapper og henges opp på et whiteboard med tidsestimat, taskID og en kort beskrivelse.

Om Sprint Backloggen ikke er fullført innen de 4 timene avsatt må den i minste grad være utformet slik at alle medlemmer av teamet kan plukke ut oppgaver og starte arbeidet sitt. I et slikt tilfelle, avsettes nødvendig tid neste dag til å fullføre Sprint Backloggen. Produkteier trenger ikke å være tilgjengelig neste dag, så spørsmål bør være avklart på det første møtet.

3.4 DAGLIG SCRUM MØTE

Hver dag skal man starte dagen med et Scrum møte. Dette møtet har en rekke klare retningslinjer som alltid skal følges:

- En tidsramme på 15 minutter.
- Alle deltakere skal stå (ingen setter seg ned).
- Møtet skal holdes på samme sted til samme tid hver dag.
- Alle deltakere må være presise, Scrum Master starter møtet til avtalt tid selv om man mangler deltakere.
- Scrum Master starter møtet ved å gi personen til venstre mulighet til å snakke, man fortsetter så med klokka til alle har svart på følgende 3 spørsmål:
    - Hva har du gjort siden forrige møte?
    - Hva skal du gjøre frem til neste møte?
    - Er det noe som hindrer deg i arbeidet og eventuelle avvik i estimert tid?
- Kun en person skal snakke omgangen, det skal ikke være noen form for diskusjoner.
- Scrum Master har ansvar for å oppdatere Sprint Backlog under møtet .

4 SPRINT

En sprint har en tidsramme på 14-21 ukedager. Dager avsatt til sprintplanlegging og debriefing er ikke en del av sprinten.

Ingen utenforstående kan gi råd eller instruksjoner til teamet under en sprint, kun når teamet selv ber om det.

Endringer i Product Backlog er ikke tillatt under selve sprinten. Product Backlog er låst frem til sprinten er fullført.

Om teamet ikke klarer å fullføre alle oppgaver gitt i løpet av en sprint skal det konsulteres med Produkteier om hvilke oppgaver som skal fjernes. Stories fra Product Backlog er prioritert, så stories med høy prioritet skal også bli prioritert først i en Sprint.

Om teamet har ledig tid etter å ha fullført alle oppgaver gitt i en sprint, kan man konsultere med Produkteier om å utvide sprinten fra Product Backlog.

Hvis omstendighetene endrer seg slik at hensikten til en påbegynt sprint faller bort, kan denne sprinten avsluttes av Scrum Master etter ønske fra teamet og/eller Produkteier. En sprint kan falle bort om brukt teknologi viser seg å ikke fungere.

Medlemmene av teamet har følgende administrative ansvar:

- Møte opp på de daglige Scrum-møtene
- Holde Sprint Backlog oppdatert med timeforbruk og estimert tid igjen per oppgave

4.1 SPRINT DEBRIEFING

Sprint debriefing møtet består av to deler. Den første delen er et sprint review møte hvor teamet presenterer arbeid som har blitt fullført i den foregående sprinten. Den andre delen er reservert for å analysere den foregående sprinten. Hensikten med den andre delen er å få avklart hva som gikk bra, og hva som bør forbedres til neste Sprint.

4.1.1 Første del – sprint review møte

- Det er avsatt 4 timer for møtet.
- Hensikten med møte er å presentere ferdig funksjonalitet, definisjonen på ferdig funksjonalitet er at oppgaven(e) som funksjonaliteten omfatter i sprint backloggen kan implementeres i produktet, og er dokumentert.
- Funksjonalitet som ikke har oppnådd ferdig status kan ikke bli presentert, dette må presenteres på neste sprint review møte.
- Møtet starter ved å presentere målet med sprinten, hvilken del av Product Backlog som ble håndtert og hva som er ferdig i Product Backlog.

- Medlemmer av teamet skal presentere funksjonalitet og svare på spørsmål fra ”stakeholders”, samt notere ned endringer som ønskes.
- Mot slutten av møtet kan ”stakeholders” komme med forslag til endringer og hvilken prioritet som ønskes på endringene.
- Produkteier diskuterer med teamet eventuelle endringer i Product Backlog på bakgrunn av tilbakemeldingen.
- Som en avslutting på møtet annonserer Scrum Master når det neste sprint review møte skal finne sted til Produkteier og eventuell ”stakeholders”.

4.1.1 Andre del – sprint retrospective møte

- Den andre delen er avgrenset til 3 timer.
- Deltakere er: Scrum Master, Produkteier og teamet. Produkteier kan selv velge om han/hun skal være tilstede.
- Medlemmer av teamet svarer på to spørsmål:
    - Hva gikk bra på forrige Sprint
    - Hva bør forbedres til neste Sprint
- Teamet diskuterer eventuelle forbedringer.
- Scrum Master skal notere ned alle svar fra teamet.
- Scrum Master sin oppgave er å dokumentere teamet sitt behov samt diskutere mulighetene for implementering av behovene.
- Emner som diskuteres skal være Scrum-relaterte, produktrelaterte forbedringer diskuteres ikke.

5 FORDELER OG ULEMPER

5.1 FORDELER

Oppfordrer til teamwork. Lettere å tilpasse seg endringer i krav/prosjektet. Kunden og sluttbrukeren involveres i større grad i utviklingsprosessen. Det jobbes i et team, der man kan dele oppgaver og i større grad hjelpe hverandre. Iterativ utvikling; man må ikke gjøre alt med en gang.

5.2 ULEMPER

Krav som ikke blir fullført blir lagt til side og kan skape problemer senere. Det er lett å tenke at man vil bli fort ferdig noe som kan føre til dårligere kode standard og dårligere sikkerhet.

6 REFERANSER

1. **DB Factory.** *Metode for utforming av user stories og tasks.* s.l. : DB Factory, 2012.

2. Mountaingoatsoftware. *Mountaingoatsoftware.* [Internett] [Sitert: 18 12 2011.] http://www.mountaingoatsoftware.com/topics/scrum.

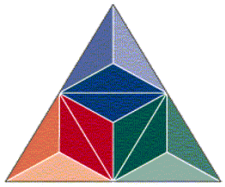
3. ScrumAlliance. *ScrumAlliance.* [Internett] [Sitert: 18 12 2011.] http://www.scrumalliance.org/learn_about_scrum.

4. **Picheler, Roger.** *Agile Product Management With Scrum: creating products that customers love.* s.l. : Addison-Wesley, 2010. ISBN: 978-0-321-60578-8.

5. **Arild Oldervoll, Marius Haraldseth, Eirik André Eidså & Olav Brandt (AMEO).** *ScrumRules.* 2011.

PRODUCT BACKLOG

HIBU STUDENTPROSJEKT 2012
CYTRAX DB FACTORY



HØGSKOLEN
i Buskerud

cytrax

DB Factory 2

Product Backlog, May 29, 2012

sto75	**Som bruker av applikasjonen er det ønskelig å kunne legge til/fjerne tabeller i GUI**	2SP
Story description Ha mulighet for å gjøre dette i GUI, ikke bare sette en tabell til not included.		*Comments*
Acceptance tests Det er mulig å legge til og fjerne tabeller i GUI. Ved lagring av fil blir de utførte handlingene oppdatert til XML-filen.		*Comments*

sto76	**Som bruker av applikasjonen er det ønskelig å kunne legge til/fjerne kolonner i GUI**	2SP
Story description Ha funksjonalitet for dette i GUI.		*Comments*
Acceptance tests Det er mulig å legge til og fjerne kolonner i GUI. Ved lagring av fil blir de utførte handlingene oppdatert til XML-filen.		*Comments*

sto77	**Som bruker av applikasjonen er det ønskelig å kunne høyreklikke på items i QTableWidget for å navigere frem og tilbake**	2SP
Story description Lage right-click menu, koble sammen handlinge rher med items som det trykkes på.		*Comments*
Acceptance tests Når man høyreklikker på en item i tablewidget kommer det opp en meny man kan bruke til å navigere. Når man velger en type navigasjon skal det nye bildet bli lastet.		*Comments*

sto79	**Som bruker av applikasjonen ønskes det å kunne ha flere prosjekter samtidig**	5SP
Story description Implementere slik at man kan ha flere åpne prosjekter i applikasjonen og representere dem med for eksemplem "prosjekt-faner".		*Comments*
Acceptance tests Implementeringen er fullstending. Bruker kan bytte mellom prosjekter og jobbe på dem etter ønske. Implementasjon følger kodestandard.		*Comments*

sto80	**Som bruker ønskes det å kunne importere flere ganger til samme prosjekt.**	3SP
Story description Det bør være mulig å importere flere ganger til samme prosjekt basert på eierskap i databasen. Tidligere valgt eierskap bør lagres slik at man ikke skriver over allerede importert data.		*Comments*
Acceptance tests Implementeringen er fullstending. Ingen data skrives over. Tidligere valgte eiere lagres. Implementasjon følger kodestandard.		*Comments*

sto78	**Som bruker av applikasjonen ønskes det å lese inn databasemodell fra SQLAnywhere10**	8SP
Story description		*Comments*
Acceptance tests Implementering av plugin er fullstendig. Databasemodell leses inn i dokumentmodellen. Unittest av implementasjon gjennomføres uten feil. Implementasjon følger kodestandard.		*Comments*

sto27	Som bruker av applikasjonen ønskes det å lese inn databasemodell fra MS SQL	13SP
Story description		*Comments*
Acceptance tests Implementering av plugin er fullstendig. Databasemodell leses inn i dokumentmodellen. Unittest av implementasjon gjennomføres uten feil. Implementasjon følger kodestandard.		*Comments*

sto28	Som bruker av applikasjonen ønskes det å lese inn databasemodell fra Postgres SQL	13SP
Story description		*Comments*
Acceptance tests Implementering av plugin er fullstendig. Databasemodell leses inn i dokumentmodellen. Unittest av implementasjon gjennomføres uten feil. Implementasjon følger kodestandard.		*Comments*

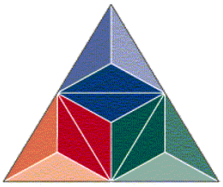
sto31	Som bruker av applikasjonen ønskes det å generere kode for C++	20SP
Story description		*Comments*
Acceptance tests Generert kode er korrekt i henhold til standarder. Implementasjonen benytter pluginsystemet. Implementasjon er fullstendig. Unittest for implemetasjon gjennomføres uten feil. Implementasjon følger kodestandard.		*Comments*

sto29	Som bruker av applikasjonen ønskes det å lese inn databasemodell fra MySQL	13SP
Story description		*Comments*
Acceptance tests Implementering av plugin er fullstendig. Databasemodell leses inn i dokumentmodellen. Unittest av implementasjon gjennomføres uten feil. Implementasjon følger kodestandard.		*Comments*

sto30	Som bruker av applikasjonen ønskes det å lese inn databasemodell fra Oracle	13SP
Story description		*Comments*
Acceptance tests Implementering av plugin er fullstendig. Databasemodell leses inn i dokumentmodellen. Unittest av implementasjon gjennomføres uten feil. Implementasjon følger kodestandard.		*Comments*

SPRINTHISTORIKK

HIBU STUDENTPROSJEKT 2012
CYTRAX DB FACTORY



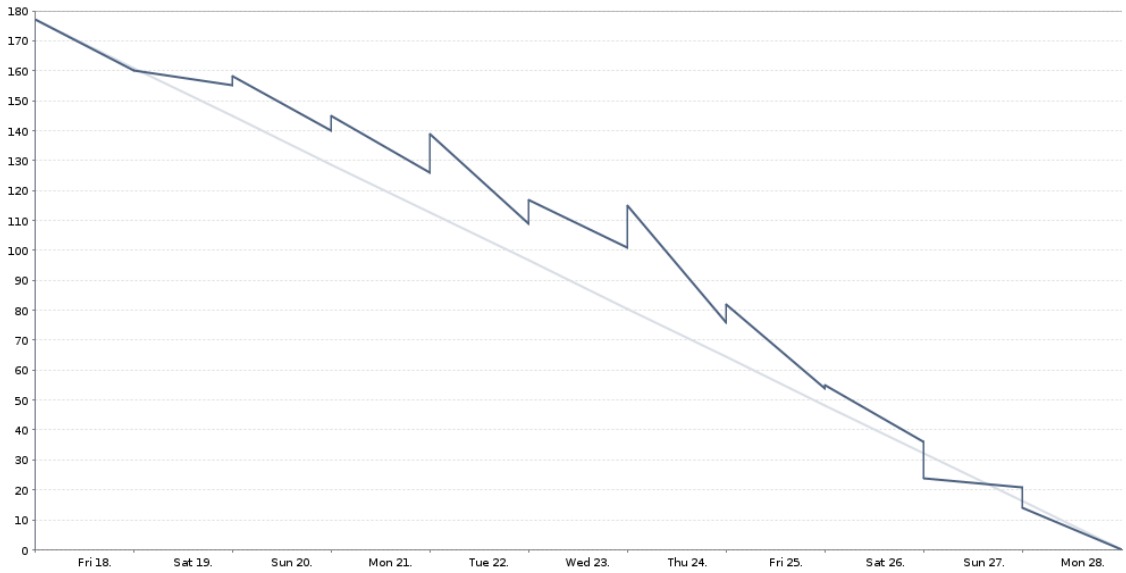
HØGSKOLEN
i Buskerud

cytrax

DB Factory 2

Sprint Report, May 28, 2012

Sprint	*spr6* Sprint 7
Period	Fri, May 18 - Mon, May 28 (10 days)
Velocity	25.0 StoryPoints
Burned work	208 hours
Product Owner	Audun, Jorgen, Tor, Vegard
Scrum Master	Jorgen, Vegard
Team	Simen, Alex, Jorgen, Tor, Vegard, Jarle



Goal

Gjøre ferdig applikasjon med mulighet for å lese inn og generere kode.

Skrive ferdig dokumentasjon for innleveringer.

Completed stories

sto19	**Som utviklere av applikasjonen ønsker vi å implementere visualisering av differanser**	*3 SP*	*32 hrs.*
tsk186 Implementere klasser som utfører dette *20 hrs.*			
tsk193 Lage GUI for visualisering av differanse mellom dokumenter *12 hrs.*			

sto74	**Som utviklere ønsker vi å klargjøre for deployment av applikasjonen**	*2 SP*	*13 hrs.*
tsk217 Gjør så generator produserer lik kode som tidligere generator *8 hrs.*			
tsk199 Sette sammen ferdigutviklet applikasjon *5 hrs.*			

sto23	**Som utviklere ønsker vi å implementere systemet for å mappe databasetyper mot type i kodespråk**	*2 SP*	*6 hrs.*
tsk187 Bestemme hvor informasjon om typemapping skal lagres *6 hrs.*			

sto54	**Som utviklere ønsker vi å designe Documentmodel wrapper**	*1 SP*	*5 hrs.*
tsk150 Dokumentere wrapper *5 hrs.*			

sto71	Som bruker av applikasjonen ønskes det å kunne velge tabeller for importering utifra owners	2 SP	6 hrs.
tsk198 Implementere i innlesingsplugin 6 hrs.			

sto73	Som utvikler av applikasjonen ønsker vi å oppdatere designdokumenter	3 SP	27 hrs.
tsk211 Oppdatere UML-diagrammer 5 hrs.			
tsk177 Oppdatere resterende dokumentasjon 7 hrs.			
tsk196 Oppdatere prosjektplan 5 hrs.			
tsk176 Oppdatere designdokument slik at det følger den nye DomDoc modellen 3 hrs.			
tsk197 Oppdatere testdokument 7 hrs.			

sto26	Som bruker av applikasjonen ønskes det å lese inn databasemodell fra SQL Anywhere v10	2 SP	3 hrs.
tsk205 Undersøke bakoverkompatibilitet mot SQLA10 3 hrs.			

sto72	Skrive veiledning for utvikling av nye plugins.	1 SP	7 hrs.
tsk201 Skrive ferdig brukerveieledning 7 hrs.			

sto25	Som bruker av applikasjonen skal man kunne generere kode via kommandolinje.	1 SP	10 hrs.
tsk215 Implementere funksjonalitet for å generere kode via kommandolinje 3 hrs.			
tsk202 Implemenetere parameterlogikk i applikasjonen 4 hrs.			
tsk203 Dokumentere bruk av parametere 3 hrs.			

sto24	Som bruker av applikasjonen ønsker man at generering av kode skal være pluginbasert	2 SP	4 hrs.
tsk175 Integrere mulighet for å velge plugins i GUI 4 hrs.			

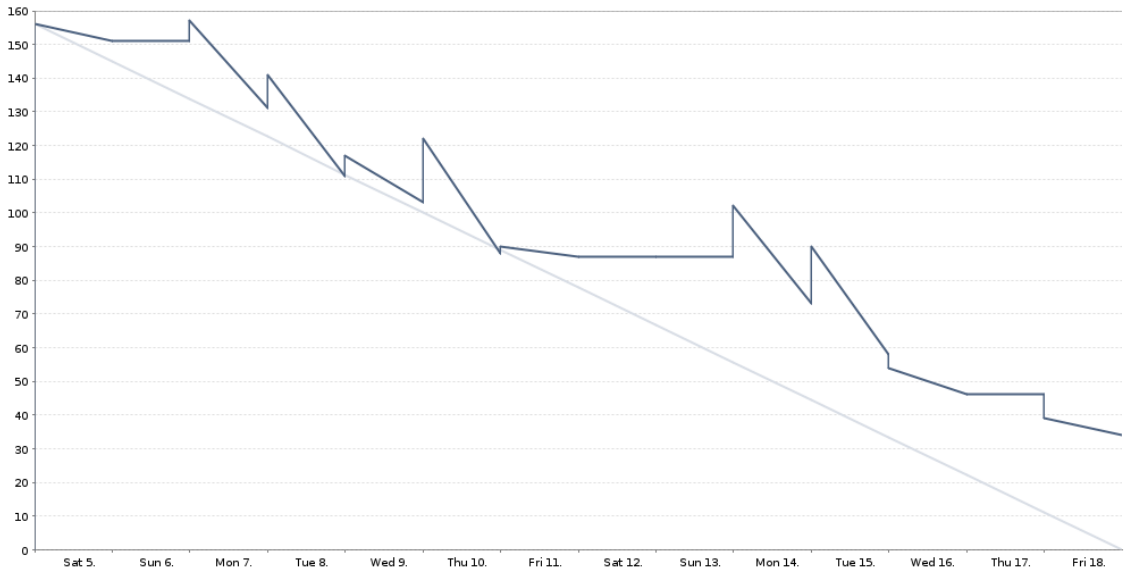
sto70	Som utviklere av applikasjonen ønsker vi å lage gui for applikasjonen	5 SP	85 hrs.
tsk214 Implementere shortcuts i applikasjonen 2 hrs.			
tsk212 Implementere gui for "about" funksjon 3 hrs.			
tsk213 Designe/implementere start side in mainwindow 2 hrs.			
tsk209 Implementere gui for options 2 hrs.			
tsk210 Implementere gui for å hente ut nylig brukte filer 22 hrs.			
tsk158 Gui for endring av typemapping 14 hrs.			
tsk179 Lage ikoner ferdig og implementere funksjonalitet 4 hrs.			
tsk206 Lage QTableWidget main vindu for visning av tabeller og kolonner med relevant info 20 hrs.			
tsk208 Implementere gui for help funksjon 8 hrs.			
tsk207 Implementere gui for generering 8 hrs.			

sto41	Som bruker av applikasjoner ønsker jeg at 0 eller flere generatorer skal kunne kobles til hverenkelt tabell via GUI	1 SP	10 hrs.
tsk194 Implementere at valg av generator populerer elementer i dokumentet 4 hrs.			
tsk195 Lage GUI for masseendring av generator for tabeller etc. 6 hrs.			

DB Factory 2

Sprint Report, May 18, 2012

Sprint	*spr5* Sprint 6
Period	Sat, May 05 - Fri, May 18 (13 days)
Velocity	18.0 StoryPoints
Burned work	186 hours
Product Owner	Audun, Jorgen, Tor, Vegard
Scrum Master	Jorgen, Vegard
Team	Simen, Alex, Jorgen, Tor, Vegard, Jarle



Completed stories

sto59	**Som utviklere ønsker vi å implementere plugin for innlesing av SQL Anywhere 12**	*3 SP*	*20 hrs.*
tsk170 Benytte parset informasjon til å populere dokumentmodellen *11 hrs.*			
tsk171 Integrere plugin i GUI *7 hrs.*			
tsk178 Integrere errorhandler og meldingslogg i plugin *2 hrs.*			

sto58	**Som utviklere ønsker vi å implementere controller**	*2 SP*	*10 hrs.*
tsk162 Implementere controlleren *10 hrs.*			

sto57	**Som utviklere av applikasjonen ønskes det å implementere parsing av systables fra SQL Anywhere 12**	*1 SP*	*3 hrs.*
tsk103 Ferdigstille parsing av informasjon hentet ut databasen *3 hrs.*			

sto55	**Som utviklere ønsker vi å implementere Documentmodel wrapper**	*2 SP*	*12 hrs.*
tsk151 Implementere Wrapper klassene *12 hrs.*			

sto22	**Som bruker av applikasjonen ønskes det å generere kode for VB.net**	*5 SP*	*33 hrs.*
tsk189 Dokumentere oppbygning og utvikling av generator *6 hrs.*			
tsk190 Implementere bruk av errorhandler i plugin *2 hrs.*			
tsk168 Implementere generatorplugin *25 hrs.*			

sto10	Som bruker av applikasjon ønsker man å kunne eksportere dokumentmodell til en prosjektfil i XML	1 SP	10 hrs.
tsk192 Oppdatere design for eksportering 5 hrs.			
tsk169 Implementere eksportering til XML fil 5 hrs.			

sto68	Som utviklere av applikasjonen ønsker vi å implementere factories	1 SP	3 hrs.
tsk181 Implementere at defaultverdi blir lagret når som attributt 3 hrs.			

sto69	Som utviklere av applikasjonen ønsker vi å designe løsning for factories	3 SP	5 hrs.
tsk183 Lage design for factories 4 hrs.			
tsk184 Dokumentere design for designdokument 1 hrs.			

Rejected stories

sto19	Som utviklere av applikasjonen ønsker vi å implementere visualisering av differanser	3 SP	11 hrs.
tsk185 Oppdatere design for QDomDocument 3 hrs.			
tsk186 Implementere klasser som utfører dette 8 hrs.			

sto23	Som utviklere ønsker vi å implementere systemet for å mappe databasetyper mot type i kodespråk	2 SP	6 hrs.
tsk187 Bestemme hvor informasjon om typemapping skal lagres 2 hrs.			
tsk188 Implementere metoder for sjekk av datatype i wrapper 4 hrs.			

sto54	Som utviklere ønsker vi å designe Documentmodel wrapper	1 SP	4 hrs.
tsk150 Dokumentere wrapper 4 hrs.			

sto73	Som utvikler av applikasjonen ønsker vi å oppdatere designdokumenter	1 SP	6 hrs.
tsk177 Oppdatere resterende dokumentasjon 6 hrs.			
tsk176 Oppdatere designdokument slik at det følger den nye DomDoc modellen 0 hrs.			

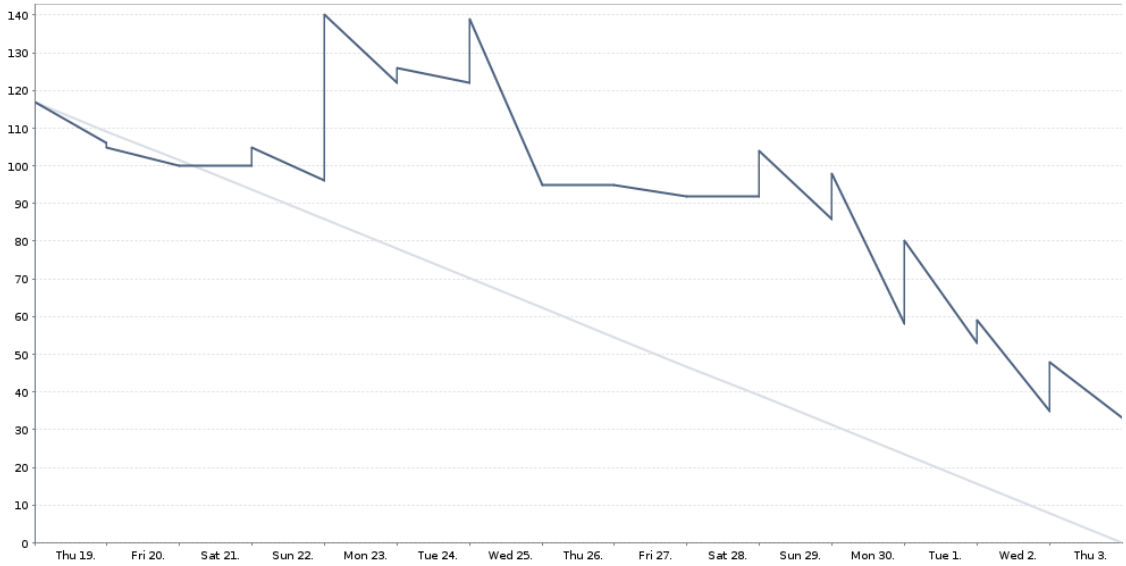
sto24	Som bruker av applikasjonen ønsker man at generering av kode skal være pluginbasert	2 SP	10 hrs.
tsk172 Implementere klasse for å representere importerte plugins 6 hrs.			
tsk173 Benytte prototype til å lese inn aktuelle plugins 0 hrs.			
tsk175 Integrere mulighet for å velge plugins i GUI 4 hrs.			

sto70	Som utviklere av applikasjonen ønsker vi å lage gui for applikasjonen	5 SP	53 hrs.
tsk158 Ending av typemapping 0 hrs.			
tsk179 Lage ikoner ferdig og implementere funksjonalitet 1 hrs.			
tsk153 Lage basissettings i applikasjonen samt mulighet til å legge til, endre og hente ut settings 52 hrs.			
tsk154 Dialog for Sammenlikning av dokumentmodeller 0 hrs.			

DB Factory 2

Sprint Report, May 4, 2012

Sprint	*spr4* Sprint 5
Period	Thu, Apr 19 - Thu, May 03 (14 days)
Velocity	5.0 StoryPoints
Burned work	218 hours
Product Owner	Audun, Tor, Jorgen, Vegard
Scrum Master	Jorgen, Vegard
Team	Simen, Alex, Tor, Jorgen, Vegard, Jarle



Goal

Implementere en fungerende applikasjon, med basisfunksjonalitet. (Ikke ferdige plugins)

Completed stories

sto64	**Som utviklere ønsker vi å implementere et system for feilhåndtering**	*1 SP*	*16 hrs.*
tsk141 Oppdatere dokumentasjon slik at den følger siste design for feilhåndtering og logging *2 hrs.*			
tsk147 Integrere logging of feilmeldinger i GUI *7 hrs.*			
tsk144 Lage kode for feilhåndtering og meldingslogging *7 hrs.*			

sto52	**Som utviklere av applikasjonen ønsker vi å designe system for hvilke tabeller som skal bruke hvilke generatorer**	*1 SP*	*1 hr.*
tsk156 gjøre endringer i analyse/designdokument og dokumentere dette *1 hrs.*			

sto67	**Som utvikler av applikasjonen ønsker vi å endre design av typemapping til å følge QDomdocument løsning**	*1 SP*	*1 hr.*
tsk157 Gjøre endringer i analyse/designdokument og dokumentere dette *1 hrs.*			

sto2	**Som utviklere ønsker vi å designe dokumentmodellen**	*2 SP*	*6 hrs.*
tsk145 Oppdatere dokumentasjon sånn at det reflekterer det nye designet *2 hrs.*			
tsk143 Lage design for ny dokumentmodell *4 hrs.*			

Rejected stories

sto58	Som utviklere ønsker vi å implementere controller	5 SP	15 hrs.
tsk162 Implementere controlleren 10 hrs.			
tsk161 Designe controlleren 4 hrs.			
tsk160 Analysere hva controlleren gjør for systemet 1 hrs.			

sto57	Som utviklere av applikasjonen ønskes det å implementere parsing av systables fra SQL Anywhere 12	3 SP	42 hrs.
tsk105 Implementere funksjon for å koble opp mot SQL Anywhere 12 6 hrs.			
tsk106 Implementere innhenting av nødvendige systemtabeller 18 hrs.			
tsk103 Implementere plugin til å fungere i applikasjonen 0 hrs.			
tsk108 Parse innhentet informasjon fra systemtabeller 18 hrs.			

sto55	Som utviklere ønsker vi å implementere Documentmodel wrapper	3 SP	16 hrs.
tsk151 Implementere Wrapper klassene 16 hrs.			
tsk152 Lage unit tests for wrapper klassene 0 hrs.			

sto22	Som bruker av applikasjonen ønskes det å generere kode for VB.net	2 SP	22 hrs.
tsk168 Implementere generatorplugin 22 hrs.			

sto54	Som utviklere ønsker vi å designe Documentmodel wrapper	3 SP	9 hrs.
tsk149 Lage design for wrapper 6 hrs.			
tsk148 Avklare hvordan wrapper skal fungere 3 hrs.			
tsk150 Dokumentere wrapper 0 hrs.			

sto10	Som bruker av applikasjon ønsker man å kunne eksportere dokumentmodell til en prosjektfil i XML	1 SP	17 hrs.
tsk167 Prototyping av løsninger 11 hrs.			
tsk169 Se på implementasjon av eksportering av XML 5 hrs.			
tsk70 Lagring av generatoregenskaper 1 hrs.			

sto68	Som utviklere av applikasjonen ønsker vi å implementere factories	3 SP	20 hrs.
tsk163 Implementere factories 20 hrs.			

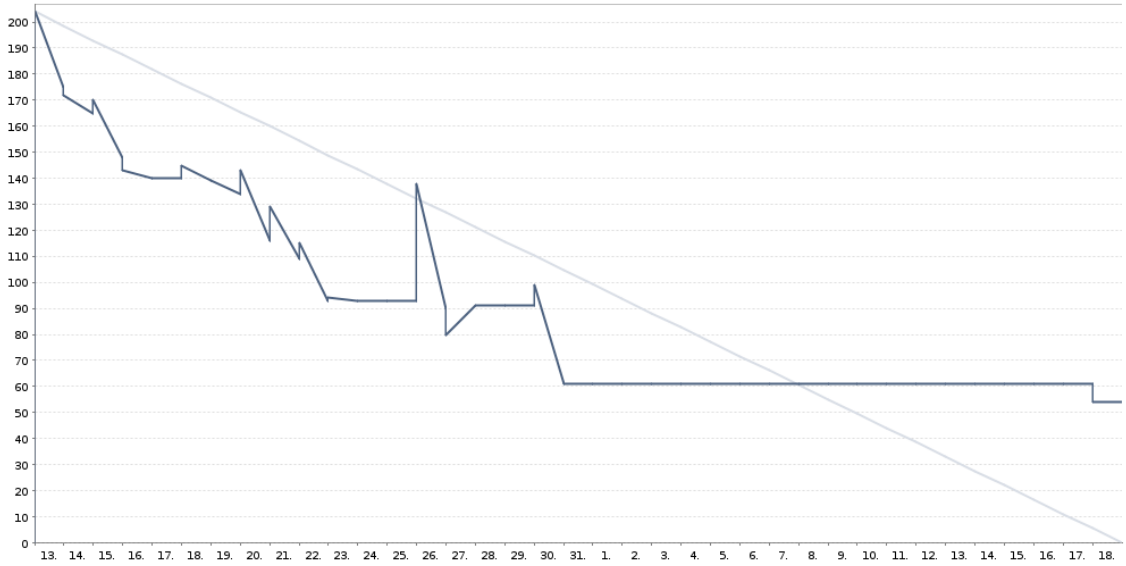
sto69	Som utviklere av applikasjonen ønsker vi å designe løsning for factories	3 SP	9 hrs.
tsk142 Analysere factories 3 hrs.			
tsk146 Designe factories 6 hrs.			

sto70	Som utviklere av applikasjonen ønsker vi å lage gui for applikasjonen	8 SP	44 hrs.
tsk158 Få gui til å kommunisere med dokumentmodell 0 hrs.			
tsk153 Ferdigstille basiskomponenter i gui 44 hrs.			
tsk154 Få gui til å kommunisere med controller 0 hrs.			

DB Factory 2

Sprint Report, April 18, 2012

Sprint	*spr3* Sprint 4
Period	Tue, Mar 13 - Wed, Apr 18 (36 days)
Velocity	25.0 StoryPoints
Burned work	217 hours
Product Owner	Audun, Jorgen, Tor, Vegard
Scrum Master	Jorgen, Vegard
Team	Simen, Alex, Jorgen, Tor, Vegard, Jarle



Goal

2. Presentasjon

Completed stories

sto63	**Som utviklere av applikasjonen ønsker vi å designe et system for feilhåndtering**	*1 SP*	*12 hrs.*
tsk134 Se på hvordan man skal knytte feilhåndtering sammen med applikasjonen og hvordan brukeren skal underrettes om feil *3 hrs.*			
tsk132 Se på muligheter i QT for feilhåndtering *1 hrs.*			
tsk133 Lage UML design *4 hrs.*			
tsk135 Dokumentere design *2 hrs.*			
tsk131 Undersøke hva klassen for feilhåndtering skal inneholde *2 hrs.*			

sto62	**Som studenter ønsker vi å oversende ny og endret dokumentasjon fra sprint 3**	*1 SP*	*6 hrs.*
tsk124 Gå gjennom dokumentasjon som skal overleveres *3 hrs.*			
tsk125 Skrive oppsummeringsdokument for sprint 3 *3 hrs.*			

sto61	**Som studenter ønsker vi å klargjøre dokumentasjon som skal leveres til skolen**	*1 SP*	*32 hrs.*
tsk120 Sette sammen mappe med dokumenter som skal innleveres *2 hrs.*			
tsk119 Kontrollere dokumentasjon mtp. oppsett *27 hrs.*			
tsk121 Sette sammen digital innlevering *3 hrs.*			

sto7	Som utviklere av applikasjon ønsker vi å utvikle API for generering av kildekode	0.5 SP	2 hrs.
tsk100 Utbedre korreksjon fra forrige sprint review 1 hrs.			
tsk104 Implementere interface i kode 1 hrs.			

sto5	Som utviklere av applikasjonen ønsker vi å analysere og designe bruk av plugin for innlesing	3 SP	12 hrs.
tsk60 Analysere tilbakemelding fra utført arbeid 1 hrs.			
tsk62 Dokumentere design 4 hrs.			
tsk61 Designe løsning for innlesning via plugin 7 hrs.			

sto60	Som studenter ønsker vi å forberede fremføringen	8 SP	60 hrs.
tsk116 Lage demoapplikasjon 20 hrs.			
tsk98 Lage powerpoint presentasjon 16 hrs.			
tsk99 Øve på fremvisning 24 hrs.			

sto42	Som utvikler av applikasjonen ønsker vi å designe et system for sammenligning av to dokumentmodeller	1 SP	1 hr.
tsk102 Lage støtte for databasemodell 1 hrs.			

sto43	Som utviklere ønsker vi å designe et system for typemapping maler	0.5 SP	6 hrs.
tsk129 Lage typemapping filer i xml 2 hrs.			
tsk80 Analysere hva dette systemet skal gjøre 1 hrs.			
tsk82 Dokumentere valgt løsning 2 hrs.			
tsk81 Designe system for typemapping maler 1 hrs.			

sto65	Som utviklere ønsker vi å designe et system for å logge meldinger i applikasjonen	1 SP	10 hrs.
tsk136 Undersøke hva klassen for feilhåndtering skal inneholde 2 hrs.			
tsk137 Lage UML design 4 hrs.			
tsk139 Dokumentere design 2 hrs.			
tsk138 Se på hvordan man skal knytte logging sammen med applikasjonen 2 hrs.			

sto53	Som utviklere ønsker vi å implementere Templatemodell	5 SP	19 hrs.
tsk112 Implementere DocumentPropertyList 2 hrs.			
tsk111 Lage Unit test for DocumentProperty 4 hrs.			
tsk114 Implementere AdditionalProperties 2 hrs.			
tsk113 Lage Unit test for DocumentPropertyList 2 hrs.			
tsk110 Implementere DocumentProperty klassen 7 hrs.			
tsk115 Lage Unit test for AdditionalProperties 2 hrs.			

sto14	Som utviklere av applikasjonen ønsker vi å utvikle et design for redigering av egenskaper i dokumentmodellen	3 SP	13 hrs.
tsk86 Analysere hva som skal endres i dokumentmodell 3 hrs.			
tsk87 Designe metoder som skal inn i dokumentmodellen 10 hrs.			

Rejected stories

sto57	Som utviklere av applikasjonen ønskes det å implementere parsing av systables fra SQL Anywhere 12	8 SP	18 hrs.
tsk105 Implementere funksjon for å koble opp mot SQL Anywhere 12 14 hrs.			
tsk106 Implementere innhenting av nødvendige systemtabeller 0 hrs.			
tsk101 Implementere interface for plugin i applikasjonen 4 hrs.			

tsk103 Se på implementasjon i GUI *0 hrs.*	
tsk108 Parse innhentet informasjon fra systemtabeller *0 hrs.*	

sto54	**Som utviklere ønsker vi å implementere Databasemodell**	*5 SP*	*9 hrs.*
tsk122 implementere databaseklassene *9 hrs.*			
tsk123 Lage nødvendige unit tests *0 hrs.*			

sto10	**Som bruker av applikasjon ønsker man å kunne eksportere dokumentmodell til en prosjektfil i XML**	*1 SP*	*3 hrs.*
tsk70 Lagring av generatoregenskaper *0 hrs.*			
tsk69 Tilpasse i henhold til template system *1 hrs.*			
tsk68 Oppdatere foreign keys til å reflektere dokumentmodell *2 hrs.*			

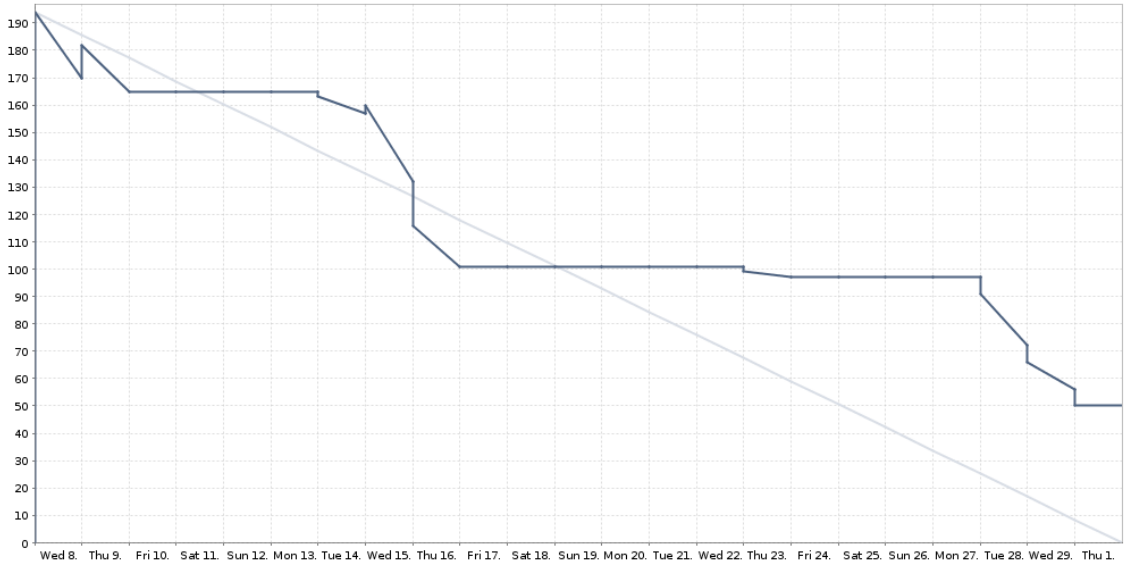
sto52	**Som utviklere av applikasjonen ønsker vi å designe system for hvilke tabeller som skal bruke hvilke generatorer**	*2 SP*	*5 hrs.*
tsk96 Utdype mulige løsninger på design *3 hrs.*			
tsk97 Begrunne løsningen som er valgt *2 hrs.*			

sto2	**Som utviklere ønsker vi å designe dokumentmodellen**	*2 SP*	*9 hrs.*
tsk128 Lage metoder for å gjøre det enklere for plugin å populere dokumentmodell *1 hrs.*			
tsk126 Oppdatere endringer i analyse og designdokumenter *2 hrs.*			
tsk127 Endre dokumentmodell sånn at alle egenskaper i klasser ligger i lister *5 hrs.*			
tsk107 Oppdatere endringer i foreign keys og typemap klassene *1 hrs.*			

DB Factory 2

Sprint Report, March 9, 2012

Sprint	*spr1* Sprint 3
Period	Wed, Feb 08 - Thu, Mar 01 (22 days)
Velocity	19.0 StoryPoints
Burned work	121 hours
Product Owner	Audun, Tor, Jorgen, Vegard
Scrum Master	Jorgen, Vegard
Team	Simen, Alex, Tor, Jorgen, Vegard, Jarle



Goal

Gjøre ferdig design på de viktigste elementene i applikasjonen

Completed stories

sto49	**Som studenter ønsker vi å vedlikeholde verktøy og webside**	*1 SP*	*2 hrs.*
tsk92 Lage gjestefunksjon i versjonskontroll *2 hrs.*			

sto48	**Som studenter ønsker vi å oversende ny og endret dokumentasjon fra sprint 2 til sensorer**	*2 SP*	*20 hrs.*
tsk89 Dokumentere prototyping for bruk av plugins i QT *2 hrs.*			
tsk91 Standardisere datomerking i dokumenter *1 hrs.*			
tsk90 Skrive oppsummeringsdokument for forrige sprint *3 hrs.*			
tsk48 Gå igjennom dokumentasjon som skal oversendes *4 hrs.*			
tsk49 Utrede verktøy for statisk testing av kode + oppdatere testdok. *5 hrs.*			
tsk50 Sette sammen analyse og designdokument *3 hrs.*			
tsk47 Oppdatere prosjektplan *2 hrs.*			

sto6	**Som utviklere av applikasjon ønsker vi å utvikle API for innlesing av databasemodell**	*2 SP*	*4 hrs.*
tsk93 Se over API og vurdere eventuelle endringer *2 hrs.*			
tsk64 Oppdatere design til å følge kodestandard *1 hrs.*			
tsk63 Legge til en ekstra QString i DBConnect som parameter *1 hrs.*			

sto38	Som utviklere ønsker vi å designe et system som tillater plugin å lagre egenskaper i dokumentmodell	3 SP	5 hrs.
tsk78 Designe hvordan egenskapene skal representeres 2 hrs.			
tsk77 Analysere hva et slikt system skal inneholde 3 hrs.			

sto8	Som utviklere av applikasjonen ønsker vi å designe GUI	8 SP	23 hrs.
tsk75 Lage førsteutkast i Qt Creator 9 hrs.			
tsk74 Designe hvordan dokumentmodellen skal presenteres 5 hrs.			
tsk73 Analysere hvilke operasjoner en bruker ønsker å gjøre via GUI 4 hrs.			
tsk72 Analysere hva GUI skal vise brukeren 5 hrs.			

sto1	Som utviklere ønsker vi å analysere dokumentmodellen	3 SP	8 hrs.
tsk53 Analysere hva template baserte egenskaper gjør for systemet 5 hrs.			
tsk52 Legge til / flytte egenskaper 3 hrs.			

Rejected stories

sto7	Som utviklere av applikasjon ønsker vi å utvikle API for generering av kildekode	2 SP	1 hr.
tsk65 Gi plugin tilgang til Dokumentmodell. 1 hrs.			

sto5	Som utviklere av applikasjonen ønsker vi å analysere og designe bruk av plugin for innlesing	3 SP	0 hrs.
tsk60 Analysere tilbakemelding fra utført arbeid 0 hrs.			
tsk62 Dokumentere design 0 hrs.			
tsk61 Designe løsning for innlesning via plugin 0 hrs.			

sto42	Som utvikler av applikasjonen ønsker vi å designe et system for sammenligning av to dokumentmodeller	5 SP	10 hrs.
tsk84 Lage klassediagram for prosessen 1 hrs.			
tsk83 Designe hvordan systemet skal sammenligne to dokumentmodeller 7 hrs.			
tsk85 Dokumentere valgt løsning 2 hrs.			

sto43	Som utviklere ønsker vi å designe et system for typemapping maler	2 SP	5 hrs.
tsk80 Analysere hva dette systemet skal gjøre 2 hrs.			
tsk82 Dokumentere valgt løsning 0 hrs.			
tsk81 Designe system for typemapping maler 3 hrs.			

sto10	Som bruker av applikasjon ønsker man å kunne eksportere dokumentmodell til en prosjektfil i XML	3 SP	12 hrs.
tsk43 Lage sekvensdiagram for generering av xml 5 hrs.			
tsk70 Lagring av generatoregenskaper 2 hrs.			
tsk69 Tilpasse i henhold til template system 3 hrs.			
tsk68 Oppdatere foreign keys til å reflektere dokumentmodell 2 hrs.			

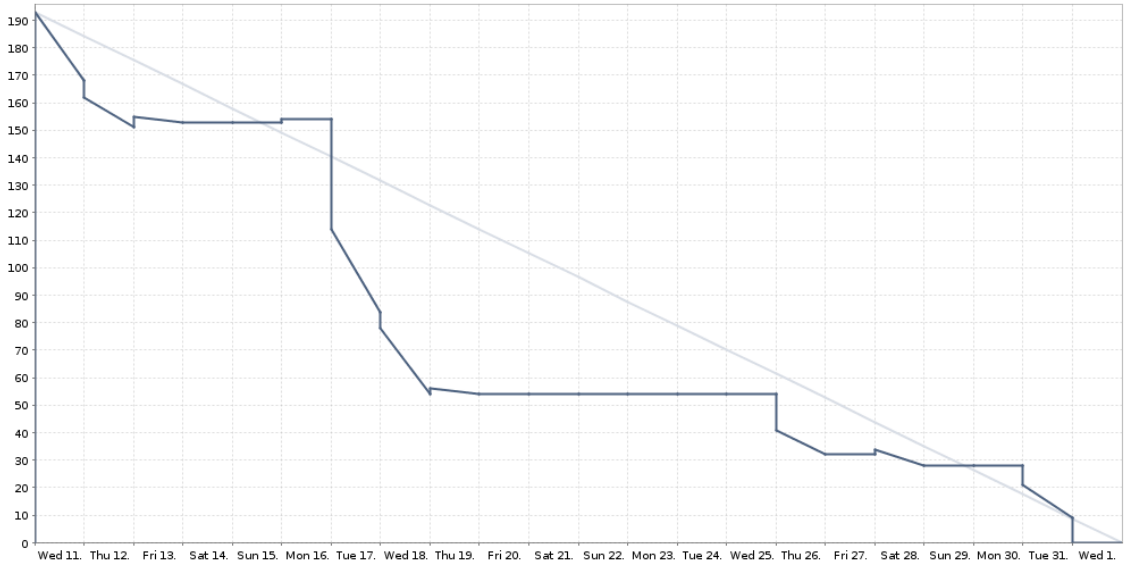
sto2	Som utviklere ønsker vi å designe dokumentmodellen	3 SP	31 hrs.
tsk94 Designe hele dokumentmodell med template og egenskaper 8 hrs.			
tsk56 Gjøre om egenskaper til template egenskaper 4 hrs.			
tsk55 Flytte foreign keys til egen klasse 1 hrs.			
tsk57 Flytte andre egenskaper som er endret i analyse 3 hrs.			
tsk59 Designe template system for egenskaper 13 hrs.			
tsk95 Dokumentere oppdateringer i designdokument 2 hrs.			

sto14	**Som utviklere av applikasjonen ønsker vi å utvikle et design for redigering av egenskaper i dokumentmodellen**	*3 SP*	*0 hrs.*
tsk88 Dokumentere endringer på dokumentmodell *0 hrs.*			
tsk86 Analysere hva som skal endres i dokumentmodell *0 hrs.*			
tsk87 Designe metoder som skal inn i dokumentmodellen *0 hrs.*			

DB Factory 2

Sprint Report, February 1, 2012

Sprint	*spr2* Sprint 2
Period	Wed, Jan 11 - Wed, Feb 01 (21 days)
Velocity	7.0 StoryPoints
Burned work	41 hours
Product Owner	Audun, Jorgen, Tor
Scrum Master	Vegard
Team	Simen, Alex, Jorgen, Tor, Vegard, Jarle



Goal

Starte på Analyse/Design fasen.

Completed stories

sto34	**Som studenter ved HIBU ønsker vi å produsere nødvendige dokumenter**	*3 SP*	*17 hrs.*
tsk34 Ferdigstille budsjett *1 hrs.*			
tsk27 Gå igjennom dokumenter som tidligere er levert *0 hrs.*			
tsk31 Ferdigstille maler med union stil *2 hrs.*			
tsk30 Lage brukermanual for GIT *6 hrs.*			
tsk37 Lage aktivitetsoversikt ink. sum i timeliste *6 hrs.*			
tsk33 Lage teknologidokument for Kunagi *2 hrs.*			

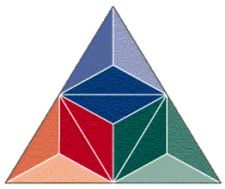
sto35	**Som utviklere ønsker vi å få satt opp prosjektmiljø**	*1 SP*	*17 hrs.*
tsk39 Kodelesnings-database *7 hrs.*			
tsk29 Flytte over data fra excel ark til prosjektverktøy *1 hrs.*			
tsk32 Sette opp server *7 hrs.*			
tsk36 Installere kunagi på server *2 hrs.*			

sto17	**Som bruker av applikasjonen skal det være mulig å importere en objektmodell fra en prosjektfil**	*3 SP*	*7 hrs.*
tsk26 Lage sekvensdiagram for importering av prosjektfil *2 hrs.*			

tsk25 Lage klassediagram for importering av prosjektfil *5 hrs.*

KVALITETSSIKRING

HIBU STUDENTPROSJEKT 2012
CYTRAXDBFACTORY



HØGSKOLEN
i Buskerud

cytrax

INNHOLDSFORTEGNELSE

REVISJONSHISTORIKK			
Versjon	Dato	Endringer	Tildelt
1	22.12.2011	Første utgave	JD
1.1	1.1.2012	- Korrekturlest - Mindre endringer	TCE
1.2	2.1.2012	- Lagt til referanser	JD
1.3	8.2.2012	-Lagt til avsnitt 4.2, referanser for dette, mindre formatteringer	JD

1 INTRODUKSJON

Vi ønsker å kvalitetssikre det arbeidet vi gjør for å oppnå best mulig kvalitet samt så få feil som mulig. Vår kvalitetssikring vil bestå av to deler, dokumentgjennomgang og testing. Testing vil utføres på kodebiter, mens dokumentgjennomgang vil foretas på alle dokumenter vi produserer i løpet av prosjektet.

1.1 SYNONYMER, FORKORTELSER OG DEFINISJONER

Synonym	Beskrivelse
Bugs	Uventete feil i en applikasjon.
User Story	Et krav levert fra for eksempel oppdragsgiver representert i Produkt Backloggen.
SCRUM	En agile(smidig) prosjektmodell.

1.2 FORFATTER

Jarle Didriksen (JD)

1.3 TILDELT

Jarle Didriksen (JD)

2 DOKUMENTGJENNOMGANG

For prosjektet vårt har gruppa blitt enige om en prosess vi skal holde oss til for å kvalitetssikre dokumenter. Når en person produserer et dokument og sier seg ferdig med det, må han sette dokumentet til statusen ”Klar til dokumentgjennomgang”. Deretter må to andre gruppemedlemmer lese gjennom og analysere dokumentet, for så å gi tilbakemelding og til slutt godkjenne det. Når dokumentet er godkjent av disse to personene får dokumentet statusen ”Ferdig”.

3 TESTING

Med testing ønsker vi å oppnå bedre kvalitet på det vi lager samt færre bugs og mindre backtracking. Testing vil gi gode tilbakemeldinger uten at vi «lurer oss selv», som programmerere er vi ofte forutinntatte. Det er lett å overse feil på det vi jobber med, med testing kan vi unngå dette. Det er viktig å tenke på at testing brukes til å vise *tilstedeværelsen* av bugs, ikke *fraværet* av bugs! Softwaretesting er en del av «verification and validation», V&V.

Verifisering: Bygger vi produktet riktig? Man evaluerer et system eller en komponent for å bestemme om det vi har laget på et visst tidspunkt i sprinten tilfredsstiller betingelsene vi har satt på starten av sprinten. For eksempel i monopol kan man *verifisere* at to spillere ikke kan eie samme hus. Gjennom verifisering forsikrer vi oss om at produktet oppfører seg slik vi vil at det skal oppføre seg.

Validering: Bygger vi det riktige produktet? Man evaluerer et system eller en komponent under, eller på slutten av en utviklingsfase for å bestemme om det tilfredsstiller spesifiserte krav. For eksempel kommer en kunde og legger ned krav for Monopol, men så kommer programmereren tilbake med Ludo. Gjennom validering forsikrer vi oss om at det ikke har blitt gjort feil under utviklingen, slik at produktet vi leverer er det kunden ønsker.

3.1 TYPER TESTING – BESKRIVELSE

Man har flere nivåer eller faser med testing. I rekkefølge fra start til slutt har vi enhetstesting, integrasjonstesting, systemtesting og aksepttesting. Et «nivå» som egentlig gjøres hele tiden, men som ikke passer inn et bestemt sted, er regresjonstesting.

Enhetstesting er nivået hvor man tester individuelle *enheter* av kildekoden for å finne ut om de fungerer.

Integrasjonstestingforegår ved å kombinere individuelle moduler og teste de som en gruppe.

Systemtesting: Systemtesting utføres på et fullstendig, integrert system for å evaluere at systemet samsvarer med de spesifiserte kravene.

Aksepttestinger på det høyeste nivået. Man tester for å beslutte om et system tilfredsstiller krav spesifisert av kunden.

Regresjonstestinggjøres ved å repetere tidligere kjørte test caser for å finne eventuelle nye feil i systemet som følge av ny kode.

I SCRUM vil alle nivåene besøkes hver sprint. På slutten av en sprint gjennomgår man aksepttesting, dvs. at en og en *user story* testes.

3.2 TEKNIKKER FOR TESTING

Det er hovedsakelig to teknikker det dreier seg om når man utfører test, de to teknikkene er *black-box-testing* og *white-box-testing*. Dette dreier seg om testerens ståsted når han ser på softwaren.

3.3 BLACK BOX-TESTING

Black-box-testing, også kjent som funksjonell testing, er testing som ignorerer interne mekanismer i et system, komponent eller lignende, og fokuserer kun på outputs som blir generert ut ifra de inputs og betingelser man velger. Det brukes ofte for *validering*.
I en black-box-test vil man kun vite at informasjon kan sendes inn, og at man kan lese noe ut. Basert på kravene vet man hva man kan forvente å få ut, og man tester for å sikre at det sendes ut det som er meningen.

Black-box-testing brukes vanligvis i alle nivåene bortsett fra enhetstesting, selv om det kan brukes der også. En black-box-test som feiler avslører et *symptom på et problem*. Testen er ikke kontrollert, og det kan være vanskelig å finne årsaken til feilen.

3.4 WHITE BOX-TESTING

White-box-testing er testing hvor de interne mekanismene til et system/en komponent er tatt i betraktning. Dette vil si at koden og alt er fullstendig synlig. Med whitebox-testing kjører man koden med forhåndsbestemt input og output, og man må ofte skrive tilleggskode for å få testet alt. Om noe kode mangler, for eksempel skal utvikles på et senere tidspunkt, må man kunne simulere den koden som ikke enda er tilstede.

White-box-testing er det som brukes til enhetstesting, og er som regel for komplisert å bruke på et høyere nivå. En white-box-test som feiler *avslører et problem*. En slik test er kontrollert, man kan identifisere de spesifikke kodelinjene som er involvert.

3.5 STATISK OG DYNAMISK TESTING

Dynamisk testing: testing som involverer å kompilere og kjøre kode for å teste. Enhetstester, integrasjonstester, systemtester og aksepttester er alle dynamiske tester.

Statisk testing er testing hvor man faktisk ikke bruker softwaren man tester. Det er vanligvis ikke en detaljert form for testing, og man sjekker vanligvis hvor forståelig koder, algoritmer og dokumenter er. Vi kommer til å bruke statisk testing i form av kodereview. Når man gjør ferdig en task blir denne lagt klar til kodereview på git, og den som vil kan gå inn og gjennomføre etkodereview. Det er satt krav til at minst to prosjektmedlemmer skalgodkjenneentask før den er ferdig.

3.6 TEST CASE

En test case er et sett med test-inputs, betingelser, og forventede resultater utviklet for et bestemt mål, for eksempel å utøve en viss stil i programmet eller, å verifisere samsvar med et spesifikt krav. Man bør lage test caser så tidlig som mulig, hvis man venter for lenge vil test caser vanligvis bli dårlige og man begynner istedenfor med adhoc-testing, dvs. alle begynner å teste det de kan komme på.

I en test case skal man ha med:

- Test-ID
- Beskrivelse
- Testtype (manuell, integrasjonstest...)
- Forventet resultat
- Faktisk resultat

I tillegg tar vi med story-ID fordi vi benytter SCRUM som prosjektmodell, dette for å kunne se hvilken *userstory*somtestes.

I beskrivelse beskriver man spesifikt et sett med steg og/eller inputs for det du vil teste, inkludert evt. forberedelser. I faktisk resultat skriver man PASS eller FAIL. Hvis FAIL er det greit å ha med beskrivelse av hva som gikk galt. Det er viktig å ha spesifikke beskrivelser for å kunne gjenskape eventuelle feil. Etter gjennomført test kan det inkluderes flere felter, f.eks:

- Hvem testen ble utført av og når
- Kommentarer

En ting som kan være lurt er å lagre alle test-caser i en enkel database. Det gir en enkel oversikt over alle testresultater og annen relevant informasjon. I prosjektet vårt finner man test casene og tilhørende user story i produktbackloggen.

3.7 TEST-NIVÅENE

3.7.1 Enhetstesting

Enhetstesting er det laveste nivået av testing, og her opererer man med *enheter*. En enhet er den minste testbare delen av en applikasjon, og skal aldri være større enn en klasse. Teknikken som brukes til enhetstesting er white-box-testing. For eksempel skriver man testkode som kaller en metode med visse parametre, og sikrer at det som returneres er som forventet. Har man for eksempel en if/else, så kan man lage en test casei tillegg som tar for seg det utfallet som den første casen ikke gjorde.

Man ønsker at alle test caser skal være uavhengig av alle andre test caser, dvs. enheten man tester må isoleres. Målet med enhetstesting blir å isolere hver del av programmet og vise at de individuelle delene stemmer. Ofte må man lage *drivere* og *stubs* når vi gjør white-box-testing, for eksempel om enheten avhenger av annen kode.

En **driver** er en softwaremodul som brukes for å starte en annen modul som testes. Den kontrollerer ofte kjøringen, inneholder ofte test-input og rapporterer resultater tilbake. Et enkelt eksempel vil være en kodelinje som kaller en metode. La oss si at vi ønsker å flytte en spiller to plasser på et brett. En driverkode for dette vil da bli:

```
movePlayer(thePlayer, 2);
```

En whitebox-test case vil da kjøre denne driver-koden, så for eksempel sjekke `Player.getPosition()` for å sikre at spilleren nå er der det er forventet at han er.

En **stub**er noe som erstatter kode som er, eller skal bli, definert et annet sted. Man kan også ha dummy-komponenter/objekter som man bruker til å simulereoppførsel. La oss si at man ikke har laget `movePlayer()` enda, da kan man bruke en stub som bare flytter en spiller til ønsket posisjon:

```
publicmovePlayer(Player player) {
      player.SetPosition(2)
}
```

Stubs og drivere er ofte "throwawaycode".

3.7.2 Integrasjonstesting

Her kombinerer man forskjellige softwarekomponenter, hardwarekomponenter, eller begge deler, og tester for å se hvordan de samhandler. Man bruker gjerne både black-box og white-box-testing til dette. Den som tester verifiserer at enhetene fungerer sammen når de integreres i en større kodebase. Selv om alle komponenter fungerer individuelt er det ikke sikkert at alle fungerer når de settes sammen eller integreres.

Som «input» samler man de relaterte enhetene som allerede har blitt testet og verifisert, og tester de som en gruppe. Som «output» får man da et integrert sub-system.

3.7.3 Systemtesting

Testing utføres på et fullstendig, integrert system for å evaluere at systemet samsvarer med de spesifiserte kravene. Systemtesting faller innenfor manuell black-box testing. Med systemtesting ønsker vi å sikre at alle subsystemene fra integrasjonstestingen fungerer sammen, og at alt er satt riktig sammen. Systemtesting har som mål å eksponere defekter i både de indre sammensetningene og systemet som helhet. Man tester gjerne *ikke-funksjonelle* egenskaper i programmet, som for eksempel:

- Stresstesting – Tester stabiliteten, man tester forbi kapasiteten til programmet.
- Ytelsetesting – Tester ytelsen til et system med tanke på responstid og stabilitet.
- Usabilitytesting – Evaluerer i hvilken grad brukere kan lære å bruke systemet.

3.7.4 Aksepttesting

Aksepttesting blir, med SCRUM, en slags funksjonell testing av en user story. En story kan ha flere acceptance tester, så mange som trengs for å sikre at funksjonaliteten fungerer. En user story er aldri ferdig før den har bestått sine acceptance tester, så nye acceptance tester må lages for hver sprint.

Det er en formell testing hvor det vanligvis er «kunden» som tester det. Dvs. at noen tar på seg rollen som «kunde». Aksepttesting ligner på systemtesting, bare at det blir utført av «kunden», og ikke teamet.

3.7.5 Regresjonstesting

Regresjonstester gjøres gjennom alle fasene. Det er en selektiv gjentesting av et system eller en komponent for å verifisere at modifikasjoner ikke har laget uønskede bieffekter.

Regresjonstester er et subset av det originale settet med test caser, og disse kjøres på nytt relativt ofte, dvs. etter signifikante endringer(bug-fixes, patcher). Det er vanligvis upraktisk å gjenteste med alle test caser. Så for å velge et sett med regresjonstester kan man følge noen retningslinjer:

- Velg et representativt utvalg av tester som går gjennom alle eksisterende funksjoner
- Velg tester som fokuserer på komponentene/funksjonene som ble endret
- Velg så flere tester som fokuserer på funksjoner som sannsynligvis kan bli påvirket

3.7.5.1 Smoke-testing

Et subset av regresjonstestene kan settes som *smoketester*. Det er en gruppe med test caser som fastslår at systemet er stabilt og at all hovedfunksjonalitet er tilstede, samt fungerer under normale betingelser. De er ofte automatiserte, og kan være smart å kjøre før man fortsetter med videre testing. Hvorfor bry seg med å teste alt, hvis systemet er ustabilt i første omgang? Hensikten med smoketesting er altså å demonstrere *stabilitet,* ikke for å *finne bugs*.

4 HJELPETEKNIKKER

I starten er det greit å begynne med å teste noe input som man vet burde passere, eller feile. Hvis slike tester ikke funker ordentlig burde man stoppe åteste og fortsette med å utvikle koden. Uansett, videre følger noen teknikker som kan hjelpe oss på vei.

4.1 BLACK BOX-TESTING

4.1.1 EquivalencePartitioning

En god test case finner en annen klasse med feil enn det som har blitt funnet av andre test caser. Equivalencepartitioning hjelper oss med dette, og reduserer antall test caser som må lages. Man deler opp input-domenet av et program inn i klasser. For hver klasse bør datasettet behandles på samme måte. Test caser designes slik at inputene ligger innenfor disse klassene.

La oss si at man har havnet i fengsel i monopol, og må betale 50kr for å slippe ut. Hvis man har over 50kr, trekkes det fra og man slipper ut, og hvis man har under 50kr taper man spillet. Dette er da to equivalenceclasses man kan partisjonere inn i "mindre enn 50kr" og "50kr eller mer". Test caser velges så fra hver partisjon. Så hvis man da har en test case hvor man tester med en spiller med 1200kr, og en med 100kr, er det lite trolig at sistnevnte finner feil som den første ikke fant. Ofte vil man lage en «gyldig» klasse og en «ugyldig» klasse.

4.1.2 Boundary Value Analysis

Boundary Value er en dataverdi som korresponderer til minimum eller maksimumverdi for input/output spesifisert for et system. For å fortsette på eksempelet, vil dette innebære å ha tester for 49, 50 og 51kr. Dette hjelper med å finne vanlige feil som oppstår pga. at man bommer med <= etc.

4.1.3 Failure Test Cases

Man skal tenke på alt som en bruker kan finne på å gjøre med et system, det innebærer også forsøk på å sabotere. Man må gjøre det robust, slik at programmet fortsatt kan svare fornuftig når det møter ulogiske brukerinputs. Dette kalles *robustness testing*, hvor man velger test caser for å teste solidheten i forhold til feil/uventet input. Noen feilsituasjoner som er fornuftig å teste er følgende:

- Kan en eller annen input forårsake deling på null?
- Hva om input-typen er feil, f.eks at man forventer int og får string?
- Hva om brukeren tar en ulogisk rute gjennom funksjonaliteten vår?
- Hva om obligatoriske felter ikke fylles ut?
- Hva om programmet avsluttes uventet?

Svarer fortsatt programmet ”elegant” i disse situasjonene?

4.2 WHITE BOX-TESTING

4.2.1 Basis Path Testing

Dette er en måte å sikre at alle uavhengige stier gjennom en kodemodul har blitt testet. En uavhengig sti betyr enhver sti gjennom koden som introduserer minst et nytt sett med statements eller en ny condition. I begynnelsen kan det være nyttig å tegne flytdiagram over kodesegmenter. Hovedpoenget er å identifisere antall beslutningspunkter i modulen. For å finne antall uavhengige stier gjennom koden kan man bruke noe som heter "thecyclomaticnumber". Den letteste måten å regne ut det tallet på er ved å telle antall "logicconditional" eller "predicate" i koden, og til slutt legge til 1. "Logic conditional" eller "predicate" vil si ting som if, else og loops. Når man har funnet alle stier vil man lage test caser som sikrer at alle de stiene testes minst én gang.

4.2.2 EquivalencePartitioning/Boundary Value Analysis

Som ved blackbox-testing kan man også bruke dette ved whitebox-testing.
Boundary Value Analysis i white-box-testing vil typisk involvere loops og lignende. Da bør man kjøre tester under, på og rett over betingelsene til den aktuelle loopen.

4.2.3 Failure test cases

Som ved blackbox-tester, man må se på strukturen til koden sin, og tenke ut alle måter en bruker kan komme til å ødelegge det, med eller uten vilje. Noen forslag kan være følgende:

- Se på alle mulige inputs i koden man tester. Håndterer man hver input hvis den er ugyldig, feil font eller for stor/liten etc.?
- Se på koden fra et sikkerhetsmessig synspunkt. Kan en bruker oversvømmeet buffer og skape et sikkerhetsproblem?
- Se på hver utregning. Er det mulig å skape oversvømmelse? Har du tatt hensyn til mulige delinger på null?

5 TESTAUTOMATISERING

Testautomatisering vil si at man bruker software for å kontrollere kjøring av tester, sammenlikne faktisk med forventet resultat, sette opp preconditions og mer. Vanligvis vil dette involvere å automatisere en ferdiglaget, manuell prosess. Man velger å automatisere noen tester fordi manuell testing er knotete og tar lang tid. Det er som regel white-box-tester som er automatiserbare. Det fins flere fremgangsmåter, men det som virker mest aktuelt er *code-driven testing*.

5.1 CODE-DRIVEN TESTING

Dette er den mest aktuelle fremgangsmåten for oss. Dette innebærer bruk av rammeverk for testing, f.eksxUnit-rammeverkene (JUnit, cppUnit) som gir oss muligheten til å kjøre enhetstester. Det gjør det også tryggere årefaktorere koden.

5.2 STATISK KODEANALYSERING/PROGRAMANALYSERING

Dette er analysering av programvare som utføres uten å kjøre selve programmet. Det utføres i de fleste tilfeller på kildekoden, eller i noen tilfeller på objektkode. Dette begrepet brukes mest om analyse som utføres av automatiserte verktøy.

Informasjon man kan få ut fra slike analyser kan være advarsler om minnelekkasjer, ressurslekkasjer, ubrukte variabler og lignende ting.

Mange verktøy for dette formålet er kommersielle eller ikke-kompatible med våre verktøy, eller begge deler. Et verktøy som høres tilstrekkelig ut til vårt bruk er *Cppcheck*, som er et verktøy med GUI og det kan sjekke én fil eller en hel mappe med filer. Et alternativ til dette er *SourceMonitor*, som også er et verktøy med GUI, men med noe andre egenskaper. Begge verktøyene er til Windows, men det finnes også en terminalbasert versjon av Cppcheck til Linux. Ellers ser det også ut til at en windows-emulator takler programmene fint.

5.3 RAMMEVERK

For C++ eksisterer det mange rammeverk. Blant det som høres best ut har vi *googletest*, som er Google's eget rammeverk for testing i C++. Det er gratis, og det er multiplattform.

I tillegg har man *QTestLib* som er for enhetstesting i QT, noe som er meget godt egnet for vår gruppe. Dette er også multiplattform og gratis for oss å bruke. Nærmere informasjon om rammeverk for testing vil finnes i teknologidokumenter.

6 HVORDAN VI SKAL UTFØRE TESTING

Basert på dette dokumentet har gruppa bestemt seg for hvordan og hva vi skal teste.

6.1 STATISK TESTING

Vi vil utføre statisk testing på to måter. Det første er ved bruk av verktøy;cppcheck. Dette vil vi gjøre på alt vi implementerer før det pushes til git, for å gjøre det mindre sannsynlig at koden inneholder mindre åpenbare feil. Når koden er tilgjengeliggjort på github vil vi utføre statisk testing på den andre måten; kodelesing. To gruppemedlemmer skal lese gjennom koden på git, og eventuelle kommentarer eller utbedringer kan legges inn. Til

slutt skriver medlemmene en kommentar i bunn av koden om de godkjenner koden eller ikke. Først når to gruppemedlemmer godkjenner koden blir den limt sammen med eksisterende kode. Denne formen for testing brukes hovedsakelig for å sjekke at koden er lesbar, har riktig implementerte algoritmer og at det følger kodestandarden. (1)

6.2 DYNAMISK TESTING

Unit-testing vil bli utført på alt vi implementerer før det pushes til github. Vi bruker QTestLib for å utføre dynamisk testing. Med dette rammeverket kan vi enkelt og effektivt kjøre mange unit-tester etter implementasjon, og raskt finne ut om ønsket oppførsel er ivaretatt.

Integrasjonstesting vil vi utføre når større komponenter skal settes sammen i system. Dette vil være i form av unit-testing mot en klasse som tar i bruk flere komponenter i systemet.

Den siste testingen vi utfører er akseptansetesting. Dette er tester som er definert for hver user-story i productbacklog. Etter hver sprint på sprintreview-møtet vil vi verifisere at testkravene er oppfylt, og godkjenne eller avvise en user-story.

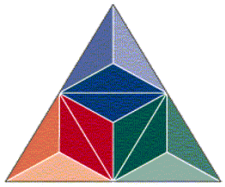
Regresjonstesting vil gjøres vha. kontinuerlig kjøring av testing i QTestLib. Etter endringer vil vi utføre tidligere kjørte tester på nytt, både for å sjekke at tidligere oppførsel er ivaretatt og at ny oppførsel er riktig.

7 REFERANSER

1. **DB Factory.***Kodestandarder.* 2011.

TEST-CASES

HIBU STUDENTPROSJEKT 2012
CYTRAX DB FACTORY



HØGSKOLEN
i Buskerud

cytrax

Test case – Test-ID: 01			
Story-ID	55	**Task**	151 – Implementere wrapper-klassene
Type test	Black-box, systemtest		
Case	Vi ønsker at wrapper-klassene skal holde dokumentmodellen oppdatert etter at det skjer handlinger i GUI.		
Krav/Kriterie	Dokumentmodellen blir oppdatert i henhold til endringer som skjer i GUI-et.		
Resultat m/kommentarer			
Endret verdier på alle forskjellige hierarkiske nivåer, lagret dokumenmodellen. Ser at de oppdaterte dataene er lagret på korrekt sted. Godkjent.			
Test nr	1\1	**Resultat**	**PASSED**
Dato utført	29.05.2012	**Utført av**	Jørgen Grøndal

Test case – Test-ID: 02			
Story-ID	59	**Task**	170 – Populere dokmod med info
Type test	Black-box, systemtest		
Case	Vi ønsker at plugin skal populere dokumentmodellen med parset informasjon fra databasen.		
Krav/Kriterie	1) Dokumentmodellen blir populert 2) Den blir populert med riktig informasjon		
Resultat m/kommentarer			
Leser inn databasemodell fra testdatabasen Hudlager.db. Alle tabeller og kolonner er tilstede med riktige egenskaper.			
Test nr	1\1	**Resultat**	**PASSED**
Dato utført	29.05.2012	**Utført av**	Jørgen Grøndal

Test case – Test-ID: 03			
Story-ID	22	**Task**	168 – Implementere generatorplugin
Type test	Black-box, systemtest		
Case	Vi ønsker å kunne generere kode for VB.net for å aksessere databasen.		
Krav/Kriterie	Gyldig, fungerende VB.net kode blir generert		
Resultat m/kommentarer			
Kode blir generert. Sammenligner koden med tidligere generert kode, koden som blir generert er gyldig og fungerende. Avvik i generering er stor forbokstav på datatypene, men det er korrekt med stor forbokstav i VB.net.			
Test nr	1\1	**Resultat**	**PASSED**
Dato utført	29.05.2012	**Utført av**	Jørgen Grøndal

Test case – Test-ID: 04			
Story-ID	10	**Task**	169 – Implementere eksportering til XML
Type test	Black-box, systemtest		
Case	Vi ønsker å kunne eksportere en dokumentmodell til en prosjektfil i XML-format.		
Krav/Kriterie	XML-fil opprettes på riktig sted med riktig informasjon		
Resultat m/kommentarer			
XML-filen ble tilsynelatende aldri opprettet og således ble informasjonen ikke eksportert. Ved nærmere undersøkelse viste det seg derimot at pathen ble brukt feil, og at vi lette etter filen på feil sted. Vi kunne til slutt konkludere med at XML-filen ble opprettet på riktig sted og med riktig innhold. Resultatet ble derfor godkjent.			
Test nr	1\1	**Resultat**	**PASSED**
Dato utført	04.05.2012	**Utført av**	Vegard Kaasin og Jarle Didriksen

Test case – Test-ID: 05			
Story-ID	70	**Task**	210 – Hente ut nylig brukte filer
Type test	Black-box, systemtest		
Case	Vi ønsker å hente ut en liste med nylige brukte filer og vise disse i GUI-et.		
Krav/Kriterie	Listen skal ha maksimum 5 oppføringer og skal ikke inneholde duplikatinformasjon.		
Resultat m/kommentarer			
Det er observert at listen aldri inneholder mer enn 5 oppføringer, og duplikatinformasjon oppstår heller ikke. Det ble i tillegg oppdaget en mindre bug. Denne oppstod ved generering av et nytt prosjekt, og feilen var at det ble lagt inn en blank oppføring for det tomme prosjektet. Dette er nå fikset. På bakgrunn av dette er testen dermed godkjent.			
Test nr	1\3	**Resultat**	**PASSED**
Dato utført	23.05.2012	**Utført av**	Jarle Didriksen

Test case – Test-ID: 06			
Story-ID	70	**Task**	210 – Hente ut nylig brukte filer
Type test	Black-box, systemtest		
Case	Vi ønsker at en fil blir åpnet når vi trykker på en tidligere brukt fil.		
Krav/Kriterie	At riktig fil blir åpnet på riktig måte og vist i GUI-et på vanlig vis.		
Resultat m/kommentarer			
Ingen filer ble åpnet eller gjort noe med. Ved nærmere undersøkelser viste det seg at metoden som skulle gjøre det aldri ble brukt. Det kan virke som at det er feil med signaler i koden slik at metoder ikke blir koblet sammen på riktig måte. Testen er dermed ikke godkjent.			
Test nr	2\3	**Resultat**	**FAILED**
Dato utført	23.05.2012	**Utført av**	Jarle Didriksen

Test case – Test-ID: 07			
Story-ID	70	**Task**	210 – Hente ut nylig brukte filer
Type test	Black-box, systemtest		
Case	Vi ønsker at en fil blir åpnet når vi trykker på en tidligere brukt fil.		
Krav/Kriterie	At riktig fil blir åpnet på riktig måte og vist i GUI-et på vanlig vis.		
Resultat m/kommentarer			
Etter forrige test ble problemområdet funnet og en ny løsning ble implementert. Denne gangen åpnet filene seg på riktig måte og ble vist i GUI-et som forventet. Testen ble derfor godkjent.			
Test nr	3\3	**Resultat**	**PASSED**
Dato utført	24.05.2012	**Utført av**	Jarle Didriksen

Test case – Test-ID: 08			
Story-ID	Ingen	**Task**	Ingen
Type test	Automatisk, statisk test, cppcheck		
Case	Vi ønsker å sikre at programmet vårt ikke lider av skjulte minnelekkasjer.		
Krav/Kriterie	At ingen minnelekkasjer blir oppdaget		
Resultat m/kommentarer			
Cppcheck ble kjørt på alle våre kodefiler uten å finne minnelekkasjer. Det ble heller ikke oppdaget andre alvorlige feil. Testen er dermed godkjent.			
Test nr	1\1	**Resultat**	**PASSED**
Dato utført	27.05.2012	**Utført av**	Jarle Didriksen

Test case – Test-ID: 09			
Story-ID	70	**Task**	212
Type test	Black-box, akseptansetest		
Case	Vi ønsker å implementere About funksjonalitet I applikasjonen		
Krav/Kriterie	Data blir presentert brukeren på en naturlig måte, Funksjonalitet virker som det skal		
Resultat m/kommentarer			
Både brukt help -> about og about knapp i GUI. About dialogen ser bra ut. Viser informasjon om hvem som har utviklet applikasjonen, logo og år den ble utviklet.			
Test nr	1\1	**Resultat**	**PASSED**
Dato utført	29.05.2012	**Utført av**	Jørgen Grøndal

Test case – Test-ID: 10			
Story-ID	70	**Task**	179
Type test	Black-box, akseptansetesting		
Case	Vi ønsker å avslutte applikasjonen		
Krav/Kriterie	Sjekker om du har lagret prosjekt, Avslutter riktig uten feil		
Resultat m/kommentarer			
Avslutter programmet med kryss i topp, høyre hjørnet. Tester alle valg; CANCEL, NO og YES. Cancel: Går tilbake, dokument lagres ikke. No: Program avsluttes, dokument lagres ikke. Yes: Program avsluttes, dokument lagres. Samme oppførsel ved bruk av File -> Exit.			
Test nr	1\1	**Resultat**	**PASSED**
Dato utført	29.05.2012	**Utført av**	Jørgen Grøndal

Test case – Test-ID: 11			
Story-ID	70	**Task**	179
Type test	Usability test		
Case	Vi ønsker at brukeren synes det er enkelt og naturlig å bruke applikasjonen		
Krav/Kriterie	Brukeren skal ikke kunne trenge hjelp for å bruke applikasjonen		
Resultat m/kommentarer			
Fungerer veldig bra og er intuitivt og lett å sette seg inn i.			
Test nr	1\1	**Resultat**	**PASSED**
Dato utført	29.05.2012	**Utført av**	Simen Russnes

Test case – Test-ID: 12			
Story-ID	70	**Task**	207
Type test	Black-box, akseptansetest		
Case	Vi ønsker kunne velge tabeller og kolonner for generering		
Krav/Kriterie	Vindu blir åpnet for generering, bruker kan velge tabeller og kolonner, kode blir generert uten tabellene kolonnene som det er fjernet hake for.		
Resultat m/kommentarer			
Kode blir generert kun for tabeller og kolonner som er valgt. Godkjent.			
Test nr	1\1	**Resultat**	**PASSED**
Dato utført	29.05.2012	**Utført av**	Jørgen Grøndal

Test case – Test-ID: 13			
Story-ID	70	**Task**	208
Type test	Black-box, systemtest		
Case	Vi ønsker å implementere "help" funksjonalitet I applikasjonen		
Krav/Kriterie	Data blir presentert brukeren på en naturlig måte, Funksjonalitet virker som det skal		
Resultat m/kommentarer			
Vinduet ble åpnet og all funksjonalitet virker som planlagt. Godkjent			
Test nr	1\1	**Resultat**	**PASSED**
Dato utført	29.05.2012	**Utført av**	Vegard Kaasin

Test case – Test-ID: 14			
Story-ID	71	**Task**	198
Type test	Black-box, systemtest		
Case	Vi ønsker å importere en database inn I applikasjonen		
Krav/Kriterie	Åpner vindu for å importere, Data skrevet inn blir håndtert riktig av applikasjonen		
Resultat m/kommentarer			
Vindu blir åpnet og data blir lest inn korrekt i dokumentmodellen. Godkjent.			
Test nr	1\1	**Resultat**	**PASSED**
Dato utført	29.05.2012	**Utført av**	Vegard Kaasin

Test case – Test-ID: 15			
Story-ID	70	**Task**	179
Type test	Black-box, systemtest		
Case	Vi ønsker å lagre en prosjektfil I applikasjonen et annet sted en tidligere spesifiser og med ett annet navn		
Krav/Kriterie	Lagrer fil med riktig navn, lokasjon og data		
Resultat m/kommentarer			
Fil blir lagret med nytt navn og på ny lokasjon. Data stemmer overens med innlest dokumentmodell. Godkjent			
Test nr	1\1	**Resultat**	**PASSED**
Dato utført	29.05.2012	**Utført av**	Vegard Kaasin

Test case – Test-ID: 15			
Story-ID	70	**Task**	179
Type test	Black-box, systemtest		
Case	Vi ønsker å lagre en prosjektfil I applikasjonen		
Krav/Kriterie	Lagrer fil med riktig navn, lokasjon og data		
Resultat m/kommentarer			
Fil ble lagret og endringer i filen ble overskrevet. Godkjent.			
Test nr	1\1	**Resultat**	**PASSED**
Dato utført	29.05.2012	**Utført av**	Vegard Kaasin

Test case – Test-ID: 16			
Story-ID	70	**Task**	179
Type test	Black-box, systemtest		
Case	Vi ønsker å lage ett nytt prosjekt I applikasjonen		
Krav/Kriterie	Korrekt data blir satt I dokumentmodell basert på hva som er skrevet inn		
Resultat m/kommentarer			
Data stemmer overens med hva som er lest inn. Godkjent.			
Test nr	1\3	**Resultat**	**PASSED**
Dato utført	29.05.2012	**Utført av**	29.05.12

Test case – Test-ID: 17			
Story-ID	70	**Task**	179
Type test	Black-box		
Case	Vi ønsker å lage ett nytt prosjekt I applikasjonen		
Krav/Kriterie	Oppsummering av data skrevet in skal vises I ett vindu etter åpning av prosjekt		
Resultat m/kommentarer			
Oppsummering stemmer. Godkjent			
Test nr	2\3	**Resultat**	**PASSED**
Dato utført	29.05.12	**Utført av**	Vegard Kaasin

Test case – Test-ID: 18			
Story-ID	70	**Task**	179
Type test	Black-box, systemtest		
Case	Vi ønsker å lage ett nytt prosjekt I applikasjonen		
Krav/Kriterie	Nytt vindu blir åpnet og informasjon blir vist på en naturlig måte		
Resultat m/kommentarer			
Intuitivt fremvisning av data. Noe kjedelig å måtte skrive inn all informasjon hver gang man lager nytt prosjekt. Men ut fra Krav: Godkjent.			
Test nr	3\3	**Resultat**	**PASSED**
Dato utført	29.05.2012	**Utført av**	Vegard Kaasin

Test case – Test-ID: 19			
Story-ID	19	**Task**	193
Type test	Black-box, akseptansetesting		
Case	Vi ønsker å sammenlikne to dokumentmodeller		
Krav/Kriterie	Vindu åpnes opp, Bruker blir presentert data på en naturlig måte		
Resultat m/kommentarer			
Vinduet åpnes og sammenlikning stemmer for både innlesning fra database og fra dokument.			
Test nr	1\1	**Resultat**	**PASSED**
Dato utført	29.05.2012	**Utført av**	Vegard Kaasin

Test case – Test-ID: 20			
Story-ID	70	**Task**	158
Type test	Black-box, systemtest		
Case	Vi ønsker å endre typemapping I applikasjonen		
Krav/Kriterie	Vindu blir åpnet, Data blir presentert for bruker naturlig		
Resultat m/kommentarer			
Vinduet blir åpnet og brukeren skjønner lett hvilke valg som blir fremstilt.			
Test nr	1\2	**Resultat**	**PASSED**
Dato utført	29.05.2012	**Utført av**	Vegard Kaasin

Test case – Test-ID: 21			
Story-ID	70	**Task**	158
Type test	Black-box, systemtest		
Case	Vi ønsker å endre typemapping I applikasjonen		
Krav/Kriterie	Data blir lagret korrekt I dokumentmodell		
Resultat m/kommentarer			
Data blir lagret korrekt for begge typer typemapping.			
Test nr	2/2	**Resultat**	**PASSED**
Dato utført	29.05.2012	**Utført av**	Vegard Kaasin

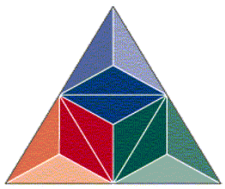
Test case – Test-ID: 22			
Story-ID	70	**Task**	158
Type test	Black-box, systemtest		
Case	Vi ønsker å importere en fil som en dokumentmodell		
Krav/Kriterie	Importering av gydlig xml-fil virker og returnerer riktig data		
Resultat m/kommentarer			
Innlesning virker, feil gis brukeren om xml fil ikke er gyldig			
Test nr	1\1	**Resultat**	**PASSED**
Dato utført	29.05.2012	**Utført av**	Vegard Kaasin

Test case – Test-ID: 23			
Story-ID	25	**Task**	-
Type test	Black-box, systemtest		
Case	Vi ønsker å kunne generere kode ved å bruke parametere i konsoll		
Krav/Kriterie	Kode blir generert korrekt ut i fra parametere som er gitt		
Resultat m/kommentarer			
Fikk problemer med at applikasjonen krasjet når jeg ga gyldige parametere. Blir testet på nytt etter bugfix. EDIT: Lokaliserte feil, typemaps var ikke lastet inn når GUI ikke ble startet på vanlig måte.			
Test nr	1\2	**Resultat**	**FAILED**
Dato utført	29.05.2012	**Utført av**	Jørgen Grøndal

Test case – Test-ID: 24			
Story-ID	25	**Task**	-
Type test	Black-box, systemtest		
Case	Vi ønsker å kunne generere kode ved å bruke parametere i konsoll		
Krav/Kriterie	Kode blir generert korrekt ut i fra parametere som er gitt		
Resultat m/kommentarer			
Etter bugfix blir kode generert korrekt basert på parametere som er gitt.			
Test nr	2\2	**Resultat**	**PASSED**
Dato utført	29.05.2012	**Utført av**	Jørgen Grøndal

AKTIVITETER

HIBU STUDENTPROSJEKT 2012
CYTRAX DB FACTORY



HØGSKOLEN
i Buskerud

cytrax

OVERSIKT OVER AKTIVITETER

AKTIVITETSNUMMER	AKTIVITETSNAVN
100	**Prosjektstyring**
101	Planlegging
102	Økonomi
103	Fremdriftsplan
104	Timelister
105	Product Backlog
106	Sprint Planlegging
199	Annen prosjektstyring
200	**Dokumentasjon**
201	Maler
202	Idedokument / Visjonsdokument
203	Forstudie
204	Kravspesifikasjon
205	Testspesifikasjon
206	Teknologidokumentasjon
207	Brukermanualer
208	Prosjektplan
209	Scrum dokumentasjon
299	Annen dokumentasjon
300	**Møter**
301	Møteforberedelser
302	Product Backlog møte
303	Sprint planleggingsmøte
304	Statusmøter
305	Planleggingsmøter
306	Veiledningsmøter
399	Andre møter
400	**Utredning**
401	Teknologi
402	Verktøy
403	Arbeidsmetodikk
500	**Analyse og design**
501	Analyse
502	Design
600	**Implementasjon**
601	Prototyping
602	Programmering
700	**Kvalitetssikring**
701	Testing

702	Dokumentasjonsgjennomgang
703	Kodegjennomgang
800	**Generelt**
801	Webside
802	Presentasjoner
803	Standarder
804	Forelesninger
899	Annet