

Branching en Merging

van Broncode

Student: Peter Verwijs

Student nr.: 1500971

Bedrijf: Thinkwise Software

1e examinator: Eric Gerlofsma

# Voorwoord

De afgelopen maanden stonden voor mij in het teken van de afstudeeropdracht ter afronding van de studie Informatica aan de Hogeschool Utrecht. De afstudeeropdracht heb ik gevolgd bij het bedrijf Thinkwise Software.

Tijdens de opdracht heb ik veel nieuwe kennis opgedaan. Ik was al enigzins bekend met C# als programmeertaal maar tijdens de opdracht heb ik veel nieuwe aspecten van deze taal ontdekt waarvan ik eerst niet wist dat ze mogelijk waren. Ook heeft het werken met een modelgedreven ontwikkelmethode me veel geleerd over de verschillende manieren waarop software ontwikkeling mogelijk is.

Verder wil ik mijn collega’s, docenten, medestudenten, familie, vrienden en tot slot mijn vriendin bedanken voor de hulp bij het maken van de opdracht en het schrijven van de scriptie. Zonder deze hulp was het maken van de opdracht niet gelukt.

# Managementsamenvatting

Met meerdere projectgroepen aan een project binnen de zelfde ontwikkelomgeving werken is vaak een moeizaam proces. De projectgroepen lopen elkaar vaak in de weg en vaak moet er gewacht worden op elkaar voor er een stabiel eindproduct neergezet kan worden. De projectgroepen zouden er dus gebaat bij zijn om hun werkzaamheden in een eigen ontwikkelomgeving uit te kunnen voeren.

Klanten die gebruik maken van de Thinkwise Software Factory, de modelgedreven ontwikkel workbench van Thinkwise Software, hebben aangegeven dat dit probleem ook voor hen aanwezig is. Er was dus vraag naar een oplossing voor dit probleem.

Deze oplossing is destijds geleverd in de vorm van branching en merging. Dit stelde gebruikers in staat om projecten te kopiëren naar een eigen ontwikkelomgeving en aan het einde van het ontwikkeltraject de wijzigingen die waren gemaakt weer op te nemen in het hoofdproject.

Deze implementatie was echter nog niet compleet. Er werd alleen functionaliteit geboden voor het branchen en mergen van het datamodel. Ondersteuning voor het branchen en mergen van de broncode die gebruikt wordt voor de functionaliteit in het eindproduct was nog niet geïmplementeerd. Gebruikers van de Software Factory hebben echter aangegeven dat dit een essentieel onderdeel is voor het parallel ontwikkelen van projecten.

Het afstudeerproject is daarom gestart met als doel onderzoek doen naar hoe deze functionaliteit in de Software Factory geïntegreerd zou kunnen worden om daarna een prototype te bouwen dat als proof of concept diende.

Om te beginnen is een functioneel ontwerp opgesteld waarin alle functionele eisen aan het prototype opgesteld werden. Hierin werden ook de schermontwerpen opgenomen die gemaakt zijn voor de schermen die in het prototype moesten worden opgenomen.

Vervolgens is er onderzoek gedaan naar bestaande componenten die de benodigde functionaliteit leverden. De resultaten van het onderzoek zijn vastgelegd in een onderzoeksverslag.

Na het onderzoek is de bouw van het prototype van start gegaan. Het prototype bestaat uit een uitbreiding van de Software Factory met een nieuw scherm. Dit scherm bevat de logica die nodig is om de broncode tussen twee projectversies met elkaar te mergen.

Verder zijn er een aantal wijzigingen gemaakt aan de Software Factory die er voor zorgen dat het branchen en mergen van broncode beter wordt ondersteund.

Zo zijn de stored procedures op de database die verantwoordelijk waren voor het toekennen van functionaliteit op verschillende plekken in het eindproduct vervangen door SQL scripts die door de GUI op het gewenste moment op de database afgevuurd kunnen worden. Dit zorgt voor een betere werking van deze functionaliteit als het in meerdere projectversies tegelijk gebruikt wordt. Voor het gebruik van de Software Factory moet er nu echter wel rekening mee gehouden worden dat gebruikers ook toegang hebben tot de harde schijf waar de scripts opgeslagen staan, terwijl voorheen toegang tot de database voldoende was.

Verder is de Software Factory aagepast zodat broncode die op de harde schijf staat opgeslagen nu automatisch wordt meegekopieerd als er een kopie van een projectversie wordt gemaakt. Voorheen was dit een handmatige actie die zorgde voor veel extra werk. Dit probleem was ook van toepassing op het aanmaken van een branch omdat hiervoor de projectversie ook gekopieerd wordt. Deze feature zorgt er dus voor dat het branchproces een stuk makkelijker verloopt en het gebruiksgemak van de Software Factory in het algemeen wordt verhoogt.

Al deze uitbreidingen zijn samen ondergebracht in het prototype en zorgen voor een goede ondersteuning van het branchen en mergen van broncode. Het prototype heeft dus bewezen dat ondersteuning voor het branchen en mergen van broncode in de Software Factory mogelijk is en laat daarbij zien hoe deze functionaliteit op een goede manier geïmplementeerd kan worden.

Het implementeren van branching en merging van code houdt echter niet in dat alle problemen rond het parallel ontwikkelen van software opgelost zijn. Conflicten die ontstaan door slechte communicatie tussen projectteams kunnen nog steeds een probleem vormen, maar dit valt buiten de scope van de opdracht. De Software Factory biedt hier overigens een oplossing voor in de vorm van requirements.

Inhoudsopgave

[Voorwoord 2](#_Toc287804778)

[Managementsamenvatting 3](#_Toc287804779)

[1 Inleiding 7](#_Toc287804780)

[2 Thinkwise Software Factory 8](#_Toc287804781)

[2.1 Modelleerschermen 9](#_Toc287804782)

[2.1.1 Requirements 9](#_Toc287804783)

[2.1.2 Datamodel 9](#_Toc287804784)

[2.1.3 GUI Model 10](#_Toc287804785)

[2.1.4 Processtromen 10](#_Toc287804786)

[2.2 Functionality Weaver 11](#_Toc287804787)

[2.2.1 Generator 12](#_Toc287804788)

[2.3 Thinkwise GUI 13](#_Toc287804789)

[2.4 Branching en merging van het datamodel 14](#_Toc287804790)

[2.5 Functionaliteit in de Software Factory 15](#_Toc287804791)

[3 Projectbeschrijving 16](#_Toc287804792)

[3.1 Probleembeschrijving 16](#_Toc287804793)

[3.2 Opdrachtomschrijving 16](#_Toc287804794)

[3.2.1 Broncode vergelijking en merging 17](#_Toc287804795)

[3.2.2 Automatische migratie van broncode 17](#_Toc287804796)

[3.2.3 Control procedures in verschillende projectversies 17](#_Toc287804797)

[3.3 Technische specificaties 18](#_Toc287804798)

[4 Projectaanpak 19](#_Toc287804799)

[4.1 Projectmanagement methode 19](#_Toc287804800)

[4.2 Fasen 19](#_Toc287804801)

[4.2.1 Plan van aanpak 19](#_Toc287804802)

[4.2.2 Onderzoek 19](#_Toc287804803)

[4.2.3 Ontwerpfase 20](#_Toc287804804)

[4.2.4 Bouw / Prototyping 20](#_Toc287804805)

[4.2.5 Afwijkingen 20](#_Toc287804806)

[5 Onderzoek 21](#_Toc287804807)

[5.1 File comparison engine 21](#_Toc287804808)

[5.2 Conclusie uit het onderzoek 22](#_Toc287804809)

[6 Implementatie 23](#_Toc287804810)

[6.1 Datamodel 23](#_Toc287804811)

[6.2 Code merging scherm 24](#_Toc287804812)

[6.3 Aanpassingen aanroep control procedures 27](#_Toc287804813)

[6.4 Automatisch kopiëren van broncode 29](#_Toc287804814)

[7 Conclusie 30](#_Toc287804815)

[8 Aanbevelingen 31](#_Toc287804816)

[8.1 Branching en merging 31](#_Toc287804817)

[8.2 Thinkwise Software Factory 31](#_Toc287804818)

[8.2.1 Opslag broncode 31](#_Toc287804819)

[8.2.2 Abstract screen modeller 31](#_Toc287804820)

[9 Bronvermelding 32](#_Toc287804821)

[Bijlage 1 33](#_Toc287804822)

[Bijlage 2 39](#_Toc287804831)

# Inleiding

Voor de afronding van de studie Informatica aan de Hogeschool Utrecht is dit project uitgevoerd binnen het kader van de afstudeeropdracht. Het afstudeerproject heeft plaatsgevonden bij Thinkwise Software.

Thinkwise Software is de leverancier van de Thinkwise Software Factory, een ontwikkel workbench waar applicaties op een model gedreven manier mee ontwikkeld kunnen worden. Werkzaamheden binnen Thinkwise zijn grotendeels gericht op de ontwikkeling, het onderhoud en de verkoop van de Thinkwise Software Factory. Daarnaast bouwt Thinkwise Software maatwerkapplicaties met de Software Factory voor haar klanten die niet over een eigen ontwikkelteam beschikken. Thinkwise Software is in 2002 opgericht en heeft momenteel 35 werknemers in dienst.

De doelgroep van Thinkwise ligt voornamelijk bij bedrijven met een interesse voor het automatiseren van hun bedrijfsprocessen. Bedrijven met een eigen ICT afdeling zijn hierbij de hoofddoelgroep, maar bedrijven zonder eigen ontwikkelaars kunnen er voor kiezen om bijvoorbeeld een eindproduct (of een deel daarvan) door Thinkwise Software te laten ontwikkelen. Onder de voornaamste klanten vallen o.a. Sligro Food Groups, KS Profiel, Royal Huisman Shipyard en Koninklijke Zeelandia.

De afstudeeropdracht betreft een onderzoek en bouw project dat binnen de context van de Thinkwise Software Factory uitgevoerd is. De opdracht is ontstaan uit het probleem dat ontstond wanneer er meerdere projectteams tegelijk aan een project werkten. Om dit te kunnen ondersteunen is er een branching en merging mechanisme ontwikkeld waarmee verschillende versies van de broncode voor een project met elkaar vergeleken en samengevoegd kunnen worden.

In deze scriptie zal een beeld gegeven worden van het ontwikkeltraject dat doorlopen is tijdens de uitvoering van het project. Er wordt een beschrijving gegeven van de Software Factory waarin de voornaamste features van de Software Factory uitgelicht worden. Er wordt verslag gedaan van het verloop van het project en de bouw van het eindresultaat. Vervolgens worden de bevindingen en resultaten van het project behandeld. Tot slot worden er een aantal conclusies over het project gegeven en zullen er aanbevelingen gedaan worden aan Thinkwise Software over onderwerpen die tijdens het afstudeertraject naar voren zijn gekomen.



Figuur 1 - Thinkwise Software

# Thinkwise Software Factory

Om de opdracht in context te kunnen plaatsen is het belangrijk om kennis te hebben van de Thinkwise Software Factory. Dit hoofdstuk zal daarom als korte rondleiding dienen door de model gedreven ontwikkelmethode die door de Software Factory gehanteerd wordt. Dit hoofdstuk zal geen allesomvattende omschrijving bevatten van alle mogelijkheden die de Software Factory biedt, maar zal de aspecten van de Software Factory die het meest relevant zijn voor de opdracht uitlichten.

De Thinkwise Software Factory is een model gedreven software ontwikkel workbench geschikt voor de bouw van o.a. administratieve en registratieve applicaties. Model gedreven ontwikkeling of Model-Driven Development in de Software Factory houdt in dat de bouw van een applicatie gebaseerd wordt op een model dat opgebouwd wordt met behulp van grafische modelleertools. Dit abstracte model wordt door de Software Factory geïnterpreteerd om zo een eindapplicatie op te bouwen. De focus van het bouwen van een applicatie verschuift dus naar het modelleren, in plaats van het schrijven van code.

De applicatiestructuur en functionele aspecten worden opgenomen in het model. Het model wordt opgeslagen in de Meta Solution Definition (MSD), ook wel het meta-niveau genoemd. Het meta-niveau dient als blauwdruk voor de eindapplicatie en hierin staan onder andere de specificatie van het datamodel, processtromen en business intelligence schermen maar ook de layout, stijl en menustructuur van de grafische interface.

Business functionaliteit wordt op een aspect georiënteerde manier in het model geweven. Door op deze manier te werken kan broncode voor business functionaliteit eenvoudig op honderden plekken in het model geplaatst worden.

De Software Factory kan de database voor het eindproduct genereren. Het eindproduct kan dan vervolgens worden opgestart met de Thinkwise GUI’s. De GUI’s bestaan uit een grafische interface die het meta-niveau runtime interpreteert en zich dynamisch opbouwt volgens het model.

## Modelleerschermen

Ontwikkeling in de Software Factory vindt voor een groot deel plaats in de modelleerschermen. Binnen deze schermen wordt het model van een applicatie gedefinieerd. De modelleerschermen worden in dit hoofdstuk toegelicht.

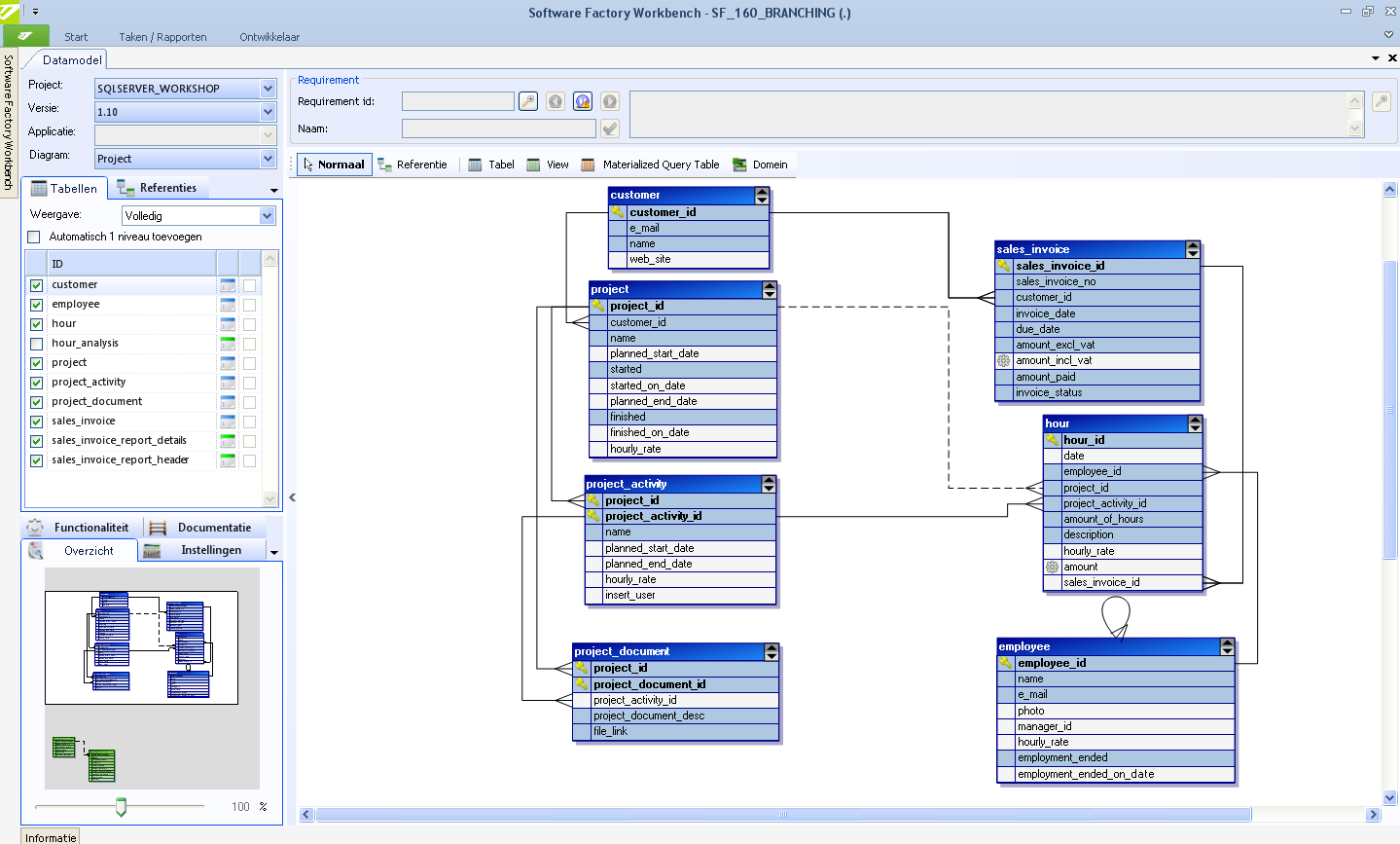
### Requirements

Bij aanvang van het ontwikkelproces kan er gekozen worden om de requirements van het project te definiëren. Dit is een optionele stap maar komt de kwaliteit en beheersbaarheid van het project wel ten goede. Het is dus aan te raden hier gebruik van te maken.

Door requirements te definiëren worden functionele eisen en criteria van het product in kaart gebracht. Door de requirements voorafgaand aan een project zorgvuldig te definiëren kan een goede planning worden gemaakt en kan de voortgang worden bewaakt. Het requirements scherm biedt hier ondersteuning in door de mogelijkheid om voor requirements een tijdsinschatting in te voeren, de goedkeuringsstatus bij te houden, het behalen van mijlpalen te registreren en een centraal punt te bieden voor het bewaren van projectmanagement documentatie. Bovendien is het ook mogelijk om rapporten te genereren waarin een overzichtelijke tijdsplanning is opgenomen.

### Datamodel

Het datamodel van de applicatie wordt gebouwd met de ERD modeller. Hiermee kunnen de data entiteiten, attributen en relaties van de applicatie worden gedefinieerd. Het bouwen van een applicatie begint over het algemeen op dit scherm.

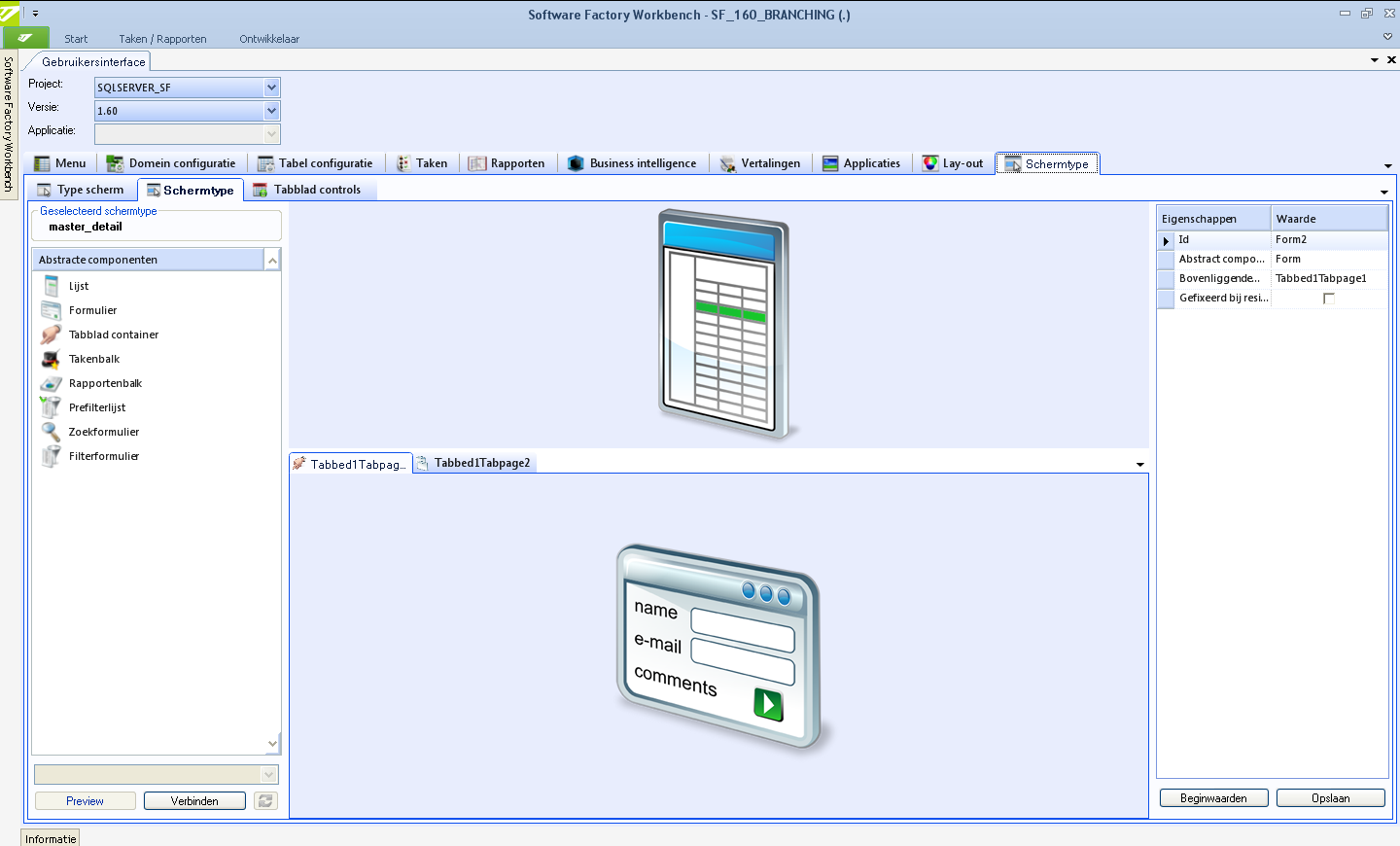


Figuur 2 - De ERD modeler met daarin weergegeven het datamodel van een eindapplicatie

### GUI Model

In de GUI modeller kan worden bepaald hoe de weergave van het model gedefinieerd word. Voor de data entiteiten en attributen wordt beschreven hoe deze in de GUI getoond moeten worden.

Voor het tonen van de data entiteiten kan een schermtype gekozen worden. Deze schermtypes bestaan uit abstracte componenten zoals een lijst, formulier en tabbladen. De schermtypes die beschikbaar worden gesteld zijn standaard schermen zoals een master detail scherm met daar in een lijst en formulier component. Als de standaardschermen niet voldoende zijn kunnen nieuwe schermtypes gemaakt worden. Dit gebeurt met behulp van de abstract screen modeller. Hiermee kunnen componenten op het scherm gesleept worden om zo een abstract scherm samen te stellen welke vervolgens als schermtype gebruikt kan worden.



Figuur 3 - De abstract screen modeller

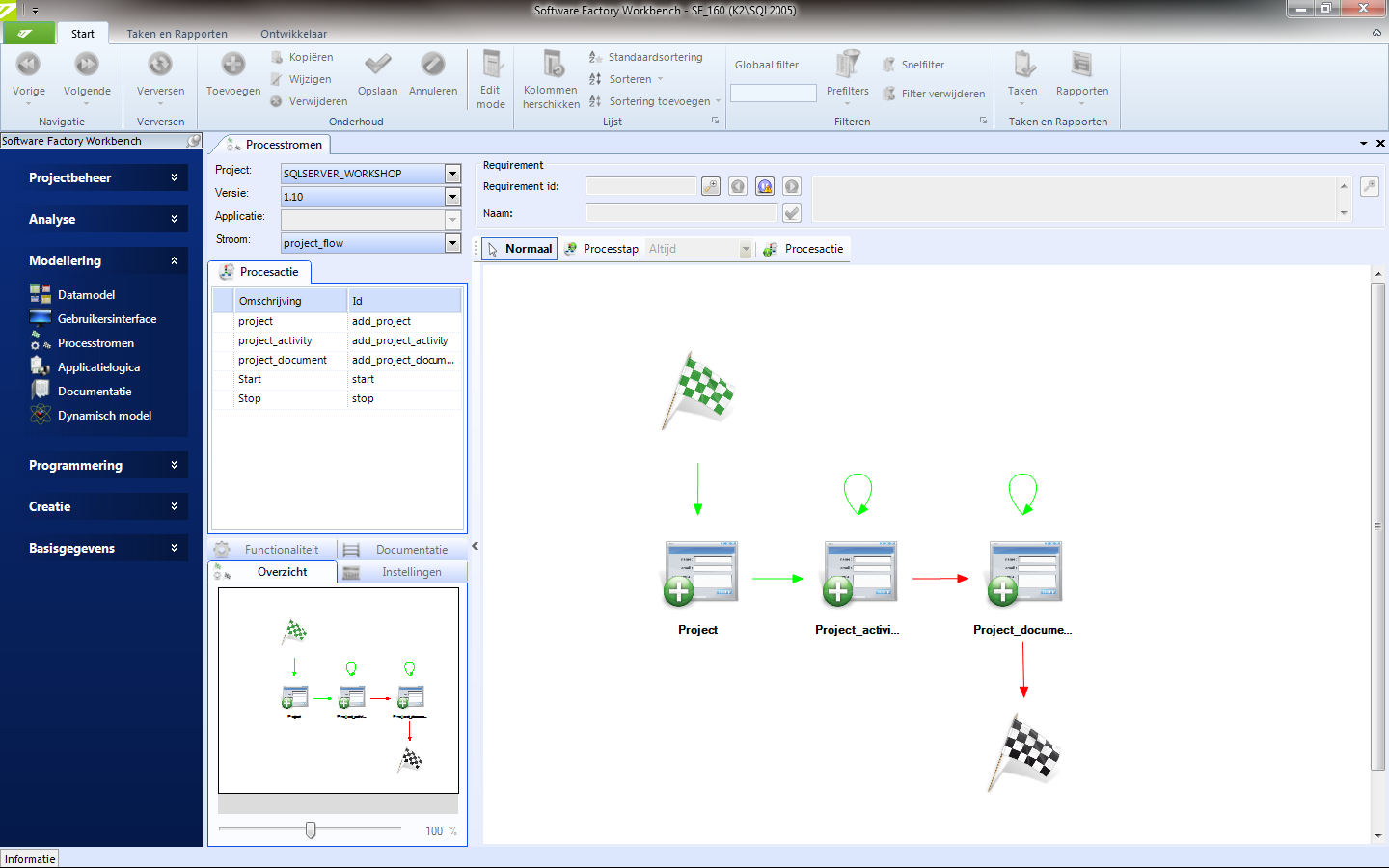
### Processtromen

Met processtromen kunnen gebruikers begeleid worden bij het uitvoeren van bepaalde acties in het eindproduct. Door gebruik van processtromen kan bijvoorbeeld tijdens de invoer van data naar het volgende invoerscherm gesprongen worden als de gebruiker klaar is met data invoeren op een eerder scherm. Een andere mogelijkheid is het automatisch uitvoeren van GUI acties na het voltooien van een eerdere actie.

Een processtroom heeft een begin- en eindpunt. Daartussen kunnen procesacties gedefinieerd worden. Deze acties kunnen verschillen van standaard DML (invoer van data in een tabel) acties tot bepaalde GUI acties (het uitvoeren van bepaalde functionaliteit). De procesacties worden met elkaar verbonden door processtappen. Een processtap kan worden gemaakt onder bepaalde condities:

* De processtap wordt genomen als de voorgaande procesactie succesvol is uitgevoerd.
* De processtap wordt genomen als de voorgaande procesactie niet succesvol is uitgevoerd.
* De processtap wordt altijd genomen, ongeacht of de voorgaande procesacties succesvol of niet succesvol uitgevoerd is.

De Thinkwise GUI zal een processtroom automatisch starten als de gebruiker een actie onderneemt die als procesactie gelijk na het startpunt van de processtroom is gedefinieerd.



Figuur 4 - Het modelleerscherm voor processtromen

## Functionality Weaver

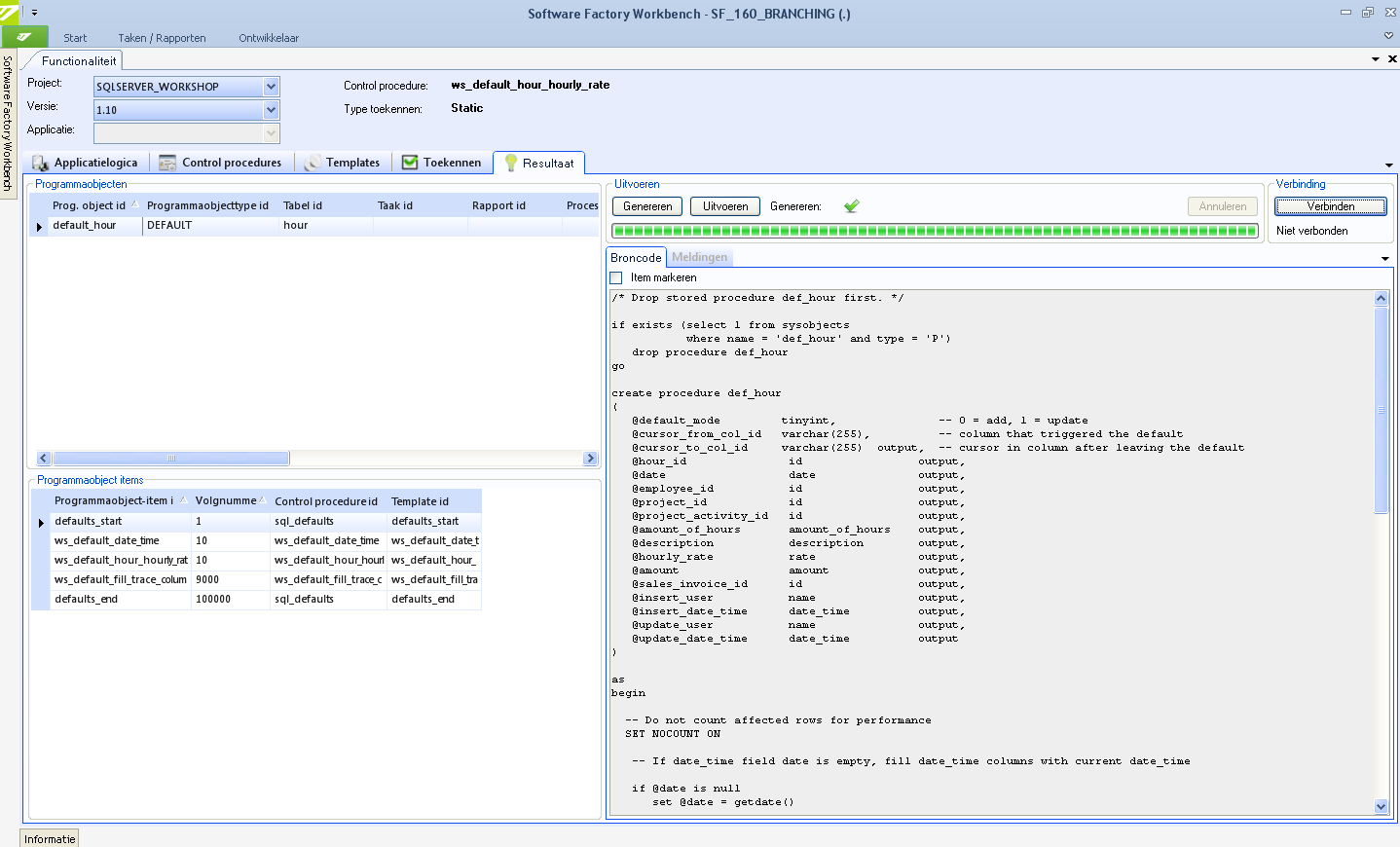
Functionaliteit binnen de Software Factory kan betrekking hebben op o.a. weergave opties, datamanipulatie en berekeningen. Met de functionality weaver kan deze functionaliteit op een aspect georiënteerde of dynamische manier in het model geplaatst worden.

De functionality weaver maakt gebruik van code templates. Dit zijn stukken code die de functionaliteit bevatten voor het eindproduct. De functionaliteit wordt verdeeld over de templates. Elke template bevat volgens conventie ook één functionaliteit.

De functionality weaver werkt door code templates toe te kennen aan objecten in het model. Deze toekenning zorgt er voor dat de functionaliteit uit de template wordt verweven met andere functionaliteit die ook aan dat object is toegewezen. Bij het genereren van de broncode wordt zo een logisch geheel van code gecreëerd dat op de database uitgevoerd kan worden.

Toekenningen zijn in twee vormen beschikbaar, te weten statisch of dynamisch. Bij statische toekenning wordt een template door de gebruiker aan de juiste objecten toegekend. Dit heeft als voordeel dat templates alleen op de plek toegekend worden waar dit nodig is. Als een template op veel plaatsen tegelijk moet worden geweven kan er dynamisch toegekend worden. Dit kan gedaan worden door met SQL code de weefcondities te definiëren. Hoofdstuk 2.5 van de scriptie gaat dieper in op de verschillende soorten functionaliteit en hoe deze aan objecten toegekend worden.

Het beheren van code binnen een projectversie gebeurt bijna exclusief binnen de schermen van de functionality weaver. De structuur die hier voor het genereren van code gehanteerd wordt is van belang voor de opdracht omdat de schermen voor het prototype dezelfde structuur aan moeten houden. Dit wordt later in de scriptie uitgebreider behandeld.



Figuur 5 - Het generatiescherm van de Functionality Weaver

### Generator

Met de generator worden de scripts gegenereerd voor het aanmaken of upgraden van de eindproduct database. Het betreft hier database scripts die voor het gespecificeerde database platform worden gegenereerd. Na het genereren kunnen de scripts direct op de database worden uitgevoerd.

De generator bevat ook een validator die het model controleert op mogelijke fouten. Validatie van het model is optioneel maar kan de gebruiker wel wijzen op fouten die tijdens het modelleren over het hoofd zijn gezien.

Bij het genereren wordt er geen GUI opgebouwd, alleen de scripts voor het opbouwen van de eindproduct database en bijbehorende functionaliteit. Het opbouwen van de GUI gebeurt runtime door het uitlezen van het meta-niveau. Hierover meer in het volgende hoofdstuk.

Het genereren van functionaliteit met de generator werkt hetzelfde als genereren met de functionality weaver met een verschil, namelijk dat de generator de functionaliteit voor alle objecten tegelijk genereert en deze groepeert in één script. Dit in tegenstelling tot de functionality weaver die functionaliteit per object apart genereert.

## Thinkwise GUI

Het opstarten en werken met eindapplicaties die met de Software Factory zijn gemaakt gebeurt met de Thinkwise GUI’s. De Thinkwise GUI’s onderscheiden zich van reguliere GUI’s door het feit dat het geen voorgebouwde schermen of applicatielogica bevat. De GUI’s worden runtime opgebouwd aan de hand van het model zoals deze in de Meta Solution Definition staat. De componenten die gebruikt worden om de schermen op te bouwen worden runtime geïninitialiseerd op de definitie van een object.

De GUI maakt gebruik van generieke componenten voor de weergave van data entiteiten. Het zelfde formulier component kan bijvoorbeeld geïnitialiseerd worden op twee verschillende tabellen. De GUI leest dan uit het meta-niveau hoe het formulier opgebouwd moet worden met betrekking tot bijvoorbeeld veldvolgorde, weergave, verplichtheid etc. Omdat de GUI met de generieke componenten wordt opgebouwd zal de werking van deze componenten hetzelfde zijn op alle schermen. Dit betekent o.a. een reductie in testwerkzaamheden omdat de werking van de componenten gewaarborgd kan worden.

Omdat de schermen runtime opgebouwd worden kunnen wijzigingen in het meta-niveau vrijwel direct getoond worden in het eindproduct. Door de weergave van een veld in het meta-niveau bijvoorbeeld op verplicht te zetten en het model opnieuw uit te lezen zal dit veld in het eindproduct nu ook verplicht zijn. De uitzondering hierop is een wijziging in het datamodel. De GUI zal proberen het datamodel uit het meta-niveau te mappen op de data entiteiten in de database van het eindproduct terwijl deze niet met elkaar overeenkomen. Om wijzigingen in het datamodel door te voeren zal er eerst een upgrade van de eindproduct database gedaan moeten worden.

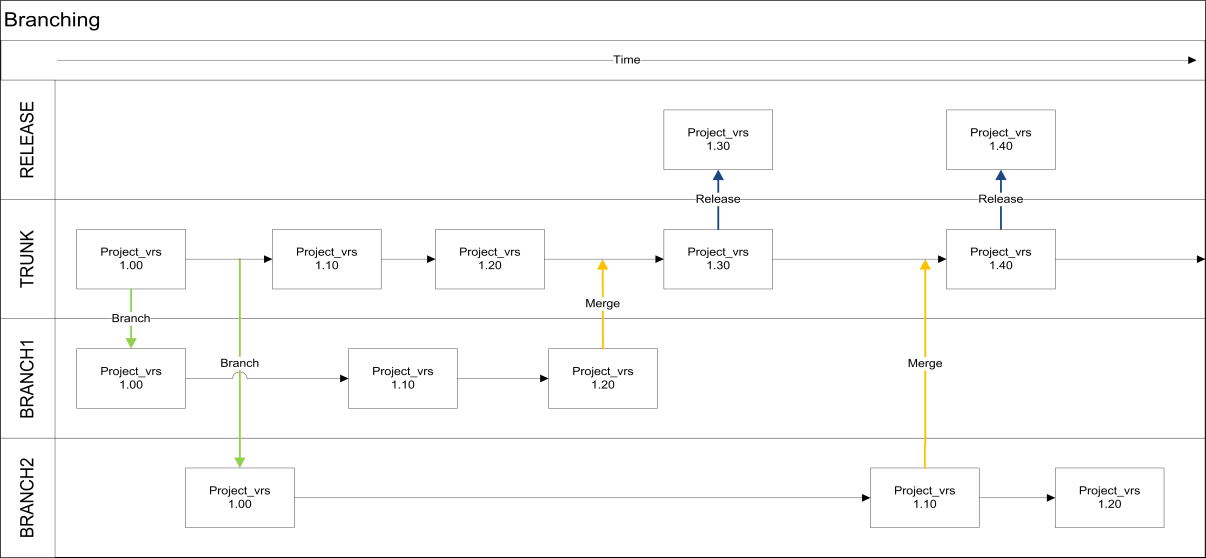
De Thinkwise GUI’s worden op een aantal platformen aangeboden. Momenteel zijn dit .NET, Java, en Java EE (web). Aan de ondersteuning van andere platformen wordt ook gewerkt. Momenteel is er een ASP.NET (web) GUI in ontwikkeling.

## Branching en merging van het datamodel

Recentelijk is de functionaliteit om te branchen en te mergen op model niveau toegevoegd aan de Software Factory. Het implementeren van deze functionaliteit is de eerste stap geweest in het ondersteunen van branching en merging. Het afstudeerproject is een uitbreiding op de functionaliteit die hier geboden wordt (hier over meer in hoofdstuk 3). Wat branching en merging toevoegt aan de Software Factory is de mogelijkheid om een project of een gedeelte ervan te kopiëren naar een eigen omgeving, genaamd de branch, waar vervolgens verder in ontwikkeld kan worden. Dit stelt meerdere projectgroepen in staat om hun eigen ontwikkeltraject te doorlopen zonder daarvoor afhankelijk te zijn van een andere projectgroep. Omdat elke projectgroep nu binnen zijn eigen omgeving werkt hoeft er bijvoorbeeld niet gewacht te worden tot het werk van een andere projectgroep stabiel is voor de werkzaamheden kunnen worden hervat.

Als het ontwikkeltraject in een branch is voltooid kan het resultaat ervan samengevoegd worden met het oorspronkelijke project, het trunkproject genaamd. Het proces van het overzetten van de wijzigingen in de branch naar de trunk wordt merging genoemd. Er wordt een verschillenanalyse gemaakt tussen de trunk en de branch waarin alle wijzigingen in het model van de branch ten opzichte van de trunk staan. Elk verschil, hier delta actie genoemd, is een wijziging die in de trunk moet worden doorgevoerd. Als er delta acties aanwezig zijn in de branch die niet op de trunk toegepast kunnen worden dan word dit een conflict genoemd. Zo lang er conflicten tussen de branch en trunk bestaan kan de merge niet voltooid worden. Om conflicten op te lossen moet er gekozen worden of de situatie uit de trunk of uit de branch behouden moet worden. De reden dat er gekozen moet worden heeft te maken met de werking van databases. Als er bijvoorbeel in de trunk een veld wordt toegevoegd waar alleen numerieke waardes ingevuld kunnen worden en in de branch wordt een veld toegevoegd met dezelfde naam waar alleen ja of nee (boolean) ingevuld mag worden levert dit een conflict op. De database kan immers niet deze twee datatypen in hetzelfde veld tegelijk gebruiken. In deze situatie moet dus voor de ene of voor de andere oplossing gekozen worden. Conflicten komen alleen in dergelijke situaties voor.

Als alle conflicten zijn opgelost kan de merge uitgevoerd worden en zal er een nieuwe projectversie gemaakt worden waarin de wijzigingen uit de trunk en de branch zijn opgenomen.



Figuur 6 - Schematische weergave van een project waar branching wordt toegepast

## Functionaliteit in de Software Factory

Functionaliteit die met de Software Factory geschreven kan worden kan een aantal vormen aannemen. De meest voorkomende daarvan staan hieronder gedefinieerd:

|  |  |
| --- | --- |
| Type functionaliteit | Beschrijving |
| Taak | Functionaliteit die expliciet door de gebruiker moet worden aangeroepen vanuit bijvoorbeeld het menu. |
| Trigger | Bevat functionaliteit die automatisch wordt uitgevoerd als er data toegevoegd, verwijdert of gewijzigd wordt. |
| Default | Wordt gebruikt om de waarde van velden te beïnvloeden op basis van waarden die in andere velden zijn ingevuld. |
| Layout | Biedt de mogelijkheid om de zichtbaarheid en muteerbaarheid van velden aan te passen. |
| Context | Wordt gebruikt om schermen te tonen of te verbergen afhankelijk van de waarden in een veld. |
| View | Functionaliteit voor het vullen van views met data. Een view is een tabel die zelf geen data bevat maar bijvoorbeeld gebruikt kan worden om data uit verschillende tabellen samen op een plek te tonen. |

Voor het schrijven van functionaliteit wordt een control procedure gemaakt. Een control procedure is een object in de database waar o.a. de naam, beschrijving en het type functionaliteit in worden opgenomen. Er zijn twee soorten control procedures, statisch en dynamisch. Het verschil tussen de twee zit voornamelijk in hoe ze templates toekennen aan objecten.

Voor de objecten die in de Software Factory gedefinieerd zijn kunnen automatisch objecten worden aangemaakt waar de functionaliteit aan kan worden toegekend. Deze objecten worden programmaobjecten genoemd. Een programmaobject is een representatie voor de functionaliteit van een object. Bijvoorbeeld, de tabel “project” heeft programmaobjecten voor taken, triggers, defaults etc.

Met behulp van de functionality weaver worden er code templates toegekend aan de programmaobjecten. De control procedures beschrijven hoe de functionality weaver de templates aan programmaobjecten toe moeten kennen. Bij het toekennen wordt zoals eerder vermeld onderscheid gemaakt tussen statische en dynamische toekenningen. Een statische toekenning houdt in dat een template door de gebruiker toegekend wordt aan een programmaobject en dynamisch toekennen betekent dat de templates aan de programmaobjecten toegekend worden aan de hand van SQL code in de vorm van een stored procedure op de database van het meta-niveau.

Voor het generen van de uiteindelijke code kijkt de Software Factory voor elk programmaobject welke templates er aan toegekend zijn. De generator voegt vervolgens alle templates samen tot een logisch geheel en genereert daaruit de code voor het eindproduct. Het eindresultaat is dus één script waar alle templates in opgenomen zijn. Om de functionaliteit uit dit script op de nemen in het eindproduct hoeft het script nu alleen nog maar afgevuurd te worden op de eindproduct database.

# Projectbeschrijving

## Probleembeschrijving

Het ontwikkeltraject van een product binnen de Software Factory speelt zich af binnen een projectversie. Omdat alle ontwikkeling binnen dezelfde projectversie plaatsvindt kan het voorkomen dat er meerdere projectgroepen tegelijk aan dezelfde projectversie werken, wat kan leiden tot een situatie waarbij de werkzaamheden van de projectgroepen met elkaar in conflict raken. Het kan bijvoorbeeld zo zijn dat de testwerkzaamheden van een projectgroep stil komen te liggen omdat een andere projectgroep nog niet klaar is met zijn werkzaamheden. Zodanig kan er nog geen stabiel eindproduct opgestart worden om de testwerkzaamheden op uit te voeren.

Om dit probleem op te lossen is de functionaliteit om het model te branchen en te mergen geïntroduceerd. Hiermee is het mogelijk om het model in een aparte ontwikkelomgeving (genaamd de branch) te kopiëren waar de projectgroep los van de andere projectgroepen in kan werken. Als het branchproject succesvol is uitgevoerd kunnen de wijzigingen in het model weer teruggebracht worden naar het oorspronkelijke project (de trunk) om zo de uiteindelijke projectversie te creëren.

Hoewel deze implementatie een goede oplossing biedt voor het model is de ondersteuning voor functionaliteit nog ontoereikend. Wijzigingen in de broncode voor de functionaliteit worden in de huidige implementatie niet bijgehouden en de Software Factory biedt geen mogelijkheid om de broncode tussen verschillende projectversies met elkaar te vergelijken.

Door het ontbreken van deze functionaliteit wordt de effectiviteit van het branchen en mergen verminderd. De gebruikers van de Software Factory hebben duidelijk gemaakt dat dit een essentieel onderdeel is dat nodig is om succesvol parallel te kunnen ontwikkelen en hebben een sterke wens uitgedrukt voor de implementatie ervan.

## Opdrachtomschrijving

De vraag die centraal stond tijdens het uitvoeren van het project was:

“Wat is er nodig om branching en merging van broncode in de Software Factory te ondersteunen en hoe kan dit geïmplementeerd worden?”

Om deze vraag te beantwoorden is er gekeken op welke punten de huidige implementatie verbeterd kon worden. Hier zijn een aantal onderdelen uit naar voren gekomen.

* Broncode vergelijking en merging
* Automatische migratie van broncode
* Control procedures in verschillende projectversies

Aanvankelijk zijn alleen de eerste twee onderdelen als benodigde punten naar voren gekomen. Tijdens de uitvoering van het project bleek echter dat de manier waarop de Software Factory omgaat met control procedures niet goed samen gaat met de manier waarop branching geïmplementeerd is. Het aanpassen van de manier waarop control procedures gebruikt worden is daarom als derde punt aan de opdracht toegevoegd.

De opdracht was het ontwikkelen van een prototype waarin deze punten geadresseerd werden en hiermee een mogelijke oplossing voor de problemen uit de probleembeschrijving te bieden.

### Broncode vergelijking en merging

De ondersteuning voor branching en merging in de Software Factory is momenteel beperkt tot het model. Omdat in een branchproject redelijkerwijs ook broncode zal worden gewijzigd is de functionaliteit om deze te mergen met de broncode van de trunk gewenst.

Het prototype moest aan de volgende eisen voldoen:

* De Software Factory detecteert automatisch welke broncode bestanden uit de trunk en de branch bij elkaar horen en toont deze in een lijst.
* De Software Factory kan de tekst uit de broncode bestanden zij aan zij tonen en markeert daarbij de plaatsen waar verschillen zijn opgetreden.
* De wijzigingen die in de branch hebben plaatsgevonden moeten in de broncode van de trunk kunnen worden overgenomen.
* Wijzigingen die met de merging tool gemaakt zijn moeten kunnen worden opgeslagen of worden teruggedraaid.

Door de Software Factory uit te breiden met deze functionaliteit zal branching en merging voortaan ondersteuning kunnen bieden voor alle aspecten van het ontwikkelproces.

### Automatische migratie van broncode

Elk project in de Software Factory heeft een pad op de harde schijf waar bijbehorende bestanden worden opgeslagen. Omdat een branchproject als apart project wordt beschouwd wordt hier ook een nieuw pad voor aangemaakt op de harde schijf. Dit blijft echter beperkt tot het aanmaken van het pad; er is nog geen functionaliteit aanwezig voor het kopiëren van de bijbehorende bestanden. Dit wordt tot nu toe nog handmatig gedaan.

Door het kopiëren van de bestanden te automatiseren kan het gebruiksgemak van de Software Factory worden verhoogd en wordt het uitvoeren van een foutgevoelige handmatige actie overbodig.

### Control procedures in verschillende projectversies

Voor het dynamisch toekennen van functionaliteit in de Software Factory wordt gebruik gemaakt van control procedures. Omdat de control procedures in de MSD zijn gedefinieerd worden ze ook in de database van de MSD opgenomen.

Deze manier van werken kan bij het werken in meerdere projectversies problematisch zijn. De control procedures krijgen op de database dezelfde naam. Als in een projectversie een aanpassing gedaan wordt aan een control procedure dan wordt deze op de database overschreven voor alle projecten die er gebruik van maken.

Om dit te voorkomen moet er een alternatieve implementatie voor de control procedures bedacht worden zodat er zonder problemen in meerdere projectversies tegelijkertijd met control procedures gewerkt kan worden.

## Technische specificaties

De technische eisen die aan het project gesteld werden waren dat het prototype ontwikkeld moest worden in C# en dat er gebruik gemaakt moest worden van een SQL Server 2005 database. De ontwikkeltools die tot de beschikking stonden waren Visual Studio 2008 en SQL Server Management Studio 2005.

De systeemeisen voor het eindproduct zijn minimaal Windows XP Service Pack 2 met .NET Framework 3.5 of hoger geïnstalleerd.

# Projectaanpak

## Projectmanagement methode

Voor het afstudeerproject is gekozen om gebruik te maken van de waterval methode. Werken volgens de waterval methode houdt in dat de bouw van een project wordt opgedeeld in fasen. Het werk aan de volgende fase begint pas als de vorige fase volledig is afgerond.

Er is gekozen voor deze projectmethode omdat het geschikt is voor kleine project teams (in dit geval één persoon) en omdat documentatie een belangrijke rol speelt. Bij de oplevering van het eindresultaat zal er voldoende documentatie aanwezig zijn om het project over te kunnen dragen aan een andere medewerker.

De fasering die in de watervalmethode wordt gehanteerd is aangepast om aan te sluiten bij het doel van het project, namelijk, het onderzoek doen naar een oplossing voor het mergen van bestanden en het bouwen van een prototype ervan. De fasering is als volgt ingedeeld:

* Plan van aanpak
* Onderzoek
* Ontwerpfase
* Bouw / Prototyping

## Fasen

In dit hoofdstuk wordt het traject dat in elke fase doorlopen is behandeld.

### Plan van aanpak

In deze fase is het plan van aanpak opgesteld. In het plan van aanpak zijn aan de hand van de opdrachtomschrijving de onderdelen benoemd die het op te leveren eindproduct moet bevatten.

In het plan van aanpak is gespecificeerd hoe deze onderdelen tot stand gebracht zullen worden, welke projectactiviteiten daarvoor benodigd zijn, hoe de kwaliteit van het eindproduct gewaarborgd wordt en wat de projectgrenzen en risico’s zijn. Daarnaast is er ook een planning gemaakt waarin het behalen van mijlpalen centraal is gesteld.

### Onderzoek

Voor de onderzoeksfase is onderzoek gedaan naar een bestaande oplossing voor het probleem dat centraal staat in de opdracht, namelijk het vergelijken van code.

Voor het onderzoek is gekeken naar een aantal oplossingen en is voor elke oplossing een testcase gemaakt waarin het component getoetst werd aan de gestelde eisen. De bevindingen hiervan zijn gedocumenteerd in een onderzoeksverslag. In het verslag wordt de uiteindelijke keuze voor een component beargumenteerd aan de hand van de bevindingen die gedaan zijn.

### Ontwerpfase

Tijdens deze fase is het functioneel ontwerp opgesteld.

In het functioneel ontwerp staan de functionele eisen gedefinieerd die op het project van toepassing zijn. Vervolgens zijn er use cases gedefinieerd waarin de functionele eisen zijn uitgewerkt tot scenario’s zoals deze in het prototype geïmplementeerd zullen worden. Tot slot zijn er schermontwerpen gemaakt die een visualisatie geven van hoe de use cases uiteindelijk in de GUI getoond zullen worden.

### Bouw / Prototyping

In de bouwfase wordt het prototype gerealiseerd. Aan de hand van de ontwerpen die in de ontwerpfase zijn gemaakt en de keuzes die gemaakt zijn tijdens het uitvoeren van het onderzoek is een functioneel prototype opgebouwd.

De bouw van het prototype dient als ondersteuning voor de het beantwoorden van de hoofdvraag in de scriptie. Er is tijdens de bouw dus zorgvuldig gekeken of het prototype aan de gestelde eisen voldeed.

### Afwijkingen

In de originele planning is het schrijven van een technisch ontwerp opgenomen. Het technisch ontwerp is na overleg echter geschrapt van de planning. De reden voor het schrappen van het technisch ontwerp is omdat er gebruik gemaakt zou worden van een extern component. Omdat de werking van het component in deze fase van het project nog onvoldoende duidelijk was is besloten om het technisch ontwerp buiten de scope van de afstudeeropdracht te plaatsen.

# Onderzoek

## File comparison engine

Bij aanvang van het project is de overweging gemaakt of er voor een bestaande file comparison tool gezocht zou worden of dat er zelf een file comparison engine geschreven zou worden. Na een kort onderzoek bleek dat het zelf schrijven van een dergelijke tool te complex zou zijn en te veel tijd in beslag zou nemen.

Er is dus gekozen voor het implementeren van een bestaande oplossing. Hier voor werd gezocht naar een component dat de mogelijkheid had om twee tekstbestanden naast elkaar te leggen en vervolgens de verschillen tussen de twee bestanden aan kon wijzen. Het gekozen component moest wel aan een aantal voorwaarden voldoen. Zo moest het component gebruikt kunnen worden binnen het .NET Framework en bij voorkeur geschreven zijn in C#. Ook moest de oplossing volledig in de Software Factory geïntegreerd kunnen worden. Dit hield in dat de tool niet buiten de Software Factory om mocht werken en dus ook geen nieuwe schermen buiten de Software Factory mocht openen. Verder was er de voorkeur dat er voor een open source oplossing gekozen zou worden. De keuze voor open source is gemaakt omdat er bij open source projecten vaak minder restricties liggen op het aanpassen van de broncode. Een open source project kon dus makkelijker naar wens aangepast worden.

Bij het zoeken naar geschikte componenten kwam er al snel een probleem naar voren. Er bleken vrij weinig losstaande file comparison engines te zijn die voldeden aan de genoemde eisen. Er waren veel applicaties te vinden die de functionaliteit bevatten die gezocht werd, maar dit waren vaak echter commerciele applicaties of applicaties die niet open source waren. De file comparison engines die in deze applicaties gebruikt werd konden dus niet in de Software Factory geïntegreerd worden.

Uiteindelijk is de keuze gevallen op een aantal file comparison engines die er veelbelovend uitzagen. Dit waren:

* DiffUtils
* Diff.Net
* Google Diff\_Match\_Patch
* A Generic, Reusable Diff Algorithm in C#

In de bijlagen is het onderzoeksverslag opgenomen. Hier worden de eigenschappen van de verschillende componenten met elkaar vergeleken.

Om een keuze te maken uit de verschillende componenten is een kleine testcase opgesteld. Voor elk component werd een klein programma geschreven dat het component als file comparison engine implementeerde. Het programma gaf de file comparison engine engine vervolgens twee tekstbestanden die op enkele punten van elkaar verschilden. Nadat de bestanden met elkaar waren vergeleken werd de output van de file comparison engine geanalyseerd. Hierbij werd gelet op zaken zoals de leesbaarheid van de output, het formaat waarin de output stond en of de output betrouwbaar was (geen gemiste verschillen of duidelijke fouten).

Vervolgens is uit de analyse een conclusie getrokken.

## Conclusie uit het onderzoek

Uit het onderzoek zijn uiteindelijk twee goede kandidaten naar voren gekomen voor de file comparison engine. Dit waren Google Diff\_Match\_Patch en het Generic Reusable Diff Algorithm (GRDA). Uit de testcases bleek dat deze twee de enige oplossingen waren waarvan het resultaat voldoende bruikbaar was. DiffUtils viel al snel af als mogelijke kandidaat omdat de engine erg lastig in C# te implementeren was. Diff.Net kwam negatief uit de test naar voren, voornamelijk omdat de broncode niet beschikbaar was en de documentatie geen goede indicatie gaf hoe er met het component gewerkt moest worden.

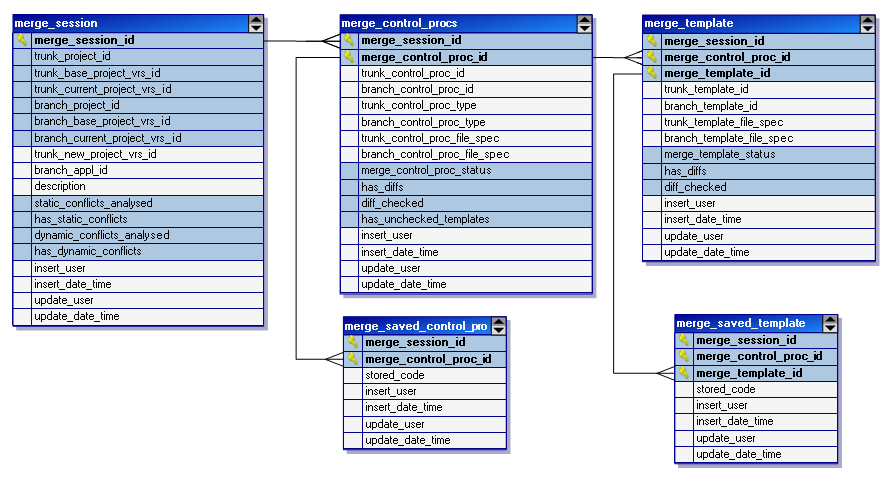
Google Diff\_Match\_Patch en GRDA bleven dus als enige opties open. In eerste instantie is de keuze gemaakt om Diff\_Match\_Patch te gebruiken voor het project. De redenatie hiervoor was dat dit project nog steeds ondersteund wordt door de ontwikkelaar. De engine was op zich krachtig genoeg om grote files met elkaar te kunnen vergelijken en de ontwikkelaar bracht regelmatig updates uit om de engine stabieler en sneller te maken.

Na enige tijd met deze engine gewerkt te hebben kwamen er echter een aantal tekortkomingen aan het licht. Het grootste probleem met Diff\_Match\_Patch was dat het een tekstvergelijking gaf op karakterbasis en niet op regelbasis. Dit houdt in dat de tekst karakter voor karakter met elkaar vergeleken wordt in plaats van een regel uit de tekst met een andere regel te vergelijken. Een vergelijking op karakterbasis is goed voor het vergelijken van gewone tekst, maar voor gestructureerde tekst, zoals SQL code, is een vergelijking op regelbasis beter omdat hiermee de structuur van de code intact blijft.

Er is dus uitendelijk gekozen voor GRDA. Deze engine maakt wel gebruik van een tekstvergelijking op regelbasis en was dus beter geschikt voor het doel van het project. Bovendien is het ook een erg krachtige file comparison engine die om kan gaan met grote files en is de implementatie volledig open source.

# Implementatie

## Datamodel

De eerste stap in het bouwen van het prototype is het samenstellen van het datamodel. Het datamodel is ontworpen met de ERD modeler in de Thinkwise Software Factory en is te zien in de onderstaande screenshot.

Figuur 7 - Het datamodel voor het prototype

Het datamodel bestaat uit de volgende tabellen:

* Merge\_session
* Merge\_control\_procs
* Merge\_template
* Merge\_saved\_control\_procs
* Merge\_saved\_template

Het proces van het mergen van de code begint zodra een merge sessie op model niveau is afgerond. In de merge sessie tabel staan de gegevens die nodig zijn om de control procedures en templates in het model te vinden, namelijk het project en projectversie van de trunk en de branch.

Aan de hand van de gegevens uit de merge sessie tabel worden de “merge\_control\_proc” en “merge\_template” tabellen gevuld. Aan de hand van een primary key vergelijking worden de control procedures en templates uit de beide projectversies met elkaar gekoppeld.

De eerste vergelijking vindt hier al gelijk plaats. Als een control procedure of template wel in de trunk voorkomt maar niet in de branch betekent dat in dit geval dat de control procedure verwijderd is. Als de situatie omgekeerd is betekent dit dus dat de control procedure of template toegevoegd is. De status wordt ingevuld in het status veld.

Op de control procedures zit een standaard sortering die sorteert op het type van de control procedure. Er wordt onderscheid gemaakt tussen dynamische en statische control procedures. Dynamische control procedures bevatten broncode, statische control procedures niet. De prioriteit van dynamische control procedures is dus hoger dan die van statische en worden zodoende als eerste getoond.

De tabellen “merge\_saved\_control\_procs” en “merge\_saved\_templates” worden gebruikt om de gewijzigde code op te slaan. De wordt hier tijdelijk opgeslagen tot de gebruiker er voor kiest om de code weg te schrijven naar de harde schijf.

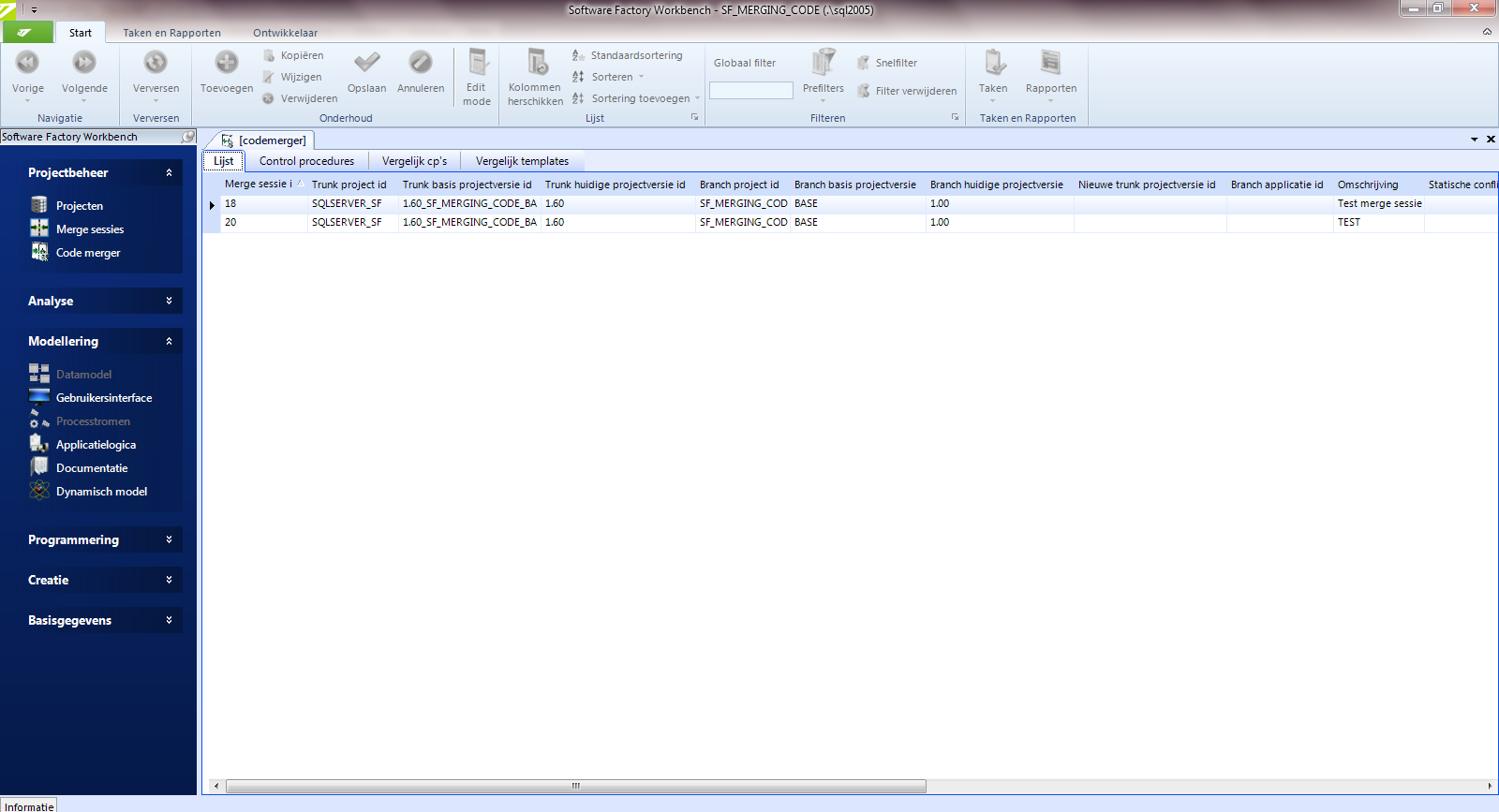
## Code merging scherm

Voor de ondersteuning van het merge proces is er aan de Software Factory een nieuw scherm toegevoegd. Hiervoor is een nieuw menu item genaamd “code merging” toegevoegd aan de menustructuur. Het scherm bestaat uit vier verschillende onderdelen welke verdeeld zijn over vier tabbladen.

In het eerste tabblad kan een merge sessie geselecteerd worden. Een merge sessie wordt gestart op model niveau (zie hoofdstuk 2.3). Pas als een merge sessie volledig doorlopen en voltooid is op model niveau wordt de merge sessie getoond op het code merging scherm. Dit is om te voorkomen dat er wijzigingen aan de broncode worden gemaakt terwijl er nog geen definitief datamodel is. Er kan dus pas een merge sessie gekozen worden als deze het merge proces op model niveau al doorlopen heeft.

In het scherm wordt een aantal eigenschappen getoond van de merge sessie, waaronder:

* Het trunk- en branchproject
* De huidige projectversies uit de trunk en de branch
* De beschrijving die bij het aanmaken van de merge sessie is opgegeven.



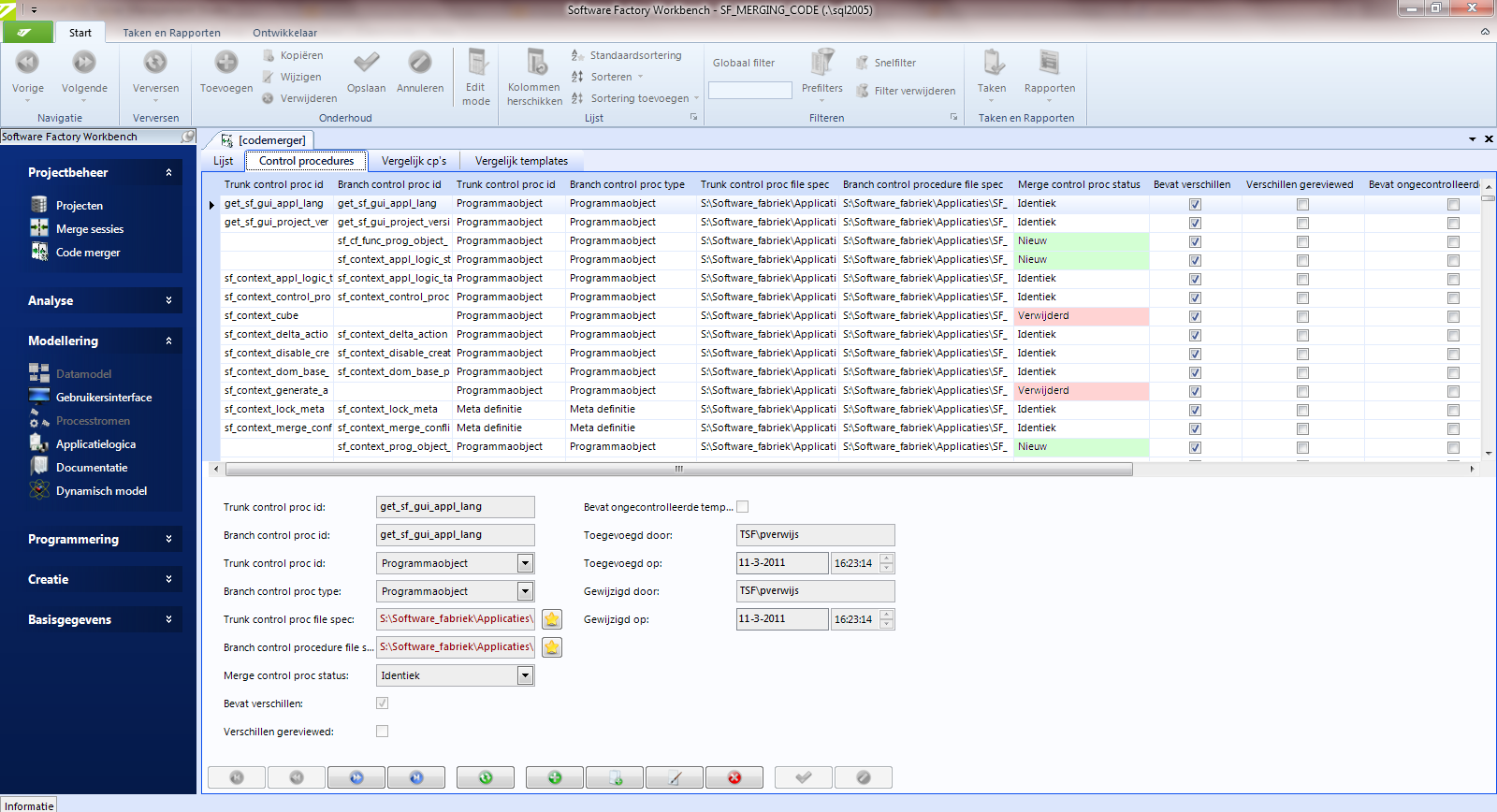
Figuur 8 - Het merge sessie scherm. Hier wordt een keuze gemaakt uit de beschikbare merge sessies.

Het volgende tabblad is “control procedures”. Hier wordt een overzicht getoond van alle control procedures die zich binnen het trunk- en branchproject bevinden.

Op dit tabblad kunnen de control procedures die vergeleken moeten worden gekozen worden. Op deze pagina staan een aantal eigenschappen genoteerd van de control procedures, namelijk:

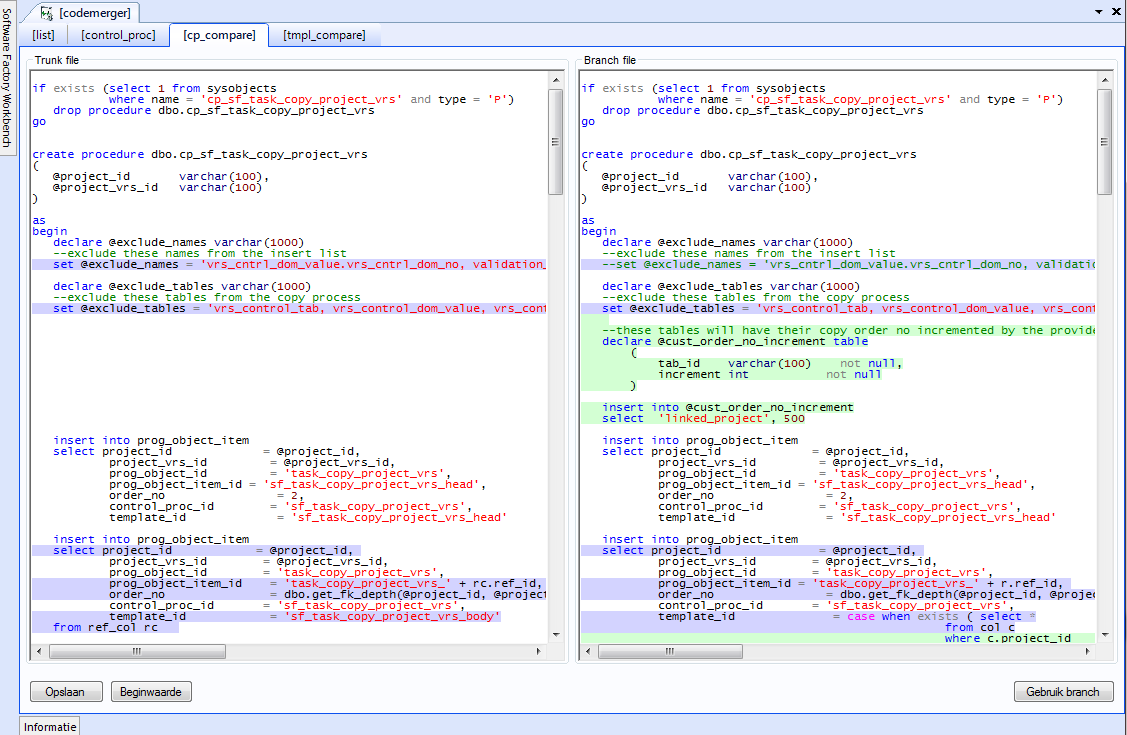
* Het **id** van de control procedure, zowel in de trunk als de branch.
* Het **type** van de control procedure.
* Het **pad** waar de control procedure op de harde schijf wordt opgeslagen (als deze bestaat).
* De **status** van de control procedure.
* Een vinkje dat aangeeft of er **verschillen** in de control procedures aanwezig zijn.
* Een vinkje dat aangeeft of er door de gebruiker al **naar de verschillen is gekeken**.
* Een vinkje dat aangeeft of de control procedures nog templates bevat waar nog niet door een gebruiker naar gekeken is.

In de lijst wordt gebruik gemaakt van kleurcodering om de status van de control procedures aan te geven. Een nieuwe control procedure wordt groen gemarkeerd en een verwijderde control procedure wordt rood gemarkeerd. Gewijzigde control procedures krijgen een blauwe markering.



Figuur 9 - Het control procedure tabblad. Hier worden alle control procedures uit de trunk en branch getoond. De status wordt met een kleur gemarkeerd zodat snel duidelijk is in welke control procedures wijzigingen zitten.

Als er een dynamische control procedure is geselecteerd kan er naar het volgende tabblad genavigeerd worden. Dit tabblad bevat het code merge component voor de control procedures. Dit tabblad zal alleen geselecteerd kunnen worden als er een dynamische control procedure is geselecteerd. Bij statische control procedures wordt dit tabblad verborgen; statische control procedures bevatten immers geen code.



Figuur 10 - Voorbeeld van het mergen van control procedures

Dit scherm is in twee delen opgedeeld. Aan de linkerzijde wordt de code zoals deze zich in de trunk bevind getoond en rechts wordt de code uit de branch getoond. Verschillen tussen de twee versies worden gemarkeerd. De kleur van de markering geeft aan om welke wijziging het gaat. Rood betekent dat een dit deel van de code verwijderd is in de branch, groen houdt in dat deze code nieuw is in de branch en blauw betekent dat er een wijziging is opgetreden tussen de trunk en de branch.

Door op een gemarkeerd stuk code te rechtsklikken wordt het context menu geopend. In het context menu kan gekozen worden of de code uit de branch of uit de trunk behouden moet worden. De optie die gekozen wordt zal in het linker tekstveld weergegeven worden. Het linker tekstveld zal daarom ook altijd de code bevatten zoals deze in de nieuwe projectversie zal komen.

Tot slot bevat dit tabblad nog een aantal knoppen, te weten:

* Opslaan
  + Hiermee wordt de tekst in het linker veld opgeslagen in de database.
* Beginwaarde
  + Met deze knop kan het linker veld weer gevuld worden met de oorspronkelijke code uit de trunk.
* Gebruik branch
  + Deze knop kopieert alle tekst uit het rechter veld naar het linker veld. Op deze manier kan alle code uit de branch op een snelle manier in de trunk geplaatst worden.

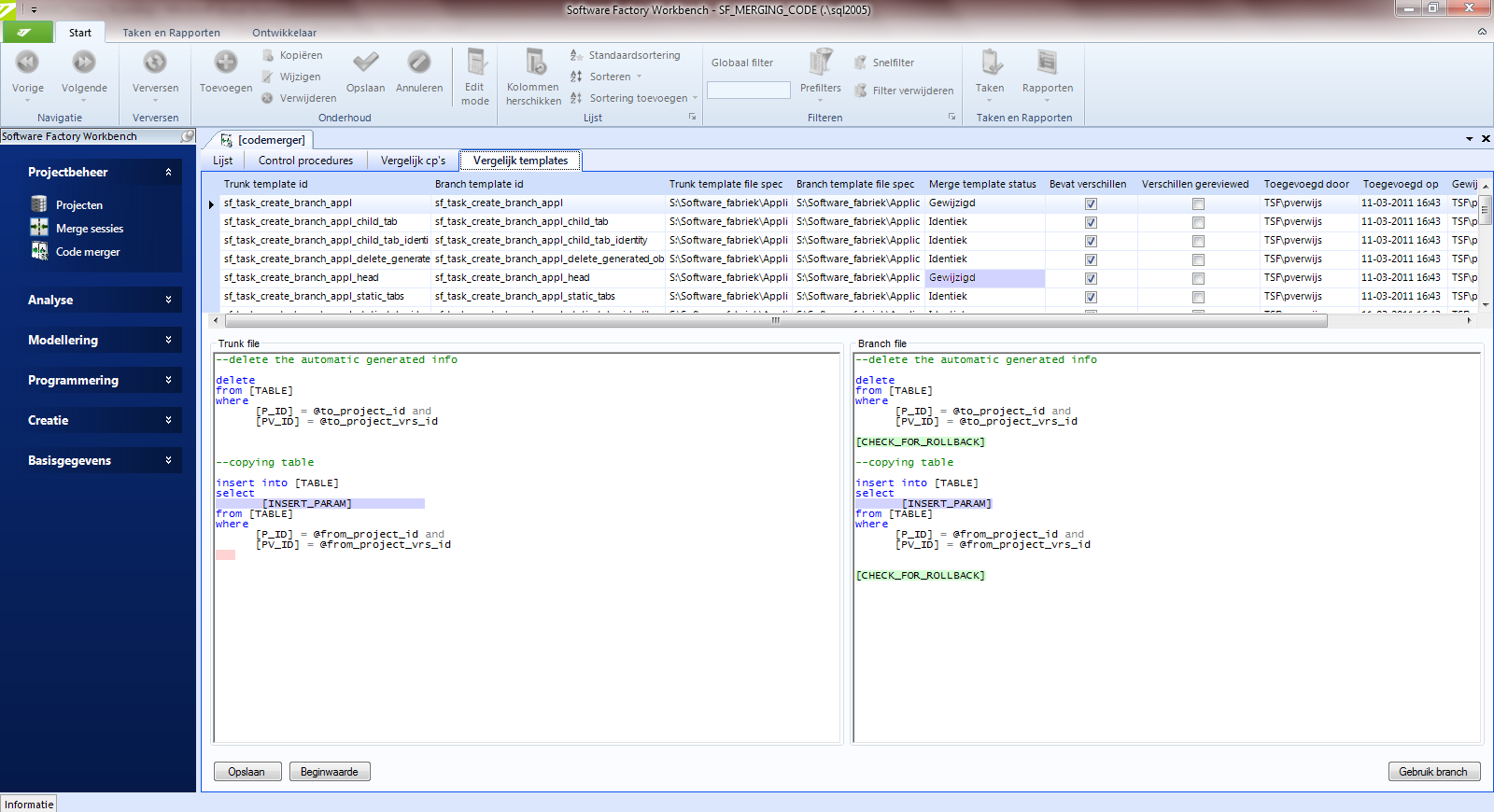
Het laatste tabblad bevat het code merge component voor de templates.

Dit tabblad bestaat uit een lijst met de beschikbare templates en een code merge component. Het code merge component op dit tabblad is qua functionaliteit gelijk aan dat op vorige tabblad. In de lijst worden de templates getoond die bij de eerder geselecteerde control procedure passen. Ook van de templates worden een aantal eigenschappen getoond in de lijst:

* Het **id** van de template, zowel in de trunk als de branch.
* Het **pad** waar de template op de harde schijf wordt opgeslagen (als deze bestaat).
* De **status** van de template.
* Een vinkje dat aangeeft of er **verschillen** tussen de templates aanwezig zijn.
* Een vinkje dat aangeeft of er door de gebruiker al **naar de verschillen is gekeken**.

Ook hier wordt gebruik gemaakt van kleurcodering om de status van de templates aan te geven. Deze is hetzelfde als voor de control procedures.

Door een template te selecteren uit de lijst zal de verschillenanalyse worden weergegeven in het merge component onder de lijst. Deze zal direct na het selecteren getoond worden.



Figuur 11 - Het template merge tabblad.Boven in het scherm is een lijst met beschikbare templates weergegeven. Onder in het scherm wordt de vergelijking tussen de geselecteerde templates getoond.

## Aanpassingen aanroep control procedures

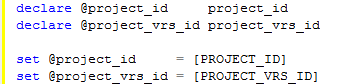
Om er voor te zorgen dat control procedures die in een branch projectversie op de database worden uitgevoerd niet de control procedures van een andere projectversie overschrijven is de huidige implementatie van de control procedures gewijzigd.

De huidige implementatie hield in dat de control procedures als stored procedures op de database stonden. Er is besloten om van dit concept af te stappen en in plaats ervan gebruik te maken van database scripts.

De functie die control procedures dienen is kort gezegd het ophalen van de data die gebruikt wordt om de parameters van een template in te vullen. De Thinkwise GUI maakt een call naar de stored procedure op de database die vervolgens het meta-niveau aanvult met de data die in de parameters van de templates moet worden geplaatst.

De nieuwe implementatie verandert dit door de stored procedure call in de GUI te vervangen met een script dat direct op de database wordt uitgevoerd. Het script dat wordt uitgevoerd is hetzelfde als de body van de stored procedure. Het verschil is dat de kop en de staart ontbreken.

In de control procedures wordt gebruik gemaakt van variabelen. Deze variabelen bevatten de project en projectversie die de context van de control procedure aangeven. Omdat de kop van de stored procedure niet in het script staat worden er dus geen waardes toegekend aan deze variabelen. Om dit op te lossen wordt er door de Thinkwise GUI een kop aan het script toegevoegd waarin de variabelen gedeclareerd worden. Deze kop ziet er als volgt uit.



Figuur 12 - Voorbeeld kop voor control procedure scripts

De geparameteriseerde waardes “[PROJECT\_ID]” en “[PROJECT\_VRS\_ID]” kunnen door de Thinkwise GUI vervangen worden. Omdat de control procedure uitgevoerd wordt op schermen waarbij de context van het project en projectversie bekend moeten zijn kan de Thinkwise GUI deze waardes gebruiken om de parameters in te vullen.

Als de GUI de kop en de body gereed heeft dan kan het script op de database uitgevoerd worden. Dit heeft dan hetzelfde effect als de control procedure als stored procedure aan te roepen.

Door de control procedures als scripts uit te voeren kunnen een aantal voordelen behaald worden. Ten eerste zullen er nu geen conflicten tussen verschillende projectversies optreden. De scripts worden vanaf de harde schijf uitgelezen en elke projectversie krijgt een eigen folder op de harde schijf waar de scripts in worden opgeslagen. Als een script wordt aangepast gebeurt dit dus alleen voor de specifieke projectversie.

Verder zorgt deze implementatie er voor dat er veel minder stored procedures op de database komen te staan en dus de grootte van de database kan worden verkleind. Bovendien is het makkelijker om overzicht te houden als het aantal stored procedures wordt gereduceerd.

Een nadeel aan deze implementatie is echter wel dat de scripts op een plek moeten staan waar de gebruiker er toegang tot heeft. Voorheen kon elke gebruiker met toegang tot de database de control procedures uitvoeren. Nu is het echter ook vereist dat de gebruiker toegang heeft tot de harde schijf waar de scripts zijn opgeslagen.

## Automatisch kopiëren van broncode

Het automatisch kopiëren van broncode is geïmplementeerd om het proces van het kopiëren van projectversies te versimpelen.

Om deze functionaliteit te implementeren is een nieuwe taak aangemaakt genaamd “copy\_project\_vrs\_source\_code”. Deze taak roept een methode aan in de Thinkwise GUI. De bestandspaden van de broncode bestanden worden uit het meta-niveau uitgelezen en aan deze methode meegegeven, alsmede het project pad van de nieuwe projectversie. De methode in de GUI zorgt er vervolgens voor dat de bestanden naar de juiste paden wordt gekopieerd.

Om er voor te zorgen dat dit automatisch gebeurt na het kopiëren van een project is er een processtroom aangemaakt. Na het succesvol uitvoeren van de taak “copy\_project\_vrs” wordt de taak “copy\_project\_vrs\_source\_code” automatisch gestart.

Het kopiëren van broncode gebeurt nu dus automatisch na het kopiëren van een projectversie. Omdat branching veel gebruik maakt van het kopiëren van projectversies is het gebruikersgemak er van aanzienlijk verbetert; het aantal handmatige acties is nu sterk teruggedrongen.

# Conclusie

Het prototype is succesvol gebouwd en geïntegreerd in de Thinkwise Software Factory. De bestaande implementatie van branching en merging is hiermee uitgebreid zodat deze voldoet aan de eisen en wensen zoals deze door Thinkwise Software en haar klanten gesteld zijn.

Het prototype biedt de mogelijkheid om de broncode tussen een trunk- en een branch projectversie met elkaar te vergelijken en de wijzigingen in de branch op te nemen in de trunk. Dankzij deze uitbreiding is branching en merging in de Software Factory een volwaardige feature geworden die nu alle aspecten van het ontwikkeltraject ondersteund.

De aanpassingen die gedaan zijn aan de Software Factory zijn van groot belang geweest voor het soepel verlopen van het branchen en mergen. Door de control procedures te vervangen met scripts is een van de grootste tekortkoming van de Software Fabriek met betrekking tot werken in meerdere projectversies opgelost. Het is voor de ontwikkelaars nu mogelijk om wijzigingen te maken aan de code zonder zich zorgen te maken over de impact die dit kan hebben op het werk van andere projectteams. Elke projectversie heeft nu dus een compleet eigen ontwikkelomgeving.

Tot slot is het automatisch kopiëren van broncode succesvol geïmplementeerd. Dit zorgt er voor dat het maken van een kopie van een projectversie, een actie die voor de werking van branching en merging zeer belangrijk is, volledig automatisch is geworden.

Het prototype heeft dus aan zijn doel voldaan. Het heeft succesvol bewezen dat het mogelijk om deze functionaliteit te ondersteunen in de Software Factory en heeft daarbij ook aangetoond hoe dit op een goede manier geïmplementeerd zou kunnen worden.

Wel moet vermeld worden dat deze functionaliteit alleen een technische oplossing kan bieden voor de verschillen die in de code zijn ontstaan. Conflicten die ontstaan door slechte communicatie tussen projectgroepen vallen onder projectmanagement, wat buiten de scope van deze tool ligt. Om dit soort conflicten te voorkomen kan er beter gebruik worden gemaakt van requirements, waar de Software Factory ook ondersteuning voor biedt.

# Aanbevelingen

## Branching en merging

Het branchen en mergen van het datamodel en broncode is verdeeld over twee schermen. Voor de eindgebruiker zou het echter duidelijker kunnen zijn als alles met betrekking tot branching en merging op hetzelfde scherm plaatsvind. Vanwege technische limitaties in de Software Factory is dit op het moment nog niet mogelijk.

Het zou in de toekomst aan te raden zijn om het code merge scherm op te nemen in het bestaande branching en merging scherm. Het is in de GUI mogelijk om tabbladen te verbergen en te tonen afhankelijk van de staat van de merge sessie. Als de merge sessie zich nog in de fase bevind van het datamodel mergen dan kunnen alleen de tabbladen met betrekking tot het datamodel getoond worden. Zodra de merge sessie in de code merge fase beland kunnen deze tabbladen weer verborgen worden en alle tabbladen voor het code mergen getoond worden.

## Thinkwise Software Factory

Voor de Thinkwise Software Factory in het algemeen kunnen er ook een aantal aanbevelingen worden gedaan.

### Opslag broncode

Hoewel er nu een oplossing is bedacht voor het mee kopiëren van broncode naar nieuwe projectversies is de huidige implementatie nog steeds niet ideaal. Afhankelijk van de werkstijl van de gebruiker kan de broncode nog steeds zeer verspreid worden opgeslagen op de harde schijf. Vanuit Thinkwise worden richtlijnen gegeven met betrekking tot het opslaan van broncode, maar de eindgebruiker wordt niet gedwongen om zich daar aan te houden.

Het opslaan van broncode in de database zou hier een definitieve oplossing voor bieden. Het datamodel van de Software Factory zou hier voor aangepast moeten worden zodat in plaats van het pad waar de code zich bevindt, de code uit het bestand zelf wordt opgeslagen in de database als tekst.

Het opslaan van code in de database zorgt er voor dat de code altijd op een centrale plaats beschikbaar is en voorkomt hiermee situaties waarbij een gebruiker wel bevoegd is om op de database te werken maar niet bevoegd is om bij de harde schijf te komen waar de broncode is opgeslagen.

### Abstract screen modeller

De abstract screen modeller biedt op het moment alleen ondersteuning voor standaard scherm componenten, zoals bijvoorbeeld een lijst, formulier of tabblad. Dit is voldoende voor het ondersteunen van klantprojecten maar voor interne ontwikkeling aan de Software Factory zou het erg handig zijn als niet standaard componenten ook ondersteund zouden worden.

Momenteel is het mogelijk om met de Software Factory gebruik te maken van custom schermen die in C# gebouwd zijn. Dit wordt gedaan door een record in een tabel op de database op te slaan waar de aanroep naar het specifieke scherm in de GUI staat. Voor de abstract screen modeller zou het mogelijk zijn om van een soortgelijke tabel waar custom componenten worden geregistreerd gebruik te maken. Dit kan een grote tijdswinst opleveren bij het bouwen van maatwerkschermen.

# Bronvermelding

1. **Wikipedia.** Comparison of file comparison tools. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Comparison\_of\_file\_comparison\_tools.

2. **de Nie, Robert Jan, Klaren, Victor en Buit, Anne.** *Handleiding Thinkwise Software Factory.* Apeldoorn : sn, 2011.

3. **Potter, Michael.** A Generic, Reusable Diff Algorithm in C# - II. *The Code Project.* [Online] 10 Juni 2004. http://www.codeproject.com/KB/recipes/diffengine.aspx.

4. **Grit, Roel.** *Projectmanagement.* sl : Wolters-Noordhoff, 2005. ISBN: 9001347037.

Bijlage 1

Onderzoeksverslag



**Onderzoeksverslag**

**Peter Verwijs**

**1500971**

# Inleiding

Dit is het onderzoeksverslag dat geschreven is om de bevindingen van het onderzoek naar een component voor het vergelijken en mergen van code vast te leggen. In dit document wordt ook de keuze voor een component gemaakt wordt deze keuze onderbouwd.

Er is gezocht met behulp van online zoekmachines. De meningen en aanbevelingen van andere mensen die gepost hebben op diverse forums en websites zijn onderzocht. Hier zijn een aantal file comparisson engines als resultaat uit gekomen. Er is een selectie gemaakt van de meest veelbelovende engines en deze zijn vervolgens getest op hun functionaliteit. Na getest te hebben en de beschikbare features van de verschillende engines vergeleken te hebben is er gekozen voor de Diff\_Match\_Patch engine van Google. De keuze hiervoor is later in het document beargumenteerd.

# Gevonden componenten

Van de gevonden componenten die geschikt leken te zijn is gekeken wat er geboden werd aan functionaliteit en in hoeverre dit geschikt was voor het behalen van het doel. In dit hoofdstuk worden de bevindingen per component besproken.

## Menees Diff.Net

<http://www.menees.com/index.html>

Diff.Net is een file comparison engine met een meegeleverde GUI. De code voor het vergelijken van de bestanden is niet open source maar wordt meegeleverd als library. Documentatie is echter weinig beschikbaar en er is geen API gepubliceerd. Het project is geschreven in C#, wat een voordeel is.

Bij Diff.Net is een GUI inbegrepen waarvan de source wel openbaar is. De GUI is momenteel echter niet volledig werkend. Het tonen van de twee files die met elkaar vergeleken worden gaat fout waardoor er dus alleen een leeg scherm te zien is. Dit is een bug, maar het is onbekend wanneer dit opgelost is. Deze GUI zal dus niet direct ingebouwd kunnen worden in de Software Factory. Het is onbekend hoeveel tijd het zelf repareren van de GUI zal kosten.

Diff.Net heeft alleen een de mogelijkheid om een vergelijking te maken tussen twee files. Het mergen ervan is niet mogelijk.

Diff.Net lijkt te zijn bedoeld als gebruiksklaar eindproduct, niet als project dat uitgebreid of geïntegreerd kan worden. Daarom is Diff.Net niet aan te raden als component.

## Diff\_Match\_Patch

<http://code.google.com/p/google-diff-match-patch/>

Diff\_Match\_Patch is een open source file comparison engine gehost op google code en is geschreven door een software engineer in dienst bij Google. Het project is volledig open source. De source code van de engine is openbaar en bovendien is er een API beschikbaar.

Bij de code is geen GUI inbegrepen. De engine vergelijkt de tekst die ingevoerd wordt, en de output is een object waar de vergelijking in opgeslagen is. Deze objecten kunnen uitgelezen worden en het resultaat kan op het scherm weergegeven worden. De engine is in verschillende programmeertalen beschikbaar, waaronder C#.

Naast het vergelijken van files is het met deze engine ook mogelijk om een file te patchen met de wijzigingen die in een andere file zijn aangebracht.

Het project is al een paar jaar oud maar is vrij recentelijk overgezet naar C#. Regelmatig worden er verbeteringen gemaakt of worden er uitbreidingen gedaan. De engine wordt al op een aantal grote projecten toegepast, zoals bijvoorbeeld Google Docs. Dit project is aan te raden als component voor het prototype.

## DiffUtils

<http://gnuwin32.sourceforge.net/packages/diffutils.htm>  
DiffUtils is een GNU licensed file comparison engine. Het oorspronkelijke project is geschreven voor een UNIX omgeving, maar er is een windows conversie beschikbaar. Deze conversie is echter vrij gedateerd en dus is het niet zeker of deze versie stabiel is.

Het project is open source en de source code is dus beschikbaar. Het project is echter in C++ geschreven wat niet ideaal is. DiffUtils is één van de eerste comparison engines en is in de loop der tijd vaak als engine gebruikt. Documentatie is ook in grote hoeveelheid aanwezig.

Hoewel dit een van de betere file comparison engines is, zal het lastig zijn om deze in de Software Factory te integreren, voornamelijk omdat het niet in de juiste programmeertaal is geschreven. Daarom is deze engine niet aan te raden voor het prototype.

## Generic, Reusable Diff Algorithm in C#

<http://www.codeproject.com/KB/recipes/diffengine.aspx>

Dit is een open source project dat geschreven is als reactie op een eerder gepubliceerd diff algoritme. Het doel van dit project is een makkelijk te gebruiken en implementeren diff algoritme leveren dat snel een vergelijking kan maken tussen zowel kleine als grote bestanden.

Omdat dit project open source is is de source code bekend. Documentatie niet in grote mate aanwezig, maar op de projectpagina van CodeProject staat voldoende uitgelegd hoe de tool gebouwd is en hoe de functionaliteit is werkt.

Bij dit project is een GUI meegeleverd waarvan de source ook openbaar is. Deze GUI komt redelijk overeen met de wensen voor het prototype, maar mist nog wel enkele features (zoals bijvoorbeeld het synchroon scrollen van de twee textvelden) en bevat nog een aantal bugs (als een regel geselecteerd wordt verliest deze zijn highlighting).

Dit project is al vrij gedateerd en wordt niet meer onderhouden. Daarom is het niet zeker of dit de juiste keuze is als component.

# Keuze en beargumentatie

Uiteindelijk is de keuze gevallen op de Diff\_Match\_Patch engine. De reden hiervoor is dat het een open source project is dat redelijk goed gedocumenteerd is. De engine is vrij krachtig en beschikt over de mogelijkheid om zowel vergelijkingen tussen files te maken en om files te patchen. Bovendien wordt het project nog onderhouden en wordt het de engine gebruikt in een aantal grote projecten. De feedback die gebruikers hadden was overwegend positief. Daarom is de uiteindelijke keuze gevallen op Diff\_Match\_Patch van Google.

Voor een visuele implementatie levert Diff\_Match\_Patch echter geen oplossing. Er is hier voor gekozen om deze zelf te schrijven. De GUI’s die meegeleverd worden met DiffDotNet en het Generic, Reusable Diff Algorithm kunnen gebruikt worden als indicatie van wat mogelijk is en hoe dit geïmplementeerd kan worden.

Bijlage 2

Functioneel Ontwerp



Functioneel ontwerp

Peter Verwijs

1500971

# Inleiding

In dit document wordt het functioneel ontwerp behandeld. In het functioneel ontwerp worden de eisen en wensen aan het eindproduct gespecificeerd. Als eerste zal er een lijst met functionele eisen gegeven worden waar het prototype aan moet voldoen. Vervolgens zullen de use cases van het project gedefinieerd worden en zullen de scenario’s die doorlopen moeten worden beschreven worden. In de use cases wordt duidelijk hoe de functionele eisen in het prototype verwerkt zullen worden.

# Functionele eisen

Hier volgt een lijst met de functionele eisen:

* Er moet de mogelijkheid zijn om twee bestanden met code naast elkaar te zien en ze zo met elkaar te vergelijken.
* Door middel van highlighting moeten de verschillen tussen de twee stukken code aangegeven kunnen worden.
* De bestanden moeten uit een lijst gekozen kunnen worden.
* Er zijn twee soorten bestanden:
  + Control procedures
  + Templates
* Templates worden gegroepeerd onder de control procedures. Dit wordt gedaan door een control procedure uit de lijst te selecteren en naar het volgende tabblad te gaan.
* In de lijst worden dezelfde delen code naast elkaar gezet. Er wordt aan de hand van de naam (dus pk) van de CP of template bepaald of het dezelfde is (zelfde naam = zelfde file, andere naam = delete van oude file en nieuwe file toegevoegd).
* Door een file te selecteren en naar het vergelijking tabblad te gaan wordt de vergelijking tussen de twee files getoond.
* Files moeten gemerged kunnen worden. Dit houdt in dat de gebruiker er voor kan kiezen om bepaalde delen van de code uit één file te behouden en een ander stuk code uit een andere file hier aan toe kan voegen.
* Voor gebruikersgemak moet het ook mogelijk zijn om in één keer een hele file te kiezen om te behouden. Bijvoorbeeld als het gewenst is om alle veranderingen in een file uit de branch mee te nemen naar de nieuwe versie moet dit met een druk op de knop kunnen.
* Er moet een manier gevonden worden om de uiteindelijke code op te slaan om deze later te genereren. Generatie van deze code kan pas uitgevoerd worden tijdens het uitvoeren van de merge sessie omdat er voor die tijd nog geen projectversie bestaat waar de code onder hoort te komen.
* Zodra de merge sessie wordt uitgevoerd moet de gemergede code gegenereerd worden. Deze code zal in de nieuwe projectversie geplaatst worden.

# Use cases

Hier worden de use cases beschreven van het project. De use cases zijn schematisch weergegeven. Allereerst wordt per case het verloop van het scenario beschreven. Als er uitzonderingen op het scenario zijn wordt het scenario voor de uitzondering beschreven.

## Use case: Branchen van de code

De gebruiker maakt een branchproject aan. Dit gebeurt door in de lijst met projectversies een taak aan te roepen. Aan de taak moeten onder andere de volgende parameters meegegeven worden:

* Naam van het trunk project
* Projectversie van de trunk project versie
* Naam van het branch project
* De map waar de templates en control procedures worden opgeslagen
* De map waar de database files worden opgeslagen
* Branch type:
  + Branch aanmaken met een lege projectversie
  + Branch aanmaken met een kopie van de trunk projectversie

Als er gekozen is voor een kopie branch worden tijdens het branchen alle control procedures en templates van het trunk project naar het branchproject gekopieerd. De bestanden die op de harde schijf zijn opgeslagen worden gekopieerd naar de map die voor het branchproject is aangemaakt. De locatie van de hoofdmap wordt bij het starten van het branchen als parameter aan de taak meegegeven. De mappenstructuur zal verder volgens de standaard implementatie van de Software Factory zijn. Deze is als volgt:

* Voor de control procedures:
  + \[basismap]\Control\_files\
* Voor de template files:
  + \[basismap]\Template\_files\

Als het branchen succesvol is verlopen kan er nu met een nieuwe kopie van de code gewerkt worden en staan alle verwijzingen binnen de Software Factory naar de nieuwe bestanden.

## Use case: Merge sessie starten

De gebruiker kiest er voor om een merge sessie te starten door een taak aan te roepen op het merge sessie scherm. De merge sessie wordt nu gecreëerd. Vervolgens wordt de lijst met control procedures van het trunk en het branch project gevuld. Deze lijst staat op een eigen tabblad binnen het merge sessie scherm. De control procedures van de twee projecten worden aan elkaar gekoppeld. Dit gebeurt door een primary key vergelijking te doen. De lijst met control procedures krijgt een standaardsortering waardoor control procedures met toekenningtype SQL bovenaan staan zodat de items die code bevatten als eerst getoond worden. Vervolgens worden de templates toegevoegd aan de template lijst op het volgende tabblad. Op dit punt wordt de use case “Control procedures vergelijken” gestart.

### Use case: Control procedures vergelijken

Het vergelijken van de control procedures gebeurt automatisch nadat de merge sessie gestart is en de control procedures en templates zijn toegevoegd. Als het een control procedure met toekenningtype SQL betreft wordt de code van de control procedures vergeleken met de comparison engine. Control procedures van het type static bevatten geen code en hoeven daarom ook niet vergeleken te worden. Als uit de vergelijking komt dat er een verschil zit tussen de code van de control procedures dan wordt dit aangegeven door de regel in de lijst te highlighten. Zodra alle control procedures met elkaar vergeleken zijn wordt de use case “Template files vergelijken” gestart.

### Use case: Template files vergelijken

Het vergelijken van de template files gebeurt automatisch nadat de merge sessie gestart is. Zodra de lijst met template files gevuld is en de templates van de branch en trunk aan elkaar gekoppeld zijn worden ze vergeleken met de comparison engine. De comparison engine laat weten of er een verschil tussen de twee files is. Als er een verschil gevonden is wordt dit aangegeven door het veld te highlighten.

## Use case: Files mergen

Het mergen van files gebeurt door uit de lijst met templates of control procedures een record te selecteren en naar het file merge tabblad te gaan. Op dit scherm staan de twee files naast elkaar weergegeven. De verschillen worden met behulp van highlighting aangetoond. Dit gebeurt vanuit het standpunt dat de file in de trunk de huidige versie is en de versie uit de branch nieuw is. Blokken code die nieuw zijn worden groen gemarkeerd, blokken die in de nieuwe versie verwijdert zijn worden rood gemarkeerd en blokken die gewijzigd zijn worden geel gemarkeerd. De file aan de linkerkant kan bewerkt worden door op een gemarkeerd blok tekst te klikken dat een verschil bevat. Er kan dan gekozen worden of het tekstblok uit de trunk of uit de branch gekozen moet worden. Als de gebruiker tevreden is met de aanpassingen kan de file opgeslagen worden door op de knop “Opslaan”te drukken. Het bestand wordt dan in de database opgeslagen. Als het scherm voor deze file opnieuw geopend wordt dan wordt de opgeslagen versie van de code geladen.

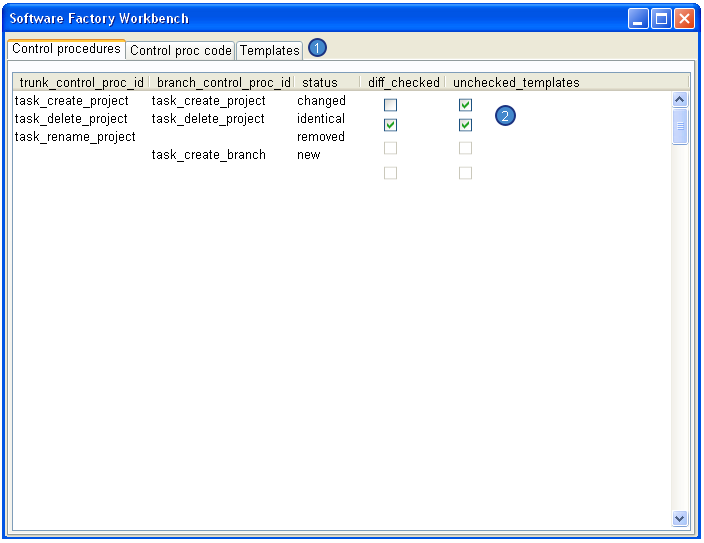
## Use case: Merge sessie uitvoeren

Als alle templates en control procedures zijn gecontroleerd en de gebruiker besloten heeft dat de code goed gemerged is kan de merge sessie uitgevoerd worden. Door op het merge sessies scherm de taak “Merge sessie uitvoeren” aan te roepen kan de merge sessie uitgevoerd worden. Deze taak zorgt er voor dat de nieuwe projectversie aangemaakt wordt. Nadat de projectversie is aangemaakt worden alle control procedures en templates gekopieerd naar de nieuwe projectversie.

# Schermontwerpen

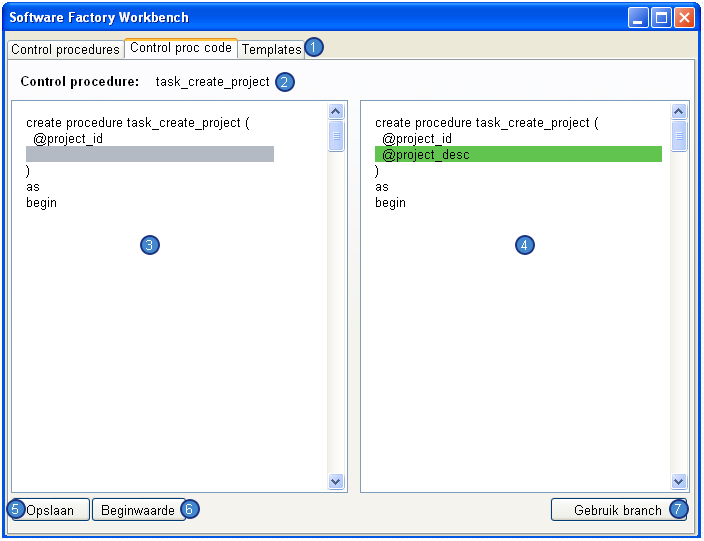
Wat volgt zijn de schermontwerpen die gemaakt zijn als concept voor de schermen die de functionaliteit in het prototype zullen bevatten. De verschillende onderdelen van de schermen zijn geannoteerd met commentaar.

## Merge control procedures



1. De tabbladselectie voor het wisselen tussen de schermen.
2. De lijst met control procedures. De lijst heeft een aantal attributen:
   * **Trunk control procedure id**: De naam van de control procedure zoals deze in de trunk gedefinieerd staat.
   * **Branch control procedure id**: De naam van de control procedure zoals deze in de branch gedefinieerd staat.
   * **Status**: De status van de control procedure. Deze status beschrijft welke wijziging er tussen de trunk en de branch heeft plaatsgevonden (indien dit het geval is). De mogelijke statussen zijn:
     + Nieuw
     + Verwijdert
     + Gewijziged
     + Identiek
   * **Diff checked:** Als de control procedure van het type “SQL” is en code bevat wordt hiermee de status van de code vergelijking aangegeven. Als de code gecontroleerd en opgeslagen is wordt dit veld aangevinkt. De werking wordt verder uitgelegd bij het scherm “Merge control procedures code”.
   * **Unchecked templates:** Met dit veld wordt aangegeven of onderliggende templates nog openstaande code vergelijkingen hebben. Als de vergelijkingen van alle onderliggende templates gecontroleerd zijn wordt dit veld leeg gemaakt. De werking wordt verder beschreven bij het scherm “Merge templates”.

## Merge control procedures code



1. De tabbladselectie voor het wisselen tussen de schermen.
2. De naam van de control procedure. Deze wordt getoond zodat de gebruiker eenvoudig kan zien aan welke control procedure gewerkt wordt.
3. De trunk control procedure file. Dit is de code zoals deze op het moment in de trunk bestaat. De verschillen ten opzichte van de branch file worden gemarkeerd. Door te rechtsklikken op een gemarkeerde regel kan gekozen worden om de wijzigingen in de branch over te nemen in deze tekst editor. Op het moment dat er wijzigingen worden aangebracht in deze code zal het niet langer als de trunk code beschouwd worden maar als code de code die in de nieuwe projectversie gebruikt zal worden.

Als de gebruiker eerder wijzigingen heeft gedaan aan de control procedure en deze heeft opgeslagen dan zal het opgeslagen bestand geladen worden in plaats van de versie uit de trunk.

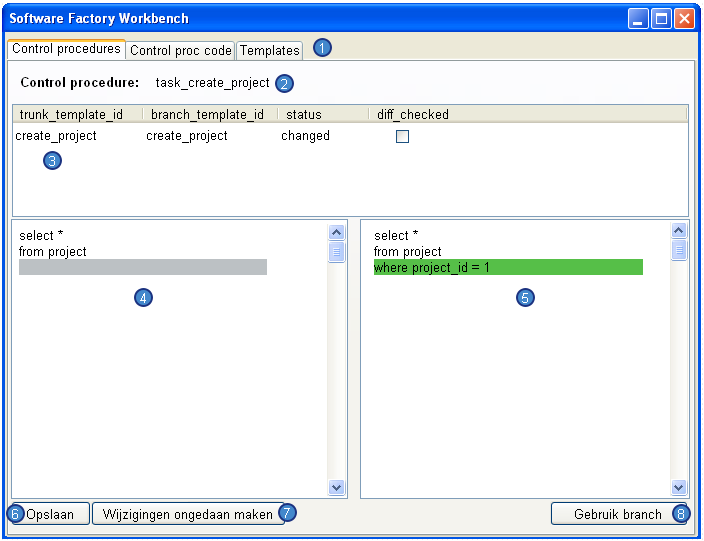
Het is belangrijk om te weten dat de code van de trunk projectversie niet direct aangepast wordt. Alle wijzigingen die worden gedaan zullen opgeslagen worden in de database en bij zal bij het uitvoeren van de merge een nieuw bestand worden. De trunk code kan ten alle tijden nog terug gehaald worden.

1. De branch control procedure file. Dit is de code zoals deze op het moment in de branch bestaat. Code die gewijzigd is ten opzichte van de trunk is gemarkeerd. Door op de gemarkeerde tekst te rechtsklikken kan gekozen worden om de wijziging in de nieuwe versie van de code mee te nemen. De gemarkeerde tekst zal dan in de linker tekst editor verschijnen.
2. Knop opslaan. Door op deze knop de klikken kan de gebruiker de code die in de linker tekst editor staat opgeslagen worden. Ook hier is het belangrijk om te weten dat het opslaan van de code er niet voor zorgt dat de code in de trunk vervangen wordt. De code wordt opgeslagen in de database en bij het uitvoeren van de merge sessie zal deze gebruikt worden als de code die in de nieuwe projectversie gebruikt wordt.

Door op deze knop te drukken geeft de gebruiker tevens aan dat het verschil gecontroleerd is en wordt het veld “Diff checked” in het scherm “Merge control procedures” aangevinkt.

1. Knop herstel. Als de gebruiker op deze knop drukt zal de trunk versie van de code weer in de editor geladen worden. Er wordt echter wel eerst een waarschuwing gegeven dat alle wijzigingen die gemaakt zijn verloren zullen gaan. Als de gebruiker bevestigd zal de code uit de trunk weer getoond worden.
2. Gebruik branch knop. Door op deze knop te drukken zal de hele branch file gekopieerd worden naar de linker editor. Deze optie is er voor gebruikers die zeker zijn van de veranderingen in de branch en snel de hele versie van de code willen gebruiken en niet elk verschil apart willen overzetten.

## Merge template



1. De tabbladselectie voor het wisselen tussen de schermen.
2. De naam van de control procedure. Deze wordt getoond zodat de gebruiker eenvoudig kan zien bij welke control procedure de templates horen.
3. De lijst met templates voor de gekozen control procedure. Uit deze lijst kan een gebruiker de template die vergeleken moet worden kiezen. De velden in deze lijst zijn:
   * **Trunk template id**: De naam van de template zoals deze in de trunk gedefinieerd staat.
   * **Branch template id**: De naam van de template zoals deze in de branch gedefinieerd staat.
   * **Status**: De status van de template. Deze status beschrijft welke wijziging er tussen de trunk en de branch heeft plaatsgevonden (indien dit het geval is). De mogelijke statussen zijn:
     + Nieuw
     + Verwijdert
     + Gewijzigd
     + Identiek
   * **Diff checked**: Dit veld geeft aan of de vergelijking tussen de templates door de gebruiker gecontroleerd en opgeslagen zijn. Als dit het geval is wordt dit veld aangevinkt.
4. De trunk template file. Dit is de code zoals deze op het moment in de trunk bestaat. De verschillen ten opzichte van de branch file worden gemarkeerd. Door te rechtsklikken op een gemarkeerde regel kan gekozen worden om de wijzigingen in de branch over te nemen in deze tekst editor. Op het moment dat er wijzigingen worden aangebracht in deze code zal het niet langer als de trunk code beschouwd worden maar als code de code die in de nieuwe projectversie gebruikt zal worden.
5. De branch template file. Dit is de code zoals deze op het moment in de branch bestaat. Code die gewijzigd is ten opzichte van de trunk is gemarkeerd. Door op de gemarkeerde tekst te rechtsklikken kan gekozen worden om de wijziging in de nieuwe versie van de code mee te nemen. De gemarkeerde tekst zal dan in de linker tekst editor verschijnen.
6. Knop opslaan. Door op deze knop de klikken kan de gebruiker de code die in de linker tekst editor staat opgeslagen worden.

Door op deze knop te drukken geeft de gebruiker tevens aan dat het verschil gecontroleerd is en wordt het veld “Diff checked” in het scherm “Merge control procedures” aangevinkt.

1. Knop herstel. Als de gebruiker op deze knop drukt zal de trunk versie van de code weer in de editor geladen worden. Er wordt echter wel eerst een waarschuwing gegeven dat alle wijzigingen die gemaakt zijn verloren zullen gaan. Als de gebruiker bevestigd zal de code uit de trunk weer getoond worden.
2. Gebruik branch knop. Door op deze knop te drukken zal de hele branch file gekopieerd worden naar de linker editor. Deze optie is er voor gebruikers die zeker zijn van de veranderingen in de branch en snel de hele versie van de code willen gebruiken en niet elk verschil apart willen overzetten.