Automatisering Reparatietraject

Eindverslag Afstudeertraject

Student

Naam: ***Sander Huisman***
Nummer: ***20006753***
Periode: ***17 mei – 8 oktober 2004***

Referaat

Dit document bevat een verslag van het project dat de hieronder genoemde student heeft uitgevoerd als afstudeeropdracht bij de onderneming Control Techniques B.V. Het doel van het bovengenoemde project was het automatiseren van het reparatietraject van aandrijfcomponenten, één van de belangrijkste processen binnen Control Techniques. Het verslag beschrijft voornamelijk het product en het proces waarmee dit product tot stand is gekomen. Hiernaast zijn tussen- en eindproducten als bijlagen opgenomen.

Afstudeerder

Dhr. S. Huisman
Studentnr: 20006753

Contactdocent

Dhr. T.H.M. Spaan,
Afdeling Informatica, Haagse Hogeschool

Bedrijf

Control Techniques B.V.
Afdeling Automatisering
Kubus 155
3364 DG Sliedrecht
+31(0) 184 420 555
http://www.controltechniques.nl

Opdrachtgever

Dhr. W. Bosman

Bedrijfsmentor

Dhr. F. Kleijwegt

Examinatoren

Dhr. T.H.M. Spaan,
Afdeling Informatica, Haagse Hogeschool

Mevr. G.S.D. Sewdajal,
Afdeling Informatica, Haagse Hogeschool

Trefwoorden

- Microsoft SQL Server
- ASP.Net
- Web-based applicatie
- IAD

- UML

Voorwoord

Graag wil ik van deze gelegenheid gebruik maken om iedereen van Control Techniques te bedanken voor de goede medewerking.

Hiernaast wil ik Walter Bosman bedanken dat hij mij de kans heeft gegeven deze uitdagende opdracht uit te voeren, en voor zijn steun en visie bij het uitvoeren hiervan.
Fred Kleijwegt voor zijn kennis van de systemen en gang van zaken binnen Control Techniques en zijn kennis van het reparatietraject.
Mickeal Verschoor voor het voorwerk wat hij gedaan heeft, en het concept dat door hem is opgezet.
Edwin Frèrejean als één van de belangrijkste gebruikers van het huidige systeem voor zijn ervaring en inzicht.

Zonder deze personen was het onmogelijk geweest een applicatie op te zetten.

Naast de mensen van Control Techniques wil ik de begeleidend docenten van de Haagse Hogeschool bedanken, de heer Spaan en mevrouw Sewdajal.

Inhoudsopgave

Bijlagen

Inleiding

Het document dat nu voor u ligt is het eindverslag van het project "Werkplaatsapplicatie Control Techniques". Het verslag is geschreven om de examinatoren en gecommitteerde, alsmede de projectbegeleiders binnen Control Techniques een inzicht te verschaffen in de uitgevoerde werkzaamheden. Het verslag is geschreven in het kader van de module afstuderen, ter afsluiting van de opleiding Informatica & Informatiekunde.

De hoofdstukken die u op het punt staat te lezen geven een uitgebreide beschrijving van de uitgevoerde werkzaamheden, de producten die zijn opgeleverd, de procesgang waarmee deze producten zijn vervaardigd en de beslissingen die zijn genomen bij het maken van deze producten.

Bij de opbouw van het document is getracht de fasen van IAD terug te laten komen. Het eerste deel van het verslag zal achtergrond informatie geven over zowel het bedrijf als de opdracht. Hoofdstuk één geeft een indruk van het bedrijf, hoofdstuk twee zal een beschrijving zijn van de opdracht.

Het tweede deel van het verslag geeft een uiteenzetting van de werkzaamheden die zijn verricht in het kader van de afstudeeropdracht. Hoofdstuk drie beschrijft de definitiefase. Hierin worden de systeemeisen vastgesteld, wordt het systeem concept bepaald en worden de pilots afgebakend. Hoofdstuk vier tot en met zeven zetten per pilot het ontwerp, en de bouw uiteen. Hoofdstuk acht beschrijft de invoering van het systeem, voor zover dit gebeurd is binnen het tijdsbestek van de opdracht.

Het derde en laatste deel van dit verslag geeft de lezer een product- en procesevaluatie. Hier staan de mening en de bevindingen van de afstudeerder zelf uiteengezet.

Let op dat getracht is de structuur van de ontwikkelmethode IAD terug te laten komen in het verslag, hierdoor is het mogelijk dat niet altijd de chronologische volgorde van gebeurtenissen wordt gehandhaafd.

Waar in het document de uitdrukking "CT" wordt gebruikt, wordt gerefereerd aan Control Techniques.

Waar in het document de uitdrukking "EWP" wordt gebruikt, wordt gerefereerd aan de elektronische werkplaats van Control Techniques.

DEEL I - Bedrijf en Opdracht

1 Het bedrijf

Dit hoofdstuk zal een beschrijving bevatten van het bedrijf Control Techniques B.V. Er zal worden uitgelegd hoe het bedrijf is ontstaan, wat de producten en diensten zijn, hoe de organisatie in elkaar zit en wat de plaats van de afstudeerder binnen het bedrijf is.

1.1 Algemeen

Control Techniques B.V. is een dochteronderneming van Control Techniques Drives Ltd, producent van elektrisch regelbare aandrijfcomponenten zoals frequentieregelaars, fluxregelaars, softstarters, gelijkstroomregelaars en zowel AC als DC servoaandrijvingen, met diverse besturingskaarten voor gecentraliseerde of gedistribueerde aansturing.

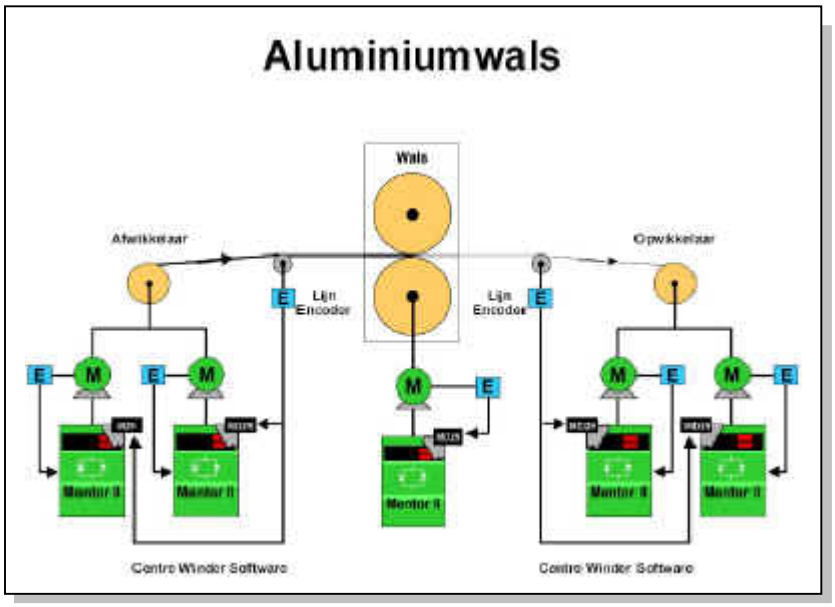
Control Techniques Drives Ltd. werd in 1973 opgericht onder de naam KTK en in 1985 kreeg het bedrijf een notering aan de Londense beurs. In januari 1995 werden de Control Techniques aandelen verkocht aan Emerson Electric Co., waarmee de Emerson groep wereldmarktleider werd in de aandrijftechniek.

Control Techniques B.V. is in 1989 opgericht. Waar men begon met een handje vol medewerkers, is dit nu uitgegroeid tot een professioneel bedrijf met ongeveer een 50 tal medewerkers.

Naast Nederland en Engeland heeft Control Techniques Ltd. vestigingen in 48 landen wereldwijd. Dit maakt Control Techniques en Emerson groep de grootste leverancier en serviceverlener van aandrijftechniek in de wereld.

1.2 Producten en Diensten

Control Techniques is een specialist in aandrijftechniek, maar wat is aandrijftechniek precies? Aandrijftechniek wordt gebruikt voor het aansturen van bijvoorbeeld hijskranen, bruggen en sluizen, liften, drukmachines en nog vele andere toepassingen. Op de volgende pagina is een voorbeeld te zien van een opstelling van een aluminiumwals. Hier is te zien hoe regelaars uit de Mentor serie een aantal elektromotoren aansturen, die op hun beurt de rollers van de wals aandrijven.



Globaal gezien heeft CT drie vormen van diensten/producten waaruit inkomsten worden verworven:

- Paneelbouw
- Verkoop losse componenten
- Component reparatie

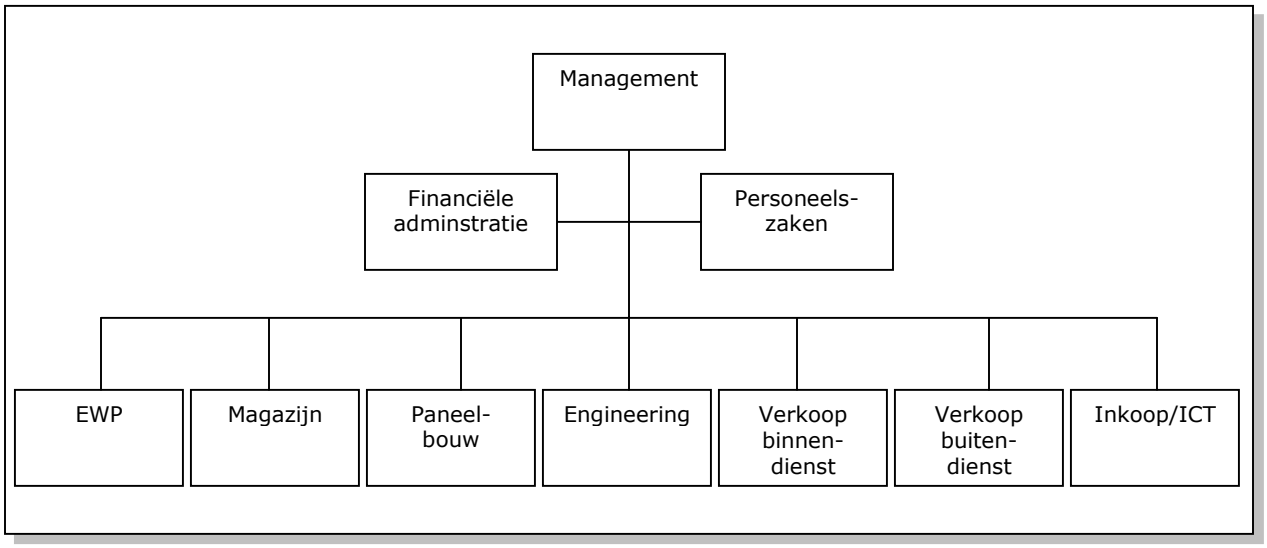
Paneelbouw
Bij de afdeling paneelbouw worden naar opdracht van een klant "panelen" gebouwd. Deze panelen worden voorzien van alle mogelijke onderdelen zoals regelaars, schakelaars, stroomgeleiders en andere elektrotechnische apparaten die voor de specifieke applicatie van de klant van toepassing zijn. Deze panelen worden in kasten gehangen die bij de klant worden geplaatst. Op deze manier wordt de klant voorzien van een volledige, op maat gemaakte aandrijf oplossing.

Verkoop losse componenten
De verkoop van losse componenten vormt samen met het bouwen van complete panelen de hoofdmoot van de inkomsten van CT. Veel verkopen vinden plaats als projecten. De klant koopt dan een aantal regelaars tegelijk, conform zijn behoeften. Naast de projectverkopen is ook verkoop van enkele eenheden mogelijk.

Component reparatie
Het repareren van regelaars is een service die CT klanten vanuit heel Europa biedt. De EWP heeft een aantal geavanceerde diagnostische werktuigen waarmee de status en een eventuele storing van een regelaar kunnen worden bepaald. Hierdoor worden de meeste reparaties in huis uitgevoerd. Reparaties die echter niet door CT zelf kunnen worden gerepareerd of die onder garantie vallen worden naar het moeder bedrijf in Engeland verscheept. Het reparatieproces vormt het uitgangspunt van deze opdracht. In dit document zal het proces onder de loep worden genomen en zullen steeds meer details aan het licht komen.

1.3 Organisatie en Cultuur

De organisatiestructuur van CT is vrij plat, en breed. Er zijn veel maar relatief korte communicatielijnen. De sfeer binnen de organisatie van CT kan gezien worden als informeel. De werknemers zijn allemaal professionals op hun vakgebied. Soms zijn de medewerkers op de afdelingen verkoop, inkoop en engineering doorgestroomd van bijvoorbeeld de paneelbouw en de elektronische werkplaats. Hierdoor is er in het hele bedrijf, op iedere afdeling mensen aanwezig met de kennis en know-how van alle producten die CT te bieden heeft. Een voordeel voor een informatieanalist is dat, door de korte communicatie lijnen en de grote hoeveelheid kennis door het gehele bedrijf, informatie vrij makkelijk is te vergaren. Echter is deze informatie niet altijd gestructureerd, maar het is aan de informatieanalist om hier structuur in aan te brengen. In het onderstaande figuur is een organigram van de onderneming opgenomen.



1.4 Plaats Afstudeerder

De afstudeerder is, zoals te verwachten op de afdeling inkoop/ICT geplaatst. De ICT afdeling is bij dit bedrijf bij de inkoop geplaatst omdat er eigenlijk geen ICT afdeling is. Het bedrijf heeft in het verleden minimaal in ICT geïnvesteerd. Hierdoor is de noodzaak van een aparte ICT afdeling gering. De huidige systemen, zowel hard- als software, worden door een persoon van de inkoop afdeling onderhouden. Daarnaast komt er één keer per week een persoon van een gespecialiseerd ICT ondersteuningsbedrijf.

Het is dan ook een nadeel voor de afstudeerder dat hij eigenlijk alleen staat en op zichzelf is aangewezen. Aan de andere kant heeft de afstudeerder alle vrijheid. Dit betekend ook dat van de afstudeerder een grote mate van zelfstandigheid wordt verwacht. Nu kan er voor deze afstudeerder worden gezegd dat zelfstandigheid geen probleem is, en dat hij goed met de hem gegeven vrijheden om dan gaan.

1.5 ICT voorzieningen

ICT is bij CT eigenlijk een ondergeschoven kindje gebleven. Tijdens mijn periode bij het bedrijf kreeg ik de indruk dat er door de jaren heen minimaal in ICT is geïnvesteerd. Zo werken veel gebruikers nog met Windows 98 op verouderde systemen en draait het artikelen beheerssysteem nog onder DOS. Niet dat dit een probleem is, alles werkt prima. Langzaamaan wordt echter Windows 98 vervangen door Windows 2000 en wordt er gezocht naar een nieuw artikelen beheerssysteem, een nieuw costomer relations management (CRM) systeem en een nieuw systeem voor het administreren van het component reparatie proces. De huidige software pakketten waar rekening mee gehouden moet worden zijn de volgende:

- Scope, een CRM systeem wat werkt met een combinatie van MS Access en Oracle in een Windows omgeving.
- Multivers, een artikelen/klanten beheerssysteem wat werkt met een (mij) onbekend soort database in een DOS omgeving.
- Reparatieprogramma, een programma voor de registratie van het reparatietraject wat werkt met MS Access in een Windows omgeving.
- MS Exchange, een mail systeem wat werkt in een Windows omgeving.

De huidige infrastructuur is opgezet rond twee centrale servers. De ene server verzorgd emailverkeer, internetverkeer, antivirus software en domein controle. De ander file-sharing, printer-sharing, applicatieserver en domein controle. De gebruikers zijn op deze servers aangesloten door middel van een 100Mbit netwerk met hier en daar een hub. Het nieuwe reparatie programma zal bij één van deze servers worden bijgeplaatst. Welke server dit is zal nog moeten blijken, waarschijnlijk zal dit de server worden welke verantwoordelijk is voor de file-sharing. Deze heeft namelijk de minste belasting.

2 De opdracht

Het nu volgende hoofdstuk zal de opdracht, waar de afstudeerder de afgelopen maanden aan gewerkt heeft, beschrijven. Het hoofdstuk bevat de opdrachtomschrijving zoals die ook ter goedkeuring aan de stage/afstudeer-coördinator is voorgelegd. Hiernaast is het plan van aanpak opgenomen.

2.1 Opdrachtomschrijving

Inleiding (organisatorische omgeving, kader, historie):

Opdrachtgever is de onderneming Control Techniques B.V. Control Techniques is een dochteronderneming van Control Techniques Drives Ltd, producent van elektrisch regelbare aandrijfcomponenten, zoals frequentieregelaars, fluxregelaars, softstarters, gelijkstroomregelaars en zowel AC als DC servoaandrijvingen, met diverse besturingskaarten voor gecentraliseerde of gedistribueerde aansturing. Control Techniques verkoopt en repareert aandrijfcomponenten voor de Europese markt. De aandrijfcomponenten worden bij het moederbedrijf in Engeland ingekocht.

Probleemstelling:

In de werkplaats waar kapotte aandrijfcomponenten worden hersteld wordt al jaren gebruik gemaakt van een applicatie die destijds door de moederonderneming in Engeland is ontworpen. Echter is Control Techniques de afgelopen jaren zo sterk gegroeid dat de applicatie het aantal gebruikers niet meer aankan. Als gevolg hiervan is men min of meer gedwongen om weer met papieren versies van de reparatiedossiers te gaan werken.

Er zijn teveel gebruikers voor het huidige systeem. Door het toegenomen aantal werknemers is het werken met de huidige applicatie niet efficiënt. De papieren dossiers die noodgedwongen worden gebruikt liggen her en der door de werkplaats verspreid wat veel irritatie oproept wanneer een klant belt over de status van de reparatie van zijn aandrijfcomponent. Een ander gevolg kan zijn dat reparaties uitlopen of worden uitgesteld.

Doelstelling van de opdracht:

Het doel van de opdracht is het ontwikkelen van een applicatie welke minimaal voldoet aan de volgende eisen:

A) De applicatie moet meerdere werkplaatsmedewerkers efficiënt in hun werk kunnen ondersteunen.

B) Met behulp van de applicatie moet informatie snel aan klanten kunnen worden verstrekt.

C) De applicatie moet een inzicht kunnen geven in de status van de lopende en aanstaande reparaties.
D) De applicatie moet in principe de functionaliteiten van de huidige applicatie bevatten.

Uitgangssituatie:

A) Benodigde software
- Standaard Windows besturingssysteem met office pakket.
- ASP.NET framework versie 1.1 of hoger.
- Internet Information Services versie 5.0 of hoger.
- MS SQL Server 2000 of hoger.
- Web Matrix of standaard tekstverwerkingsprogramma.
- MS Access.

B) Benodigde hardware
- Standaard pc voor ontwikkeling en testen.
- Server voor het draaien van MS SQL, en de uiteindelijke applicatie.

C) Beschikbare rapporten
- Huidige opzet werkplaatssysteem.
- Documentatie en gebruikershandleiding van de huidige applicatie.

D) Aanwezige ideeën
- De huidige applicatie geheel herschrijven.
- Gebruik maken van een MS SQLServer database.
- Gebruik maken van COM+ objecten.

Concrete werkzaamheden:

A) Uit te voeren activiteiten
- Opstellen plan van aanpak.
- Opstellen definitiestudie.
  - Definiëren van ontwikkelscenario.
  - Analyseren van huidige applicatie.
  - Analyseren van huidige systeem opzet.
  - Definiëren van systeemeisen.
  - Bepalen systeem concept.
  - Beschouwen technische structuur.
  - Beschouwen organisatorische inrichting.
  - Opstellen pilotplan.
- Ontwikkelen pilots.
  - Specificeren globaal-functionele structuur pilot.
  - Specificeren globaal-technische structuur pilot.
  - Specificeren globaal-organisatorische inrichting pilot.
  - Opstellen pilot ontwikkelplan.
  - Ontwerpen software bouweenheden.
  - Bouwen software bouweenheden.
  - Bouwen conversietools pilot.
  - Ontwerpen handmatige procedures pilot.
  - Opstellen opleidingsmateriaal pilot.
  - Opstellen handleidingen pilot.
  - Integreren bouweenheden.
  - Beoordelen en testen pilots.

- Invoeren pilot(s)/systeem.
  - Opstellen plan van aanpak.

- Invoeren pilot(s)/systeem.
- Verzorgen opleidingen.
- Uitvoeren pilot acceptatie.
- Overdragen pilot.
- Feedback verzamelen.
- Ondersteunen pilots/systeem.

In de definitiefase zal worden bepaald hoe de bovenstaande activiteiten iteratief worden uitgevoerd.

B) Te hanteren methoden

IAD zal de gebuikte methode worden. Aan de hand van het plan van aanpak zal worden bepaald of alle aspecten van IAD aan de orde gaan komen, of dat er wegens irrelevantie onderdelen van IAD worden geschrapt.

C) Te gebruiken technieken

UML zal worden gebruikt voor het modelleren van de applicatie. Ook hier geldt weer dat aan de hand van het plan van aanpak wordt bekeken welke aspecten van UML worden uitgewerkt. Verder zullen de ERD en DD technieken worden toegepast voor het ontwerpen van de database.

Resultaten voor de opdrachtgever (op te leveren producten):

- Plan van aanpak.
- Definitiestudie.
- Pilot ontwerp(en).
- Pilot(s) voor de werkplaatsmedewerkers van Control Techniques.
- Testrapport(en).
- Gebruikers handleiding.

2.2 Plan van aanpak

Projectorganisatie

Het project zal worden uitgevoerd door de volgende personen:
- Sander Huisman Systeem ontwikkelaar

Vanuit Control Techniques zullen de volgende personen het project begeleiden:
- F. Kleijwegt Systeembeheer
- W. Bosman Inkoop manager

Methoden en technieken

- Alle opgeleverde documenten en rapporten zullen voldoen aan de eisen zoals deze gesteld door de module AV-03 van de opleiding IVIT;
- Alle documentatie en stukken zullen worden gemaakt in MS-Word;
- Het systeem zal worden ontworpen aan de hand van de IAD methode;
- De software zal worden ontworpen met behulp van de technieken die UML (Unified Modeling Language) te bieden heeft;

Er is voor de ontwikkelmethode IAD gekozen om een drietal redenen. Eén, de systeemeisen zijn bij aanvang van het project nog onduidelijk, en de mogelijkheid dat de eisen zullen wijzigen is aanwezig. Twee, de specificaties zullen waarschijnlijk alleen te toetsen zijn aan de hand van werkende prototypen. Drie, de organisatie hecht veel waarde aan de inbreng van de eindgebruikers in het ontwikkelproces. Daarnaast is de beschikbare tijd voor het project, met kijk op het afstuderen, gering en is de project relatief klein.

Voor de modelleringstechniek UML is gekozen omdat ten eerste de ontwikkelomgeving object georiënteerd is. Ten tweede omdat dit soort informatiesysteem zich goed leent voor een object georiënteerde aanpak. Als laatste omdat het voor mijzelf als afstudeerder een goede ervaring is op het gebied van de object oriëntatie.

Benodigde hulpmiddelen

Er zijn een aantal hulpmiddelen nodig om het project tot een goed einde te brengen. Hier volgt een opsomming:

- PC met:
    - Windows 98 of hoger
    - MS Access en MS SQL Server 2000
    - Internet Information Services 5.0 of hoger
    - .NET Framework 1.0 hoger
    - Ontwikkelsoftware zoals DreamWeaver, VisualStudio.NET of WebMatrix
    - Internet Browser
    - Tekstverwerkingsprogramma
    - Teken programma
    - Documentatie betreffende de database en programmeeromgeving.

- Server met:
  - Windows server 2000 of hoger
  - MS SQL Server 2000
  - Internet Information Services 5.0 of hoger
  - .NET Framework 1.0 hoger

Risicofactoren

Risico	Factor	Maatregel
Ziekte, onvoorziene afwezigheid van een lid van de projectgroep.	Middel	Ruim plannen.
Onvoldoende beschikbaarheid van eind gebruikers voor het vergaren van input.	Laag	Dit is niet te voorkomen, nog te verhelpen. Ook hier kan ruim worden gepland.

Kosten & baten

Vergoeding stagiair (€ 160,00 p/m)	=	€ 800,00
Aanschaf SQL server 2000 standaard editie	=	€ 1500,00
Kantoor benodigdheden	=	€ 100,00

		Totaal € 2400,00

De kosten zijn bij benadering

De baten van het uiteindelijke software product zijn niet direct in geld uit te drukken. Hier moet meer gedacht worden aan winsten in efficiëntie van werknemers en efficiëntie in informatievoorziening richting klant.

Mijlpaalproducten

- Plan van aanpak
- Definitiestudie
- Pilot ontwikkelrapport "Pilot 1: K-Formulieren"
- Pilot 1: K-Formulieren
- Pilot ontwikkelrapport "Pilot 1: Communicatie"
- Pilot 2: Communicatie
- Pilot ontwikkelrapport "Pilot 1: Planning"
- Pilot 3: Planning
- Pilot ontwikkelrapport "Pilot 1: Beheer"
- Pilot 4: Beheer
- Systeem handleiding

Planning

De planning kan worden teruggevonden in bijlage I.

DEEL II - Definitiefase en Pilotontwikkeling

3 Definitiefase

In het hoofdstuk wat nu voor u ligt, zal worden uitgelegd hoe het traject van reparatie van een component in elkaar zit. Hieraan gekoppeld wordt het huidige programma onder de loep genomen. Ook wordt de opzet voor een nieuw systeem die gemaakt is door een ex-medewerker beschouwd. Aansluitend zullen de systeemeisen aan bod komen en hierop voortbouwend een systeem concept.

3.1 Analyse huidig reparatiesysteem en -proces

Het is belangrijk om te weten hoe nu het reparatieproces in zijn werk gaat. Om dit te begrijpen volgt hier een gedetailleerde beschrijving.

Het hele proces begint bij het binnenkomen van een (kapot)aandrijfcomponent in het magazijn. Deze worden aangeleverd per transportdienst, koerier of door de klant zelf. Een magazijnmedewerker neemt de componenten in ontvangst en schrijft deze in het systeem in. Hij maakt een nieuw K-formulier aan met een K-nummer en vult de initiële gegevens in. Als alle componenten voor die dag zijn ingeschreven worden er in een batch bewerking labels afgedrukt die aan de componenten worden gehangen. Hierna worden de componenten in een stellage van de EWP geplaatst. De aangemaakte K-formulieren worden uitgedraaid en samen met een kopie van eventuele bijlagen die bij een component zijn meegeleverd, in een map van de EWP gestopt. De medewerkers van de EWP bladeren door de map en pakken er naar eigen goeddunken K_formulieren uit. Hierna zoeken zij het bijbehorende component in de stellingen en gaan hiermee aan de slag. Tijdens het testen van het component wordt een testformulier opgesteld. Aan de hand hiervan wordt bepaald welke onderdelen er nodig zijn. De benodigde onderdelen worden op het K-formulier bijgeschreven, samen met een schatting van de benodigde tijd. Hierna vervolgt het K-formulier zijn weg naar het management waar een prijs wordt bepaald en een goedkeuring voor reparatie wordt gegeven, of dat een component naar het moederbedrijf moet worden verscheept, of dat het component moet worden verschroot, of dat er terugkoppeling met de klant moet plaatsvinden. Wordt er direct goedkeuring gegeven, dan wordt het K-formulier terug naar de werkplaats gestuurd waar het component wordt gerepareerd. Moet het component naar Engeland, dan gaat het K-formulier naar de inkoop afdeling waar een medewerker het transport voorbereid. Hierna gaat een transport order naar het magazijn. Moet een component worden verschroot of is er terug koppeling van de klant gewenst, dan gaat het K-formulier naar de afdeling verkoop waar medewerkers contact zoeken met de klant. Is er goedkeuring van de klant voor reparatie, dan gaat het K-formulier weer richting de EWP. Bij verschroting gaat het richting inkoop waar transport wordt voorbereidt. Nadat een component is gerepareerd wordt het weer in een stelling geplaatst. Het bijbehorende K-formulier gaat richting inkoop waar transport naar de klant wordt voorbereid en wordt gefactureerd. Alle transport orders komen uiteindelijk weer in het magazijn terecht waar de componenten op transport worden gezet. Het papieren K-formulier ligt nu nog steeds bij de inkoop afdeling waar een medewerker de gegevens van het papier weer in het reparatiesysteem invoert. Ten slotte vervolgd het papieren K-formulier zijn weg naar het archief.

Wat zijn nu de problemen? Een probleem is dat men niet kan achterhalen waar een K-formulier zich op een willekeurig tijdstip in het traject of de organisatie bevindt. Hierdoor kan het zijn dat K-formulieren zoek raken, lang blijven liggen, of een klant niet goed kan worden geïnformeerd naar de vordering van zijn reparatie. Een ander probleem is dat er geen duidelijk beeld is van hoelang een K-formulier is ingeschreven. Een gevolg hiervan is dat soms de reparatie streeftijd van drie weken wordt overschreden. Het is van groot belang dat deze problemen worden opgelost en het reparatietraject efficiënter wordt doorlopen. Het repareren van aandrijfcomponenten is namelijk een groot deel van de inkomsten en één van de belangrijkste processen van CT.

3.2 Visie toekomstig reparatiesysteem en -proces

Hoe ziet men bij CT nu het nieuwe systeem, en wat zijn de ideeën over hoe de gesignaleerde problemen moeten worden opgelost? Er zal een gecentraliseerde database zijn waarin de gegevens van een reparatie worden vastgelegd. Zaken als bijlagen, testrapporten, pakbonnen en dergelijke zullen hierin ook terug te vinden zijn. Het systeem zal hiernaast ook een planning bevatten zodat iedere werknemer weet wat hij nog moet doen. Het systeem heeft een duidelijke, gebruiksvriendelijke gebruikersinterface. Iedere medewerker die met het reparatietraject te maken heeft zal toegang hebben tot het systeem. De werking van het proces blijft min of meer gelijk, het enige verschil is dat er nu met digitale versies van de K-formulieren zal worden gewerkt. Doordat de reparatiegegevens centraal, digitaal staan opgeslagen zal het niet meer voorkomen dat een K-formulier zoek raakt, dat een klant sneller van informatie kan worden voorzien en dat K-formulieren niet meer blijven liggen.

3.3 Analyse nieuwe systeem opzet

De optie om een standaardpakket aan te schaffen is overwogen, maar door de grote hoeveelheid specifieke informatie die moet worden opgeslagen is deze optie minder aantrekkelijk dan een maatwerkpakket. Ook zullen de kosten voor een standaardpakket aanzienlijk groter zijn dat van een maatwerkpakket gemaakt door een student, of dit nu een stagiair of een afstudeerder is. Wat ontwikkeltijd betreft zal er weinig verschil zijn tussen de twee opties. Ook aan een standaardpakket zullen aanpassingen nodig zijn geweest voor het efficiënt functioneren binnen CT. Deze argumenten in beschouwing nemend, is er gekozen voor een maatwerkoplossing in de vorm van een stage/afstudeeropdracht.

Een opzet voor een nieuw systeem is al eens gemaakt door Mickeal Verschoor. Mickeal is een voormalig part-time medewerker, en nu informatica-student aan de Hogeschool van Utrecht. Zijn opzet is beperkt gebleven tot het efficiënt rondsturen van K-formulieren, wat praktisch neerkomt op pilots 1 en 2 uit mijn opdracht. Hij is zo vriendelijk geweest mij zijn visie uit te leggen. Zijn opzet komt globaal op het volgende neer:

- Een 3-Tier (drie lagen applicatie (databaselaag, applicatielaag, interfacelaag))
- Een cliënt-side programma gebruikmakend van server-side COM+ objecten. (decentrale aanpak)
- Ontwikkelen in VisualBasic6.0 of later in C++.
- MS SQL Server als achterliggende database.

- Gebruik maken van stored procedures.
- Objectgeoriënteerde aanpak.
- Koppeling met bestaande klant en artikel databases.

Mijn mening is dat zijn opzet verre van volledig is, wat hij ook heeft bevestigd. Daarnaast vind ik zijn opzet te ingewikkeld. Niet dat ik het niet begrijp, ik begrijp het zelfs heel goed. Als ik zijn opzet lees dan krijg ik het gevoel dat het niet de werkelijkheid representeert. Het hart van zijn opzet zijn de verschillende "managers" en de UserInterfaceFactory. Deze verschillende klassen zullen waarschijnlijk de interne werking van het systeem moeilijk te doorgronden maken, omdat objecten uit de werkelijkheid uit verschillende klassen worden opgebouwd. Een ander nadeel vind ik persoonlijk het gebruik van een cliënt-side programma met COM+ objecten. Met COM+ is in principe niets mis, en met cliënt-side programma's ook niet. Maar bij cliënt-side applicaties ligt het overgrote deel van de business logica bij de cliënt waardoor deze zwaarder belast wordt. Een server is echter beter geconfigureerd om met dit soort belastingen om te gaan. Daarnaast zijn de werkstations bij CT al een jaar of drie oud en dus niet meer "top of the line". Een ander nadeel is dat, als er updates van de software zijn, deze op iedere cliënt moeten worden geïnstalleerd. Als laatste moeten in de toekomst klanten zelf de status van hun ter reparatie aangeboden componenten via het web kunnen raadplegen. Op deze punten na heeft zijn concept ook een aantal hele goede punten. Zo is bijvoorbeeld de 3-Tier applicatie structuur heel goed, tegenwoordig is dit de standaard voor object georiënteerde oplossingen. Op deze manier kunnen de database-, applicatie- en interfacelaag goed worden gescheiden. Ook het gebruik van MS SQL Server is een goed voornemen. MS SQL Server is namelijk een krachtig en stabiel platform voor grote databases. Daarnaast kan met MS SQL Server optimaal gebruik worden gemaakt van de stored procedures die het biedt. Stored procedures maken dataselectie uit een database snel en gemakkelijk. Stored procedures worden geschreven in een SQL-dialect wat Transact-SQL of T-SQL wordt genoemd. Met deze eenvoudige programmeertaal is het mogelijk om functies, die normaal gesproken in de applicatielaag van een applicatie terug zijn te vinden, te verplaatsen naar de databaselaag. Dit zorgt voor minder communicatie tussen de database- en applicatielaag, wat de applicatie sneller zal maken. Tevens is er een snelheidswinst omdat de stored procedures voor gecompileerd zijn. Objectoriëntatie is een goede manier om snel een efficiënte applicatie te ontwikkelen.

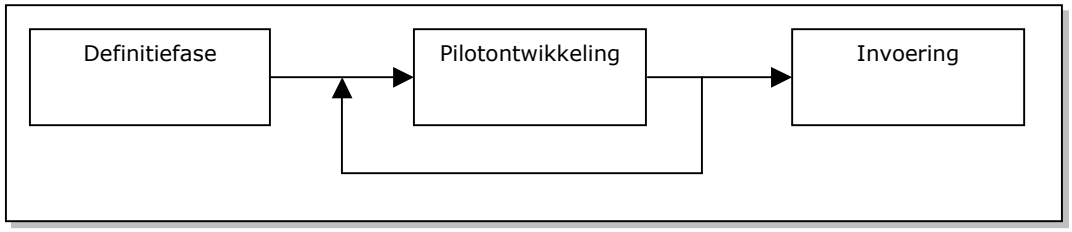
Gezien de genoemde bezwaren, vooral de decentralisatie, is er in overleg met de opdrachtgever besloten een andere weg in te slaan. Namelijk de weg van centralisatie en het thin-cliënt concept. Hoe de structuur eruit gaat zien zal verderop in dit verslag worden besproken.

3.4 Ontwikkelscenario

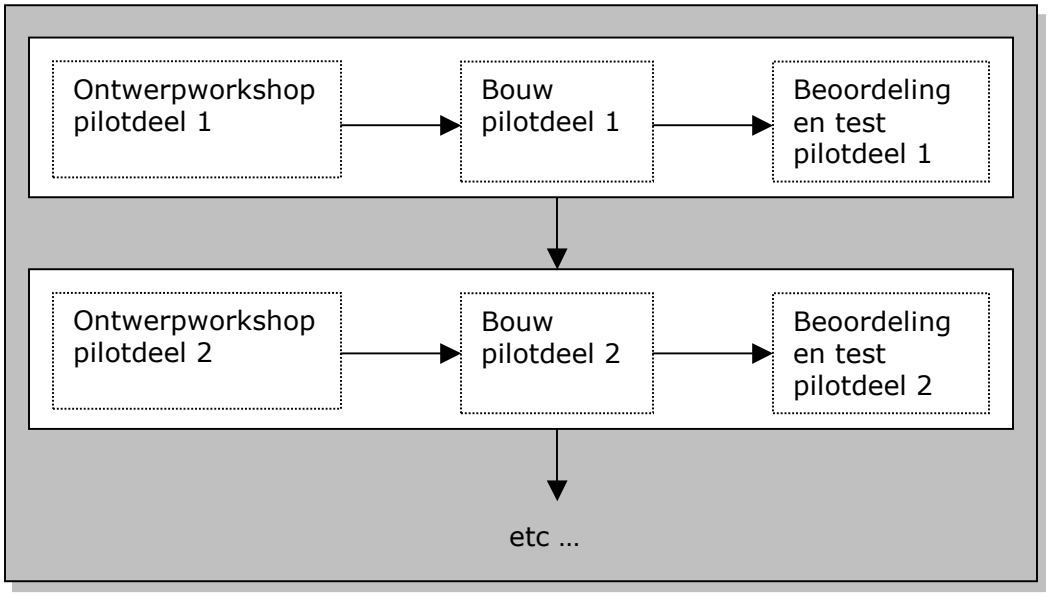
Omdat er voor de IAD ontwikkelmethode is gekozen moet een ontwikkelscenario worden bepaald. Het IAD ontwikkeltraject is verdeeld in een drietal fasen, te weten: "Definitiefase", "Ontwikkelfase", "Invoeringsfase". Het ontwikkelscenario geeft aan hoe de fasen iteratief worden doorlopen. Globaal zijn er vier verschillende scenario's. Het "Evolutionair ontwikkelen", het "Incrementeel opleveren", het "Incrementeel ontwikkelen" en het "Big Bang invoeren". Voor een gedetailleerde beschrijving van alle vier de scenario's wil ik verwijzen naar het boek "IAD Het evolutionair ontwikkelen van informatiesystemen" van R.J.H. Tolido (ISBN: 90-395-0401-6). In dit geval is er gekozen om het systeem te ontwikkelen

volgens het "Incrementeel ontwikkelen" scenario. Dit zal ik dan ook nader toelichten.

Incrementeel ontwikkelen is een strategie waarbij eerst de systeemeisen volledig in kaart worden gebracht. Aan de hand van de systeemeisen wordt dan één systeem concept opgezet. Hierna wordt het gehele systeem door middel van een aantal iteratieslagen ontwikkeld. Vervolgens wordt het gehele systeem in één keer ingevoerd. Dit scenario ziet er globaal als volgt uit:



Binnen de pilotontwikkelingsfase bestaan nog drie fasen. De fasen "Ontwerp-workshop", "Bouw", "Beoordeling en test". In de manier waarop deze fasen worden doorlopen zijn twee smaken. Eén, eerst voor alle iteratieslagen "Ontwerpworkshop", daarna per iteratie "Bouw" en "Beoordeling en test". Twee, de drie fasen iteratie voor iteratie. Tenslotte is er ook de mogelijkheid om de pilots parallel te ontwikkelen. Echter omdat er voor dit project maar één "A-team" is, is het niet verstandige deze manier te volgen. De manier waarvoor is gekozen in het kader van dit project, is de tweede en ziet er als volgt uit:



3.5 Systeemeisen

Voor aanvang van de opdracht heeft de opdrachtgever een lijst van eisen en wensen opgesteld. Deze lijst is te vinden in bijlage II pagina 6.

Omdat een informatieanalist met deze lijst op zich niets kan, is er verder onderzoek gedaan naar de werking van het huidige reparatieproces en het huidige systeem. De analyse van het huidige proces en systeem is te vinden in paragraaf 3.1. Naast deze analyse is er aan experimentele prototyping gedaan. Dit houdt in dat, voor dit project, eerst schetsen van de interfaces worden gemaakt. Deze schetsen kunnen leiden tot nieuwe inzichten in eisen en wensen, en misschien zelfs wel nieuwe eisen en wensen aan het licht brengen. Uit eerdere ervaringen met deze werkwijze is gebleken dat tot een zeer volledig eisenpakket kan worden gekomen. Tevens krijgt de opdrachtgever al een beetje inzicht in hoe het uiteindelijke product zal worden, wat vaak als prettig wordt ervaren. De prototypen van de interfaces zijn te vinden in de bijlagen van bijlage II.

Uit het onderzoek en verschillende gehouden interviews is, samen met de opdrachtgever, de volgende lijst met definitieve eisen en wensen opgesteld.
Eisen:

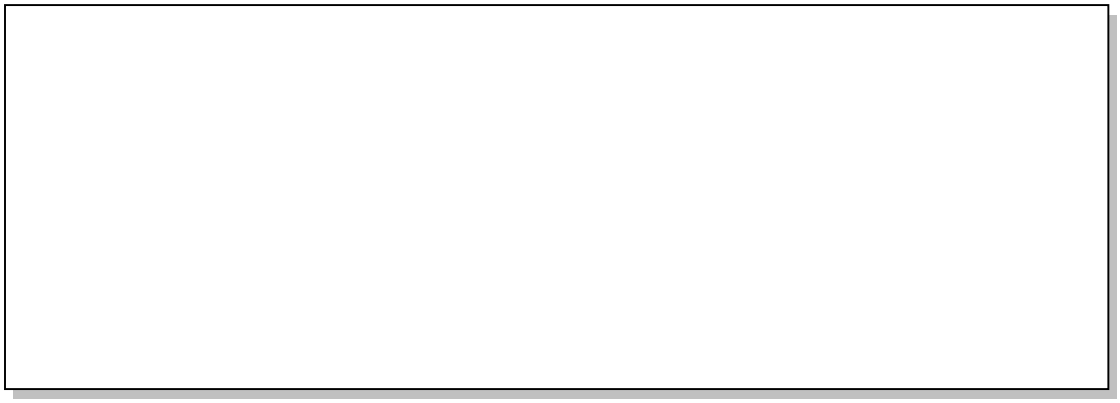
- K-formulier moet kunnen worden ingevoerd.
- Klant moet kunnen worden aangegeven.
- Drivetype moet kunnen worden aangegeven.
- Gebruikte onderdelen moeten kunnen worden aangegeven.
- Labels met opdrachtinformatie moeten kunnen worden uitgedraaid.
- Er moet kunnen worden bijgehouden welke communicatie er omtrent een K-formulier is geweest.
- Bijlagen moeten digitaal kunnen worden bijgevoegd.
- Een testrapport moet digitaal kunnen worden bijgevoegd.
- Het k-formulier moet kunnen worden uitgedraaid voor het archief.
- Een orderbevestiging moet kunnen worden uitgedraaid.
- Een pakbon richting klant moet kunnen worden uitgedraaid.
- Een pakbon richting Engeland moet kunnen worden uitgedraaid.
- K-formulieren moeten virtueel door de onderneming kunnen worden gestuurd.
- Een planning met alle lopende reparaties moet kunnen worden gegenereerd.
- Er moet naar opdracht/k-formulier kunnen worden gezocht op verschillende criteria.
- Er moeten verschillende overzichten kunnen worden gegenereerd.
- Verschillende gebruikers, verschillende rechten.

Wensen:

- Exporteren van overzichten naar Microsoft Excel.
- Exporteren van documenten naar Microsoft Word.
- Stabiele en consistente database en applicatie.
- De applicatie moet gebruiksvriendelijk zijn.
- Er is een koppeling met huidig bestaande databases gewenst.
- Het systeem moet op de huidige ICT faciliteiten kunnen werken.
- Het is gewenst dat met een streepjescode scanner gewerkt kan worden.

3.6 Systeemconcept

Om tot een inzichtelijk systeem concept te komen is gebruik gemaakt van use-cases zoals deze in de UML techniek worden gebruikt. De use-cases bestaan uit use-case diagrammen en een beschrijving van iedere use-case gedefinieerd in deze diagrammen. "Use-case" is letterlijk vertaald "gebruiksmogelijkheid". Dit is letterlijk wat ze beschrijven, de gebruiksmogelijkheden van het systeem. Deze methode is zeer geschikt om de opdrachtgever en/of gebruikers kennis te laten maken met de toekomstige mogelijkheden van het systeem. Tevens is het voor de opdrachtgever/gebruikers gemakkelijk na te gaan of zij of de systeemontwikkelaar geen systeemeisen over het hoofd hebben gezien. Er moet wel in acht worden genomen dat er op deze manier niet steeds meer en meer wensen in plaats van eisen in het systeemconcept sluipen. Daarom zijn de use-cases in dit project alleen met de opdrachtgever overlegd. De use-cases zijn opgenomen in bijlage II. Voor het overzicht zijn de use-cases per type gebruiker gegroepeerd. Het gebruikerstype beheerder is speciaal. Dit is namelijk een toevoeging aan één van de andere gebruikerstypen, waarschijnlijk wordt dit de inkoop medewerker. Ter verduidelijking zal één interessante use-case worden uitgelicht en verder worden toegelicht. Deze use-case is: "Opslaan Bijlagen".



Bij het definiëren van use-cases wordt het systeem gezien als een "black-box". Het is in dit stadium namelijk nog niet bekend hoe het systeem werkt, het is nu dan ook veel belangrijker om te weten wat het systeem doet. In dit geval zal het opslaan van de bijlagen onder de loep worden genomen. De details van de use-case zijn opgenomen in use-case templates. Hieronder is de use-case template van de use-case "Opslaan Bijlagen" opgenomen.

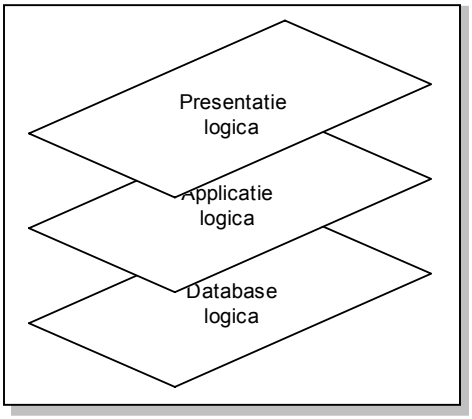
Naam	Opslaan bijlagen.
Samenvatting	Gescande bijlagen worden in het systeem opgeslagen.
Actoren	Magazijn Medewerker, EWP medewerker, Verkoper, Inkoper, Manager.
Aannamen	De medewerker heeft de bijlagen gescand als plaatje (.JPG, .JPEG of .GIF extensie). Medewerker heeft het scherm "K-formulier" open staan.
Beschrijving	(1) De magazijn medewerker klikt op de knop "Bijlagen". (2) Het bijlagen scherm wordt geopend. (3) De pagina "Bijlagen" wordt geladen. (4) De magazijn medewerker geeft het pad aan waar de bijlage file staat. (5) De magazijn medewerker klikt op "Opslaan". (6) De gegevens worden opgeslagen in de database.

	(7) Het bijlage bestand wordt opgeslagen op de server. (8) De pagina "Bijlagen" wordt herladen. (9) Herhaal indien nodig stap 4 t/m 7.
Uitzonderingen	[Sluiten] De gebruiker klikt op "Sluiten" en het bijlagenscherm wordt gesloten.
Resultaat	Een bijlage bestand is fysiek opgeslagen op de server, een verwijzing naar de bijlage is opgeslagen in de database.

Zoals te zien is de use-case template een korte maar volledige beschrijving van de functie "Opslaan Bijlagen". Zonder in te gaan op de interne werking van het systeem wordt toch duidelijk gedefinieerd wat het systeem moet doen.

3.7 Technische structuur

De applicatie is opgezet naar het zogenaamde 3-tier concept. Dit houdt in dat de applicatie in drie delen is opgedeeld. Presentatielogica, applicatielogica, en de databaselogica. De presentatielogica laag bevat alle interfaces. In dit geval zijn dat de .aspx en .html bestanden. Deze bestanden bevatten de knoppen, invoervakken en dergelijke die door de gebruikers kunnen worden benut. De applicatielogica laag bestaat uit de klassen die samenwerken en het systeem maken tot wat het is. Het verzorgt de informatie van database naar interface en andersom. Verder bevat het alle logica die nodig is om de interface aan te sturen, data te transformeren etc. In dit geval zijn dit de bestanden met .cs en .aspx.cs extensies. De databaselogica laag bevat de tabellen en stored procedures, nodig voor de opslag van data. Globaal kan het 3-tier concept op de onderstaande manier worden gevisualiseerd.



De presentatie logica is opgebouwd uit HTML. HTML staat voor hypertext markup language en is ontwikkeld in de 90'er jaren. Versie 4.01 is ten tijde van het schrijven van dit document de meest recente versie. Een HTML pagina wordt opgebouwd met behulp van "tags". Tags zijn blokjes code die een bepaald HTML-element definiëren. Zo zal het onderstaande codefragment een invoervak en een knop onder elkaar in de web-browser laten zien.

```
<input id="Vak" type="text" NAME="Vak" /><br>
<input id="Knop" type="button" name="Knop" value="Knop" />
```

De tags die hier zijn gebruikt zijn `<input>` en `<br>`. `<input>` staat voor een invoer element, `type` geeft hier aan wat voor soort invoer element. `<br>` staat voor een regel break. Omdat er voor dit project met ASP.NET wordt gewerkt, is er nog een ander soort HTML beschikbaar, genaamd ASP. Zie het onderstaande fragment.

```
<asp:TextBox id="Vak" Runat=server></asp:TextBox><br>
<asp:Button ID="Knop" Text="Knop" Runat=server></asp:Button>
```

Deze twee code fragmenten zullen voor het oog exact dezelfde output generen. Echter zijn er een aantal enorme verschillen. De elementen die door standaard HTML worden gegenereerd bestaan alleen lokaal op de computer die de betreffende HTML pagina opvraagt. De ASP elementen bestaan echter naast lokaal bij de opvragende computer ook op de server. Dit geeft een enorme potentie voor server-side scripts. Het feit dat ze op de server bestaan wordt mogelijk gemaakt doordat een instantie van de klasse "TextBox" of "Button" met dezelfde naam als het HTML element kan worden aangemaakt. Het ASP.NET framework biedt een grote hoeveelheid aan klassen voor het maken van krachtige webapplicaties. Zo bevat het niet alleen klassen voor gebruikers invoer, maar ook voor onder andere fysieke schijf bewerkingen, Active Directory bewerkingen, database bewerkingen, en email bewerkingen. Het is zelfs mogelijk om met behulp van het .NET framework verschillende soorten documenten te generen, van Word tot Excel tot PDF.

De applicatielogica laag is opgebouwd uit de klassen die het systeem maken tot wat het is. Deze klassen kunnen worden onderverdeeld in twee groepen, de "code-behind" klassen en de "object" klassen. De code-behind klassen verzorgen de functionaliteiten achter de HTML/ASP pagina's en de object klassen reflecteren objecten uit de werkelijkheid, zoals bijvoorbeeld een klant of een opdracht. Deze klassen bevinden zich in de .cs en .aspx.cs bestanden, de structuur van deze bestanden ziet er als volgt uit.

```
using System;       // Lijst van namespaces waar gebruik van wordt gemaakt.

namespace CTEWP     // Namespace waar de klasse in bestaat.
{
       /// <summary>
       /// Summary description for Class1.
       /// </summary>
       [serializable]
       public class Class1 // Naam van de klasse.
       {
              public int Integer1;     // Variabelen declaraties.

              public Class1()    // Deze code wordt aangeroepen bij het
              {                     initialiseren van de klasse
                     //
                     // TODO: Add constructor logic here
                     //
              }

              /// <summary>
              /// Summary description for Method1.
              /// </summary>
              public void Method1()    // Methoden declaraties van de
              {                        // klasse.
                     //
                     // TODO: Add method logic here
                     //
              }
       }
}
```

Alle klassen waar de applicatie uit opgebouwd is, hebben deze structuur. De code zelf is geschreven in de taal C# (uitgesproken als C-sharp). ASP.NET is namelijk geen programmeertaal, maar een framewerk wat een aantal standaard functies verschaft die door de programmeur kunnen worden benut. De eigenlijke code waarmee deze functies kunnen worden gemaakt en/of aangeroepen wordt geschreven in C#, VB.NET, C++.NET, of J#.NET.

Doordat gekozen is voor ontwikkelen met behulp van VisualStudio.Net is het mogelijk gebruik te maken van speciale functies. Zo is het mogelijk code-behind pagina's te gebruiken. Deze code-behind pagina's bevatten de functies om de elementen op de HTML/ASP pagina's aan te sturen. Omdat het klassen zijn worden ze mee gecompileerd met de object-klassen door de pre compiler. Zou er geen code-behind worden gebruikt, dan zouden de scripts in de HTML/ASP pagina's zelf staan en worden gecompileerd door de run-time compiler als de pagina wordt opgevraagd. Door gebruik te maken van de pre compiler worden de klassen gecompileerd tot een "dynamic link library" (DLL) bestand. De run-time compiler hoeft nu deze code niet meer te compileren maar kan de DLL gewoon uitlezen. Dit geeft een enorme prestatiewinst voor de applicatie. Een andere functie van VisualStudio.Net is het automatisch genereren van technische documentatie. Door het automatisch laten genereren van technische documentatie worden de `///<summary></summary>` blokken uitgelezen en verwerkt tot HTML pagina's. Deze funtctie is handig wanneer er met een groep mensen wordt ontwikkeld, of wanneer later aanpassingen moeten worden gedaan. Ook bevat VisualStudio.Net een debugger waarmee fouten in de applicatie kunnen worden opgespoord.

De databaselogica laag bestaat uit MS SQL Server 2000. Er is voor SQL Server 2000 gekozen omdat het een enorm stabiel database platform is wat snel een grote hoeveelheid aan data kan opslaan en verwerken. Daarnaast is er een keur aan beheersmogelijkheden waarmee het onderhoud van de databases gemakkelijk geautomatiseerd kan worden. Ook is CT op zoek naar een nieuw customer relations management (CRM) systeem wat ze ook op SQL server willen laten gaan draaien. Door MS SQL Server te gebruiken kan gebruik worden gemaakt van stored procedures. Om data op te halen uit een database, wordt gebruik gemaakt van query's. Deze query's zouden normaal gesproken tussen de programmacode staan. Stored procedures daarentegen zorgen ervoor dat query's uit de programmacode kunnen worden gehaald, en in de database worden geplaatst. Doordat de query's op deze manier dichter bij de data staan, en zijn voorgecompileerd, zullen database bewerkingen sneller worden uitgevoerd. Tevens is er minder communicatie nodig tussen de applicatielogicalaag en de databaselogicalaag, en de queries staan bij elkaar in plaats van verspreid over de code. Stored procedures zien er als volgt uit.

```
CREATE PROCEDURE dbo.StoredProcedure1
/*
	(
		@parameter1 datatype = default value,
		@parameter2 datatype OUTPUT
	)
*/
AS
	/* Code to execute */
	RETURN
```

Stored procedures worden geschreven in Transact SQL ofwel T-SQL. Dit is gewoon een simpele programmeertaal geënt op databasebewerkingen en dataconversie.

Hoe komt de gebruiker nu aan zijn data? De gebruiker zal nooit en te nimmer rechtstreeks de tabellen uit de database openen. Sterker nog, de gebruiker zal alleen maar met de interfacelogica laag communiceren. Dit voorkomt dat een gebruiker geen bewerkingen kan uitvoeren die hij niet mag uitvoeren. Deze interfacelogica roept de gebruiker op via zijn web-browser. Als hij de link naar de applicatie opent, zal hij contact maken met de webserver waar de interface en applicatielogica worden geparsed. Dat houdt in dat de code door de webserver wordt uitgevoerd, en de HTML uitvoer naar de cliënt wordt gestuurd. Tevens zal de database server door de web-server worden aangeroepen en worden stored procedures uitgevoerd. Naast de database server kunnen ook de active directory server en de mail server worden aangeroepen. De active directory kan op het niveau de databaselogica laag worden geplaatst, en de mail-server op het niveau van de applicatielogica laag.

3.8 Organisatorische veranderingen voor de onderneming

De veranderingen in de organisatie zullen minimaal zijn. Met het ontwerpen van de applicatie is er rekening mee gehouden dat de werkwijze zoals deze nu is, zoveel mogelijk in stand wordt gehouden. Vooral wat betreft de communicatie, ofwel de weg van een reparatiedossier door de organisatie. Dit heeft als voordeel dat de medewerkers redelijk snel zullen wennen aan de nieuwe applicatie. Zoals gezegd zal er organisatorisch niet veel veranderen, de medewerkers behouden allemaal hun eigen verantwoordelijkheden en nieuwe verantwoordelijkheden zullen er niet veel bijkomen, wel kunnen ze iets wijzigen. Nu moeten bijvoorbeeld magazijn medewerkers bijlagen die binnenkomen bij een reparatiecomponent in scannen en digitaal bij een k-formulier voegen, in plaats van achter een uitgeprint k-formulier nietten. Ook zullen ze een orderbevestiging uit moeten gaan draaien.

3.9 Pilots

De IAD ontwikkelmethode schrijft voor dat het ontwikkelen van het systeem verdeeld wordt in zogenaamde pilots. Deze pilots zijn op zichzelf staande, ongeveer even grote delen van het systeem. In het geval van dit project is dit niet helemaal het geval. Het systeem is opgedeeld in vier delen. Deze delen zijn echter niet allemaal ongeveer even groot en niet geheel op zichzelf staand. De volgende vier pilots zijn onderscheiden:

- K-formulier
- Communicatie
- Planning en overzichten
- Beheer

De pilot “k-formulier” is het grootste deel van het systeem en vormt eigenlijk de basis voor de rest van het systeem. Dit is namelijk het deel wat te maken heeft met alle gegevens van een reparatie, het is eigenlijk een digitale versie van het huidige papieren k-formulier. Er wordt op aangegeven wat voor aandrijfcomponent het betreft, wie de klant is, wat de symptomen en diagnose van het component is, wat de kosten voor reparatie zijn, of het eventueel naar Engeland moet etc. Hier komen nog bij de bijlagen, het testrapport, orderbevestiging, pakbon richting klant

en het afdrukken van labels. Deze pilot vormt dus de basis voor de andere pilots, want wat kun je plannen als er nog geen k-formulieren zijn en wat moet je doorsturen als er nog geen k-formulieren zijn. De pilot k-formulieren heeft dus de hoogste prioriteit.

De pilot "communicatie" behelst eigenlijk alles wat met het informeren van andere personen van doen heeft. Het omvat het virtueel sturen van de k-formulieren naar andere personen binnen de onderneming en het bijhouden van interne-, externe-, telefoon en faxcommunicatie. Deze pilot leek misschien wel het meest gecompliceerd voor aanvang van het ontwerpen, echter bleek het later vrij simpel te realiseren. Doordat communicatie een belangrijk deel van het gehele proces is heeft deze pilot de op één na hoogste prioriteit gekregen.

In de pilot "planning en overzichten" zijn eigenlijk twee onderdelen samengenomen omdat ze erg overeenkomen. "Overzichten" omvat het kunnen opbouwen en ophalen van overzichten, en "planning" omvat in wezen eigenlijk ook een overzicht van de openstaande k-formulieren per persoon. Echter zal de planning actiever zijn dan een standaard overzicht omdat er constant zaken aan veranderen als een k-formulier door de onderneming wordt gestuurd. Deze pilot krijgt de derde prioriteit.

De laatste pilot is "beheer" en dit bevat alles wat te maken heeft met het onderhouden van de applicatie en de toegang tot de applicatie. Zo wordt er mogelijk gemaakt statussen, medewerkers, afdelingen, etc. toe te voegen, te verwijderen en te wijzigen. Ook zal het de toegang tot de applicatie omvatten door het middel van een inlog scherm. Deze pilot heeft de laagste prioriteit.

4 Pilot 1: K-formulieren

In het nu volgende hoofdstuk zal worden ingegaan op het ontwerp en de bouw van de pilot "K-formulieren". Besproken zullen worden onder andere het systeemontwerp (klassediagram, sequentie diagrammen), het database ontwerp in de vorm van een ERD en een aantal interessante code fragmenten. Bedenk dat de besproken producten volgens een aantal iteratie slagen zijn verkregen.

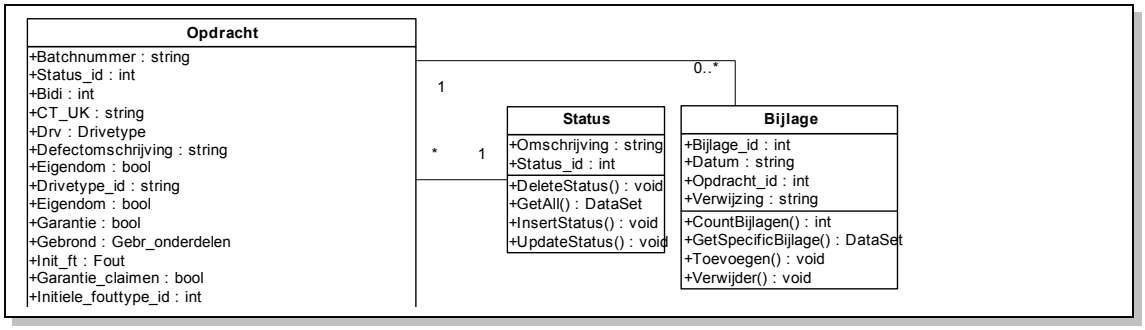
4.1 Globaal-functionele structuur

De globaal functionele structuur van de pilots zal worden gedefinieerd aan de hand van het klassediagram uit UML. Het klassediagram is een statische weergave van de werkelijkheid. Objecten uit de werkelijkheid zijn gemodelleerd met hun variabelen en functies. Teneinde een klassediagram te maken zal eerst een lijstje van kandidaat klassen moeten worden opgesteld. Dit lijstje ziet er als volgt uit.

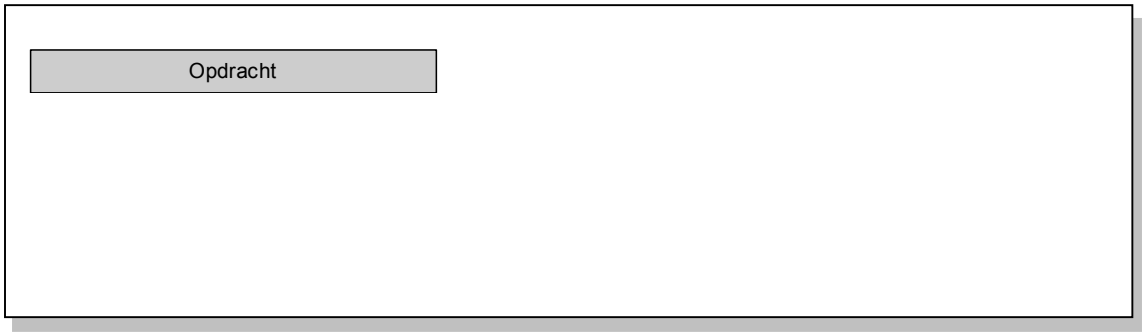
Kandidaat	**Beslissing**	**Kandidaat**	**Beslissing**
Opdracht	Klasse	Bijlage	Klasse
K-formulier	Implementatie	Analyse	= fout
Drivetype	Klasse	Fout	Klasse
Drive	Instantie van Drivetype	Testrapport	Klasse
Klant	Klasse	Label	Implementatie
Pakbon	Klasse	Magazijn	Instantie van Afdeling
Reparatie	= opdracht	Locatie	Attribuut
EWP	Instantie van Afdeling	Serienummer	Attribuut
Status	Klasse	Gebruikt onderdeel	Instantie van Onderdeeltype
Onderdeeltype	Klasse	Gebruikte tijd	Instantie van Tijd
Medewerker	Klasse	Afdeling	Klasse
Tijd	Klasse		

Een toelichting is hier op zijn plaats. Bijvoorbeeld de kandidaat klassen "opdracht" en "k-formulier". Waarom is nu de één een klasse en ander een implementatie? In feite representeren opdracht en k-formulier dezelfde dingen, namelijk de reparatie van een aandrijfcomponent. Opdracht is echter als klasse genomen en niet k-formulier omdat opdracht meer generiek is, het zegt meer dan k-formulier. K-formulier is dan een implementatie van de klasse opdracht.

Aan de hand van deze lijst is het klassediagram zoals in bijlage III opgesteld. Zoals is te zien, is de klasse "Opdracht" de meest belangrijke en gecompliceerde. Deze klasse bevat alle specifieke gegevens van een reparatie opdracht. De attributen zijn gekozen naar aanleiding van het huidige programma en het huidige k-formulier. De methoden zijn na verloop van tijd gegroeid aan de hand van de sequentie diagrammen. Verder spreekt het klassediagram voor zich. Hieronder is een deel van het klassediagram opgenomen ter illustratie. De klassen Opdracht, Status en Bijlage zijn te zien. Er is te zien dat een Opdracht altijd één Status heeft en dat een Status bij meerdere opdrachten kan horen. Er is te zien dat een Bijlage altijd bij één opdracht hoort, en dat een Opdracht nul of meerdere bijlagen heeft.



Aan het klassediagram zit, naar mijn mening, de database structuur gekoppeld. Immers moeten alle attributen die op het klassediagram zijn gedefinieerd terugkomen in de database structuur. Het database ontwerp is gemaakt met de ERD (Entity Relationship Diagram) techniek. Deze techniek vind ik persoonlijk zeer gebruiksvriendelijk. Het ERD is opgenomen in bijlage III. Bij het ontwerpen van de database is rekening gehouden met efficiëntie van het opslaan en ophalen van gegevens, en het consistent houden van de data in de database. Een deel van het ERD is in de volgende figuur opgenomen. De tekentechniek is echter niet ERD, het principe erachter is hetzelfde. De tekentechniek ERD is niet gebruikt omdat het modelleringsprogramma MS Visio deze niet als template bied.

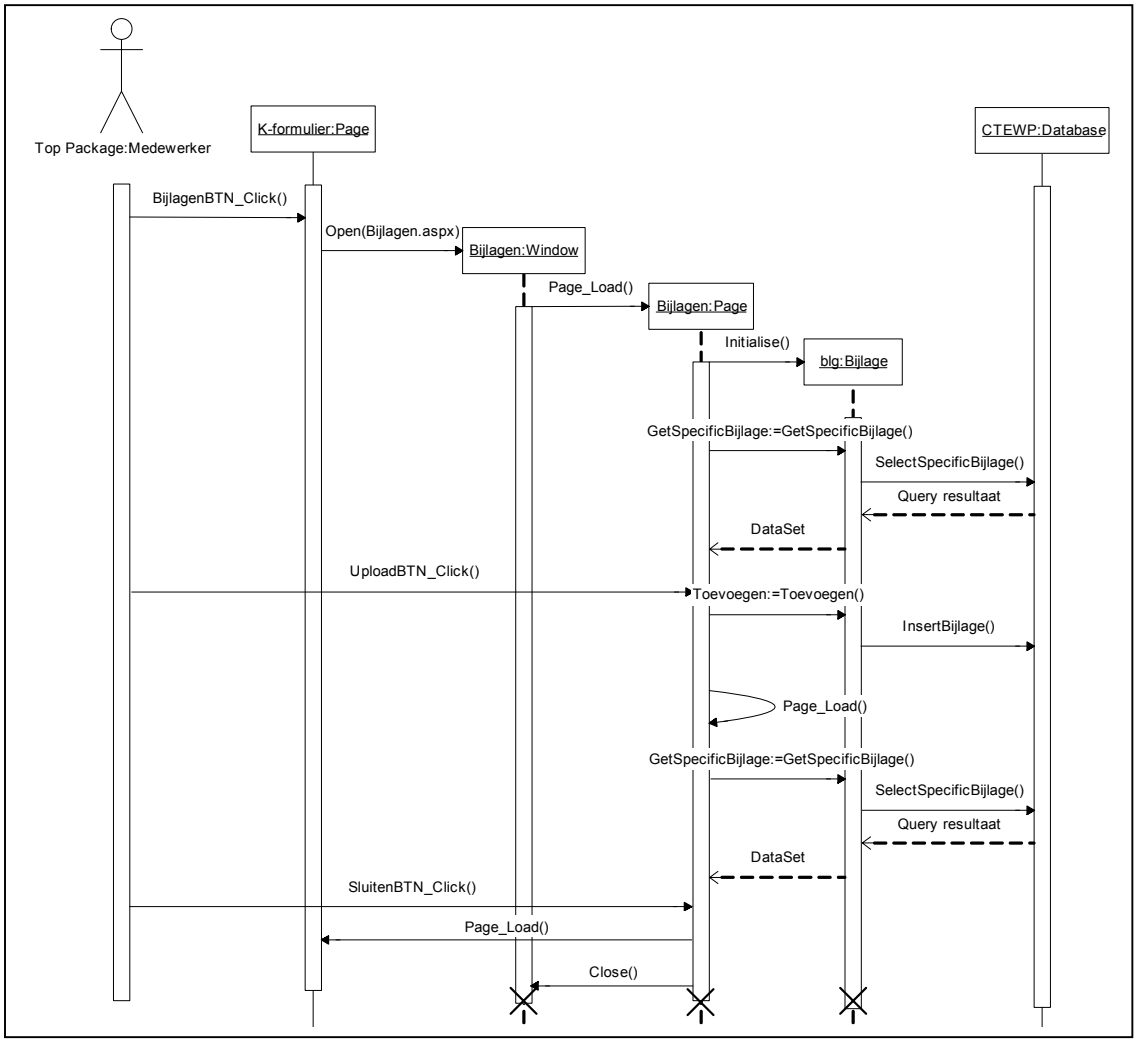


Waar in de ERD techniek een entiteit met een rechthoek wordt aangegeven, met attributen als ballonnetjes gekoppeld. Hier wordt een entiteit direct met zijn attributen gedefinieerd. Tevens is aangegeven of een attribuut een primaire of vreemde sleutel is. De primaire sleutels staan in het vak rechtsboven, de overige attributen in het vak rechtsonder. In de linker kolom wordt aangegeven of een attribuut een primaire of vreemde sleutel is, of beiden.

Ook de definitie van de relaties is enigszins anders. In de ERD techniek wordt een combinatie van een ruit en lijnen gebruikt om het type relatie en de cardinaliteit aan te geven. In de gebruikte manier van tekenen wordt een relatie gedefinieerd met behulp van een pijl en een lijn. De pijl wordt aan het entiteit met de primaire sleutel gekoppeld. Een nadeel van deze manier van tekenen is dat niet meteen de cardinaliteit zichtbaar is. Mijn persoonlijke mening is dat met de gebruikte manier van tekenen goed te werken is. De definitie van de entiteiten is duidelijker. De definitie van de relaties minder duidelijk, maar dit is niet onoverkoombaar.

4.2 Globaal-technische structuur

Om duidelijkheid te krijgen in de technische structuur en de samenwerking van de klassen is gebruik gemaakt van sequentie diagrammen. De sequentie diagrammen geven per use-case aan hoe het systeem werkt. Aan de hand van deze diagrammen kunnen de methoden van een klasse worden bepaald. Men weet immers waar de use-case begint en wat het resultaat van de use-case moet zijn. Hierdoor kan men vrij gemakkelijk bepalen wat er intern in het systeem moet gebeuren. Ter illustratie zal één sequentie diagram worden uitgelicht. Om in de trend van voorgaande voorbeelden te blijven zal het sequentie diagram van de use-case "Opslaan bijlagen" als voorbeeld worden genomen. Voor de volledigheid is de database als object opgenomen in de diagrammen. Hierdoor wordt het mogelijk de stored procedures te definiëren. Deze worden dan gezien als functies/methoden van het database object.



Het diagram geeft aan wat het systeem doet als om een actie wordt gevraagd. In dit geval "vraagt" de gebruiker de code achter de knop BijlagenBTN uit te voeren. Dit wordt gevraagd aan het object "K-formulier", een instantie van de klasse "Page". Deze klasse wordt door het .NET framework gegeven. De pagina creëert een nieuwe instantie van de klasse "Window", welke door internet explorer wordt

aangegeven. Het venster creëert een nieuwe instantie van de klasse "Page" genaamd "Bijlagen" en vraagt de Page_Load functie uit te voeren. Het object "Bijlagen" creëert bij initialisatie een instantie van de klasse Bijlage. De Page_Load functie van de pagina "Bijlagen" vraagt het object Bijlage de functie GetSpecificBijlage uit te voeren, deze vraagt op zijn beurt aan het database object om "SelectSpecificBijlage" uit te voeren. Hierna klikt de gebruiker op de knop "UploadBTN". Hierdoor vraagt het object Bijlagen aan de paginaklasse Bijlage om de functie "Toevoegen" uit te voeren. Het object Bijlage vraagt aan het object CTEWP om de stored procedure "InsertBijlage" uit te voeren. Doordat een postback naar de server optreed zal de functie Page_Load weer worden uitgevoerd en zal de functie GetSpecificBijlage weer worden uitgevoerd, en de stored procedure "SelectSpecificBijlage". De gebruiker klikt hierna op de knop "SluitenBTN", waarna de pagina "K-formulier" opnieuw wordt geladen en het bijlagen venster wordt gesloten.

4.3 Globaal-organisatorische structuur

Zoals in paragraaf 3.8 is beschreven zullen de organisatorische veranderingen gering zijn. Hierdoor is het niet noodzakelijk hier nog eens op de veranderingen in te gaan.

4.4 Pilot ontwikkelplan

Bij het maken van het pilotplan is gebruik gemaakt van de planning als basis voor time-boxing. Voor het ontwikkelen van het totale systeem zijn vijftien weken uitgetrokken. Omdat de pilot "K-formulieren" de grootste is, is hier voor vijf weken uitgetrokken. Bij de klassen zijn de stored procedures inbegrepen. Bij de interface zijn de interface klassen inbegrepen.

4.5 Ontwerp software bouweenheden.

Voor deze pilot zijn vijf software bouweenheden gedefinieerd. Te weten:

- Database tabellen
- Stored procedures
- Klassen
- Interface
- Interfaceklassen

Deze bouweenheden hebben een zekere onderlinge prioriteit. Zo is het verstandig (of soms zelfs noodzakelijk) eerst de databasetabellen, relaties en stored procedures te maken. Hier is dan dus ook mee begonnen. Hierna zijn de applicatieklassen gemaakt. Als laatste de interface en interfaceklassen. De ontwerpen van de tabellen liggen vast in het ERD. De werkingen van de stored procedures is af te leiden aan interface ontwerpen en het klassediagram. De ontwerpen van de klassen zijn opgenomen in het klassediagram en de sequentie diagrammen. De ontwerpen van de interface zijn de schetsen die in de definitiefase gemaakt zijn en de werking van de interface klassen is af te leiden aan de use-case omschrijvingen, de interface en de sequentiediagrammen. Voor de overige ontbrekende specificaties voor de interne werking van de klassen wordt een beroep gedaan op de creativiteit van de programmeur. Dit is bewust gedaan vanwege de grote omvang van het project en de geringe tijd die ervoor beschikbaar is. De basis die de gemaakte ontwerpen en diagrammen bieden moet genoeg zijn om tot een systeem te komen welke aan de eisen voldoet.

4.6 Bouw software bouweenheden.

Het resultaat van de bouw van de software bouweenheden voor deze pilot is te vinden in bijlage IV. De bouwfase ging over het algemeen vrij vlekkeloos. Problemen en knelpunten zijn allemaal opgelost of omzeild. Ter toelichting wil ik weer de bijlagen als voorbeeld nemen. Er zal worden uitgelegd hoe de klassen zijn opgebouwd, hoe de stored procedures eruit zien en hoe de interface en interface klassen zijn opgebouwd. Er zal met de klasse worden begonnen. De volledige code van de klasse is te vinden in de bijlagen, voor het gemak zal ik de code in fragmenten verdelen om ze te bespreken.

Namespace declaraties:

```
1 using System;
2 using System.Data;
3 using System.Data.SqlClient;
4 using System.Data.SqlTypes;
5 using System.Configuration;
6
7 namespace CTEWP
8 {
```

Deze regels declareren de delen van het ASP.NET framewerk waar de klasse gebruik van maakt, de zogenaamde namespaces. Bij de applicatie klassen zullen de in het fragment genoemde meestal altijd wel voorkomen. De onderste regel geeft aan dat de klasse bestaat in de namespace CTEWP. Deze namespace ligt ongeveer op hetzelfde niveau als de System namespace.

Klasse declaratie en variabele declaraties:

```
9       /// <summary>
10      /// Standaard applicatie klasse. Bevat alle functionaliteit
11      /// geralateerd aan bijlagen.
12      /// </summary>
13      [Serializable]
14      public class Bijlage
15      {
16              /// <summary>
17              /// Unieke identificatie van de bijlage.
18              /// </summary>
19              public int bijlage_id;
20
21              /// <summary>
22              /// Unieke identificatie van de gerelateerde opdracht.
23              /// </summary>
24              public int opdracht_id;
25
26              /// <summary>
27              /// Fysieke locatie van de bijlage op de server.
28              /// </summary>
29              public string verwijzing;
30
31              /// <summary>
32              /// Bijvoegdatum van het bijlage.
33              /// </summary>
34              public string datum;
35
```

In dit code fragment wordt de klasse gedeclareerd in regel 14. Regel 13 geeft aan dat de klasse serialiseerbaar is, wat nodig is om de klasse in de sessionstate op te kunnen nemen. De sessionstate wordt aangemaakt als er een sessie met de server wordt aangegaan, dus als een gebruiker inlogt op de applicatie. In de sessionstate kunnen variabelen worden opgenomen, maar ook volledige objecten. Als een object in de sessionstate is opgenomen kan deze op iedere plek in de applicatie weer uit de sessie worden gehaald. De waarde van de sessionstate zijn dus globaal. Regels 16 t/m 34 geven de variabelen declaraties van de klasse, met hun datatype en hun toegankelijkheid. In dit geval zijn ze allemaal publiekelijk toegankelijk.

```
36          /// <summary>
37          /// Toevoegen van een bijlage.
38          /// </summary>
39          public void Toevoegen()
40          {
41                SqlConnection conn = new
      SqlConnection(ConfigurationSettings.AppSettings.Get("connecti
      onString"));
42                conn.Open();
43                SqlCommand command = new
      SqlCommand("InsertBijlage", conn);
44                command.CommandType =
      CommandType.StoredProcedure;
45
46                command.Parameters.Add("@knummer",
      SqlDbType.Int);
47                command.Parameters["@knummer"].Value =
      opdracht_id;
48
49                command.Parameters.Add("@verwijzing",
      SqlDbType.VarChar, 150);
50                command.Parameters["@verwijzing"].Value =
      verwijzing;
51
52                command.Parameters.Add("@datum",
      SqlDbType.VarChar, 10);
53                command.Parameters["@datum"].Value = datum;
54
55                command.ExecuteNonQuery();
56
57                conn.Close();
58
59                command.Dispose();
60                conn.Dispose();
61
62                return;
63          }
64
65          /// <summary>
66          /// Ophalen van bijlagen behorende bij een specifieke
67          /// opdracht a.d.h.v. opdracht_id.
68          /// </summary>
69          /// <returns> DataSet met database uitvoer </returns>
70          public DataSet GetSpecificBijlage()
71          {
```

```
72                SqlConnection conn = new
      SqlConnection(ConfigurationSettings.AppSettings.Get("connecti
      onString"));
73                conn.Open();
74                SqlCommand command = new
      SqlCommand("SelectSpecificBijlage", conn);
75                command.CommandType =
      CommandType.StoredProcedure;
76
77                command.Parameters.Add("@knummer",
      SqlDbType.Int);
78                command.Parameters["@knummer"].Value =
      opdracht_id;
79
80                SqlDataAdapter da = new SqlDataAdapter (command);
81                DataSet ds = new DataSet();
82                da.Fill (ds, "Table");
83
84                conn.Close();
85                da.Dispose();
86                ds.Dispose();
87
88                command.Dispose();
89                conn.Dispose();
90
91                return ds;
92          }
93
94          /// <summary>
95          /// Verwijderen van een specifieke bijlage a.d.h.v.
96          /// bijlage_id
97          /// </summary>
98          public void Verwijder()
99          {
100               SqlConnection conn = new
      SqlConnection(ConfigurationSettings.AppSettings.Get("connecti
      onString"));
101               SqlCommand command = new
      SqlCommand("DeleteBijlage", conn);
102               command.CommandType =
      CommandType.StoredProcedure;
103
104               command.Parameters.Add("@bijlagenr",
      SqlDbType.Int);
105               command.Parameters["@bijlagenr"].Value =
      bijlage_id;
106
107               conn.Open();
108
109               command.ExecuteNonQuery();
110
111               conn.Close();
112
113               command.Dispose();
114               conn.Dispose();
```

```
115
116                 return;
117           }
118
119           /// <summary>
120           /// Telling van alle bijlagen behorende bij een
121           /// specifieke opdracht a.d.h.v. opdracht_id.
122           /// </summary>
123           /// <returns>
124           /// Integer met aantal bijlagen bij de opdracht
125           /// </returns>
126           public int CountBijlagen()
127           {
128                 SqlConnection conn = new
      SqlConnection(ConfigurationSettings.AppSettings.Get("connectio
      nString"));
129                 SqlCommand command = new
      SqlCommand("CountBijlagen", conn);
130                 command.CommandType = CommandType.StoredProcedure;
131
132                 command.Parameters.Add("@opdracht_id",
      SqlDbType.Int);
133                 command.Parameters["@opdracht_id"].Value =
      opdracht_id;
134
135                 command.Parameters.Add("@aantal_bijlagen",
      SqlDbType.Int);
136                 command.Parameters["@aantal_bijlagen"].Value = 0;
137                 command.Parameters["@aantal_bijlagen"].Direction =
      ParameterDirection.Output;
138
139                 conn.Open();
140
141                 command.ExecuteNonQuery();
142
143                 conn.Close();
144
145                 command.Dispose();
146                 conn.Dispose();
147
148                 return
      Convert.ToInt32(command.Parameters["@aantal_bijlagen"].Value;
149           }
150     }
151 }
```

Het bovenstaande code fragment laat de declaraties van de methoden van de klasse zien. De eerste methode is “Toevoegen()”, met “void” als return type. In deze methode wordt een nieuw SqlConnection object aangemaakt genaamd conn. Deze krijgt direct een connectiestring mee in regel 41. De connectiestring is een globale variabele die uit het web.config bestand wordt gehaald. Het web.config bestand is zoals de naam al doet vermoeden een configuratie bestand voor de webapplicatie. Hierin staan een aantal zaken geregeld voor autorisatie, sessionstate en authenticatie. Maar er kunnen er globale variabelen worden gedefinieerd. De database connectiestring is er hier één van, de ander is het pad naar de active directory. De web.config bevind zich in bijlage IV. In het codefragment wordt nu de connectie open gezet in regel 42, en een nieuw

SqlCommand object wordt aangemaakt met de naam command in regel 43. Er wordt meteen een sql commando meegegeven (“InsertBijlage”), en de connectie. Tevens wordt aangegeven dat het commando van het type stored procedure is. Hierna worden in regels 46 t/m 53 drie parameters, met een datatype, aan het SqlCommando toegevoegd, te weten @knummer, @verwijzing en @datum. Deze parameters komen overeen met de parameters van de stored procedure. De waarde van de parameters word gekoppeld aan de variabelen van de klasse, respectievelijk opdracht_id, verwijzing en datum. Het SqlCommand wordt nu uitgevoerd in regel 55. Meteen wordt de connectie weer gesloten en de objecten command en conn worden vernietigd. De methode verwijder is op dezelfde manier opgebouwd.

Een andere methode is “GetSpecificBijlage()”, met DataSet als return type. DataSet is een klasse klasse uit het .NET framework. De klasse bestaat in de namespace System.Data.SqlClient, vandaar dat de namespaces in het begin van de klasse zijn gedeclareerd. Was dit niet het geval geweest dan moest het return type aangegeven worden met System.Data.SqlClient.DataSet, maar omdat de namespace al is gedeclareerd kan worden volstaan met alleen DataSet. Het opzetten van de connectie naar de database en het toevoegen van de parameters aan het sqlcommando gaan op dezelfde manier als bij de methode “Toevoegen”. Echter gaat het uitvoeren van het commando anders. In regel 80 wordt een nieuw SqlDataAdapter object, met de naam da, aangemaakt en krijgt het object het sqlcommando mee. In regel 81 wordt het DataSet object ds aangemaakt wat ook uit de methode zal worden terug gegeven. In regel 82 wordt de Fill() functie van de SqlDataAdapter aangeroepen welke de DataSet vult met de gegevens uit de database onder de naam “Table”. Hierna word de connectie gesloten en de gebruikte objecten vernietigd. Als laatste in regel 91 wordt het DataSet object ds terug gegeven. Waarom er bij deze functie voor een DataSet is gekozen zal later worden uitgelegd.

De laatste methode van deze klasse is de methode “CountBijlagen()” met een integer als return type. De opbouw is vergelijkbaar met die van de methoden “Toevoegen()” en “Verwijderen()”. Echter heeft de stored procedure “CountBijlagen” uitvoer parameters. In regel 137 wordt aangegeven dat de richting van de stored procedure uitvoer is. In regel 148 wordt dan de waarde van de parameter naar een getal geconverteerd om uit de methode terug te worden gegeven. Dit concludeert de bespreking van de klasse Bijlage. Er zal met de bespreking van de stored procedures worden vervolgd.

De eerste procedure heet “InsertBijlage” en wordt gebruikt door de methode “Toevoegen” uit de klasse Bijlage. Het betreft hier een standaard insert statement.

```
ALTER PROCEDURE dbo.InsertBijlage
(
	@knummer int,
	@verwijzing varchar (255),
	@datum varchar(10)
)
AS
	INSERT INTO Bijlage (Opdracht_id, Verwijzing, Datum)
	VALUES (@knummer, @verwijzing, @datum)
	RETURN
```

De tweede procedure heet "SelectSpecificBijlage" en wordt gebruikt door de methode "GetSpecificBijlage" uit de klasse Bijlage. Het betreft hier een standaard select statement.

```
ALTER PROCEDURE dbo.SelectSpecificBijlage
(
	@knummer int
)
AS
	SELECT  Bijlage_id, Verwijzing, Datum
	FROM Bijlage WHERE Opdracht_id = @knummer
	RETURN
```

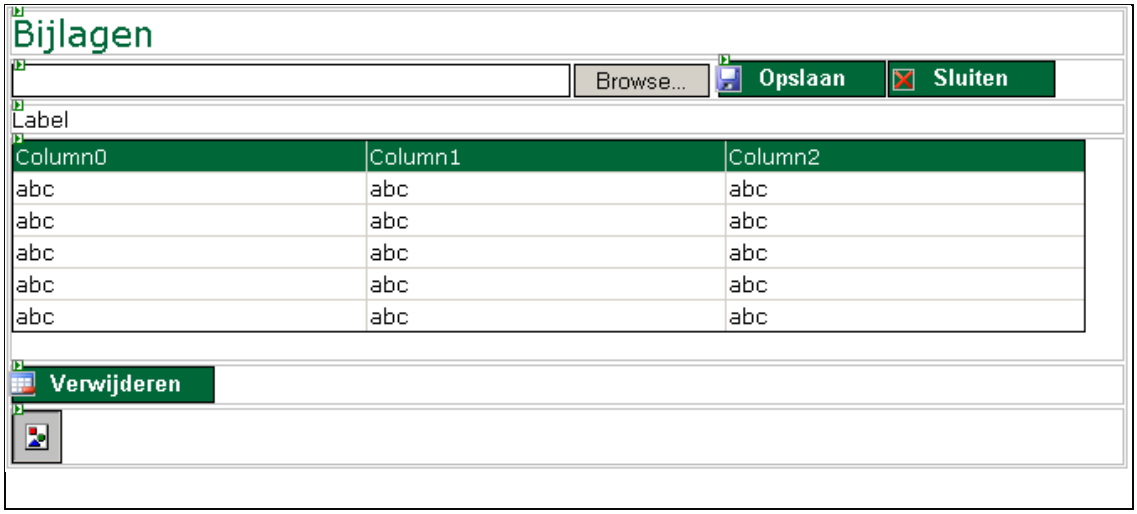
De volgende procedure heet "DeleteBijlage" en wordt gebruikt door de methode "Verwijder" uit de klasse Bijlage. Het betreft hier een standaard delete statement.

```
ALTER PROCEDURE dbo.DeleteBijlage
(
	@bijlagenr int
)
AS
	DELETE FROM bijlage WHERE bijlage_id = @bijlagenr
	RETURN
```

De laatste procedure heet "CountBijlage" en wordt gebruikt door de methode "CountBijlage" uit de klasse Bijlage. Het betreft hier een select statement van een telling van alle bijlagen met een specifieke opdracht_id.

```
ALTER PROCEDURE dbo.CountBijlagen
(
	@opdracht_id int,
	@aantal_bijlagen int OUTPUT
)
AS
	SELECT @aantal_bijlagen = COUNT(bijlage.bijlage_id)
	FROM bijlage
	WHERE bijlage.opdracht_id = @opdracht_id
	RETURN
```

Vervolgens zal de interface worden besproken. De interface kan op twee manieren worden bekeken, de eerste is ontwerp modus, de tweede is HTML modus. In de ontwerp modus ziet de interface er als volgt uit.



In deze interface (en bijna alle anderen) is een tabel gebruikt voor het uitlijnen van de elementen. Deze tabel is te onderscheiden aan de licht grijze lijnen. In dit geval is er één kolom met zes rijen gebruikt. Verder zijn er een aantal elementen te zien. Deze elementen zijn op te delen in twee groepen. Namelijk de server-side en de cliënt-side controls. De server-side controls zijn aangegeven met het symbool. In deze interface zijn dat er zeven, te weten:

- Een labeltype voor de titel.
- Een inputtype met file upload invoervak en knop.
- Een buttontype voor het opslaan van een bijlage.
- Een label voor meldingen aan de gebruiker.
- Een datagridtype voor het laten zien van database uitvoer.
- Een buttontype voor het verwijderen van een bijlage.
- Een imagetype voor het laten zien van een bijlage.

Naast de server-side controls zijn er vier cliënt-side controls te zien, te weten:

- Een img type voor het opslaan plaatje.
- Een img type voor het sluiten plaatje.
- Een inputtype voor het sluiten van de pagina.
- Een img type voor het verwijderen plaatje.

Verdere verduidelijking zal plaatsvinden aan de hand van de HTML modus.

```
1 <%@ Page language="c#" Codebehind="Bijlagen.aspx.cs"
AutoEventWireup="false" Debug="false" Inherits="CTEWP.Bijlagen"%>
2 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
3 <HTML>
4     <HEAD>
5           <title>CT EWP - Bijlagen</title>
6           <LINK href="stylesheet/stylesheet.css" type="text/css"
7                 rel="stylesheet"></LINK>
8           <script language="javascript"
9                 src="Javascript/DisableRightClick.js"></script>
10          <script language="javascript"
11                src="Javascript/SelectDatagridRow.js"></script>
12          <script language="javascript"
13                src="Javascript/DisableKeys.js"></script>
14          <script language="jscript" event="onclick" for="BijlagenDGD">
15                selectRow( "BijlagenDGD" );
16          </script>
17          <script>
18                function CloseWindow()
19                {
20                      window.opener.__doPostBack('LinkButton1', '');
21                      window.close();
22                }
23          </script>
24    </HEAD>
25    <body>
26          <form id="Form1" method="post" encType="multipart/form-data"
27                runat="server">
28                <table>
29                      <tr>
30                            <td>
31                                  <asp:label id="Label1"
32                                  runat="server"
33                                  CssClass="bigtextlabel">
34                                        Bijlagen
35                                  </asp:label>
36                            </td>
37                      </tr>
```

```
38                      <tr>
39                            <td>
40                                  <INPUT id="FilUpload"
41                                  type="file" size="50"
42                                  name="File1" runat="server">
43                                  <asp:button id="UploadBTN"
44                                  runat="server"
45                                  CssClass="button"
46                                  Text="Opslaan"
47                                  Width="100px"></asp:button>
48                                  <INPUT id="Sluiten"
49                                  style="WIDTH: 100px; COLOR:
50                                  #ffffff; BACKGROUND-COLOR:
51                                  #006738"
52                                  onclick="CloseWindow()"
53                                  type="button" value="Sluiten">
54                                  <IMG style="LEFT: -102px;
55                                  POSITION: relative; TOP: 0px"
56                                  src="./images/Sluiten.gif">
57                                  <IMG style="LEFT: -226px;
58                                  POSITION: relative; TOP: 1px"
59                                  src="./images/Opslaan.gif">
60                            </td>
61                      </tr>
62                      <tr>
63                            <td>
64                                  <asp:label id="OutputLBL"
65                                  runat="server" Visible="False">
66                                        Label
67                                  </asp:label>
68                            </td>
69                      </tr>
70                      <tr>
71                            <td>
72                                  <asp:datagrid id="BijlagenDGD"
73                                  runat="server" Width="640px"
74                                  CssClass="datagrid">
75                                        <SelectedItemStyle
76                                        CssClass="datagrid
77                                        selecteditem">
78                                        </SelectedItemStyle>
79                                        <ItemStyle
80                                        CssClass="datagriditem">
81                                        </ItemStyle>
82                                        <HeaderStyle
83                                        CssClass="datagridheader">
84                                        </HeaderStyle>
85                                        <Columns>
86                                              <asp:ButtonColumn
87                                              Visible="False"
88                                              Text="Select"
89                                              CommandName="Select
90                                              ">
91                                              </asp:ButtonColumn>
92                                        </Columns>
93                                  </asp:datagrid>
94                            </td>
95                      </tr>
96                      <tr>
97                            <td>
98                                  <asp:button id="VerwijderenBTN"
99                                  runat="server" CssClass="button"
100                                 Text=" Verwijderen"></asp:button>
```

```
101                                         <IMG style="LEFT: -124px;
102                                         POSITION: relative; TOP: 0px"
103                                         src="./images/Verwijderen.gif">
104                                 </td>
105                         </tr>
106                         <tr>
107                                 <td>
108                                         <asp:image id="PictureIMG"
109                                         runat="server" CssClass="textbox"
110                                         BackColor="Silver">
111                                         </asp:image>
112                                 </td>
113                         </tr>
114                 </table>
115         </form>
116     </body>
117 </HTML>
```

In regel nummer 1 worden een aantal zaken vastgelegd met betrekking tot de configuratie van de pagina. Er wordt vastgelegd dat de standaard script taal C# is, dat de code-behind pagina "bijlagen.aspx.cs" is, dat events niet automatisch worden gekoppeld, dat er niet wordt gedebuged, en dat er wordt geërfd van de klasse CTEWP.Bijlagen (dit is niet de klasse Bijlage die eerder is beschreven maar de interfaceklasse Bijlagen die later zal worden beschreven). Regel 2 is wat algemene informatie over het type document. Regels 4 t/m 24 geeft de kop van de HTML pagina. In deze kop wordt de titel gedefinieerd, wordt een link gemaakt naar een stylesheet en wordt javascript code geplaatst. Een stylesheet is een extern bestand waarin generieke opmaak kan worden gedefinieerd. Op deze manier kan iedere pagina verwijzen naar deze opmaak en hoeft niet voor iedere pagina de opmaak telkens opnieuw worden vastgelegd. Javascript is cliënt-side code. Dit houdt in dat de code niet op de server wordt uitgevoerd maar op de cliënt. Aan deze pagina zijn drie externe javascripts gekoppeld. Omdat deze drie in bijna alle pagina's worden gebruikt is het handig als aan deze scripts kan worden gerefereerd. Het eerste script zorgt ervoor dat er niets gebeurt als er op de rechtermuisknop wordt gedrukt in een scherm. Dit is nodig om het standaard rechtermuisknop scherm van Internet Explorer te verbergen. Het tweede script zorgt ervoor dat er op een willekeurige kolom in het datagrid element kan worden geklikt, en dat dan de gehele rij wordt geselecteerd. Dit is namelijk standaard niet het geval. Het laatste script maakt een aantal standaard knoppen in Internet Explorer zoals bijvoorbeeld F3 (zoeken), F4 (adresbalk), F5 (refresh), etc niet toegankelijk. Het eerste en het derde script zijn gemaakt om de omgeving van de applicatie robuuster te maken. Hoe minder opties fouten kunnen veroorzaken, hoe soepeler de applicatie zal werken. In de regels 14 t/m 16 wordt het mogelijk gemaakt uit een specifiek datagrid een volledige rij te selecteren. In de regels 17 t/m 23 staat de code voor de sluiten knop. Let op, dit is niet het kruisje rechtsboven maar de sluiten knop in de interface zelf. In de code wordt bepaald dat "LinkButton1" uit het openende scherm (K-Formulier) moet worden geactiveerd en dat het huidige scherm moet worden gesloten. Tussen regels 25 t/m 117 staat wat de gebruiker op zijn scherm zal zien.
In regel 26 is als eerste een Form aangemaakt. Een form is de basis waarop alle elementen en controls worden geplaats. In regel 28 wordt de tabel geopend die ook in de ontwerp modus te zien was. De daaropvolgende `<tr></tr>` tags omvatten een rij en de `<td></td>` tags omvatten een cel. Vervolgens word in regel 31 het eerste element van het type `<asp:label>` aangemaakt. Dit label heeft de id "Label1", de CssClass "bigtextlabel" en wordt op de server uitgevoerd. De `id` is in het geval van dit element niet zo belangrijk omdat er niet verder met het element

zal worden gewerkt. Het volgende element wat wordt gedefinieerd is van het type input/file. Dit element maakt het browsen van de schijf van de gebruiker mogelijk. Ook dit element draait op de server zodat deze in de code kan worden benaderd. In dezelfde tabel rij zijn twee knoppen gedefinieerd. Eén van het type `<asp:button>` en één van het type `<INPUT type="button">`. De eerste is een server-side knop de tweede een cliënt-side knop. Er is voor gekozen om waar het kan cliënt-side knoppen te gebruiken. Een server-side knop zal namelijk altijd een interactie met de server genereren, terwijl een cliënt-side knop dit niet doet. Het gebruik van zo veel mogelijk cliënt-side knoppen resulteert dus in minder belasting voor de server. Cliënt-side knoppen zijn echter alleen nuttig voor het openen en sluiten van vensters. Hiervoor wordt namelijk javascript gebruikt wat een cliënt-side scripting taal is. De volgende twee declaraties zijn die van plaatjes door middel van de `<IMG>` tag als icoontjes voor de knoppen. Door middel van het `style` attribuut wordt een relatieve positie aan de plaatjes gegeven waardoor ze over de knoppen heen komen te liggen. In de volgende rij van de tabel wordt een weer een `<asp:label>` gedefinieerd waarmee berichten aan de gebruiker worden getoond. De volgende tabel rij bevat een element van het type `<asp:datagrid>`. Een datagrid is een element om niet alleen database uitvoer te laten zien, maar om deze ook gelijk te verwijderen, te wijzigen en te verwijderen. Het biedt vele mogelijkheden en is volledig aan te passen aan de wensen van de programmeur. In het datagrid is maar één kolom gedefinieerd, dit is een kolom voor het selecteren van een rij. De andere kolommen worden automatisch gegenereerd aan de hand van de database uitvoer. De volgende rij bevat een knop voor het verwijderen van bijlagen, met weer een bijbehorend icoon. De laatste rij bevat een server-side plaatjes element, waar de voorgaande gedeclareerde plaatjes niet server-side waren. Dit plaatjes element is voor het laten zien van de bijlagen.

Aan de hand van het volgende code fragment zal de werking achter de interface worden uitgelegd.

```
1 using System;
2 using System.Collections;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Web;
7 using System.Web.SessionState;
8 using System.Web.UI;
9 using System.Web.UI.WebControls;
10 using System.Web.UI.HtmlControls;
11 using CTEWP;
12 using System.IO;
13
14 namespace CTEWP
15 {
16     /// <summary>
17     /// Summary description for Bijlagen.
18     /// </summary>
19     public class Bijlagen : System.Web.UI.Page
20     {
21           protected System.Web.UI.WebControls.DataGrid BijlagenDGD;
22           protected System.Web.UI.WebControls.Image PictureIMG;
23           protected System.Web.UI.WebControls.Button UploadBTN;
24           protected System.Web.UI.HtmlControls.HtmlInputFile FilUpload;
25           protected System.Web.UI.WebControls.Label OutputLBL;
26           protected System.Web.UI.WebControls.Button VerwijderenBTN;
27           protected System.Web.UI.WebControls.Label Label1;
```

```
28          protected Opdracht opd = new Opdracht();
29          protected Bijlage blg = new Bijlage();
30
31          private void Page_Load(object sender, System.EventArgs e)
32          {
33                  opd = (Opdracht)this.Session["opdracht"];
34
35                  blg.opdracht_id = opd.opdracht_id;
36                  BijlagenDGD.DataSource = blg.GetSpecificBijlage();
37                  BijlagenDGD.DataBind();
38          }
39
40          #region Web Form Designer generated code
41          override protected void OnInit(EventArgs e)
42          {
43                  //
44                  // CODEGEN: This call is required by the ASP.NET Web
45                  // Form Designer.
46                  //
47                  InitializeComponent();
48                  base.OnInit(e);
49          }
50
51          /// <summary>
52          /// Required method for Designer support - do not modify
53          /// the contents of this method with the code editor.
54          /// </summary>
55          private void InitializeComponent()
56          {
57                  this.UploadBTN.Click += new
    System.EventHandler(this.UploadBTN_Click);
58                  this.BijlagenDGD.ItemDataBound += new
    System.Web.UI.WebControls.DataGridItemEventHandler(this.DGDDataBind);
59                  this.BijlagenDGD.SelectedIndexChanged += new
    System.EventHandler(this.IndexChanged);
60                  this.VerwijderenBTN.Click += new
    System.EventHandler(this.VerwijderenBTN_Click);
61                  this.Load += new System.EventHandler(this.Page_Load);
62          }
63          #endregion
64
65          private void DGDDataBind(Object sender, DataGridItemEventArgs
    e)
66          {
67                  if (e.Item.ItemType == ListItemType.Header)
68                  {
69                          e.Item.Cells[2].Font.Bold = true;
70                          e.Item.Cells[3].Font.Bold = true;
71                  }
72
73                  e.Item.Cells[1].Visible = false;
74                  e.Item.Cells[2].Width = 550;
75                  e.Item.Cells[3].Width = 100;
76          }
77
78          public void IndexChanged(object sender, EventArgs e)
79          {
80                  PictureIMG.ImageUrl =
    BijlagenDGD.SelectedItem.Cells[2].Text;
81          }
82
83          public void UploadBTN_Click(object sender, EventArgs e)
84          {
85                  string sSavePath;
```

```
86                  sSavePath = "./bijlagen/" + opd.opdracht_id + "/";
87
88                  if (!Directory.Exists(sSavePath))
89                  {
90                        DirectoryInfo di =
      Directory.CreateDirectory(sSavePath);
91                  }
92
93                  // If file field isn't empty
94                  if (FilUpload.PostedFile != null)
95                  {
96                        // Check file size (mustn't be 0)
97                        HttpPostedFile myFile = FilUpload.PostedFile;
98                        int nFileLen = myFile.ContentLength;
99                        if (nFileLen == 0)
100                       {
101                             OutputLBL.Text = "No file was uploaded.";
102                             OutputLBL.Visible = true;
103                             return;
104                       }
105
106                       // Check file extension (must be JPG)
107                       if
      (System.IO.Path.GetExtension(myFile.FileName).ToLower() != ".jpg" &&
      System.IO.Path.GetExtension(myFile.FileName).ToLower() != ".jpeg" &&
      System.IO.Path.GetExtension(myFile.FileName).ToLower() != ".gif")
108                       {
109                             OutputLBL.Text = "Het bestand moet een
      .jpg, .jpeg of .gif extensie hebben.";
110                             OutputLBL.Visible = true;
111                             return;
112                       }
113
114                       // Read file into a data stream
115                       byte[] myData = new Byte[nFileLen];
116                       myFile.InputStream.Read(myData,0,nFileLen);
117
118                       // Make sure a duplicate file doesn't exist.  If
119                       // it does, keep on appending an incremental
120                       // numeric until it is unique
121                       string sFilename =
      System.IO.Path.GetFileName(myFile.FileName);
122                       int file_append = 0;
123                       while
      (System.IO.File.Exists(Server.MapPath(sSavePath + sFilename)))
124                       {
125                             file_append++;
126                             sFilename =
      System.IO.Path.GetFileNameWithoutExtension(myFile.FileName)
      + "-" + file_append.ToString() + ".jpg";
127                       }
128
129                       // Save the stream to disk
130                       System.IO.FileStream newFile = new
      System.IO.FileStream(Server.MapPath(sSavePath + sFilename),
      System.IO.FileMode.Create);
131                       newFile.Write(myData,0, myData.Length);
132                       newFile.Close();
133
134                       // Check whether the file is really a JPEG by
135                       // opening it
136
137                       try
138                       {
```

```
139                                 Bitmap myBitmap;
140                                 myBitmap = new
      Bitmap(Server.MapPath(sSavePath + sFilename));
141
142                                 blg.verwijzing = sSavePath + sFilename;
143                                 blg.datum =
      Convert.ToString(DateTime.Today.Day) + "-" +
      Convert.ToString(DateTime.Today.Month) + "-" +
      Convert.ToString(DateTime.Today.Year);
144                                 blg.Toevoegen();
145
146                                 OutputLBL.Text = "Bestand succesvol
      opgeslagen.";
147                                 OutputLBL.Visible = true;
148                                 myBitmap.Dispose();
149                           }
150                           catch (ArgumentException errArgument)
151                           {
152                                 // The file wasn't a valid jpg file
153                                 OutputLBL.Text = "Het bestand is geen
      geldig .jpg, .jpeg of .gif bestand";
154                                 OutputLBL.Visible = true;
155                                 System.IO.File.Delete(Server.MapPath
      (sSavePath + sFilename));
156                           }
157                     }
158                     BijlagenDGD.DataSource = blg.GetSpecificBijlage();
159                     BijlagenDGD.DataBind();
160               }
161
162               public void VerwijderenBTN_Click(object sender, EventArgs e)
163               {
164                     if (BijlagenDGD.SelectedIndex != -1)
165                     {
166                           string temp =
      BijlagenDGD.SelectedItem.Cells[2].Text.TrimStart('.');
167                           blg.bijlage_id =
      Convert.ToInt32(BijlagenDGD.SelectedItem.Cells[1].Text);
168                           blg.Verwijder();
169                           System.IO.File.Delete("c:/inetpub/wwwroot/CTEWP"
      + temp);
170                           // Create a reference to the current directory.
171                           DirectoryInfo di = new
      DirectoryInfo("c:/inetpub/wwwroot/CTEWP/bijlagen/" +
      opd.opdracht_id);
172                           // Create an array representing the files in the
173                           // current directory.
174                           FileInfo[] fi = di.GetFiles();
175                           if (fi.Length == 0)
176                           {
177     System.IO.Directory.Delete("c:/inetpub/wwwroot/CTEWP/bijlagen/" +
      opd.opdracht_id);
178                           }
179
180                           BijlagenDGD.SelectedIndex = -1;
181                           BijlagenDGD.DataSource =
      blg.GetSpecificBijlage();
182                           BijlagenDGD.DataBind();
183
184                           PictureIMG.ImageUrl = "";
185                     }
186               }
187     }
188 }
```

Regel 1 t/m 14 geeft weer de namespace declaraties. Regel 19 declareert de naam van de klasse. `Bijlagen : System.Web.UI.Page` betekend dat de klasse Bijlagen een subklasse van de klasse Page is, en dat dus de attributen en methode van de klasse Page worden geërfd. In regel 21 t/m 29 worden de variabelen gedeclareerd. Merk op dat de server-side elementen die in de interface zijn gedeclareerd hier weer terug komen. De labels, buttons en het datagrid worden dus als variabelen gedeclareerd. Regel 31 t/m 38 geeft aan wat er moet gebeuren als de pagina wordt geladen. Regels 40 t/m 63 bevatten automatisch gegenereerde code waarbij functies aan een aantal van de events van de variabelen worden gekoppeld. Er staat hier een waarschuwing niet de inhoud van de functie te `InitializeComponent()` wijzigen. Echter is dit wel gedaan met regel 58. Hier wordt een functie gekoppeld aan het creëren van een regel in het datagrid. Deze functie is gecodeerd in de regels 65 t/m 76. In deze functie wordt de tekst in de header (kop) van het datagrid vet gemaakt, wordt een kolom op onzichtbaar gezet en wordt de breedte van de overige kolommen aangepast. Dit moet dynamisch worden gedaan omdat de kolomen ook dynamisch worden gegenereerd aan de hand van de database output. In de regels 78 t/m 81 wordt de functie `IndexChanged()` gecodeerd. Deze functie wordt aangeroepen als de geselecteerde rij uit het datagrid veranderd. Met andere woorden, als een gebruiker een andere rij aanklikt. In de functie wordt het ImageUrl attribuut van PictureIMG (een image control) gezet op een waarde uit de database (via het datagrid) zodat de geselecteerde bijlage wordt getoond. Regels 83 t/m 160 bevatten de functie voor het uploaden van een bijlage naar de server. Er wordt een nieuwe directory aangemaakt per K-formulier als deze nog niet bestaat. Er wordt gekeken of er een file is opgegeven en of deze file wel een .jpg, .jpeg, of .gif extensie heeft. De file wordt ingelezen en er wordt bepaald of de file al bestaat. Zo ja, dan wordt er een nummer toegevoegd. Een verwijzing naar de file wordt in de database opgenomen, de volledige file in de database opnemen zou de database explosief doen groeien. De file zelf wordt weggeschreven en weer geopend om te controleren of alles goed is gegaan. Als laatste wordt het datagrid opnieuw gegenereerd aan de hand van de functie `blg.GetSpecificBijlage()`. De uitkomst van deze functie is van het type DataSet. DataSets kunnen aan datagrids worden gekoppeld, vandaar dat dit type als uitvoer is gekozen. Regels 162 t/m 186 bevatten de functie `VerwijderenBTN_Click()` welke wordt geactiveerd als op de knop "Verwijderen" wordt geklikt. In deze functie wordt het bestand van de schijf verwijderd en wordt de verwijzing uit de database verwijderd. Betreft het de laatste file in de directory dan wordt meteen de volledige directory verwijderd. Ten slotte wordt het datagrid weer opnieuw gegenereerd. Dit beëindigd de bespreking van de klasse bijlage, de stored procedures DeleteBijlage, CountBijlage, InsertBijlage en SelectSpecificBijlage, de interface en de interface klasse Bijlagen.

Verder zijn in de pilot de volgende problemen opgetreden en opgelost.

- Als een actie naar de server (postback) op treed, dan wordt de pagina opnieuw geladen en de scroll-positie van het scherm niet overgenomen. Denk er bijvoorbeeld aan als iemand ergens onderaan een pagina op een knop klikt en de pagina schiet weer naar boven, dit kan hoogst irritant zijn.
    - De oplossing hiervoor is gevonden in een gratis control van een derde partij genaamd StrengthControls. De control is gecompileerd in een .dll bestand wat kan worden geïnclude in een interface bestand. De control maakt het mogelijk de scroll-positie van zowel de X als Y-as te behouden tussen postbacks.

- Een ander punt is cliënt-side printen. Omdat de taakbalken van internet explorer worden weggelaten in de applicatie om deze robuuster te maken, zijn de menu's bestand, bewerken, beeld, favorieten, etc. niet beschikbaar. Hierdoor is ook de print functie niet meer bereikbaar. Een oplossing was nodig waarbij gebruikers op een knop in de applicatie klikken en dan het print dialoog werd geopend.
  - Deze oplossing is gevonden in een ActiveX component van MeadCo. Dit component komt in een .cab (cabinet) bestand en is te includen in een interface bestand. Dit .cab bestand moet van de server af worden gedownload als de interface pagina wordt geopend. Het component maakt het mogelijk de header/footer van de pagina in te stellen, landscape/portrait in te stellen, automatisch te laten printen maar ook het print dialoog te laten zien.

- Een probleem wat is opgetreden is het verlies van sessievariabelen. Een sessie wordt aangemaakt als een gebruiker inlogt op de applicatie. Het bleek dat, na verloop van tijd, de waarden van variabelen die in de sessiestate waren geplaatst verdwenen waren. Dit probleem bleek te komen door de garbadge collection van het ASP.Net proces. Iedere keer als garbadge collection werd uitgevoerd waren de sessievariabelen verdwenen.
  - Na onderzoek bleek dat er drie mogelijkheden zijn om de sessiestate te onderhouden. 1) in_proc (standaard), dit houdt in dat de sessiestate in het ASP.Net proces draait. 2) stateserver, dit houdt in dat de sessiestate in een apart sessieproces draait. 3) Sql server, dit houdt in dat de sessiestate in Sql server wordt opgeslagen. Iedere methode heeft zijn voor en nadelen. Zo is de eerste methode het snelst omdat het in het zelfde proces als ASP.Net draait, nadeel is verlies van sessiedata. Methode twee is nog altijd zeer snel omdat het sessieproces ook in het geheugen van de server aanwezig is. Een nadeel is dat er een extra proces op de server draait. De derde optie is langzamer ten opzichte van de eerste twee omdat er interactie moet plaatsvinden met de Sql server. Echter zullen de sessievariabelen nooit zomaar verdwijnen. Er is gekozen voor de tweede optie omdat het belangrijk is dat de sessievariabelen worden behouden voor zolang als de sessie duurt, en interactie met een Sql server te langzaam zou zijn. Om deze optie mogelijk te maken moet de "ASP.NET State" service worden gestart. Deze service is te vinden in configuratiescherm -> systeembeheer -> services, als het ASP.net framework is geïnstalleerd. In de configuratie file van de applicatie moet dan nog wel worden aangegeven dat de stateserver wordt gebruikt.

4.7 Bouw conversietools pilot.

Omdat er data uit andere systemen met elk hun eigen database wordt gehaald, is het nodig geweest conversies uit te voeren. De systemen waar data uit gehaald moet worden zijn:

- Scope, CRM pakket.
- Multivers, Artikelen/klant administratie pakket.
- Een MS Excel werkblad, reparatieonderdelen gegevens.

Scope is een pakket wat werkt onder een Oracle database. Uit dit systemen moeten de kortingen per klant worden gehaald. Echter is de Oracle database meegeleverd met het Scope pakket, en is de database zo geïnstalleerd dat niemand, behalve een medewerker van Scope, de database kan benaderen met behulp van een database tool. Dit is echter wel logisch omdat de database engine bij het software pakket is inbegrepen, en het niet de bedoeling is dat hier verdere databases op gaan draaien. Dat zou niet helemaal kloppen met licenties en dergelijke. Was het wel mogelijk geweest de database te benaderen met behulp van een database tool, dan was het exporten van de data naar MS SQL Server waarschijnlijk vrij eenvoudig geweest.

Multivers is een pakket wat draait in een DOS omgeving met een (mij) onbekend type database. Gelukkig bied Multivers een tool waarmee data kan worden geëxporteerd naar een aantal bestandsformaten. Deze tool heet MuSQLe. De voormalige systeembeheerder bij CT is zo vriendelijk geweest om voor mij een handleiding te maken over de werking van MuSQLe. Met behulp van deze handleiding ben ik tot de volgende export functies gekomen.

Repart.sql

```
CONNECT '094';
SELECT
    ARTIKEL.ART 'ART'
   ,ARTIKEL.OMSCHR1 'OMSCHR1'
   ,ARTIKEL.OMSCHR2 'OMSCHR2'
   ,ARTIKEL.ZOEKNAAM 'ZOEKNAAM'
FROM
    ARTIKEL, ARTVOORRAAD
WHERE
    ARTVOORRAAD.MAG = 'C'
    AND ARTVOORRAAD.ART = ARTIKEL.ART
    AND ARTIKEL.ART < '7&'

;
FORMAT SEP 1 BLANK
    COL   1 WIDTH  12
    COL   2 WIDTH  30
    COL   3 WIDTH  30
    COL   4 WIDTH  30
    INCL ONLY (   1   2   3  4)
;
TRANSFER '\\TESTSERVER\CTEWP\IMPORTS\REPART.DBF', 'DBF';
```

Repdeb.sql

```
CONNECT '094';
SELECT
    DEBITEUR.DEB 'DEB'
   ,DEBITEUR.NAAM 'NAAM'
   ,DEBITEUR.PERSOON 'PERSOON'
   ,DEBITEUR.STRAAT 'STRAAT'
   ,DEBITEUR.WOONPLAATS 'WOONPLAATS'
   ,DEBITEUR.POSTCODE 'POSTCODE'
   ,ALGLAND.OMSCHR 'OMSCHR'
   ,DEBITEUR.TELEFOON 'TELEFOON'
   ,DEBITEUR.TELEFAX 'TELEFAX'
FROM
    DEBITEUR
    LEFT JOIN (ALGLAND)
    ON  DEBITEUR.LAND          = ALGLAND.LAND
    AND ALGLAND.SOORT02       = '02'

;
FORMAT SEP 1 BLANK
    COL   1 WIDTH   8
    COL   2 WIDTH  30
    COL   3 WIDTH  30
    COL   4 WIDTH  30
    COL   5 WIDTH  30
    COL   6 WIDTH   7
    COL   7 WIDTH  30
    COL   8 WIDTH  15
    COL   9 WIDTH  15
    INCL ONLY (   1   2   3   4   5   6   7   8   9)
;
Transfer '\\TESTSERVER\CTEWP\IMPORTS\repdeb.DBF', 'DBF';
```

De bovenste regel verzorgt de connectie met de database. De syntax voor het selecteren van de gegevens komt overeen met de standaard SQL syntax. Het format blok geeft informatie over het formaat van de uitvoer. De onderste regel geeft het uitvoerformaat en pad aan, in dit geval is het een DBaseIV(.dbf) bestand wat op de locatie \\TESTSERVER\CTEWP\IMPORTS\repdeb.DBF. Er is voor het DBaseIV bestandsformaat gekozen omdat dit het makkelijkst te importeren bleek in MS SQL Server.

Met behulp van de volgende stukjes code zijn batchoperaties gemaakt waarmee MuSQLe scripts worden uitgevoerd. Deze batchoperaties kunnen met behulp van de taak planner van Windows automatisch worden uitgevoerd. Momenteel is ingesteld dat de batchoperaties iedere nacht plaatsvinden.

Repart.bat

```
ECHO OFF
CLS
O:
CD\
CD MULTIVER
CD WORK
..\musqle\musqlrun -u musqle -p unit4  "repart.SQL"
```

Repdeb.bat

```
ECHO OFF
CLS
O:
CD\
CD MULTIVER
CD WORK
..\musqle\musqlrun -u musqle -p unit4  "repdeb.SQL"
```

De uitvoer van deze batchoperaties is dus een .dbf bestand. Om de data uit deze bestanden in de MS SQL database te krijgen, moet de data worden geïmporteerd. Hiervoor heeft MS SQL de Import/Export wizard. Er moet worden aangegeven wat het type en de locatie van de te importeren gegevens zijn. In dit geval is dat een DBaseIV bestand op de locatie \\TESTSERVER\CTEWP\IMPORTS\. Er moet worden aangegeven wat de bestemming is van de te importeren gegevens. In dit geval is dat de MS SQL Server op localhost (de huidige machine). Als laatste moet er worden aangegeven welk dataveld uit de bron, naar welk dataveld in de bestemming moet worden gekopieerd. Deze wizards kunnen worden opgeslagen als Data Transformation Services (DTS) package. Op die manier hoeven niet iedere de wizards opnieuw worden ingesteld en kunnen ze worden ingesteld om automatisch uitgevoerd te worden. In dit geval worden ze automatisch uitgevoerd als “Jobs”. Naast deze twee jobs is er nog een job aangemaakt voor het automatisch maken van een back-up van de database en een job voor het “shrinken” van de database. Bij het shrinken van een database wordt alle gereserveerde ruimte voor data die niet is benut weer vrijgegeven, dit houdt de database klein en snel.

Als laatste wordt er met een Excel werkblad gewerkt waarin de onderdelen staan die gebruikt worden bij een reparatie met een beschrijving, een nummer en de prijs. Daarnaast staat er in dit werkblad een prijsindicatie voor reparaties per aandrijfcomponent type. Deze data kan niet zomaar uit de Excel werkbladen worden geïmporteerd. Hiervoor zal de data uit de werkbladen eerst naar een schoon werkblad moeten worden gekopieerd, waarna het kan worden opgeslagen als een .dbf bestand. Dit geld voor zowel de onderdelen als de indicatieprijzen, er moeten dus twee aparte dbf. Bestanden worden geïmporteerd. Een beschrijving van hoe dit moet zal in de systeemhandleiding worden opgenomen. Het maken van DTS packages gaat eender als in de voorgaande paragraaf is beschreven. Echter is er het verschil dat deze packages niet worden ingeroosterd voor automatische uitvoer. Voor deze packages zijn stored procedures gemaakt zodat de .dbf bestanden worden uitgelezen als de stored procedures worden uitgevoerd. Deze stored procedures zien er als volgt uit.

RunOnderdelenImport:

```
ALTER PROCEDURE dbo.RunOnderdelenImport
AS
EXEC    master..xp_cmdshell    'dtsrun    /Slocalhost    /Usa    /Ptuinhuis
/NOnderdelenImport'
```

RunStdRepPrijsImport:

```
ALTER PROCEDURE dbo.RunStdRepPrijsImport
AS
EXEC master..xp_cmdshell 'dtsrun /Slocalhost /Usa /Ptuinhuis
/NStdRepPrijsImport'
```

In het programma zijn, in het beheers gedeelte, twee knoppen gemaakt waarmee de stored procedures worden uitgevoerd en de data wordt ingevoerd.

In de toekomst, als alle tests zijn uitgevoerd en het programma is geaccepteerd, zal de data uit het oude systeem moeten worden geïmporteerd naar het nieuwe. Bij het converteren van MS Access naar MS SQL verwacht ik niet veel problemen. Het zal alleen een leuke puzzel zijn om uit te vinden welk dataveld uit het oude systeem naar welk dataveld in het nieuwe systeem moet worden overgezet.

4.8 Handmatige procedures pilot.

Scannen bijlagen
De pilot "K-formulieren" heeft één handmatige procedure: het scannen van de bijlagen die door een klant worden meegeleverd bij een aandrijf component. Hoe gaat deze handmatige procedure in zijn werk? De magazijn medewerker opent het scan programma. Hij stelt het formaat van de in te scannen papieren in op zwart-wit foto. De schaal moet op 50% worden gezet omdat anders de afbeelding te groot wordt. Hij plaatst het in te scannen papier in de scanner en klikt op de scan knop. Hij geeft een bestandformaat op, in dit geval jpeg en klikt op opslaan. Hierna zal hij de bijlagen in het systeem moeten opslaan. Dit doet hij door in een k-formulier op de bijlagen knop te klikken. In het volgende scherm moet hij de locatie van de bijlage opgeven en op opslaan klikken. De bijlage is opgenomen in het systeem en de handmatige procedure is beëindigd. Overigens komen alle beschrijvingen van handmatige procedures in de systeemhandleiding te staan.

4.9 Opleidingsmateriaal pilot.

Nog voor de afzonderlijke pilots, nog voor het volledige systeem zal opleidingsmateriaal worden verzorgd. De interface van het programma is namelijk zo opgezet dat het voor een gebruiker vrij eenvoudig te doorgronden is. Verdere toelichtingen zullen in de handleiding terug te vinden zijn. Wel zal er voor alle gebruikers een presentatie gehouden worden waarin de verschillende functies van het systeem worden uitgelegd.

4.10 Handleidingen pilot.

Voor de afzonderlijke pilots zullen geen handleidingen worden gemaakt omdat de pilots niet afzonderlijk worden vrijgegeven. De gebruikers zullen dus niet met de afzonderlijke pilots werken. De gebruikershandleiding wordt aan het eind van het project gemaakt, als alle pilots zijn samengevoegd tot één systeem. De technische handleidingen kunnen automatisch worden gegenereerd, door gebruik te maken van de mogelijkheden van de VisualStudio.Net ontwikkelomgeving. Ten tijde van het schrijven van dit verslag is echter nog niet alle code voorzien van commentaar.

4.11 Integratie bouweenheden.

Het integreren van de bouweenheden is ongemerkt tijdens het ontwikkelen van de bouweenheden gegaan. Zoals al is aangegeven zit er een zekere volgorde in het ontwikkelen van de bouweenheden. Het heeft bijvoorbeeld pas zin de database bewerkingscode in de klassen te bouwen als de bijbehorende tabellen in de database bestaan.

4.12 Beoordelen en testen pilot.

Het beoordelen en testen van de pilot is eigenlijk alleen tot het beoordelen gebleven. Het beoordelen van de pilot in de vorm van een vergadering uitgevoerd. Hierbij waren aanwezig de heren Bosman, van Genderen, Klijwegt en de afstudeerder. Tijdens deze sessies heeft de afstudeerder de pilot gepresenteerd, waarna er een overleg plaats vond. Hieruit werd telkens een lijst met actiepunten opgesteld. Deze actiepunten werden naar prioriteit georganiseerd en afgewerkt. Het testen van de pilot is gebeurd in een periode van ongeveer een maand wanneer alle pilots zijn samengevoegd.

5 Pilot 2: Communicatie

In het nu volgende hoofdstuk zal worden ingegaan op het ontwerp en de bouw van de pilot "Communicatie". Besproken zullen worden onder andere het systeemontwerp (klassediagram, sequentie diagrammen), het database ontwerp in de vorm van een ERD en een aantal interessante code fragmenten. Bedenk dat de besproken producten volgens een aantal iteratie slagen zijn verkregen.

5.1 Globaal-functionele structuur

De globaal functionele structuur van de pilots zal worden gedefinieerd aan de hand van het klassediagram uit UML. Het klassediagram is een statische weergave van de werkelijkheid. Objecten uit de werkelijkheid zijn gemodelleerd met hun variabelen en functies. Teneinde een klassediagram te maken zal eerst een lijstje van kandidaat klassen moeten worden opgesteld. Dit lijstje ziet er als volgt uit.

Kandidaat	**Beslissing**	**Kandidaat**	**Beslissing**
Bericht	Klasse	Fax	Type communicatie
Email	= bericht	Telefoon	Type communicatie
Intern	Type communicatie	Extern	Type communicatie
Communicatie	Te vaag	Mailbox	Interface
Communicatie notitie	Klasse	Taak	Klasse

Uit deze lijst van kandidaat klassen zijn drie kandidaten tot klassen gemaakt. De overige kandidaten zoals Fax, Telefoon, Intern en Extern zijn waarden. De kandidaat mailbox is een interface voor berichten. De kandidaat communicatie is te vaag.

Aan de hand van deze lijst is het klassediagram zoals in bijlage V opgesteld. In dit klassediagram zijn de nieuwe klassen, en de oude klassen opgenomen als deze een link hebben met de nieuwe klassen.

Het database ontwerp van de pilot, opgenomen in bijlage V, reflecteert het klassediagram. Het enige punt is dat de klasse communicatie van twee tabellen gebruik maakt, namelijk "communicatie_history" en "communicatie_history_type". In "communicatie_history" worden alle communicatie notities opgeslagen per opdracht. In de tabel "communicatie_history_type" worden alle soorten communicatie opgeslagen, zodat deze aan een communicatie notitie kunnen worden gekoppeld. De tabel is overigens gevuld met de kandidaat klassen die als "Type communicatie" staan bestempeld.

5.2 Globaal-technische structuur

De sequentie diagrammen die zijn gemaakt voor deze pilot gaan door in de trant van de diagrammen uit de voorgaande pilot. De sequentie diagrammen spreken vrij voor zich, dus is verdere uitleg hier overbodig. Voor een uitleg over de werking en notatie wijze van een sequentie diagram wil ik verwijzen naar het boek "Praktisch UML 2e editie" van Jos Warmer en Anneke Kleppe.

5.3 Globaal-organisatorische structuur

Zie paragraaf 4.3

5.4 Pilot ontwikkelplan

Bij het maken van het pilotplan is gebruik gemaakt van de planning als basis voor time-boxing. Voor het ontwikkelen van het totale systeem zijn vijftien weken uitgetrokken. Voor de pilot "Communicatie" zijn vier weken uitgetrokken. Bij de klassen zijn de stored procedures inbegrepen. Bij de interface zijn de interface klassen inbegrepen.

5.5 Ontwerp software bouweenheden.

Zie paragraaf 4.5

5.6 Bouw software bouweenheden.

De bouw van de software bouweenheden is op een zelfde manier gegaan als de pilot "K-formulieren". Bij de bouw zijn geen bijzonderheden opgetreden. Het bouwen ging bijzonder vlot, en er is zelfs tijdswinst mee geboekt. Waar een aantal weken waren ingepland, zijn een paar dagen nodig geweest. Dit is mede te danken aan de ervaringen die met de vorige pilot zijn opgedaan en de steeds groter wordende vaardigheid in de .NET ontwikkelomgeving.

5.7 Bouw conversietools pilot.

Voor deze pilot zijn geen conversietools nodig geweest.

5.8 Handmatige procedures pilot.

Externe email
Als er communicatie is geweest met de klant via email over een bepaalde reparatieopdracht, en de klant stuurt een email terug, dan moet de medewerker handmatig de inhoud van de email kopiëren naar een externe email communicatie.

5.9 Opleidingsmateriaal pilot.

Zie paragraaf 4.9

5.10 Handleidingen pilot.

Zie paragraaf 4.10

5.11 Integratie bouweenheden.

Zie paragraaf 4.11

5.12 Beoordelen en testen pilots.

Zie paragraaf 4.12

6 Pilot 3: Planning en overzichten

In het nu volgende hoofdstuk zal worden ingegaan op het ontwerp en de bouw van de pilot "Planning en overzichten". Besproken zullen worden onder andere het systeemontwerp (klassediagram, sequentie diagrammen), het database ontwerp in de vorm van een ERD en een aantal interessante code fragmenten. Bedenk dat de besproken producten volgens een aantal iteratie slagen zijn verkregen.

6.1 Globaal-functionele structuur

Voor deze pilot is geen lijstje met trefwoorden gemaakt zoals in de voorgaande pilots. Er is voor gekozen om voor "Overzicht" geen nieuwe klasse te maken. Er is voor gekozen om de functionaliteiten voor overzichten de klasse "Tool" te stoppen. Er is een functie om eigengemaakte query's uit te voeren. Deze functie is in de klasse "Tool" gestopt. Omdat verder voor de overzichten maar een klein aantal functies nodig zijn, en geen attributen, is er gekozen om hiervoor niet een nieuwe klasse aan te maken.

In de database is een nieuwe tabel aangemaakt, de tabel overzichten. Omdat er verder geen wijzigingen/toevoegingen zijn, wordt hier niet verder op in gegaan.

6.2 Globaal-technische structuur

Zie paragraaf 5.2

6.3 Globaal-organisatorische structuur

Zie paragraaf 4.3

6.4 Pilot ontwikkelplan

Bij het maken van het pilotplan is gebruik gemaakt van de planning als basis voor time-boxing. Voor het ontwikkelen van het totale systeem zijn vijftien weken uitgetrokken. Voor de pilot "Planning en overzichten" zijn drie weken uitgetrokken. Bij de klassen zijn de stored procedures inbegrepen. Bij de interface zijn de interface klassen inbegrepen.

6.5 Ontwerp software bouweenheden.

Zie paragraaf 4.5

6.6 Bouw software bouweenheden.

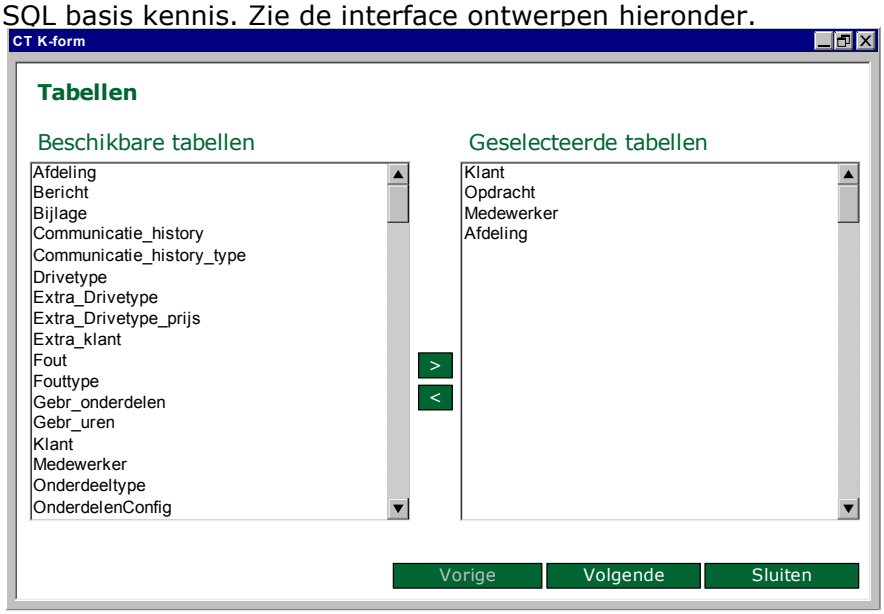
De bouw van de software bouweenheden is op een zelfde manier gegaan als de voorgaande pilots. De problemen waar tegenaan is gelopen zijn de volgende.

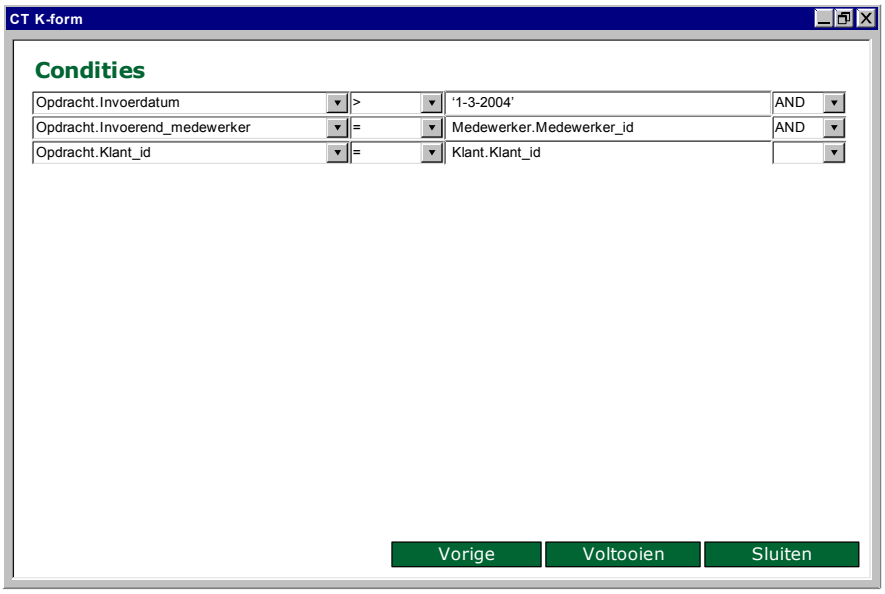
- Als gebruikers een overzicht aanmaken, moet dit overzicht kunnen worden opgeslagen voor later gebruik. Althans niet het overzicht zelf moet worden

opgeslagen maar de query waarmee het overzicht uit de database wordt gehaald. Nu kan de query in de database worden opgeslagen, maar beter is er een stored procedure van te maken.

- Na een onderzoek is gebleken dat het mogelijk is om dynamisch stored procedures aan te maken. In feite gaat dit op dezelfde manier als het maken van stored procedures met de enterprise manager van SQL Server. De stored procedures worden aangemaakt met het "create procedure" statement. Bijvoorbeeld: "CREATE PROCEDURE <procedure naam> AS <query>". Door middel van dit statement is het gemakkelijk om programmatisch stored procedures aan te maken.

- Een ander probleem wat is opgetreden is dat de query builder die ik voor ogen had niet werkte. De opzet was een soort "Wizard" systeem. Echter bleek dit systeem niet gebruiksvriendelijk. De gebruiker moest weten in welke tabel welke data staat, en de gebruiker moest handmatig de tabellen aan elkaar knopen (de primaire en secundaire sleutel verwijzingen). Hiervoor is kennis van de onderliggende database nodig evenals een zekere SQL basis kennis. Zie de interface ontwerpen hieronder.

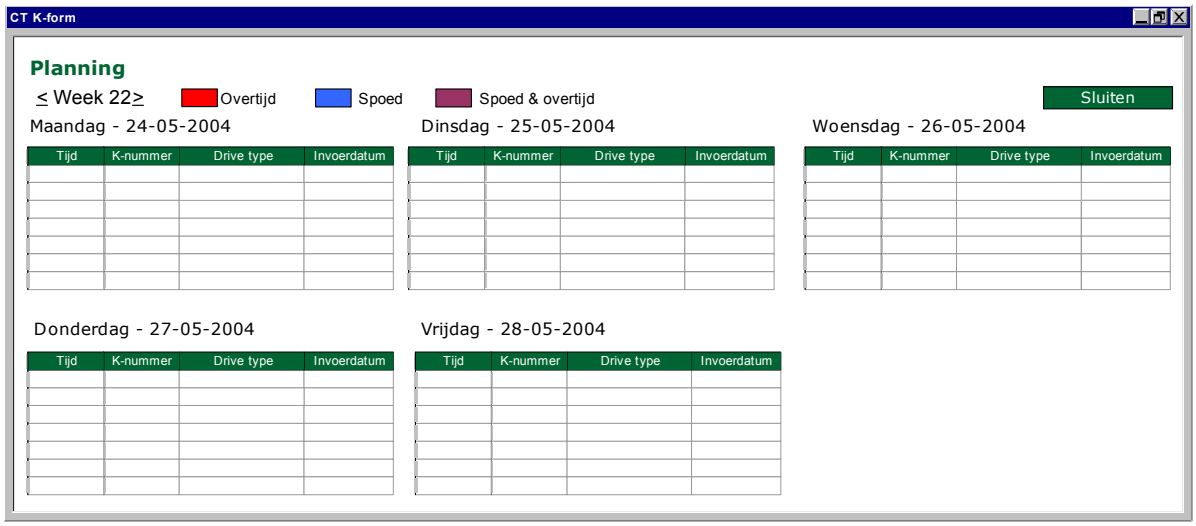




Het wizard systeem is zeer flexibel, en alle gewenste data kan uit de database worden opgehaald. Het gebrek aan gebruiksvriendelijkheid is echter onacceptabel.

- Voor de oplossing tot dit probleem is een methode gevonden waarbij de gebruikers alleen maar de gewenste kolommen hoeven aan te klikken, en eventueel condities hoeven aan te geven. Deze methode is minder flexibel, maar flexibel genoeg. Het gebruiksgemak is echter vele malen groter dan het wizard systeem. Voor de interface ontwerpen wil ik verwijzen naar een bijlage van bijlage II. De oplossing maakt gebruik van zogenaamde views. Een view is een virtuele tabel. In dit geval zal de gebruiker zijn overzicht uit de view "TotaleOpdrachtInformatie" halen. Deze view bevat als het ware alle informatie van een opdracht in een ongenormaliseerde vorm. Alle kolommen in de view zijn weergegeven in de interface zodat de gebruiker alleen de gewenste kolommen hoeft aan te klikken, condities op te geven en op uitvoeren te klikken. Het gebruik van views is een vrij grote belasting voor het systeem, vooral als er veel data in de database staat. De virtuele tabel moet namelijk eerst worden opgebouwd voordat er data uitgehaald kan worden. De virtuele tabel kan groot zijn. De code voor de views is opgenomen in bijlage IX.

- Naast de eerste versie van de query builder bleek dat ook de eerste versie van de planning niet goed zou werken. Deze planning was opgezet als een echte planning. Het had een soort kalender structuur. Opdrachten konden worden ingepland op datums, worden verplaatst en verwijderd. Door middel van navigatieknoppen kon door de weken worden gebladerd. Tijdens een sessie met testgebruikers bleek dat de planning niet overzichtelijk genoeg zou zijn. Zie het onderstaande beeldscherm ontwerp.



- Een oplossing is gevonden in simpelheid. Nu is de planning niet meer dan een overzicht per gebruiker, met openstaande opdrachten. Zie bijlage IX. Ik moet zeggen dat ik het met de gebruikers eens ben dat de huidige planning veel overzichtelijker is.

6.7 Bouw conversietools pilot.

Voor deze pilot zijn geen conversietools nodig geweest.

6.8 Handmatige procedures pilot.

Deze pilot heeft geen handmatige procedures.

6.9 Opleidingsmateriaal pilot.

Zie paragraaf 4.9

6.10 Handleidingen pilot.

Zie paragraaf 4.10

6.11 Integratie bouweenheden.

Zie paragraaf 4.11

6.12 Beoordelen en testen pilots.

Zie paragraaf 4.12

7 Pilot 4: Beheer

In het nu volgende hoofdstuk zal worden ingegaan op het ontwerp en de bouw van de pilot "Beheer". Besproken zullen worden onder andere het systeemontwerp (klassediagram, sequentie diagrammen), het database ontwerp in de vorm van een ERD en een aantal interessante code fragmenten. Bedenk dat de besproken producten volgens een aantal iteratie slagen zijn verkregen.

7.1 Globaal-functionele structuur

Bij deze pilot is, net als de vorige, geen lijst met trefwoorden gemaakt. Er zijn namelijk geen klassen bijgekomen, wel zijn bestaande klassen van extra functionaliteit voorzien. In het klassediagram zijn dan ook alleen de klassen opgenomen waar een wijziging in is opgetreden. Deze wijzigingen zullen vooral te maken hebben met het verwijderen en wijzigen van data in de database.

Ook in de database zelf is niets veranderd.

7.2 Globaal-technische structuur

Zie paragraaf 5.2

7.3 Globaal-organisatorische structuur

Zie paragraaf 4.3

7.4 Pilot ontwikkelplan

Bij het maken van het pilotplan is gebruik gemaakt van de planning als basis voor time-boxing. Voor het ontwikkelen van het totale systeem zijn vijftien weken uitgetrokken. Omdat de pilot "Beheer" zijn vier weken uitgetrokken. Bij de klassen zijn de stored procedures inbegrepen. Bij de interface zijn de interface klassen inbegrepen.

7.5 Ontwerp software bouweenheden.

Zie paragraaf 4.5

7.6 Bouw software bouweenheden.

De bouw van de software bouweenheden is op een zelfde manier gegaan als de voorgaande pilots. De problemen waar tegenaan is gelopen zijn de volgende.

- Er moest een manier worden gevonden om de gebruikers te autoriseren aan de hand van de Active Directory. Op deze manier hoeven niet nog eens wachtwoorden te worden opgeslagen in het programma. Gebruikers daarentegen zullen wel in het programma moeten worden opgenomen,

omdat er attributen worden gebruikt die niet in de Active Directory voorkomen.

- In een artikel op het internet is een code fragment gevonden wat zeer bruikbaar is gebleken. Na een aantal kleine aanpassingen is het nu mogelijk gebruikers bij het inloggen te autoriseren aan da hand van de Active Directory. Voor de code van de Active Directory autorisatie wil ik graag verwijzen naar bijlage X.

- Er moest een manier worden gevonden om via het programma data uit Excel sheets te importeren in de database.
  - Voor het importeren zijn DTS packages gemaakt net als bij de conversietools voor de pilot "k-formulieren". Nu is het mogelijk om via een stored procedure deze DTS packages uit te voeren. Zie de onderstaande code fragmenten.

```
ALTER PROCEDURE dbo.RunOnderdelenImport
AS
EXEC master..xp_cmdshell 'dtsrun /S<servernaam> /U<gebruiker>
/P<wachtwoord> /NOnderdelenImport'
```

```
ALTER PROCEDURE dbo.RunStdRepPrijsImport
AS
EXEC master..xp_cmdshell 'dtsrun /S<servernaam> /U<gebruiker>
/P<wachtwoord> /NStdRepPrijsImport'
```

Door middel van deze code kan programmatisch een DTS package worden uitgevoerd. Aan deze code fragmenten is goed te zien dat T-SQL, de programmeertaal voor stored procedures, meer is dan alleen een taal om gegevens uit de database op te halen.

7.7 Bouw conversietools pilot.

Voor deze pilot zijn geen conversietools nodig geweest.

7.8 Handmatige procedures pilot.

Deze pilot heeft geen handmatige procedures.

7.9 Opleidingsmateriaal pilot.

Zie paragraaf 4.9

7.10 Handleidingen pilot.

Zie paragraaf 4.10

7.11 Integratie bouweenheden.

Zie paragraaf 4.11

7.12 Beoordelen en testen pilots.

Zie paragraaf 4.12

8 Invoering

Het nu volgende hoofdstuk zal een inzicht geven in de introductie van het systeem in het reparatie traject. Er zal worden verhaald over het opleidingsmateriaal, de handleidingen, het beoordelen en testen van het systeem, de acceptatie en natuurlijk de invoering zelf.

8.1 Opleidingsmateriaal.

Nog voor de afzonderlijke pilots, nog voor het volledige systeem zal opleidingsmateriaal worden verzorgd. De interface van het programma is namelijk zo opgezet dat het voor een gebruiker vrij eenvoudig te doorgronden is. Verdere toelichtingen zullen in de handleiding terug te vinden zijn. Wel zal er voor alle gebruikers een presentatie gehouden worden waarin de verschillende functies van het systeem worden uitgelegd.

8.2 Handleidingen.

Het schrijven van de gebruikershandleiding is één van dingen die nog moet worden gedaan. Het zal moeten gebeuren voordat de applicatie "live" gaat, zodat iedere gebruiker aan de handleiding kan refereren. In de handleiding zal de interface als uitgangspunt gaan dienen, dit is immers waar de gebruikers tegenaan kijken. Een punt waar extra aandacht aan besteed zal moeten worden is de querybuilder. Om gebruik te maken van de querybuilder moet in mindere mate SQL worden uitgelegd. De technische handleidingen kunnen automatisch worden gegenereerd, door gebruik te maken van de mogelijkheden van de VisualStudio.Net ontwikkelomgeving. Ten tijde van het schrijven van dit verslag is echter nog niet alle code voorzien van commentaar.

8.3 Integratie bouweenheden.

Het integreren van de bouweenheden is (ongemerkt) tijdens het ontwikkelen van de bouweenheden gegaan. Telkens is voort geborduurd op de eerste pilot. Op deze manier is de lijn tussen pilots erg vaag.

8.4 Beoordelen en testen systeem.

Na iedere iteratieslag heeft een functionele beoordeling door de opdrachtgever plaatsgevonden. Deze beoordeling werd uitgevoerd in de vorm van een kleine presentatie van de afstudeerder. De opdrachtgever gaf hierna zijn goedkeuring en eventuele op en aanmerkingen. Nu de applicatie eenmaal is gereed voor gebruikerstests, is er een dergelijke presentatie voor een select groepje gebruikers gehouden. Eventuele op en aanmerkingen zijn ter kennisgeving aangenomen en kritieke punten zijn verbeterd. Momenteel draait het systeem in een testfase van een maand. Het selecte groepje gebruikers werkt nu met het oude, en het nieuwe systeem. Gevraagd is of zij niet kritieke op en aanmerkingen willen opschrijven, kritieke opmerkingen/fouten zullen direct worden verholpen. Is het systeem naar tevredenheid door de maand van testen heen gekomen, dan zal de data uit het oude systeem worden overgezet en het oude systeem worden vervangen.

8.5 Invoering.

Zoals in de voorgaande paragraaf is uitgelegd, zal het nieuwe systeem het oude gaan vervangen nadat de maand proefdraaien is afgesloten. Echter voordat het systeem wordt ingevoerd, zal de feedback van de testgebruikers worden geanalyseerd en zal er eventueel naar worden gehandeld. Hierna kan het systeem volledig worden ingevoerd. Dit invoeren houdt in dat de programmatuur van de test server naar een productie server zal worden verplaatst. Dat de data vanuit het oude systeem zal worden geïmporteerd. Dat de handleiding aan de gebruikers ter beschikking wordt gesteld. Voordat de programmatuur naar de productie server kan worden verplaatst, moet één en ander op de server worden geïnstalleerd. Software als SQL Server 2000, ASP.NET framework en MS Word.

8.6 Systeem acceptatie.

Het systeem is geaccepteerd als het volledig is ingevoerd en het een nog nader te bepalen periode zonder grote problemen live heeft gedraaid. De opdrachtgever zal te kennen geven als het systeem door hem is geaccepteerd.

8.7 Overdracht systeem.

Het systeem is overgedragen als de programmatuur gekopieerd is naar de productie server, en een kopie van de programmatuur in het beheer van CT is gegeven.

8.8 Feedback

Omtrent het geven van feedback door de gebruikers tijdens de test fase zijn de volgende afspraken gemaakt.

- Op- en aanmerkingen worden opgeschreven. Eén keer per week zal de afstudeerder deze op- en aanmerkingen ophalen.
- Kritieke meldingen en fouten waardoor het programma niet naar behoren functioneert of de gebruiker zijn werk niet kan doen kunnen direct worden gemeld, en zullen zo snel mogelijk worden verholpen.

Voor deze constructie is gekozen om de afstudeerder zo veel mogelijk te ontlasten aangezien het schrijven van dit eindverslag en het testen van het systeem min of meer tegelijkertijd plaatsvindt.

8.9 Systeem ondersteuning

Met de opdrachtgever is afgesproken dat de afstudeerder nog een aantal weken zal blijven om ondersteuning aan de gebruikers te geven, en om nog een aantal openstaande werkzaamheden af te ronden. De ondersteuning zal worden opgezet als een soort helpdesk. De gebruikers kunnen bellen naar afstudeerder bellen met vragen en problemen. Problemen en vragen zullen worden genoteerd en verholpen. Eventueel kan de gebruikershandleiding worden herzien aan de hand van de gestelde vragen.

8.10 Nog af te ronden werkzaamheden

Wat nog te doen om het project volledig af te ronden?

- Verzamelen van gebruikersfeedback.
- Eventueel doorvoeren van aanpassingen aan de hand van gebruikersfeedback.
- Schrijven gebruikershandleiding.
- Omzetten data oude systeem naar nieuwe systeem.
- Gebruik handscanner voor serienummer scanning onderzoeken.
- Gebruik multi-feed documentscanner voor bijlagen onderzoeken.
- Koppeling met Scope onderzoeken.
- Koppeling met Serienummer registratie onderzoeken.

DEEL III - Evaluatie

9 Evaluatie

Het nu volgende hoofdstuk zal een evaluatie geven van zowel product als proces.

9.1 Productevaluatie

Persoonlijk ben ik erg tevreden met het uiteindelijke softwareproduct. Ik denk dat er een product is gemaakt waarmee de werknemers, die betrokken zijn bij het reparatie traject van aandrijfcomponenten, hun efficiëntie en de efficiëntie richting de klant kunnen verbeteren. Deze gedachte wordt versterkt door de enthousiaste reacties van de testgebruikers. Tijdens de eerste tests bleek dat het digitale K-formulier al bij de verkoper lag voordat het papieren K-formulier de werkplaats had verlaten. De testgebruikers zijn zeer te spreken over het inzicht en overzicht wat ze nu over hun eigen werk hebben. Het is dan als beginnend informaticus leuk om te zien dat een eigen product zoveel positieve reacties oproept.

De technische kant van het product betreft zou misschien nog iets beter kunnen. Persoonlijk kom ik maar net kijken in de wereld van object georiënteerd ontwerpen en programmeren. Ik weet zeker dat de klassen met hun methoden efficiënter kunnen worden gemaakt door nog meer functionaliteit uit de interface klassen naar de object klassen te verplaatsen. Misschien dat dit in de komende weken nog zal worden aangepast. Het uiteindelijke systeem is in mijn mening een goede reflectie van het systeem wat tijdens de ontwerpfase is gedefinieerd.

De documentatie en ontwerpen zijn naar mijn mening goed en gedetailleerd. Ik ben van mening dat de eisen die tijdens het project naar boven zijn gekomen in de ontwerpen zijn verwerkt. Het systeem is een goede implementatie van de ontwerpen.

9.2 Procesevaluatie

Met het verloop van het proces ben ik redelijk tevreden. Het proces van ontwikkelen met IAD is redelijk goed verlopen. In het begin van het project was het lastig een volledig eisenpakket samen te stellen, omdat de eisen voor de opdrachtgever nog niet geheel vastlagen. Om deze reden zijn er in de loop van de tijd eisen bijgekomen en gewijzigd. Het ontwerpen ging steeds beter omdat steeds meer kennis van de applicatie/programmeer omgeving werd opgedaan. Ook het programmeren ging steeds beter en sneller. Ik ging steeds meer inzicht krijgen in de mogelijkheden van de ASP.NET programmeer omgeving en voel me nu als een vis in het water in deze omgeving. Ik mag denk ik wel stellen dat ik wat programmeren in ASP.NET geen beginner meer ben. Het is jammer dat de tijd te kort is voor een dergelijk groot project, ik had het graag volledig afgerond in het kader van het afstuderen.

Tijdens de afstudeerstage ben ik zeer zelfstandig bezig geweest. Bij CT is er namelijk geen echte ICT afdeling, laat staan een systeem ontwikkelaar of programmeur. Dit is jammer, er zijn momenten geweest dat ik hulp kon gebruiken. Op deze momenten is een goede internet verbinding onmisbaar. Op het internet is veel informatie te vinden omtrent systeem ontwikkeling en ontwikkel/database platforms. Veelal zijn forums een goede bron van informatie, het komt namelijk zelden voor dat je de enige bent met een bepaald probleem of vraag.

Literatuurlijst

Literatuur

- Tolido, R.J.H., IAD Het evolutionair ontwikkelen van informatiesystemen Pilot 2, Academic Service, ISBN 90-395-0401-6.
- Warmer, J., Kleppe, A., Praktisch UML 2e editie, Addison Wesley, ISBN 90-430-0494-4.
- Microsoft, SQL Server Books Online.

Websites

- C# Help
  http://www.csharphelp.com/
- Datagrid Girl!
  http://www.datagridgirl.com/
- ASP.NET Resource Index
  http://www.411asp.net/
- Experts Exhange
  http://www.experts-exchange.com/
- Microsoft Developer Network
  http://msdn.microsoft.com/
- The Code Project
  http://www.thecodeproject.com/
- C# Corner
  http://www.c-sharpcorner.com/
- .NET 247
  http://www.dotnet247.com
- ASP.NET
  http://www.asp.net/
- MeadCo's ScriptX
  http://www.meadroid.com/scriptx/

Verklarende woordenlijst

IAD
Iterative Application Development. IAD is een methode voor het ontwikkelen van applicaties op een iteratieve manier. Het idee achter IAD is evolutie. Het "waterval" model wordt aan de kant geschoven en een manier van herhaling wordt opgepakt. De producten die uit een IAD ontwikkelproces voortkomen zullen in latere stadia weer worden aangepast.

UML
Unified Modeling Language. UML biedt een aantal technieken waarmee op een gestructureerde manier de werkelijkheid kan worden gemodelleerd.

C#
C-sharp. C-sharp is de programmeertaal die wordt gebruikt voor het coderen van de server-side code van de applicatie.

ASP.NET
ASP.NET is een raamwerk wat een programmeur een grote hoeveelheid standaard functies en klassen biedt.

SQL
Structured Query Language. SQL is een gestandaardiseerde manier waarmee data uit een databron kan worden gehaald.

T-SQL
Transact-SQL. T-SQL is een dialect van het standaard SQL gebruikt door MS SQL Server 2000 in stored procedures.

DTS
Data Transformation Service. DTS is een service waarmee gemakkelijk data kan worden geëxporteerd en geïmporteerd van en naar een SQL Server 2000 database.