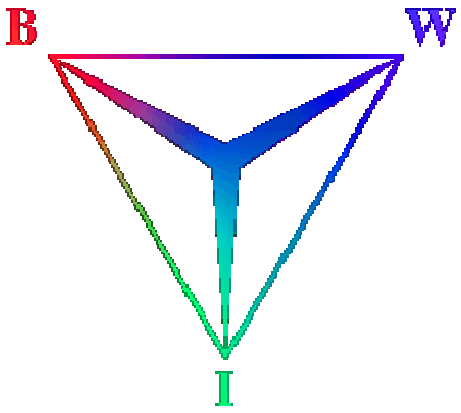
Plannen, Uitvoeren en Herplannen

een praktische benadering van projectplanning

Wouter Radder
Juni 2006



Vrije Universiteit Amsterdam
Faculteit Exacte Wetenschappen
Bedrijfswiskune en Informatica

De Boelelaan 1081
1081 HV Amsterdam

efficiency online

Houtmankade 45
1013 MR Amsterdam

Voorwoord

De studie Bedrijfswiskunde en Informatica wordt traditioneel afgesloten met een stage, waarbij de student gedurende een periode van minimaal zes maanden de tijdens de studie opgedane kennis toepast op een praktisch probleem binnen een organisatie. Ik heb deze stage afgelegd bij Efficiency Online B.V. te Amsterdam, het bedrijf waar ik ook mijn duale werkperiode volbracht heb. De stage heeft plaatsgevonden in de periode van februari 2005 tot en met september 2005. Tijdens de stage heb ik een model ontwikkeld voor het plannen van projecten, en dit model vervolgens geïmplementeerd in de vorm van een prototype binnen de software generator van Efficiency Online. In dit document wordt verslag gedaan van deze stage. Hierbij zal zowel aandacht besteed worden aan het verloop van de stage als de behaalde resultaten.

Alvorens hieraan te beginnen wil ik graag eerst een aantal mensen bedanken voor het mogelijk maken van de stage. Ten eerste de bedrijfsleiders van Efficiency Online, Daniel van der Wallen en Sjoerd Geraedts. Zij hebben mij zowel bij het uitzoeken van een onderwerp als tijdens de stage een grote mate van vrijheid gegeven waarvoor ik hun erg dankbaar ben. Ondanks hun enorm drukke schema zijn ze altijd bereid geweest me te helpen, te sturen en te motiveren op de momenten waarop dit het nodig was. Verder wil ik graag mijn begeleider vanuit de VU, Rob van der Mei, bedanken voor zijn inzet en inspanningen gedurende de stage. Ten slotte wil ik iedereen binnen Efficiency Online bedanken voor de goede tijd die ik zowel gedurende mijn stage als mijn duale werkperiode met jullie gehad heb.

Samenvatting

Efficiency Online maakt maatwerk browser software voor haar klanten. Deze software wordt op projectbasis ontwikkeld met behulp van een zelf ontwikkelde *software generator*. De ontwikkelprojecten duren in de praktijk vaak langer dan vooraf ingeschat. Het management van Efficiency Online heeft aangegeven dat in veel gevallen haar grip op het project verloren gaat. Er is binnen Efficiency Online behoefte om op een meer gestructureerde manier om te gaan met het plannen van projecten en het bijsturen bij afwijkingen op deze planning.

Het past binnen de filosofie van Efficiency Online om dit probleem vanuit een generiek oogpunt te benaderen: er is niet specifiek naar de planning van Efficiency Online gekeken, maar naar projectplanning in het algemeen. Er is een model ontwikkeld dat activiteiten inroostert op basis van het zogenaamde *kritieke pad*. Het gebruik van dit model dwingt de planner tot een *systematische*, *gestructureerde* werkwijze. Het model is dusdanig opgezet dat het een hoge mate van *flexibiliteit* bezit: uitkomsten kunnen met de hand aangepast worden en planningen kunnen *meegroeien* met de voortgang van projecten. Dit model is gedeeltelijk geïmplementeerd, in de vorm van een werkend *prototype* binnen de genoemde software generator.

In de toekomst zal dit prototype verder ontwikkeld worden om in eerste instantie als test ingezet te worden voor de *interne projectplanning* van Efficiency Online. Tijdens deze test zal het model *gevalideerd* worden op praktische bruikbaarheid. Hierna kan het prototype omgebouwd worden tot een concreet *product* dat aan klanten aangeboden kan worden.

Hoofdstuk 1: Inleiding

In dit hoofdstuk wordt een een framewerk opgezet voor de uitvoering van de stage. Het stagebedrijf (paragraaf 1.1) en het probleem dat binnen dit bedrijf speelt (paragraaf 1.2) worden besproken en de stageopdracht wordt geformuleerd in de vorm van een doelstelling (paragraaf 1.3). Vervolgens bespreken we in paragraaf 1.4 en 1.5 achtereenvolgens beknopt het plan van aanpak en de verdere opbouw van dit verslag.

1.1 Efficiency Online

Efficiency Online is een klein en jong bedrijf dat database gestuurde webapplicaties ontwikkelt voor haar klanten. Het bedrijf is in het begin van 2003 ontstaan uit de samenwerking tussen twee bedrijven, te weten Dantec B.V. en E.N.G. Efficiency B.V., en bestaat momenteel uit een tiental medewerkers welke verdeeld zijn over een aantal afdelingen. Alle software wordt ontwikkeld wet behulp van een unieke, zelf ontwikkelde *softwaregenerator*. Deze generator vormt dan ook het hart van het bedrijf en is (in)direct betrokken bij alle activiteiten van het bedrijf.

1.1.1 Generiek denken

Het idee achter de software generator is dat elk specifiek probleem een instantie is van een generieke klasse van problemen. Door een oplossing voor deze generieke klasse van problemen te vinden, wordt het specifieke probleem opgelost. Dit is niet alleen de filosofie achter de generator, maar achter de gehele werkwijze van Efficiency Online.
Ter illustratie het volgende voorbeeld: een klant vraagt om een pagina in zijn applicatie waar bezoekers een lijstje vragen kunnen beantwoorden die hun tevredenheid meet. Efficiency Online stelt zichzelf dan de vraag: “is deze specifieke vraag een voorbeeld van een breder inzetbaar probleem?”. In dit geval is het antwoord ja: een dergelijke vragenlijst valt op te nemen in een generiek enquête systeem. Vervolgens wordt er door middel van marktonderzoek gekeken of er een breed genoeg draagvlak is om dit probleem ook daadwerkelijk generiek te implementeren.

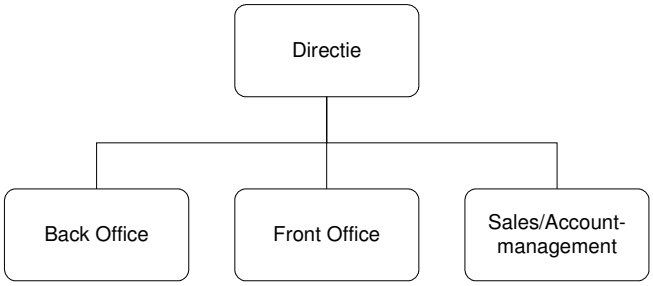
1.1.2 Klanten

Efficiency Online heeft klanten van zeer uiteenlopende aard. Grofweg zijn zij in te delen in twee categorieën: *commerciële instellingen* en *maatschappelijke organisaties*. Onder de commerciële instellingen heeft het bedrijf een aantal grote klanten, zoals *Akzo Nobel*, waarvoor Efficiency Online onder de naam *marktkompas* een geografisch informatie systeem heeft ontwikkeld. Ook zitten er kleinere bedrijven zoals *De Monnik Apartments* in het klantenbestand van Efficiency Online. Naast de commerciële bedrijven is Efficiency Online ook actief in de goede doelen wereld. De bekendste klant uit deze categorie is *Unicef*, voor wie Efficiency Online een aantal systemen ontwikkeld heeft, waaronder een enquêtetool en een programma voor het opstellen en versturen nieuwsbrieven aan haar leden.

1.1.3 Organisatiestructuur

Vanwege de kleine omvang van het bedrijf is de organisatiestructuur van Efficiency Online relatief eenvoudig. Om te beginnen is er de directie, welke bestaat uit de twee

eigenaren van het bedrijf. Zoals men vaak ziet bij kleinere bedrijven houden zij zich, naast het strategische beleid, vooral met operationele zaken bezig.
Onder de directie bevinden zich twee ontwikkelafdelingen, de *Frontoffice* en de *Backoffice*. De frontoffice bestaat uit medewerkers die allen direct verantwoordelijk zijn voor de uitvoering van een aantal projecten. Hiernaast hebben zij de taak om *support* aan klanten te geven. De backoffice is primair verantwoordelijk voor ontwikkelingen aan de generator en het beheren van de infrastructuur binnen het bedrijf. Hiernaast is er een afdeling sales/accountmanagement die verantwoordelijk is voor het binnenhalen van opdrachten. In de onderstaande figuur is deze structuur in een organogram weergegeven.



Figuur 1: het organogram van Efficiency Online.

1.2 Probleembeschrijving

De ontwikkeling van software voor klanten gebeurt binnen Efficiency Online op projectbasis. Bij de start van een project wordt er een inschatting gemaakt van de hoeveelheid werk benodigd voor de voltooiing van het project. Deze schatting wordt vervolgens voornamelijk gebruikt voor het opstellen van de offerte en voor het uitkiezen van de uitvoerende medewerkers. Naarmate het project vordert, wordt er vaak afgeweken op de oorspronkelijke planning. Dit wordt door Efficiency Online als onvermijdelijk beschouwd, de vraag is echter hoe hiermee om te gaan. Momenteel wordt er in een dergelijke situatie weinig tot geen feitelijke actie ondernomen. Dit heeft tot gevolg dat het management maar al te vaak de grip op een project verliest. Efficiency Online is op zoek naar een meer gestructureerde methode om systematisch om te gaan met de planning en bijsturing van haar projecten.

1.3 Opdracht en doelstelling

Het past in de filosofie van Efficiency Online om ook voor dit probleem een meer generieke aanpak te hanteren. Daarom gaan we tijdens de stage niet specifiek naar projectplanning binnen Efficiency Online kijken, maar meer generiek, naar projectplanning in het algemeen. Daarom formuleren we de volgende doelstelling:

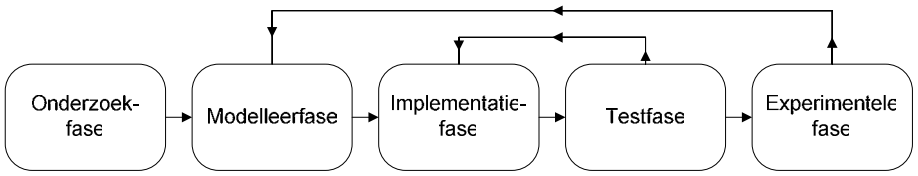
*Het **doel** van de stage is het opstellen van een model voor projectplanning en het maken van een (werkend) prototype dat dit model implementeert.*

Voorwaarden
Voor het bereiken van deze doelstelling stellen we een aantal technische voorwaarden waar het model aan moet voldoen. De volgende voorwaarden beschouwen we als cruciaal voor het behalen van de doelstelling:

1. *Transparantie van het model*: voor een verantwoord gebruik van een dergelijk systeem is het van belang dat de gebruiker te allen tijde door heeft wat hij exact doet.
2. *Beperkte complexiteit van het model*: managers hebben vaak moeite beslissingen te baseren op modellen die ze zelf niet begrijpen. Daarom willen we dat de technische complexiteit beperkt blijft.
3. *Hanteerbaarheid van het model*: het is van belang dat het model hanteerbaar blijft, vooral met betrekking tot de omvang van de vereiste invoer van het model.
4. *Flexibiliteit van de uitkomsten*: er moet altijd de mogelijkheid blijven om een gegenereerde planning handmatig aan te passen. Hiernaast moet een bestaande planning 'mee kunnen groeien' met de realiteit en aangepast kunnen worden naar onvoorziene omstandigheden.

1.4 Plan van aanpak

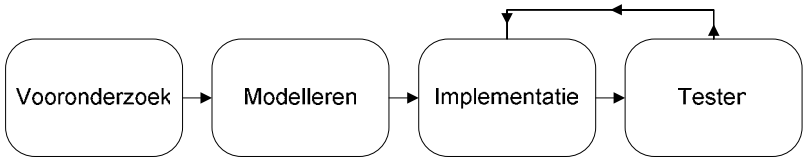
Voor de ontwikkeling van een volwaardige webapplicatie dienen een aantal fasen doorlopen te worden. In de onderstaande figuur worden deze fasen schematisch weergegeven.



Figuur 2: schematische weergave van het ontwikkelproces.

De onderzoek-, modelleer- en implementatiefasen worden achtereenvolgens doorlopen. Vervolgens wordt er in de testfase *geverifieerd* of het model correct geïmplementeerd is. Zolang dit niet het geval is, vindt er terugkoppeling naar de implementatiefase plaats, welke gedeeltelijk opnieuw doorlopen zal worden. In de experimentele fase wordt *gevalideerd* of het model in de praktijk bruikbare oplossingen genereert.

Omdat er tijdens de stage niet een volledige applicatie maar slechts een prototype ontwikkeld zal worden (zie de doelstelling), houden we de onderstaande fasering aan:



Figuur 3: schematische weergave van de fasering die tijdens de stage aangehouden zal worden.

De experimentele fase valt dus buiten de stage. De overige fasen worden in het vervolg van dit verslag besproken.

1.5 Het verslag

In dit verslag worden de vier in de vorige paragraaf gedefinieerde fasen in chronologische volgorde behandeld. We kiezen voor een resultaat georiënteerde benadering: we presenteren de verkregen resultaten en motiveren de hierbij gemaakte keuzen. In hoofdstuk 2 worden de resultaten van het vooronderzoek gegeven. Vervolgens wordt in hoofdstuk 3 het model stukje bij beetje opgebouwd. In hoofdstuk 4 presenteren we het resulterende prototype. Ten slotte wordt in hoofdstuk 5 teruggekeken op de stage en wordt er beschouwd in welke mate de doelstelling behaald is. Om de geïntroduceerde theorie kracht bij te zetten zullen we vanaf hoofdstuk 3 gebruik maken van een zogenaamd *running example*. Hiervoor gebruiken we een vereenvoudigde versie van de projecten van Efficiency Online.

Hoofdstuk 2: Projectplanning

In dit hoofdstuk leggen we verslag van het tijdens de stage uitgevoerde vooronderzoek. In het eerste gedeelte van dit hoofdstuk gaat de aandacht uit naar begrippen die nauw met projectplanning samenhangen. Zo wordt er om te beginnen in paragraaf 2.1 een definitie van het begrip *project* gegeven. Vervolgens wordt in paragraaf 2.2, op basis van deze definitie, de structuur van projecten nader beschouwd. In paragraaf 2.3 behandelen we de organisatie van projecten en in paragraaf 2.4 hebben we het over projectbeheersing. Vanaf paragraaf 2.5 concentrreren we op projectplanning en zullen we een aantal vraagstukken introduceren. In paragraaf 2.6 beschrijven we de methodiek die wij zullen gebruiken om tot antwoorden op de gedefinieerde vraagstukken te komen.

2.1 Projecten

Door een aantal verschillende definities uit de literatuur te vergelijken hebben we de volgende definitie voor een project opgesteld:

Definitie 1:

*Een **project** is een verzameling van activiteiten die met het gebruik van beperkte middelen uitgevoerd dienen te worden voor het bereiken van een vooraf gedefinieerd, uniek resultaat.*

In de onderstaande secties bespreken we de drie belangrijkste elementen van deze definitie: het *resultaat*, de *activiteiten* en *beperking van middelen*.

2.1.1 Het project resultaat

In onze definitie stellen we dat het resultaat *uniek* van aard is. Hiermee doelen we op het eenmalige karakter van dit resultaat. Dit is een belangrijk kenmerk van projecten, omdat dit is wat een project onderscheidt van de “reguliere productie” binnen een bedrijf. Een tweede kenmerk van het resultaat van een project is dat het in zekere mate vooraf gedefinieerd moet kunnen worden. Dit is niet altijd even eenvoudig, in de praktijk is er soms een vooronderzoek nodig om het resultaat van het project te definiëren.

2.1.2 Activiteiten

We spreken in onze definitie van een verzameling van activiteiten. Al deze activiteiten dienen (indirect) bij te dragen aan het bereiken van het beoogde resultaat. Tussen de verschillende activiteiten bestaan afhankelijkheden die de volgorde van de activiteiten in grote lijnen vastleggen.

2.1.3 Beperkte middelen

De activiteiten dienen uitgevoerd te worden met een beperkt aantal middelen. Over het algemeen wordt bij de aanvang van het project bepaald welke en hoeveel middelen er beschikbaar worden gesteld ter uitvoering van het project. Deze middelen zijn te onderscheiden in de volgende vier categorieën:

1. *Mensen en deskundigheid*: voor het uitvoeren van activiteiten zijn mensen nodig. Het uitvoeren van elke activiteit vereist één of meerdere deskundige vaardigheden.
2. *Tijd*: tijd is vaak één van de belangrijkste beperking tijdens de uitvoering van project. In deze context kent 'tijd' twee dimensies: arbeidsuren en doorlooptijd.
3. *Materieel*: vaak is er voor het uitvoeren van het project materieel nodig. Dit materieel valt op te delen in grondstoffen, infrastructuur en overige hulpmiddelen.
4. *Budget*: projecten dienen uitgevoerd te worden binnen een vooraf bepaald budget.

2.2 Structuur van projecten

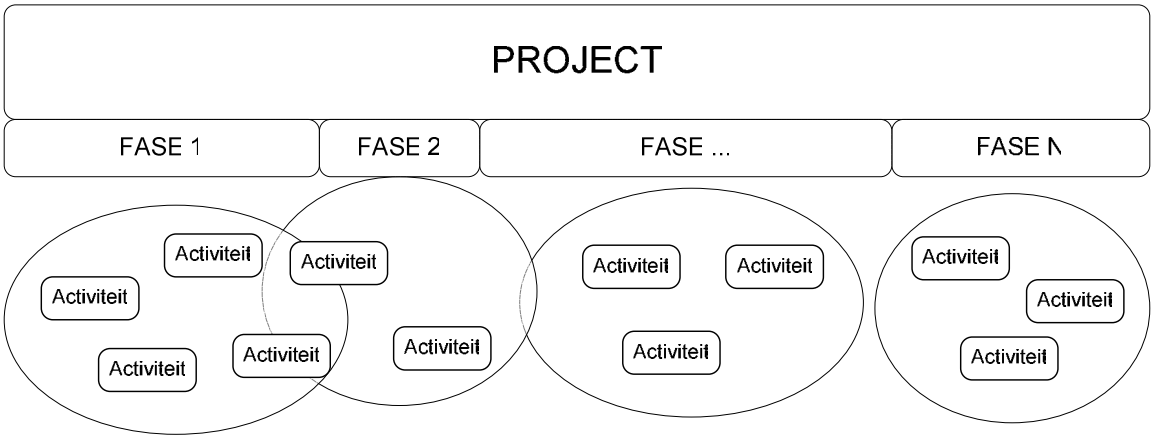
Met deze definitie als basis analyseren we de structuur van de activiteiten verzameling binnen een enkel project. Vervolgens plaatsen we deze structuur in de gehele projectportfolio van een organisatie.

2.2.1 Fasen en activiteiten

Projecten worden vaak opgedeeld in fasen. Elke fase bestaat uit een aantal activiteiten welke samen leiden tot de oplevering van één of meerdere deelresultaten of *milestones*. In figuur 4 wordt een voorbeeld gegeven van de fasen zoals je die tegen zou kunnen komen binnen een project.

Figuur 4: een voorbeeld van verschillende fasen die binnen een project plaatsvinden.

Deze figuur impliceert dat er een lineaire structuur in de fasen zit waarbij de activiteiten uit de verschillende fasen in strikt chronologische volgorde doorlopen worden. In de praktijk zal het echter vaak voorkomen dat activiteiten uit verschillende fasen parallel plaatsvinden. De verschillende fasen geven dus niet per definitie een chronologisch overzicht van het project. Om deze reden zullen wij een fasering binnen een project slechts beschouwen als een differentiatie van de activiteiten naar aard. Elke activiteit is dus te plaatsen binnen één van de fasen van het project, maar de indeling van activiteiten in fasen geeft ons geen informatie over de relaties tussen de activiteiten. We beschouwen de verschillende fasen dus als kenmerken van de activiteiten, en niet als op zich zelf staande onderdelen van het project. Dit wordt schematisch weergegeven in figuur 5:



Figuur 5: schematische weergave van het overlappen van activiteiten uit verschillende fasen in een project.

Afhankelijk van de grootte en complexiteit van het project en de omgeving waarbinnen het project plaats vindt zullen de activiteiten in meer of mindere mate verder op te delen zijn in subactiviteiten. In paragraaf 2.6 zullen we nader op deze opdeling in subactiviteiten in gaan.

2.2.2 Projecten, programma's en portfolio's

Binnen grote bedrijven vinden vaak talloze projecten tegelijk plaats. Deze projecten worden uitgevoerd om een op hoger niveau gesteld doel te bereiken. Vaak worden meerdere projecten uitgevoerd voor het bereiken van één doel. Men spreekt in dit geval van een programma. Op het hoogste niveau binnen de organisatie kunnen de programma's op hun beurt weer gegroepeerd worden in portfolio's. Programma's en portfolio's hebben dezelfde eigenschappen als projecten en we kunnen de in paragraaf 2.1 gevonden definitie dan ook gemakkelijk aanpassen voor programma's en portfolio's:

Definitie 2:

*Een **programma** is een verzameling van projecten die met het gebruik van beperkte middelen uitgevoerd dienen te worden voor het bereiken van een vooraf gedefinieerd, uniek resultaat.*

Definitie 3:

*Een **portfolio** is een verzameling van programma's die met het gebruik van beperkte middelen uitgevoerd dienen te worden voor het bereiken van een vooraf gedefinieerd, uniek resultaat.*

2.3 Projectorganisatie

In deze paragraaf beschouwen we de organisatie van projecten. Allereerst bekijken we welke verschillende partijen er doorgaans bij een project betrokken zijn en wat hun belangen en verantwoordelijkheden zijn. Voor de uitvoering van een project worden de

projectmedewerkers meestal verdeeld in groepen. Vervolgens beschrijven we hoe dit binnen grotere projecten veelal georganiseerd is. De indeling die we geven is een richtlijn, in de praktijk hebben de meeste bedrijven hiervoor hun eigen systeem.

2.3.1 Betrokken partijen

Over het algemeen zijn er bij projecten twee partijen die direct belang hebben bij het project: de *uitvoerende partij* en de *opdrachtgever*. De opdrachtgever wil een product of dienst afnemen van de uitvoerende partij. Over het algemeen wil hij dat dit zo snel en goedkoop mogelijk gebeurt, zonder dat de kwaliteit van het resultaat in gevaar komt. Simpel gezegd komt het er op neer dat de uitvoerende partij geld wil verdienen, of op zijn minst zo min mogelijk geld wil uitgeven. Vaak wil hij om deze reden de doorlooptijd en kosten van het project zoveel mogelijk drukken. Dit kan wel eens ten koste gaan van de kwaliteit van het resultaat of het project. Voor beide partijen is het van belang om dit te voorkomen. Hiervoor is het belangrijk dat de opdrachtgever voldoende bij het project betrokken wordt. De opdrachtgever moet zich dan ook realiseren dat voor succesvolle voltooiing van het project ook door hem de nodige inspanningen gemaakt zullen moeten worden.

Naast de opdrachtgever en de uitvoerende partij zijn er nog een aantal andere partijen die bij projecten betrokken kunnen zijn. Ten eerste zijn er bij een project vaak *leveranciers* betrokken. Vooral vanuit het perspectief van de uitvoerende partij kunnen deze een belangrijke rol spelen: tijdige voltooiing van het project is in sommige gevallen direct afhankelijk van tijdige levering door de leveranciers. Andere partijen die een rol kunnen spelen bij projecten zijn bijvoorbeeld *overheidsinstellingen* zoals gemeenteraden, maar ook onafhankelijke instellingen zoals *milieuorganisaties* kunnen invloed uitoefenen op het verloop van projecten.

Over het algemeen geldt dat hoe meer verschillende partijen er bij een project betrokken zijn, hoe groter de kans op uitloop van het project wordt. Indien mogelijk is het aan te raden het aantal betrokken partijen zo beperkt mogelijk te houden. In de praktijk is dit meestal echter niet mogelijk. Er zal dan bij het plannen van het project rekening gehouden moeten worden met een vergrote kans op vertraging.

2.3.2 Projectgroepen

Zoals vermeld worden de verschillende werkzaamheden en projectmedewerkers vaak verdeeld in een aantal groepen. In Koetsier (2001) worden in de context van automatiseringsprojecten vier soorten groepen onderscheiden. Deze indeling is echter ook bruikbaar voor projecten in het algemeen. De vier soorten groepen zullen hieronder kort beschreven worden.

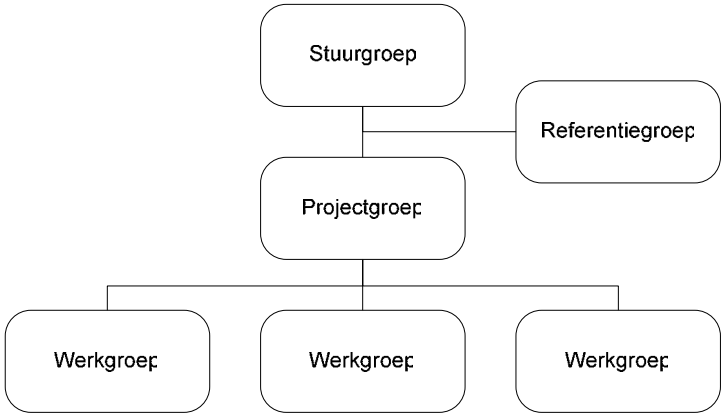
Stuurgroep

Deze groep is verantwoordelijk voor het “grote lijnen beleid” van het project. De stuurgroep bestaat doorgaans uit de projectmanager, een aantal leden van de directie van de uitvoerende partij en de afdelingshoofden van betrokken afdelingen.

Projectgroep
De projectgroep is verantwoordelijk voor de uitvoering van het project. Voorzitter van de groep is de projectmanager. Verder bestaat de groep uit de projectmedewerkers, afkomstig uit diverse afdelingen van de verschillende betrokken partijen.

Werkgroepen
Bij grotere projecten wordt de projectgroep vaak opgedeeld in werkgroepen, die elk verantwoordelijk zijn voor de uitvoering van bepaalde gedeelten van het project. In dit geval zitten alleen de voorzitters van deze werkgroepen ook in de projectgroep.

Referentiegroep
De referentiegroep dient in eerste instantie als adviesgroep en is te vergelijken met een stafdienst binnen de reguliere organisatie. In figuur 6 wordt structuur van groepen schematisch weergegeven.



Figuur 6: verdeling van een project in projectgroepen

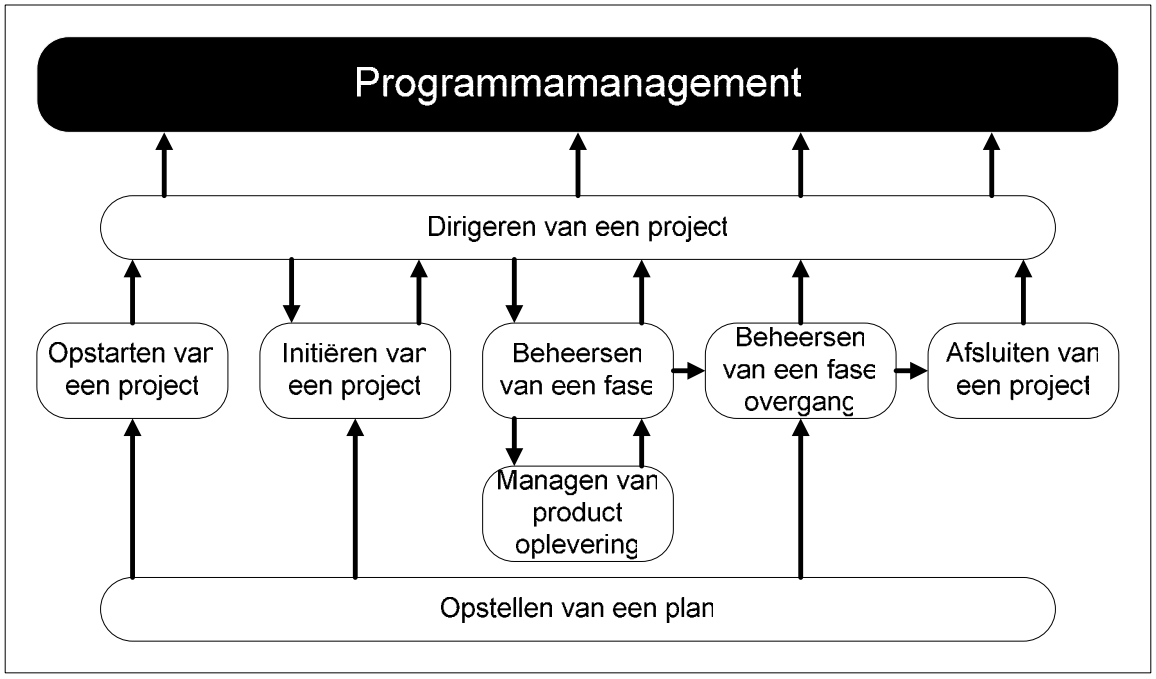
Binnen deze structuur vindt de planning op verschillende niveaus plaats. De lange termijn planning zal meestal binnen de stuurgroep plaatsvinden. In veel gevallen is deze lange termijn planning echter gebaseerd zijn op een optelling van korte termijn planningen, welke op het niveau van de projectgroep gemaakt worden.

2.4 Beheersen van projecten

Het beheersen (ofwel managen) van een project is van groot belang voor de goede afloop van het project. Het doel van deze paragraaf is het verkrijgen van inzicht in de verschillende activiteiten die komen kijken bij het beheersen van projecten, zonder hierbij diep op de details in te gaan. We doen dit aan de hand van PRINCE 2, een veel gebruikte methodiek voor projectbeheersing in de IT.

Projectbeheersing volgens PRINCE 2

PRINCE staat voor *Projects IN Controlled Environments* en is in 1989 ontwikkeld door de Central Computer and Telecommunications Agency (CCTA). De methode verdeelt het managen van een project in acht processen. Deze processen zijn georiënteerd op de verschillende fasen die een project doorloopt. In de volgende figuur wordt de samenhang tussen de verschillende processen volgens PRINCE 2 weergegeven.



Figuur 7: het PRINCE 2 procesmodel

De acht processen bestaan uit activiteiten die door het project-managementteam uitgevoerd dienen te worden. Hieronder worden deze processen en een aantal van de bijbehorende activiteiten kort beschreven.

1. Opstarten van een project
Dit proces is het eerste proces volgens PRINCE 2 en dient uitgevoerd te worden voor de start van het project. Het doel van dit proces is ervoor te zorgen dat de er aan de uitgangsbenodigdheden voor het initiëren van het project voldaan wordt. Het proces draait om de volgende drie kernactiviteiten:

1. Ontwerpen en aanstellen van het project management team
2. Zeker stellen dat de benodigde informatie voor het project team aanwezig is
3. Opstellen van het faseplan voor het initiëren van het project

2. Initiëren van een project
Tijdens dit proces wordt er in kaart gebracht wat er precies bereikt moet worden met het project, hoe dit bereikt dient te worden en waarom. De verantwoordelijkheden van de verschillende betrokken partijen worden geïdentificeerd zodat er een draagvlak ontstaat voor de uitvoering van het project.

3. Dirigeren van een project
Dit proces vindt gedurende het gehele project plaats en is bedoeld om de kans dat het project met succes afgesloten wordt te vergroten. De activiteiten van dit proces worden uitgevoerd door de stuurgroep van het project en komen neer op het nemen van beslissingen. Een aantal voorbeelden van deze beslissingen zijn:

- Geven van goedkeuring geven voor het initiëren van een project
- Geven van goedkeuring geven voor het uitvoeren van een fase
- Formele afsluiting van het project

4. Beheersen van een fase
Dit proces omvat het dagelijkse management van de werkzaamheden binnen een fase. Het doel van het proces is het verzekeren dat de betreffende fase binnen de gestelde tijdsduur en met gebruik van de beschikbaar gestelde middelen de gewenste resultaten oplevert.

5. Managen van fase overgangen
PRINCE 2 stelt dat aan het eind van iedere fase van een project het management de levensvatbaarheid van het verdere verloop van het project dient te beoordelen. Hiernaast kent dit proces de volgende doelstellingen:

- Formeel afsluiten van de huidige fase
- Het verkrijgen van autorisatie voor het uitvoeren van een volgende fase
- Vastleggen van alle leerpunten en uitslagen van meetpunten

6. Managen van productoplevering
Het doel van dit proces is het zeker stellen dat de resultaten worden gemaakt en opgeleverd volgens het plan dat is goedgekeurd door de stuurgroep. Hierbij hoort onder andere de controle of het uitgevoerde werk overeenkomt met de gestelde kwaliteitsverwachtingen en acceptatiecriteria.

7. Afsluiten van een project
Dit proces richt zich op het afsluiten van het project. De activiteiten bestaan voornamelijk uit het voorbereiden van informatie die de stuurgroep nodig heeft voor de formele afsluiting van het project.

8. Opstellen van een plan
Dit proces wordt gedurende het gehele project meerdere malen herhaald. Een plan bestaat uit een beschrijving van wat, hoe, wanneer en door wie een specifiek doel binnen het project bereikt dient te worden. Enkele voorbeelden van dergelijke plannen zijn:

- initiatieplan
- projectplan
- faseplan
- afwijkingsplan

We zien dus dat planning slechts een klein onderdeel is van het beheersen van projecten. In de volgende paragrafen zullen we verder inzoomen op dit specifieke gedeelte van projectbeheersing.

2.5 Planningsvraagstukken

In deze paragraaf zullen we de problemen die bij projectplanning spelen omzetten in een aantal concrete planningsvraagstukken. Dit doen we door eerst een aantal basisvragen op te stellen. Door deze basisvragen vervolgens verder uit te werken komen we vanzelf terecht bij de vraagstukken waar we in geïnteresseerd zijn.
Wanneer we een planner vragen wat de eerste problemen zijn die bij hem opkomen tijdens het plannen van een project, krijgen we vaak het volgende antwoord: “Wat gaan we doen?”, “Wanneer is het af ?” en “Wat gaat het kosten?”. Dit zijn voor het management van de organisatie de drie meest belangrijke zaken betreffende de planning van een project. We zullen deze drie vragen nu verder uitsplitsen in een aantal concrete planningsvraagstukken.

Wat gaan we doen?
Met deze vraag doelen we niet op een gedetailleerde beschrijving van het beoogde projectresultaat, maar op de samenstelling van de activiteiten die uitgevoerd moeten worden voor het bereiken van dit resultaat. Uit deze vraag volgen de volgende planningsvraagstukken:

1. Welke activiteiten dienen uitgevoerd te worden voor het bereiken van het gedefinieerde projectresultaat?
2. Welke relaties bevinden zich tussen deze activiteiten?

Wanneer we deze vragen beantwoord hebben, gaan we ons bezighouden met de benodigde middelen voor de verschillende activiteiten.

Wat gaat het kosten?
De kosten van een project worden bepaald door het gebruik van middelen voor het uitvoeren van de verschillende activiteiten. Dit leidt tot het volgende planningsvraagstuk:

3. Welke en hoeveel middelen zijn er nodig voor het uitvoeren van de verschillende activiteiten binnen het project?

Zodra voor iedere activiteit vastgelegd is welke middelen benodigd zijn, kunnen we beginnen een antwoord te vinden op de vraag wanneer het project voltooid is.

Wanneer is het af?
Dit is de belangrijkste vraag in de projectplanning. In de praktijk is al vaak genoeg gebleken dat het beantwoorden van deze vraag niet gemakkelijk is. Om tot een betrouwbaar antwoord op deze vraag te komen moet er met veel verschillende zaken

rekening gehouden worden. We definiëren de volgende planningsvraagstukken die aan deze vraag gerelateerd zijn:

4. Welke werknemers hebben de tijd en deskundigheid voor de uitvoering van de activiteiten van het project?
5. Welke en hoeveel middelen zijn er wanneer beschikbaar binnen de organisatie?
6. Welke werknemers dienen de verschillende activiteiten uit te voeren en wanneer?

Deze combinatie van vragen geeft ons in principe antwoord op de vraag wanneer het project af is, of liever gezegd, wanneer het project volgens de planning af is. In de praktijk zal in veel gevallen namelijk blijken dat de oorspronkelijke planning niet gehaald wordt. Daarom relateren we alle in deze paragraaf gedefinieerde vraagstukken aan de huidige stand van zaken. We willen dus op ieder willekeurig tijdstip binnen het project, gegeven de huidige stand van zaken een antwoord kunnen krijgen op de vraagstukken, zodat we adequaat op afwijkingen van de planning kunnen reageren.
Daarnaast willen we voor een gemaakte planning iets kunnen zeggen over de risico's op afwijking van deze planning. Daarom definiëren we de nog een tweetal vraagstukken:

7. Hoe groot is de kans op uitloop voor verschillende activiteiten?
8. Wat is het gevolg van het uitlopen van een activiteit op de verdere voltooiing van het project?

We hebben nu acht planningsvraagstukken gedefinieerd. In de volgende paragraaf zullen we een proces beschrijven om met behulp van een systematisch proces een antwoord te vinden op deze vragen.

2.6 Een proces voor projectplanning

In deze paragraaf zullen we een methode beschrijven om tot een planning te komen die de besproken vraagstukken zo volledig mogelijk beantwoord. Samengevat komen deze vraagstukken neer op de volgende vier *kernproblemen*:

1. Wat moet er gebeuren?
2. Door wie moet dit gebeuren en waarmee?
3. Wanneer moet dit gebeuren?
4. Hoe groot is het risico op uitloop voor de verschillende activiteiten?

Het planningsproces draait om het beantwoorden van deze vragen, en het bijwerken van de antwoorden naarmate een project vordert. In deze paragraaf geven we onze visie op de aanpak van deze problemen. We beschouwen projectplanning dus als meer dan alleen het maken van een planning aan de start van een project. Gedurende het gehele project dient de voortgang vergeleken te worden met de planning. Wanneer de realiteit (te veel) afwijkt van de planning dient er ingegrepen te worden. In het bedrijfsleven wordt er daarom vaak gesproken van "planning & control".

2.6.1 Kernactiviteiten van het planningsproces

Voor het 'oplossen' van de kernproblemen definiëren we een aantal *kernactiviteiten*, welke we hier zullen bespreken. Het uitgangspunt van het planningsproces is een doelstelling. In het geval van projectplanning is deze van de vorm "het opleveren van het gedefinieerde resultaat binnen de gestelde periode met het gebruik van beperkte middelen". We beschouwen het specificeren van het resultaat dus niet als onderdeel van het planningsproces, maar zien de hieruit resulterende specificatie als input voor het planningsproces. We onderscheiden de volgende kernactiviteiten:

1. Opstellen *work breakdown structure*
2. Schatten benodigde middelen
3. Inplannen activiteiten
4. Signaleren afwijkingen en bijsturen

2.6.2 Opstellen work breakdown structure

De eerste kernactiviteit die we onderscheiden is het opstellen van een zogenaamde w*ork breakdown structure* (WBS). Een WBS is een resultaat-georiënteerde opdeling van het uit te voeren werk. Simpel gezegd komt het opstellen van een WBS neer op het beantwoorden van de eerste vraag: "wat moet er allemaal gebeuren om tot het gewenste resultaat te komen?". In de praktijk is het opstellen van een WBS echter niet zo simpel. Er zijn twee uitgangspunten om tot een WBS te komen: de productgeoriënteerde WBS en de activiteitgeoriënteerde WBS. In de productgeoriënteerde WBS wordt allereerst vastgelegd welke deelresultaten er achtereenvolgens nodig zijn voor de oplevering van het eindresultaat. Vervolgens wordt er bepaald welke activiteiten er nodig zijn om tot deze deelresultaten te komen en wat de relaties tussen deze activiteiten zijn. De activiteitgeoriënteerde WBS draait dit om en gaat uit van de activiteiten die nodig zijn voor het behalen van het resultaat. Deze tweede methode is vooral geschikt bij kleinere projecten waarbij het aantal *deliverables* relatief beperkt is.

In paragraaf 2.2 hebben we de basis van de WBS al gelegd. We gingen uit van activiteiten die gegroepeerd kunnen worden in fasen en opgedeeld in subactiviteiten. We verfijnen dit enigszinds door de activiteiten in drie categorieën op te delen:

1. In het eenvoudigste geval is een activiteit een concrete *taak*. Hierbij houden we de volgende definitie van een taak aan:

Definitie 4:

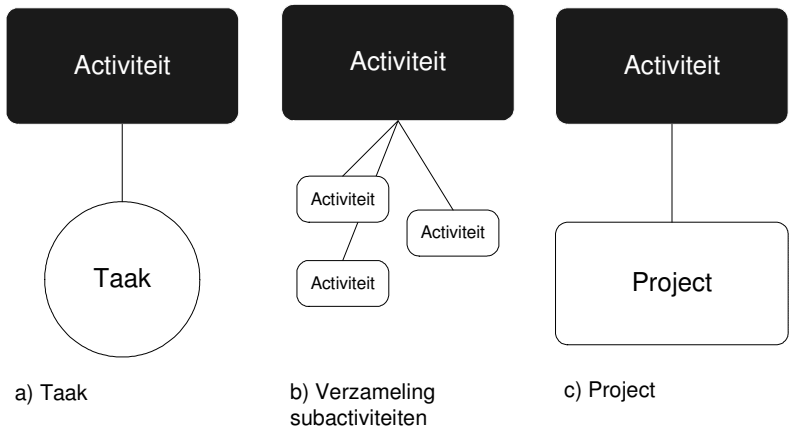
> *Een **taak** is een afgebakende groep werkzaamheden waarvan de verantwoordelijkheid direct bij de uitvoerende medewerker(s) ligt en waarbij alle uitvoerende medewerkers dezelfde rol vervullen.*

2. Wanneer een activiteit complexer en/of groter is, kan de activiteit opgedeeld worden in een aantal subactiviteiten. We zullen in dit geval spreken van een *abstracte activiteit.* Eventueel kan binnen deze activiteit ook een fasering

aangebracht worden. Elk van de subactiviteiten is dan weer te plaatsen binnen één van de fasen van de bovenliggende activiteit.

3. In sommige gevallen zal een activiteit zo groot en complex zijn dat deze als een apart, op zich zelf staand project gedefinieerd wordt.

In de onderstaande figuur worden de beschreven vormen van activiteiten grafisch weergegeven.



Figuur 8: drie verschillende vormen van activiteiten

Tussen de verschillende activiteiten binnen een project bestaan een aantal soorten relaties. We onderscheiden de volgende twee soorten relaties:

1. Relaties van de vorm *bestaat uit*: dit betreft relatie die een abstracte activiteit opbreken in subactiviteiten (zie figuur 6b).

2. *Afhankelijkheden*: tussen de verschillende activiteiten bestaan *expliciete* en *impliciete* afhankelijkheden.

    a. De *expliciete afhankelijkheden* zijn van de vorm "benodigd voor"; sommige activiteiten kunnen alleen uitgevoerd worden als één of meerdere andere activiteiten (in zekere mate) succesvol voltooid zijn. Deze afhankelijkheden maken deel uit van de WBS en zullen dus bij het opstellen hiervan geïdentificeerd moeten worden.
    b. De *impliciete*, "*verborgen*" *afhankelijkheden* hebben te maken met de beperking van middelen. Het kan bijvoorbeeld zo zijn dat twee verschillende activiteiten dezelfde machine nodig hebben. Tussen de twee activiteiten zit dan de impliciete relatie dat ze niet tegelijk uitgevoerd kunnen worden. Deze impliciete afhankelijkheden komen niet terug in de WBS.

2.6.3 Wijzigingen in de WBS

Het opstellen van een WBS vindt in eerste instantie bij de start van het project plaats. Afhankelijk van een aantal eigenschappen van het project zal de aanvankelijke WBS een

verschillende mate van detail bevatten. Zo zal bij sommige projecten relatief duidelijk zijn welke activiteiten uitgevoerd moeten worden, bijvoorbeeld omdat er ervaring is met vergelijkbare projecten. In dit geval kan de eerste versie van de WBS een vrij gedetailleerde beschrijving van de structuur van de uit te voeren activiteiten bevatten. Bij andere projecten is bij aanvang soms zelfs het gewenste resultaat nog vaag. In dit geval zal in de WBS slechts een beschrijving op hoog niveau van de structuur van het project bevatten. De WBS zal dan gedurende het project meerdere malen verfijnd worden. Vaak worden deelresultaten gebruikt als input voor het verfijnen van de WBS van de volgende fase.

Een WBS is dus niet statisch maar wordt gedurende een project vaak meerdere malen aangepast, waarbij deelresultaten en/of activiteiten toegevoegd of geschrapt worden. Hierbij kunnen ook de afhankelijkheden tussen de activiteiten veranderen. Op basis van oorzaak onderscheiden we een aantal soorten wijzigingen:

1. *Beschikbaar komen van informatie*. Deze categorie betreft wijzigingen die voortkomen door het beschikbaar komen van meer informatie met betrekking tot het gewenste resultaat. De activiteiten uit de aanvankelijke WBS worden hierbij verfijnd door ze op te delen in (sub)activiteiten en de afhankelijkheden worden tot op een lager niveau vastgelegd. Bij deze wijzigingen wordt de structuur van de WBS dus niet gewijzigd, maar wordt deze alleen in de diepte uitgebreid door middel van verfijning in subactiviteiten.

2. *Veranderingen in het beoogde resultaat*. Deze veranderingen in het resultaat komen ofwel voort uit veranderende wensen van de opdrachtgever, ofwel uit onduidelijke afspraken tussen de opdrachtgever en de uitvoerende partij. Bij deze wijzigingen in de WBS kunnen zowel activiteiten en/of afhankelijkheden toegevoegd als geschrapt worden. Deze wijzigingen kunnen de structuur van de WBS dus ingrijpend veranderen.

3. *Onvolledigheid of onjuistheid*. Tijdens het opstellen van de aanvankelijke WBS kunnen er zaken over het hoofd gezien of verkeerd beoordeeld zijn. Deze wijzigingen kunnen allerlei gevolgen hebben op de structuur van de WBS.

Al met al is het opstellen van een WBS een lastige, doch belangrijke taak. Door in de beginperiode van een project al goed na te denken over de structuur van de activiteiten kunnen mogelijke knelpunten al in een vroeg stadium geïdentificeerd worden, hetgeen de succeskans van een project aanzienlijk kan vergroten.

3.6.4 Schatten benodigde middelen

Naast het opstellen van de WBS dient er voor alle activiteiten geïdentificeerd te worden welke middelen er nodig zijn voor de uitvoering van deze activiteiten en hoeveel. Hierbij gaan we uit van de in paragraaf 2.2 genoemde categorieën van middelen: mensen en deskundigheid, tijd, materieel en geld. We definiëren vier activiteiten voor het tot stand komen van deze schattingen:

1. *Bepalen benodigde deskundige vaardigheden*. Is er personeel met deze deskundigheid aanwezig of moet er extra personeel worden aangetrokken? In sommige gevallen is het misschien wel beter om bepaalde specialistische activiteiten volledig uit te besteden. In de praktijk zal dit niet op basis van 'losse' vaardigheden gebeuren, maar wordt er gekeken naar groepen van samenhangende vaardigheden. Op basis van deze groepen worden profielen gemaakt. Deze profielen komen vaak in grote mate overeen met 'gewone' functieprofielen binnen het uitvoerende bedrijf.

2. *Schatten benodigde tijd*. De benodigde tijd wordt geschat in manuren, gedifferentieerd naar het vereiste profiel van de uitvoerende medewerkers. Dit zal in de meeste gevallen in eerste instantie gebeuren door middel van schattingen op relatief hoog niveau door het management, eventueel geadviseerd door specialisten. Afhankelijk van de mate van detail van de WBS zullen de schattingen ook op lagere niveaus plaatsvinden. Naarmate het niveau van de activiteiten daalt, dienen er steeds meer uitvoerende werknemers betrokken te worden bij het maken van deze schattingen. Uiteindelijk kunnen de schattingen op de verschillende niveaus met elkaar vergeleken worden. Wanneer er hierbij inconsistenties geconstateerd worden, dient er onderzocht te worden waar deze inconsistenties uit voortkomen en zal de schatting op één of meerdere niveaus aangepast moeten worden. In de praktijk blijkt dat het zelfs voor ervaren managers en specialisten ontzettend moeilijk is om een goede inschatting te maken van hoe lang een activiteit gaat duren. In sommige gevallen kan de aanwezigheid van data over, en ervaring met vergelijkbare activiteiten uitkomst bieden.

3. *Schatten benodigd materieel*. Er dient gekeken te worden wat beschikbaar is en wat eventueel besteld moet worden. Extra aandacht dient hierbij uit te gaan naar materieel dat niet bij bekende leveranciers besteld kan worden. Het benodigde materieel is onder te verdelen in drie categorieën: *grondstoffen*, *infrastructuur* en *overige hulpmiddelen*.

4. *Uitgedrukken in geld*. Er dient een budget opgesteld te worden waarbinnen het project uitgevoerd behoort te worden. Dit budget speelt een cruciale rol in de besluitvorming betreffende de goedkeuring van het project. De stuurgroep en het management van de organisatie zullen grotendeels op basis van het budget al dan geen groen licht geven voor de start van het project. Vooral bij grotere bedrijven kan het voorkomen dat managers met opzet te lage inschattingen doen om zo toestemming te krijgen een bepaald project te starten.

3.6.5 Inplannen activiteiten

Wanneer de WBS opgesteld is en de benodigde middelen voor de verschillende activiteiten bepaald zijn kan er begonnen worden aan het feitelijke plannen van de activiteiten. De vraag "Wanneer en door wie worden de verschillende activiteiten uitgevoerd?" staat centraal in dit gedeelte van het planningsproces. Voor alle activiteiten uit de WBS zal een verantwoordelijke medewerker aangewezen moeten worden. Op de hogere niveaus zal dit iemand uit het management zijn, op het niveau van taken zal het de

uitvoerende medewerker zijn. Ook moet elke activiteit een geplande start en einddatum krijgen. Hierbij moet er met een aantal zaken rekening gehouden worden:

1. *Expliciete afhankelijkheden*. Deze afhankelijkheden die in de WBS zijn opgenomen bepalen in grote mate de volgorde van de uit te voeren activiteiten.

2. *Beschikbaar komen van middelen*. Het kan voorkomen dat er middelen benodigde zijn die pas gedurende het project beschikbaar komen. Deze middelen hebben vaak betrekking op grondstoffen die geleverd moeten worden door externe leveranciers, maar het kan bijvoorbeeld ook om medewerkers gaan die pas vanaf een bepaalde datum op het project ingezet kunnen worden.

3. *Risico's*. Bepaalde typen activiteiten hebben een grotere kans op uitloop dan andere. Een degelijke planning houdt enigszins rekening met deze risico's, bijvoorbeeld door de activiteiten met een groter risico uit te spreiden over verschillende medewerkers. In veel gevallen zal dit slechts gebeuren op basis van ervaring van de planner en het gebruik van gezond verstand en zullen deze risico's in de planning opgenomen worden door middel van ruimere schattingen voor de duur van de activiteiten. Wanneer activiteiten binnen de projecten van een bedrijf een sterk herhalend karakter hebben kan er met behulp van een statistische analyse gekeken worden naar de risico's van de verschillende soorten activiteiten.

Al met al is het inplannen van activiteiten een complex proces, zeker voor grote projecten. Door de grootte en de complexe structuur van het project is het vaak lastig om een goed overzicht te krijgen op de activiteiten en de (impliciete) afhankelijkheden tussen deze activiteiten. Het te ontwikkelen model zal zich in grote mate richten op de ondersteuning van deze kernactiviteit.

3.6.6 Signaleren en Bijsturen

Het signaleren van afwijkingen op de planning en het bijsturen van de uitvoering van het project zijn belangrijke onderdelen van het planningsproces. Het blijkt in de praktijk onvermijdelijk dat op een zeker moment gedurende het project de realiteit afwijkt van de planning. Voor deze afwijkingen zijn een aantal mogelijke oorzaken te identificeren:

1. *Wijzigingen in de WBS*. Het is duidelijk dat wanneer er een nieuwe activiteit toegevoegd wordt aan de WBS, dit gevolgen heeft op de planning.

2. *Uitlopen van activiteiten*. Het komt maar al te vaak voor dat activiteiten meer tijd in beslag nemen dan aanvankelijk geschat was.

3. *Later beschikbaar komen van middelen*. Hierdoor kunnen bepaalde activiteiten vaak pas later gestart worden, en zullen deze (indien er geen actie ondernomen wordt) in de meeste gevallen ook later afgerond worden.

4. *Onvoorziene gebeurtenissen*. Hieronder verstaan we alle gebeurtenissen die invloed op de planning hebben en die volledig buiten de controle van alle bij het project betrokken partijen vallen.

Het tijdig signaleren van afwijkingen die een bedreiging vormen voor de succesvolle voltooiing van het project is een essentiële maar lastige taak. Wanneer het halen van de planning door één van de genoemde oorzaken in gevaar komt, dient er actie ondernomen te worden door het management. We identificeren de volgende vier methoden om hiermee om te gaan:

1. *Alsnog opleveren*. Vaak zal getracht worden het gedefinieerde resultaat alsnog op de geplande deadline op te leveren. Dit kan bijvoorbeeld door het toekennen van extra middelen aan één of meerdere activiteiten. Dit zal helaas niet altijd mogelijk zijn.

2. *Uitstellen*. Wanneer het gedefinieerde resultaat onder geen mogelijk meer binnen de gestelde deadline gerealiseerd kan worden is dit vaak de enige mogelijkheid. Indien dit het geval is, is het zaak om zo snel mogelijk contact met de opdrachtgever op te nemen, het kan namelijk voor komen dat uitstel voor de opdrachtgever geen optie is.

3. *Aanpassen van de resultaatdefinitie*. In sommige gevallen zal een opdrachtgever deze optie verkiezen boven het uitstellen van de deadline. Vooral in de softwarewereld is dit vaak een overweegbare optie.

4. *Afblazen*. Wanneer geen van deze drie opties tot een bevredigende oplossing leidt, zal in sommige gevallen het project volledig afgeblazen moeten worden. In dit geval zijn beide partijen verliezer.

Vooral bij grote projecten is het aan te raden dat er van tevoren afspraken met de opdrachtgever gemaakt worden betreft de te ondernemen acties bij afwijkingen op de planning. Dit gebeurt dan in de vorm van een afwijkingsplan, waarin beschreven staat wat er bij mogelijke afwijkingen moet gebeuren en welke partij de (financiële) verantwoordelijk heeft voor het omgaan met deze afwijkingen.

Al met al concluderen we dat het tijdig signaleren van knelpunten en het adequaat ingrijpen veel problemen kan voorkomen en dus van groot belang is.

2.7 Conclusie

Projecten zijn te beschouwen als verzamelingen activiteiten die met gebruik van beperkte middelen uitgevoerd dienen te worden om een vooraf gespecificeerd, uniek resultaat te bereiken. Het plannen van projecten draait om het beantwoorden van de vragen “wat moet er gebeuren?”, “door wie en waarmee moet dit gebeuren?” en “wanneer moet dit gebeuren?”. Om op een systematische wijze een antwoord op deze vragen te vinden hebben we een proces beschreven dat uit de volgende vier onderdelen bestaat:

1. Opstellen Work Breakdown Structure:
2. Schatten benodigde middelen
3. Inplannen activiteiten
4. Signaleren en bijsturen

Het model dat we gaan opstellen zal primair gericht zijn op het geven van ondersteuning voor deze vier processen.

Hoofdstuk 3: Een model voor projectplanning

In dit hoofdstuk bouwen we het model op. Om te beginnen beschrijven we in paragraaf 3.1 de verschillende stappen van het model. In paragraaf 3.2 introduceren we de verschillende wiskundige onderdelen waarmee we het model gaan opbouwen. Vervolgens beginnen we in paragraaf 3.3 met het feitelijke opstellen van het model. In deze paragraaf definiëren we de op organisatie niveau benodigde modelonderdelen. In paragraaf 3.4, 3.5 en 3.6 werken we de beschreven stappen verder uit. Vervolgens breiden we in paragraaf 3.7 het model uit om met wijzigingen in de planning om te kunnen gaan. Ten slotte geven we in paragraaf 3.8 een beknopt overzicht van de functionaliteit van het opgestelde model.

3.1 Een stappenplan voor projectplanning

We hebben een proces beschreven voor het tot stand komen en bijwerken van een planning (paragraaf 2.6). In deze paragraaf zullen we dit proces verder uitwerken tot een stappenplan voor projectplanning. Dit stappenplan bestaat uit drie onderdelen en vormt de basis van ons model.

1. *Definiëren van het project*. Tijdens deze stap wordt vastgelegd wat er moet gebeuren, hoe de verschillende activiteiten zich tot elkaar verhouden en wat allemaal benodigd is voor de uitvoering van deze activiteiten. De uitvoering van deze stap bestaat voornamelijk uit het geven van input door de gebruiker. We noemen deze fase de *projectdefinitie fase*.

2. *Plannen*. De tweede stap betreft het oplossen van het model, oftewel het maken van een planning voor het project. Dit gebeurt in een aantal stappen.

    a. Ten eerste wordt de opgegeven WBS geanalyseerd. Dit gebeurt door middel van de zogenaamde kritieke pad analyse.
    b. Met behulp van deze analyse wordt er een *theoretisch optimale planning* bepaald.
    c. Vervolgens worden er werknemers toegekend aan de activiteiten.
    d. Wanneer dit gebeurd is worden de activiteiten ingeroosterd.

3. *Bijsturen*. Zodra alle activiteiten ingeroosterd zijn, is de eerste versie van de planning voltooid. Deze eerste versie van de planning kan gedurende het project op verschillende niveaus aangepast worden. Het aanpassen van de planning beschouwen we als een wezenlijk onderdeel van het model.

3.2 De basisonderdelen van het model

Het model dat we gaan opstellen bestaat uit een aantal verschillende onderdelen. In deze paragraaf zullen we de verschillende onderdelen die we in het model gaan gebruiken introduceren.

3.2.1 Entiteiten

Om te beginnen hebben we binnen ons model te maken met verschillende *entiteiten*. Voorbeelden van entiteiten zijn projecten, werknemers, activiteiten etc. We noteren deze entiteiten door middel van één of meerdere hoofdletters, en geven deze een index om specifieke instanties uit elkaar te kunnen houden. Zo zouden we een activiteit bijvoorbeeld kunnen noteren als:

Activity: A_i

Een entiteit kan bepaalde eigenschappen hebben. Eigenschappen die betrekking op één enkele entiteit hebben, leggen we vast door middel van *attributen*. Zo zouden we aan een activiteit bijvoorbeeld het attribuut *duur* kunnen toevoegen. Dit noteren we als volgt:

A_i.duur

Voor elk attribuut definiëren we een type, om aan te geven welke waarden het attribuut kan aannemen. Het attribuut duration is van het type *positive integer*. We definiëren de volgende attribuut types:

- *integer (de natuurlijke getallen)*
- *positive integer (alle positieve natuurlijke getallen)*
- *date (een datum)*
- *date-time (een datum-tijd combinatie)*
- *entity refference (een verwijzing naar een instantie van een entiteit)*
- *percentage (een percentage)*
- *binary (0 of 1)*

Naast een type hebben we om de waarde van een attribuut te kunnen interpreteren vaak ook een eenheid nodig. In het geval van *duration* zou dit bijvoorbeeld dagen of uren kunnen zijn. De mogelijke eenheden worden niet alleen vastgelegd door het type van het attribuut, maar vooral ook door de interpretatie van het attribuut. We zullen bij het definiëren van attributen dus aan moeten geven of er een eenheid is, en zo ja welke dat is.

3.2.2 Verzamelingen

We introduceren een aantal soorten verzamelingen in ons model. We beginnen met de zogenaamde *entiteiten verzamelingen*. In deze verzamelingen groeperen we alle entiteiten van een bepaald type. We noteren de verzamelingen met dezelfde lettercombinatie als de entiteit, alleen dan in cursief en zonder index. We kunnen de activiteiten dus als volgt weergeven:

Activities: $A_i \in A$

Naast de *entiteiten verzamelingen*, die per definitie alle entiteiten van een bepaald type bevatten, hebben we ook zogenaamde *afgeleide verzamelingen*. Een afgeleide verzameling is een deelverzameling van één van de entiteiten verzamelingen en wordt gebruikt om op een handige manier alle entiteiten met een specifieke eigenschap weer te geven. We noteren deze afgeleide verzamelingen als *functies* op de entiteiten verzameling of op een specifieke entiteit uit deze verzameling. De functie noteren we cursief en in hoofdletters. Hiernaast kan de verzameling één of meerdere indices hebben, waarvan de betekenis context afhankelijk is.
Zo zouden we bijvoorbeeld een afgeleide verzameling van *A* kunnen definiëren, met daarin alle voltooide activiteiten. Deze verzameling kunnen we als volgt noteren:

Completed activities: COMPLETED(A)

Dit is dus een functie op de entiteiten verzameling *A*. We zouden ook een verzameling kunnen definiëren met daarin alle subactiviteiten van een zekere activiteit A_k. Deze verzameling is dan dus een functie op een specifieke entiteit. We noteren dit als volgt:

A_k's Subactivities: SUB(A_k)

In sommige gevallen kan de inhoud van de afgeleide verzameling expliciet gedefinieerd worden. In andere gevallen zijn hier impliciete (recursieve) notaties of zelfs algoritmen voor vereist.

3.2.3 Relaties

We hebben eerder attributen gedefinieerd als eigenschappen die betrekking hebben op één enkele entiteit. In ons model willen we echter ook eigenschappen kunnen opnemen die betrekking op meerdere entiteiten hebben. Dit doen we door middel van *relaties*. We definiëren een relatie als zijnde een functie op minimaal twee (instanties van) entiteiten. Evenals de attributen hebben de relaties een type en eventueel ook een eenheid. We noteren de relaties als functies op verschillende entiteiten, waarbij we de functienamen in kleine letters opschrijven. We zouden bijvoorbeeld het vaardigheidsniveau van een bepaalde werknemer (E_k) in een bepaalde competentie (C_l) kunnen weergeven als een percentage. Dit noteren we dus:

skill(E_k, C_l)

Het type van deze relatie is dus *percentage*, hierbij hoort geen eenheid.

3.2.4 Algoritmen

Het volgende onderdeel van het model zijn de *algoritmen*. Een algoritme bestaat uit één of meerdere functies, die samen één of meerdere veranderingen aanbrengen in de hierboven gedefinieerde modelonderdelen. Om de leesbaarheid van de algoritme zo groot mogelijk te houden hebben we ervoor gekozen om in de algoritmen gebruik te maken pseudo code in plaats van exacte wiskundige notatie.

3.2.5 Wijzigingen in het model

De entiteiten verzamelingen in ons model zijn dynamisch. Dat wil zeggen dat we entiteiten kunnen toevoegen en verwijderen uit deze verzamelingen. Bij het verwijderen van entiteiten kunnen we echter op een probleem stuiten. We kunnen namelijk te maken hebben met relaties en andere entiteiten die verwijzingen naar de betreffende entiteit hebben. Er zijn een aantal verschillende mogelijkheden om hiermee om te gaan. We willen voor de verschillende soorten relaties en attributen kunnen vastleggen hoe we hiermee omgaan. We zullen nu, in termen van ons model, een mechanisme introduceren om hier mee om te gaan. Voor elke combinatie van entiteit en relatie dan wel attribuut, definiëren we de volgende relatie:

del(Ent_k ε *ENT*, X) := de actie met betrekking tot relatie/attribuut X, die ondernomen dient te worden bij het verwijderen van entiteit Ent_k uit entiteitenverzameling *ENT*.

We definiëren de volgende twee waarden die del(Ent_k ε *ENT*, X) kan aannemen:

1. "delete": indien de entiteit Ent_k verwijderd wordt, dient ook de relatie X, dan wel de bij attribuut X behorende entiteit verwijderd te worden.
2. "restrict": entiteit Ent_k kan niet worden verwijderd zolang er relaties, dan wel attributen X bestaan die naar Ent_k verwijzen.

We hebben nu de verschillende bouwstenen van ons model geïntroduceerd. Samengevat bestaat het model dus uit *entiteiten* met *attributen*, die gegroepeerd zijn in *verzamelingen*. Op deze entiteiten/verzamelingen bestaan functies die *afgeleide verzamelingen* opleveren. Ook zijn *relaties* tussen verschillende entiteiten in de vorm van functies in het model opgenomen. Deze genoemde onderdelen krijgen een concrete invulling door middel van gebruikers invoer en *algoritmen*.

3.3 Het model op organisatieniveau

Om projecten te kunnen plannen hebben we informatie nodig over de uitvoerende organisatie. In deze paragraaf wordt deze informatie in termen van ons model vastgelegd. Het betreft voornamelijk informatie over de werknemers van de organisatie en hun vaardigheden. Om te beginnen nemen we deze werknemers in ons model op door middel van de entiteit *employee*:

Employee: E_i ε *E*

Om te kunnen roosteren willen we in ons model informatie opnemen over de werktijden van de werknemers. Hiervoor is het voldoende om te weten hoeveel uren de werknemer op een gegeven dag werkt. Daarom nemen we het volgende attribuut in ons model op:

E_i.hrs_j (integer): het aantal uren dat werknemer E_i werkt op dag j van de week.

Hierbij is j een getal tussen 1 en 7, waarbij 1 staat voor maandag en 7 voor zondag. Om in ons model op een handige manier met groepen werknemers om te kunnen gaan, definiëren we zogenaamde *resource pools*. Een resource pool is in feite niets meer dan een *verzameling* van werknemers.

Resource Pool: RP_i ε *RP*

Elke resource pool is een deelverzameling van de werknemersverzameling *E*. Om vast te leggen welke werknemers er in welke resource pools zitten maken we de volgende *binary* relatie aan:

emprp(E_i,RP_j) (binary) = 1 als werknemer E_i deel uitmaakt van resource pool RP_j, 0 anders

Met behulp van deze relatie definiëren we een afgeleide verzameling met daarin alle werknemers binnen de resource pool:

E(RP_q) = {E_p ε *E* | emprp(E_p, RP_q) = 1}

We willen iets kunnen zeggen over de vaardigheden van de verschillende werknemers. Dit doen we met behulp van de entiteit *competence*.

Competence: C_i ε *C*

Vervolgens koppelen we de werknemers en competenties aan elkaar. Hierbij willen we niet alleen weten of een werknemer een bepaalde competentie bezit, maar willen het niveau van de vaardigheid ook kwantificeren. Hiervoor definiëren we een *relatie* van het type *percentage* op de combinatie werknemer en competentie:

skill(E_i,C_j) ε (0, 100): "het *vaardigheidsniveau* van werknemer E_i in competentie C_j"

Een waarde van 0 betekent dus dat werknemer E_i de betreffende competentie C_j niet bezit. Voor de handigheid definiëren we voor elke competentie een *afgeleide verzameling*, die bestaat uit alle werknemers die de competentie (in zekere mate) bezitten:

E(C_i) := {E_k ε *E* | *skill*(E_k,C_i)> 0}: "de verzameling van werknemers die competentie C_i bezitten".

Wanneer we de competenties gaan gebruiken om aan te geven aan welke eisen een werknemer moet voldoen om een bepaalde activiteit uit te kunnen voeren, willen we de competenties niet één voor één uit hoeven zoeken. Om dit te voorkomen definiëren we rollen:

Role: RL_i ε *RL*

We modelleren de rollen als verzamelingen van competenties. Net als bij de resource pools definiëren we een *binary* relatie om aan te geven welke competentie bij welke rol hoort.

`crl(`C_i`, `RL_j`) (binary) = 1 als competentie `C_i` deel uitmaakt van rol `RL_j`, 0 anders`

Dit geeft ons voor iedere rol een afgeleide verzameling met daarin de bijbehorende competenties:

$C(RL_q) = \{C_p \mid rlc_{p,q} = 1\}$: de benodigde competenties voor het vervullen van rol RL_q.

Hiermee hebben we het eerste gedeelte van ons model opgesteld. We zullen het in deze paragraaf geïntroduceerde gedeelte van het model nu uitwerken voor ons voorbeeldproject.

3.3.1 Het voorbeeldproject

Om de verschillende onderdelen van het model nader te illustreren, maken we gebruik van een voorbeeldproject. Hiervoor gebruiken we een vereenvoudigde versie van de klantprojecten van Efficiency Online. Naarmate het model verder opgebouwd wordt werken we het voorbeeld verder uit. In deze sectie geven we invulling aan het gedefinieerde organisatorische onderdeel van het model.

We beginnen met de werknemersverzameling *E*. In de onderstaande tabel vinden we de werknemers van Efficiency Online.

Employee E_i	Naam
E_1	Pascal Cramer
E_2	Wouter Ewalds
E_3	Sjoerd Geraedts
E_4	Tom Hendrix
E_5	Jeroen de Hertog
E_6	Otto Moerbeek
E_7	Wouter Radder
E_8	Michel Schaake
E_9	Quirine Veth
E_{10}	Daniel van der Wallen

Tabel 1: de werknemers verzameling *E*.

Merk op dat de namen van de werknemers strikt gezien geen deel van het model uitmaken, maar zijn toegevoegd om de leesbaarheid te verhogen. Voor het gemak gaan we er vanuit dat alle werknemers standaard werkweken van 40 uur maken. Met betrekking tot $E_i.hrs_j$ (het aantal uren dat werknemer E_i werkt op de j-de dag van de week) geldt dus:

- $E_i.hrs_j = 8$ voor alle j, voor i ε {1, 2, 3, 4, 5}
- $E_i.hrs_j = 0$ voor alle j, voor i ε {6, 7}

Vervolgens geven we invulling aan de resource pool verzameling *RP*. Voor elk van de afdelingen van het bedrijf (zie paragraaf 1.1) maken we een resource pool aan. Verder maken we een resource pool development aan die bestaat uit zowel de frontoffice als de backoffice. Dit levert ons de volgende *resource pools* op:

Resource Pool RP_i	Naam
RP_1	Backoffice
RP_2	Directie
RP_3	Development
RP_4	Front Office
RP_5	Sales

Tabel 2: de resource pool verzameling *RP*.

In ons model wordt door middel van de relatie $\texttt{emprp}(E_i, RP_j)$ vastgelegd welke werknemers deel uit maken van de verschillende resource pools. We zijn in principe alleen geïnteresseerd zijn in de combinaties werknemer en resource pool waarvoor deze relatie gelijk aan 1 is, en zullen de volledige matrix met de waarden van $\texttt{emprp}(E_i, RP_j)$ dan ook achterwegen laten. In plaats daarvan geven we in de onderstaande tabel weer welke werknemers in welke resource pools zitten.

Resource Pool RP_i	$E(RP_i)$
Backoffice (RP_1)	Jeroen de Hertog (E_5) Otto Moerbeek (E_6) Daniel van der Wallen (E_1)
Directie (RP_2)	Sjoerd Geraedts (E_3) Daniel van der Wallen (E_{10})
Development (RP_3)	Pascal Cramer (E_1) Wouter Ewalds (E_2) Tom Hendrix (E_4) Jeroen de Hertog (E_5) Otto Moerbeek (E_6) Wouter Radder (E_7) Michel Schaake (E_8) Daniel van der Wallen (E_{10})
Front Office (RP_4)	Pascal Cramer (E_1) Wouter Ewalds (E_2) Sjoerd Geraedts (E_3) Tom Hendrix (E_4) Wouter Radder (E_7)
Sales (RP_5)	Sjoerd Geraedts (E_3) Quirine Veth (E_9)

Tabel 3: werknemers per resource pool.

De volgende verzameling die we invullen is de competentie verzameling *C*. In tabel 4 vinden we een lijst met alle competenties. Voor iedere competentie is de afgeleide verzameling $E(C_i)$, met daarin alle werknemers die de betreffende competentie bezitten, weergegeven.

Comptences C_i	Naam	$E(C_i)$
C_1	Projectmanagement technieken	Sjoerd Geraedts (E_3) Daniel van der Wallen (E_{10})
C_2	Programmeren (basis)	Pascal Cramer (E_1) Wouter Ewalds (E_2) Tom Hendrix (E_4) Jeroen de Hertog (E_5) Otto Moerbeek (E_6) Wouter Radder (E_7) Michel Schaake (E_8) Daniel van der Wallen (E_{10})
C_3	Programmeren (gevorderd)	Pascal Cramer (E_1) Jeroen de Hertog (E_5) Otto Moerbeek (E_6) Daniel van der Wallen (E_{10})
C_4	Configureren met de EOL Generator	Pascal Cramer (E_1) Wouter Ewalds (E_2) Sjoerd Geraedts (E_3) Tom Hendrix (E_4) Jeroen de Hertog (E_5) Otto Moerbeek (E_6) Wouter Radder (E_7) Michel Schaake (E_8) Daniel van der Wallen (E_{10})
C_5	Programmeren aan de EOL Generator	Wouter Ewalds (E_2) Jeroen de Hertog (E_5) Otto Moerbeek (E_6) Daniel van der Wallen (E_{10})
C_6	Datamodelleren	Pascal Cramer (E_1) Wouter Ewalds (E_2) Sjoerd Geraedts (E_3) Tom Hendrix (E_4) Wouter Radder (E_7)
C_7	Informatieanalyse	Pascal Cramer (E_1) Wouter Ewalds (E_2) Sjoerd Geraedts (E_3) Tom Hendrix (E_4) Wouter Radder (E_7)
C_8	Sales technieken	Sjoerd Geraedts (E_3) Quirine Veth (E_9)
C_9	Webdesign	Tom Hendrix (E_4)

Tabel 4: de competentie verzameling *C*, met voor iedere competentie de lijst met werknemers die de competentie bezitten.

Ten slotte zullen we een aantal rollen aanmaken die we nodig hebben voor de verschillende activiteiten van het project. In tabel 5 vinden we de benodigde rollen en per rol de vereiste competenties.

Roles RL_i	Naam	$C(RL_i)$
RL_1	Project Manager	Projectmanagement technieken
RL_2	Ontwerper	Informatieanalyse Datamodelleren
RL_3	Web Designer	Webdesign
RL_4	Applicatie Ontwikkelaar	Configureren met de EOL Generator Programmeren (basis)
RL_5	Sales Manager	Sales technieken
RL_6	Generator Programmeur	Programmeren (basis) Programmeren (gevorderd) Programmeren aan de EOL Generator

Tabel 5: de rollen verzameling *RL*, met voor iedere rol de lijst met vereiste competenties.

We hebben nu alle organisatorische gegevens die we in ons model gedefinieerd hebben ingevuld voor een vereenvoudiging van de situatie binnen Efficiency Online. We zullen nu in de volgende paragraaf ons model uitbreiden met de projectdefinitie.

3.4 De projectdefinitie

In deze paragraaf zullen we deze fase nader bespreken en uitwerken.
Deze stap bestaat uit de kernactiviteiten “opstellen WBS” en “schatten benodigde middelen”. We zetten het model dusdanig op dat deze activiteiten zowel gescheiden als door elkaar heen plaats kunnen vinden. Concreet zal de uitvoering van deze fase neerkomen op het definiëren en specificeren van activiteiten en relaties tussen deze activiteiten.

3.4.1 Definiëren van activiteiten

De basis van de projectdefinitie wordt gevormd door activiteiten. Daarom definiëren we de entiteit *activity*:

$Activity: A_i \varepsilon A$

We hebben we de activiteiten opgedeeld in drie categorieën, te weten abstracte activiteiten, taken en deelprojecten (zie paragraaf 2.6). Om de complexiteit van het model binnen de perken te houden kiezen we er in eerste instantie voor om deelprojecten buiten het model te laten. We zullen dus uitgaan van abstracte activiteiten en taken. Dit leidt tot onze eerste modelaanname:

Modelaanname 1:
Alle activiteiten binnen een project zijn ofwel concrete taken, ofwel abstracte activiteiten die onder te verdelen zijn in subactiviteiten.

In de projectdefinitie fase dient het betreffende project volledig opgebroken te worden in activiteiten uit deze twee categorieën. Zo ontstaat er een boomstructuur van activiteiten. De wortel in deze boom is het project zelf. Onder de wortel zitten, afhankelijk van de grootte van het project, verschillende niveaus van activiteiten. In ons model definiëren we hiervoor voor elk project een verzameling met activiteiten:

Project Activities: *PROJECT*(A_p) := "de verzameling van activiteiten binnen project A_p".

Om de inhoud van deze verzameling (wiskundig) vast te leggen hebben we binnen het model een aantal andere onderdelen nodig. Om te beginnen leggen we de genoemde boomstructuur binnen deze verzameling vast. Dit doen we door middel van de volgende *relatie*:

$sub(A_i,A_j)$ ε *SUB*

De interpretatie hiervan is als volgt:

$sub(A_i,A_j)$ (binary) = 1 als A_j een subactiviteit van A_i is, 0 anders.

Om handig met deze structuur om te kunnen gaan definiëren we voor ieder project A_p een afgeleide verzameling met daarin alleen de relevante subrelaties, en voor iedere activiteit A_i een afgeleide verzameling met daarin alle onderliggende activiteiten.

SUB(A_p) := {$sub(A_i,A_j)$ | A_i, A_j ε *PROJECT*(A_p) & $sub(A_i,A_j)$ = 1}
de verzameling van subactiviteit-relaties binnen project A_p

CHILD(A_i) := {A_k ε *A* | *sub*(A_i,A_k)=1 OR ($sub(A_i,A_q)$=1 && A_k ε *CHILD*(A_q))}
alle onderliggende activiteiten van activiteit A_i

De inhoud van *PROJECT*(A_p) leggen we vast als:

PROJECT(A_p) = CHILD(A_p) v {A_p}

We stellen als eis dat alle laagste niveau activiteiten uit deze boom *taken* zijn. Dit is nodig omdat we deze laagste niveau activiteiten willen gaan inroosteren. We willen kunnen rekenen met de duur van deze activiteiten. Een abstracte, niet verder gespecificeerde activiteit kunnen we niet concreet inplannen omdat we informatie missen over de verdeling van de subactiviteiten. Bovendien kiezen we er in deze eerste versie van het model voor om aan te nemen dat dit één-mans-taken zijn. Deze eis leidt tot onze tweede modelaanname:

Modelaanname 2:
Alle laagste niveau activiteiten in de WBS zijn één-mans-taken.

Voor later gebruik binnen ons model definiëren we de volgende *afgeleide verzameling*:

- $TASKS(A_i)$:= $\{A_k \in CHILD(A_i) \mid sub(A_i,A_k) = 0$ voor alle $k\}$
  de laagste niveau activiteiten van project A_p, oftewel de taken

Het definiëren van de activiteiten bestaat dus uit het herhaaldelijk opdelen van een abstracte activiteit in subactiviteiten, net zolang totdat alle activiteiten op het laagste niveau één-mans-taken zijn. In ons model komt dit neer op het invullen van de verzamelingen $PROJECT(A_p)$ en $SUB(A_p)$.

3.4.2 Specificeren van activiteiten

Om te kunnen plannen hebben we gegevens nodig over de benodigde middelen voor de uitvoering van de verschillende activiteiten. We hebben we de volgende vier categorieën van middelen gedefinieerd (zie paragraaf 2.1):

1. Mensen en deskundigheid
2. Tijd
3. Materieel
4. Budget

We zullen ons binnen het model in eerste instantie beperken tot de eerste twee categorieën, dit geeft ons een volgende modelaanname:

> ***Modelaanname 3:***
> *De benodigde middelen voor de uitvoering van een activiteit bestaan uit de categorieën "tijd" en "mensen en deskundigheid".*

We hebben dus input van de gebruiker nodig betreffende de benodigde middelen uit deze twee categorieën. We kiezen ervoor de gebruiker de mogelijkheid te geven om voor elke activiteit uit de WBS kunnen we een aantal gegevens invullen die betrekking hebben op benodigde middelen. Om het project te kunnen plannen is het echter niet noodzakelijk dat deze gegevens voor alle activiteiten ingevuld worden. Alleen voor het laagste niveau is dit het geval, voor alle abstracte activiteiten is het geven van deze informatie optioneel.

Tijd

We zullen nu de benodigde input verder specificeren, te beginnen met de categorie "tijd". Voor elke activiteit uit de WBS kunnen vier gegevens ingevuld worden die betrekking hebben op deze categorie, te weten:

1. Duur
2. Minimale startdatum
3. Maximale einddatum
4. Minimale achtereenvolgende werkduur

In ons model creëren we voor de entiteit *activity* de volgende drie attributen:

- A_i.duration (integer): de duur van activiteit A_i in uren.
- A_i.minsdate (date): de minimale startdatum van activiteit A_i.
- A_i.maxedate (date): de maximale einddatum van activiteit A_i.
- A_i.minsuchours (integer): minimale achtereenvolgende werkduur van activiteit A_i.

Onder de duur van een activiteit verstaan we het totaal aantal arbeidsuren benodigd voor de uitvoering van de activiteit. Om een project te kunnen plannen hebben we voor alle *taken* uit het project een schatting van de duur van de taak nodig. Hieruit ontstaat onze eerste invoereis*:*

Invoereis 1:
Voor alle taken uit het project dient een schatting voor de duur ingevuld te worden:
Voor project A_p: voor alle A_k ε *TASKS(A_p)*: A_k.duration ≥ 0

Ook voor de abstracte activiteiten kan een schatting van deze duur als input gegeven worden. Wanneer er voor een abstracte activiteit een duur opgegeven is moet gelden dat deze duur consistent is met de duur van de onderliggende activiteiten. Concreet komt dit erop neer dat de duur van een abstracte activiteit gelijk moet zijn aan de som van de duur van alle onderliggende *taken.* Dit is onze tweede invoereis:

Invoereis 2:
Voor elke abstracte activiteit geldt: de schatting van duur van de activiteit is ofwel ongedefinieerd, ofwel gelijk aan de som van de duur van alle direct onderliggende activiteiten:

```
foreach (project Ap) {
   foreach(Ak in PROJECT(Ap) – TASKS(Ap)){
      Ak.duration = SUM(Al in CHILD(Ak)): Al.duration
      OR
      Ak.duration = 0
   }
}
```

Naast de duur kunnen we voor alle activiteiten ook een minimale startdatum en een maximale einddatum opgeven. Het opgeven van een startdatum (respectievelijk einddatum) betekent dat de uitvoering van de activiteit niet voor (respectievelijk na) deze datum plaats kan vinden. We stellen twee invoereisen op die ervoor zorgen dat deze data voor abstracte activiteiten consistent zijn met de onderliggende activiteiten.

Invoereis 3:
Voor elke abstracte activiteit geldt: de minimale startdatum van de activiteit is ofwel ongedefinieerd, ofwel kleiner of gelijk aan de kleinste minimale startdatum van alle onderliggende activiteiten:

```
foreach (project Ap) {
      foreach(Ak in PROJECT(Ap) -  TASKS(Ap)){
      Ak.minsdate ≤ minimum(Aq in CHILD(Ak)): Aq.minsdate
   }
}
```

Invoereis 4:
Voor elke abstracte activiteit geldt: de maximale einddatum van de activiteit is ofwel ongedefinieerd, ofwel groter of gelijk aan de grootste maximale einddatum van alle onderliggende activiteiten:

```
foreach (project Ap) {
      foreach(Ak in PROJECT(Ap) -  TASKS(Ap)){
      Ak.maxedate ≥ maximum(Aq in CHILD(Ak)): Aq.maxedate
   }
}
```

Ten slotte kan door de gebruiker worden aangegeven wat de minimale achtereenvolgende tijd (in uren) is dat er aan een activiteit gewerkt dient te worden. Deze invoer is alleen relevant voor taken, omdat alleen deze later concreet ingeroosterd zullen worden. Wanneer de *minsuchours* niet opgegeven is, wordt het minimum van 1 uur aangehouden.

Mensen en deskundigheid
Ook voor middelen uit de categorie "mensen en deskundigheid" hebben we input van de gebruiker nodig. Om te kunnen plannen willen we namelijk voor alle *taken* uit het project weten welke werknemers de taak eventueel kunnen uitvoeren. Door middel van input legt de gebruiker voor alle taken impliciet een lijst met potentiële uitvoerders vast. Dit realiseren we met behulp van de *resource pools* en *rollen* (zie paragraaf 3.2).
Het uitgangspunt is dat alle werknemers binnen het model een activiteit kunnen uitvoeren. Door één of meerdere resource pools aan een activiteit toe te kennen, beperken we de potentiële uitvoerders tot de werknemers uit de gekozen resource pools. Door een rol aan de activiteit te koppelen schrappen we alle werknemers uit de lijst die niet alle competenties van de betreffende rol bezitten. Op deze manier kan de planner op een handige manier aangeven welke werknemers een bepaalde activiteit kunnen uitvoeren. Deze werkwijze levert een aantal toevoegingen aan ons model op. Ten eerste willen we dat er aan elke activiteit één rol gekoppeld kan worden. Hiervoor definiëren we de rol als eigenschap van een activiteit:

`Ai.role ε RL (entity refference): de rol die vereist is voor de uitvoering van activiteit Ai.`

De eigenschap A_i.role bestaat dus uit een verzameling competenties. Elk van deze competenties heeft op zijn beurt weer een afgeleide verzameling: de werknemers die de competentie bezitten. De doorsnede van al deze verzamelingen geeft ons alle werknemers die een bepaalde rol kunnen uitvoeren. Deze verzameling noteren we als volgt:

$E(A_i.role)$:= INTERSECTION(C_k ε $C(A_i.role)$): $E(C_k)$
de verzameling van werknemers die de rol van activiteit A_i kunnen vervullen

Naast een rol kennen we aan de activiteit nul of meerdere resource pools toe. Hiervoor definiëren we de volgende *verzameling*:

$RP(A_i) \subset RP$:= de verzameling van resource pools die aan activiteit A_i toegekend zijn.

Elk van deze resource pools bestaat uit een aantal werknemers. De vereniging van al deze verzamelingen geeft ons een verzameling van werknemers. Deze verzameling noteren we als:

$E(RP(A_i))$:= UNION (RP_k ε $RP(A_i)$): $E(RP_k)$
de verzameling van werknemers die ontstaat uit de vereniging van alle resource pools van activiteit A_i".

Wanneer we de doorsnede nemen van deze twee verzamelingen krijgen we de verzameling van potentiële werknemers voor de betreffende activiteit. Dit noteren we als volgt:

$E(A_i) = E(A_i.role) \cap E(RP(A_i))$

3.4.3 Definiëren van afhankelijkheden

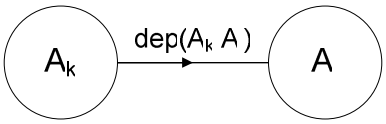
We hebben nu de boomstructuur van activiteiten opgenomen in ons model. Ook hebben we een aantal eigenschappen aan de activiteiten toegekend die door de planner concrete invulling dienen te krijgen. Wat we echter nog missen zijn de afhankelijkheden. We willen voor een activiteit kunnen aangeven dat de uitvoering van deze activiteit pas van start kan gaan wanneer één of meerdere andere activiteiten (succesvol) afgerond zijn. Dit doen we door middel van afhankelijkheden. Deze definiëren we door middel van de volgende relatie:

$dep(A_i, A_j)$ (binary) = 1 als activiteit A_j afhankelijk is van de voltooiing van activiteit A_i, 0 anders

De complete verzameling van afhankelijkheden binnen project A_p noteren we als:

$DEP(A_p) = \{dep(A_i,A_j) \mid A_i, A_j \varepsilon PROJECT(A_p) \ \& \ dep(A_i,A_j) = 1\}$: de verzameling van afhankelijkheden binnen project A_p.

Voor de beeldvorming worden de afhankelijkheden vaak weergegeven als pijlen, en de activiteiten als cirkels. In de onderstaande figuur is de afhankelijkheid dep(A_k,A_l) weergegeven.



Figuur 9: schematische weergave van afhankelijkheid dep(A_k, A_l).

We zullen spreken van *uitgaande* en *inkomende afhankelijkheden*. Voor activiteit A_k is de weergegeven afhankelijkheid dus een uitgaande afhankelijkheid, terwijl het voor A_l een inkomende afhankelijkheid is. Wanneer we het over de afhankelijkheid dep(A_k,A_l) hebben, zullen we de activiteiten A_k (respectievelijk A_l) aanduiden met de *van-activiteit* (respectievelijk *naar-activiteit*) van de afhankelijkheid.

We willen cyclische afhankelijkheden niet toestaan. Dit zou immers betekenen dat we in een oneindige cirkel terechtkomen. Hiervoor stellen we de volgende invoereis op:

Invoereis 4:
De verzameling van afhankelijkheden DEP(A_p) mag geen cyclische afhankelijkheden bevatten.

3.4.4 De Projectdefinitie van het voorbeeldproject

Nu we de volledige projectdefinitie in ons model hebben opgenomen, zullen we deze uitwerken voor ons voorbeeldproject. Binnen de klantprojecten kunnen we op het hoogste niveau een vijftal activiteiten onderscheiden:

1. initiatie
2. ontwerp
3. ontwikkeling
4. testen
5. formele acceptatie

We gaan er vanuit dat deze activiteiten achtereenvolgens uitgevoerd dienen te worden, en stellen daarom afhankelijkheden tussen deze activiteiten op. Vervolgens gaan we deze activiteiten verder opbreken in subactiviteiten. Om het voorbeeld overzichtelijk te houden gaan we hierbij niet te ver in op de details van de activiteiten.

1. *Initiatie*:
   de activiteit initiatie breken we voor het gemak niet verder op in subactiviteiten. Dit betekent dat we deze activiteit in ons voorbeeld dus als een één-mans taak

beschouwen. We schatten dat de taak ongeveer 8 uur in beslag zal nemen en stellen dat de rol van de uitvoerder *Project Manager* dient te zijn. We specificeren geen resource pools voor deze activiteit: de uitvoerder kan dus uit alle beschikbare resource pools komen.

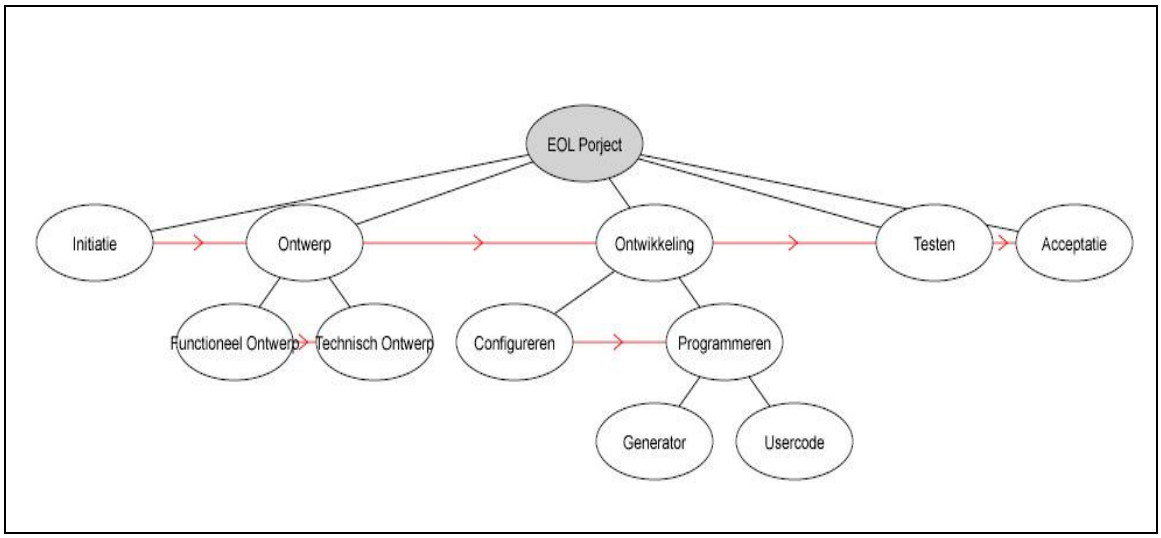
2. *Ontwerp*
   De activiteit ontwerp breken we op in de taken *functioneel ontwerp* en *technisch ontwerp*, welke in deze volgorde uitgevoerd dienen te worden. We voegen dus een afhankelijkheid "van" de activiteit functioneel ontwerp "naar" de activiteit technisch ontwerp toe aan de WBS. We schatten dat het functionele ontwerp ongeveer 16 uur in beslag zal nemen, en het technische ontwerp 8 uur. Voor beide taken kiezen we voor de rol *Ontwerper* en willen we dat de uitvoerder uit de resource pool *Frontoffice* afkomstig is.

3. *Ontwikkeling*
   Vervolgens beschouwen we de activiteit ontwikkeling. Deze ontwikkeling gebeurt met behulp van de in hoofdstuk 1 genoemde software generator en komt neer op programmeren en het configureren van schermen. Voor het gemak nemen we aan dat we eerst moeten configureren alvorens en vervolgens pas programmeren. We verdelen het programmeren in het maken van zogenaamde *usercode* en het maken van wijzigingen/toevoegingen in de generator zelf. Voor het configureren en het programmeren van de usercode zoeken we een *Applicatie Ontwikkelaar* uit de *Frontoffice*. Voor programmeren van de wijzigingen/toevoegingen in de generator zoeken we een *Generator Programmeur* uit de *Backoffice*. We schatten dat het configureren 20 uur zal duren, het programmeren van de usercode 20 uur, en het programmeren aan de generator 40 uur.

4. *Testen*
   Onder het testen van de applicatie verstaan we niet alleen het testen op correcte werking, maar ook het verhelpen van eventuele problemen. Voor het overzicht besluiten we toch om deze activiteit in het voorbeeld niet verder op te breken. We schatten de totale tijd nodig voor het testen en verwerken van de testresultaten op 100 uur, en stellen dat dit uitgevoerd dient te worden door een *Applicatie Ontwikkelaar* uit de *Frontoffice*.

5. *Formele acceptatie*
   De laatste activiteit betreft de formele acceptatie, de formele afsluiting van het project. Deze activiteit wordt uitgevoerd door een *Project Manager*, en zal ongeveer 8 uur in beslag nemen.

We laten de minimale achtereenvolgende werkduur (A_i.minsuchours) op de standaardwaarde voor alle activiteiten. Verder geven we alleen het project zelf de minimale startdatum "1 januari 2006". Hiermee hebben we (een sterk vereenvoudigde) projectdefinitie opgesteld voor de projecten van Efficiency Online. In de onderstaande figuur geven we de WBS schematisch weer.



Figuur 10: de WBS van het voorbeeldproject

In de onderstaande tabel vinden we een lijst van alle activiteiten en de daarbij behorende gegevens die we gespecificeerd hebben.

A_i	Naam	A_i.duration	A_i.minsdate	A_i.role	$RP(A_i)$
A_1	EOL Project	220	1-1-2006		
A_2	Initiatie	8		Project Manager	
A_3	Ontwerp	24		Ontwerper	Frontoffice
A_4	Ontwikkeling	80			
A_5	Testen	100			
A_6	Acceptatie	8		Project Manager	
A_7	Functioneel Ontwerp	16		Ontwerper	Frontoffice
A_8	Technisch Ontwerp	8		Ontwerper	Frontoffice
A_9	Configureren	20		Applicatie Ontwikkelaar	Frontoffice
A_{10}	Programmeren	60			
A_{11}	Programmeren Generator	40		Generator Programmeur	Backoffice
A_{12}	Programmeren Usercode	20		Applicatie Ontwikkelaar	Frontoffice

Tabel 6: een overzicht van de middelenschatting van het voorbeeldproject

3.5 De theoretisch optimale planning

Wanneer het project gedefinieerd is en aan alle gestelde invoereisen voldaan is, gaan we de structuur van het project analyseren om zo tot een theoretisch optimale planning te komen. Dit doen we door middel van een kritieke pad analyse op de laagste niveau activiteiten van het project. Het kritieke pad bestaat uit alle activiteiten waarvoor geldt dat uitloop van deze activiteiten per definitie tot uitloop van het project leidt. De som van de duur van alle activiteiten op dit kritieke pad geeft ons de minimale duur van het project. Op deze manier kunnen we dus bepalen wat de theoretisch optimale planning voor het project is. Of deze planning in de praktijk gehaald kan worden hangt af van de beschikbaarheid van de benodigde middelen. Voor het tot stand komen van deze optimale planning moeten we een aantal stappen uitvoeren. Om te beginnen moet de door de gebruiker ingevoerde WBS enigszins uitgebreid worden, om deze vervolgens te vertalen in een *kritiek pad netwerk*. Pas wanneer dit netwerk bepaald is kunnen we berekenen welke activiteiten op het kritieke pad liggen.

3.5.1 Opstellen kritiek pad netwerk

Voordat we de WBS kunnen vertalen in een kritiek pad netwerk voegen we eerst een *resultaat activiteit* A_r toe aan activiteiten verzameling $PROJECT(A_p)$. Het voltooien van deze activiteit komt overeen met voltooiing van het project. We beschouwen deze activiteit als een subactiviteit van het project A_p, in termen van ons model:

```
sub(Ap,Ar) = 1
```

We gaan er vanuit dat iedere door de planner gedefinieerde activiteit uit het project op de één of andere manier bijdraagt aan het bereiken van dit resultaat, en dus voegen we afhankelijkheden “naar” het resultaat toe “vanuit” alle “hoogste niveau” activiteiten die nog geen “uitgaande” afhankelijkheden hebben. Dit doen we met het volgende algoritme:

Algoritme 1:

```
foreach (Ak in PROJECT(Ap)){
   if (sub(Ap,Ak)=1 && Ak NOT IS Ar){
      if (foreach Al PROJECT(Ap): dep(Ak, Al) == 0){
         dep(Ak,Ar) = 1
      } else {
         dep(Ak,Ar) = 0
      }
   }
}
```

Vervolgens beginnen we met het opstellen van het kritieke pad netwerk. Net als de WBS bestaat dit netwerk uit activiteiten en afhankelijkheden tussen deze activiteiten. We definiëren een nieuwe activiteiten verzameling. Deze (afgeleide) verzameling bestaat uit alle laagste niveau activiteiten uit de WBS (de taken) samen met het resultaat (A_r):

$CPNW(A_p)$ = TASKS(A_p) v {A_r}: de verzameling van kritieke pad netwerk activiteiten.

Naast de activiteiten bestaat het kritieke pad netwerk uit afhankelijkheden. Hiervoor definiëren we de volgende relatie:

cpdep(A_i, A_j) (binary) = 1 als activiteit A_j in het kritieke pad netwerk afhankelijk is van de voltooiing van activiteit A_i, 0 anders.

De verzameling van kritieke pad afhankelijkheden noteren we als volgt:

CPDEP(A_p) = {dep(A_i,A_j) | A_i, A_j ε *PROJECT(A_p)* & cpdep(A_i,A_j) = 1}: de verzameling van afhankelijkheden binnen het kirtieke pad netwerk van project A_p.

Het bepalen van de afhankelijkheden die in het kritieke pad netwerk zitten is niet triviaal. Om te beginnen horen alle afhankelijkheden tussen twee activiteiten uit *CPNW*(Ap) in het netwerk thuis. We initiëren de verzameling van de kritieke pad netwerk afhankelijkheden dan ook met deze afhankelijkheden. Binnen de WBS zijn er echter op hoger niveau afhankelijkheden waar we ook rekening mee moeten houden. Met betrekking tot deze afhankelijkheden nemen we de volgende aanname:

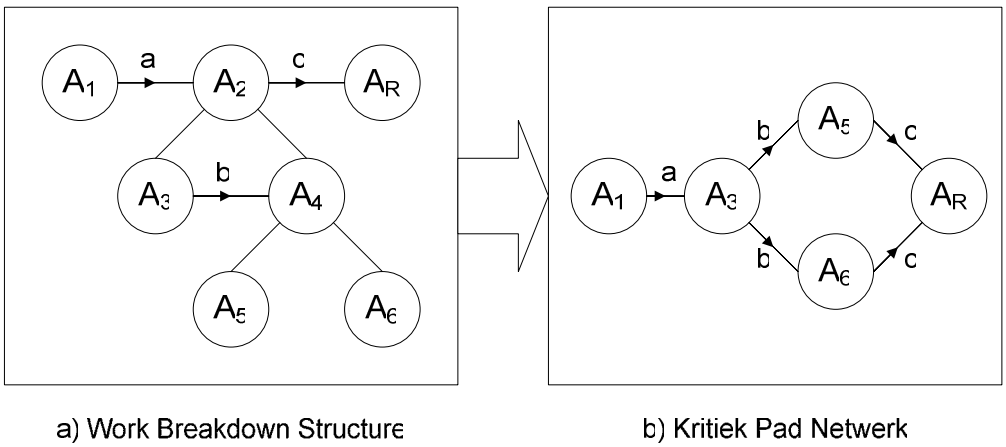
Modelaanname 4:
Als activiteit A_j afhankelijk is van activiteit A_i, betekent dit voor A_j dat iedere onderliggende activiteit pas van start kan gaan als alle onderliggende taken van A_i voltooid zijn.

Tussen de verschillende activiteiten uit *CPNW(A_p)* bestaan dus afhankelijkheden die impliciet vastgelegd zijn door afhankelijkheden op hoger niveau. Deze afhankelijkheden willen we opnemen in de verzameling van kritieke pad afhankelijkheden. Om deze afhankelijkheden te identificeren maken we gebruik van algoritme 2. Vanwege de complexiteit van het algoritme beperken we ons hier tot het uitleggen van het idee achter het algoritme. In appendix A2 is de volledig uitgewerkte versie van het algoritme te vinden.
Er wordt achtereenvolgens alle door de gebruiker gedefinieerde afhankelijkheden uit de WBS afgelopen. Voor elke afhankelijkheid wordt er gekeken of beide bijbehorende activiteiten in het kritieke pad netwerk zitten. Dit levert vier mogelijke gevallen op:

1. Beide activiteiten maken deel uit van het kritieke pad netwerk
2. Alleen de “van-activiteit” maakt deel uit van het kritieke pad netwerk
3. Alleen de “naar-activiteit” maakt deel uit van het kritieke pad netwerk
4. Beide activiteiten maken geen deel uit van het kritieke pad netwerk

In het eerste geval wordt de activiteit direct toegevoegd aan het kritieke pad netwerk. In de overige drie gevallen wordt er voor de activiteiten die niet in het kritieke pad netwerk zitten gekeken naar de onderliggende taken. Het idee is dat al deze taken een afhankelijkheid dienen te hebben. In figuur 9 wordt schematisch weergegeven hoe het algoritme omgaat met deze drie gevallen. Net als voorheen worden de activiteiten

weergegeven als cirkels en de afhankelijkheden weer weergegeven als pijlen. Een pijl van A_i naar A_k betekent dat activiteit A_k afhankelijk is van activiteit A_j.



Figuur 11: schematische weergave van de vertaling van WBS naar hritiek pad netwerk.

In figuur 8 zien we een gedeelte van een WBS, met daarin drie afhankelijkheden a, b en c. In figuur 8b zien we hoe dit vertaald is in een kritiek pad netwerk. Voor elk van de afhankelijkheden in 8b is aangegeven uit welke oorspronkelijke afhankelijkheid uit de WBS hij voortkomt.

3.5.2 Bepalen kritiek pad

Vervolgens gaan we het kritieke pad bepalen. We beschrijven het hiervoor gebruikte algoritme in woorden:

> Voor alle activiteiten bepalen we de zogenaamde *earliest* en *latest start*. De *earliest start* van een activiteit is de datum waarop deze activiteit op zijn vroegst kan beginnen. Voor alle activiteiten die geen *inkomende afhankelijkheden* hebben geldt dat de *earliest start* gelijk is aan de startdatum van het project: voor de start van de uitvoering van deze activiteiten hoeven immers geen andere activiteiten voltooid te zijn. Voor de overige activiteiten wordt de *earliest start* bepaald door voorgaande activiteiten: de activiteit kan op zijn vroegst beginnen wanneer alle voorgaande activiteiten voltooid zijn.
> De *latest start* van een activiteit is de datum waarop de activiteit op zijn laatst kan beginnen, zonder dat het project als geheel daar vertraging door oploopt. De *latest start* van het projectresultaat A_r is per definitie gelijk aan de *earliest start* hiervan. Voor alle overige activiteiten word de *latest start* vastgelegd door de opvolgende activiteiten. Vervolgens bekijken we voor alle activiteiten het verschil tussen de *earliest start* en de *latest start*. Dit verschil wordt ook wel de *slack* van een activiteit genoemd, en geeft aan hoeveel speling er zit in de uitvoering van de activiteit. Wanneer voor een activiteit geld dat de *earliest start* gelijk is aan de *latest start* (met andere woorden: de *slack* van de activiteit is gelijk aan nul) betekent dit dat deze activiteit geen enkele speling heeft: uitloop van deze activiteit levert gegarandeerd uitloop van het project op, en dus behoort de activiteit tot het kritieke pad.

Omdat we de gedurende het algoritme berekende gegevens later nog gaan gebruiken voegen we hiervoor attributen toe aan de activiteiten:

- A_i.es (date-time): de *earliest start* van activiteit A_i
- A_i.ef (date-time): de *earliest finish* van activiteit A_i
- A_i.ls(date-time): de *latest start* van activiteit A_i
- A_i.lf(date-time): de *latest finish* van activiteit A_i
- A_i.slack (integer): de speelruimte voor de uitvoering van activiteit A_i

Naast de *earliest* (respectievelijk *latest*) *start* hebben we ook een *earliest* (respectievelijk *latest*) *finish* en *slack* toegevoegd. Bij het rekenen met deze tijdstippen wordt uitgegaan van standaard werkweken (oftewel maandag tot en met vrijdag van negen tot vijf). Door middel van een aantal voorbeelden zullen we uitleggen wat we hier precies mee bedoelen:

- Vrijdag 18-11-2006 9:00 + 5 uur = Vrijdag 18-11-2006 14:00
- Vrijdag 18-11-2006 9:00 + 8 uur = Vrijdag 18-11-2006 17:00 = Maandag 21-11-2006 9:00
- Vrijdag 18-11-2006 9:00 + 11 uur = Maandag 21-11-2006 12:00

Hierachter zit de volgende aanname:

Modelaanname 5:
Bij het bepalen van de theoretisch optimale planning gaan we uit van standaard werktijden van 40 uur per week, maandag tot en met vrijdag, van negen tot vijf.

Deze aanname heeft in eerste instantie dus alleen betrekking op de theoretische planning. Het model is relatief eenvoudig uit te breiden met de mogelijkheid om dit *werkschema* als gebruikers invoer op te nemen. We hebben er in verband met de beperkte tijd van deze stage echter voor gekozen deze mogelijkheid niet in ons model op te nemen. Verder voegen we een *binary* attribuut *in_cp* toe aan de activiteiten:

A_i.in_cp (binary) = 1 als A_i in het kritieke pad zit, 0 anders

De volledige uitwerking van het kritieke pad algoritme is te vinden onder algoritme 3 in appendix A2. Na het toepassen van dit algoritme hebben we onze theoretisch optimale planning bepaald op het niveau van de taken. We kunnen nu de earliest/latest start/finish doorberekenen naar de hogere niveaus om ook hier een planning te kunnen weergeven. Dit doen we met het volgende algoritme:

Algoritme 4:

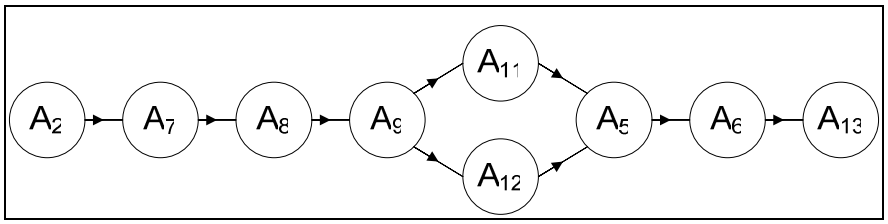
```
foreach (Ak in Project(Ap)) {
   if (Ak not in Tasks(Ap)) {
      Ak.es = minimum(Aq in Tasks(Ak)): Aq.es
      Ak.ef = maximum(Aq in Tasks(Ak)): Aq.ef
      Ak.ls = minimum(Aq in Tasks(Ak)): Aq.ls
      Ak.lf = maximum(Aq in Tasks(Ak)): Aq.lf
   }

}
```

Wanneer we ook dit algoritme toegepast hebben, is onze theoretische planning voltooid. We hebben nu voor alle activiteiten uit het project bepaald wat de optimale periode is voor uitvoering van de activiteit. Hiernaast hebben we inzicht verkregen in welke activiteiten cruciaal zijn voor tijdige voltooiing van het project.
De volgende stap in ons model betreft het toekennen van werknemers aan de taken, en vervolgens het daadwerkelijk inroosteren van deze taken. In de volgende paragraaf zullen we deze stap beschrijven, maar eerst zullen we de theoretisch optimale planning uitwerken voor het voorbeeldproject.

Het voorbeeldproject
We berekenen nu de theoretisch optimale planning voor ons voorbeeldproject. Om te beginnen voegen we een projectresultaat (A_{13}) toe aan de activiteitenverzameling en plaatsen deze "onder" het project: sub(A_1, A_{13}) = 1. Vervolgens voeren we Algoritme 1a uit om afhankelijkheden naar het resultaat toe te voegen. Dit levert ons één nieuwe afhankelijkheid op, van de activiteit *Acceptatie* naar het resultaat: dep(A_6,A_{13}) = 1. Nu kunnen we beginnen met het opstellen van het kritieke pad netwerk. Hierin zitten de volgende activiteiten:

$$CPNW(A_1) = TASKS(A_1) \text{ v } \{A_r\} = \{A_2, A_5, A_6, A_7, A_8, A_9, A_{11}, A_{12}\} \text{ v } \{A_{13}\} = \{A_2, A_5, A_6, A_7, A_8, A_9, A_{11}, A_{12}, A_{13}\}$$

Met behulp van algoritme 2 bepalen we vervolgens welke afhankelijkheden er in het kritieke pad netwerk thuishoren. Het resulterende kritieke pad netwerk is weergegeven in figuur 10.



Figuur 12: het kritieke pad netwerk van het voorbeeldproject.

Op basis van dit netwerk voeren we vervolgens algoritme 3 uit om het kritieke pad te bepalen. In tabel 7 vinden we voor alle activiteiten de earliest (respectievelijk latest) start en finish datum.

Activiteit A_i	A_i.duration	A_i.es	A_i.ef	A_i.ls	A_i.lf	A_i.slack
A_2	8	2-1-2006 9:00	2-1-2006 17:00	2-1-2006 9:00	2-1-2006 17:00	0
A_7	16	3-1-2006 9:00	4-1-2006 17:00	3-1-2006 9:00	4-1-2006 17:00	0
A_8	8	5-1-2006 9:00	5-1-2006 17:00	5-1-2006 9:00	5-1-2006 17:00	0
A_9	20	6-1-2006 9:00	10-1-2006 13:00	6-1-2006 9:00	10-1-2006 13:00	0
A_{11}	40	10-1-2006 13:00	17-1-2006 13:00	10-1-2006 13:00	17-1-2006 13:00	0
A_{12}	20	10-1-2006 13:00	12-1-2006 17:00	13-1-2006 9:00	17-1-2006 13:00	20
A_5	100	17-1-2006 13:00	2-2-2006 17:00	17-1-2006 13:00	2-2-2006 17:00	0
A_6	8	3-2-2006 9:00	3-2-2006 17:00	3-2-2006 9:00	3-2-2006 17:00	0
A_{13}	0	3-2-2006 17:00	3-2-2006 17:00	3-2-2006 17:00	3-2-2006 17:00	0

Tabel 7: de resultaten van het kritieke pad algoritme voor ons voorbeeldproject.

In de tabel zien we dat alleen activiteit A_{12} een slack van groter dan 0 heeft, en dus niet in het kritieke pad thuishoort. Met behulp van Agoritme 5 berekenen we nu ook voor de abstracte activiteiten de earliest/latest start en finish datum. Omdat dit voor ons voorbeeldproject redelijk triviaal is, laten we de resultaten hiervan achterwegen. Hiermee hebben we de theoretisch optimale planning voor ons voorbeeldproject bepaald.

3.6 Roosteren van activiteiten

Nu we bepaald hebben wat de optimale planning voor het project is gaan we opzoek naar een planning die we in de praktijk kunnen gebruiken. Dit doen we door de taken één voor één in te roosteren. Voor we een taak kunnen inroosteren moeten we eerst vastleggen welke werknemer de uitvoering van de taak op zich gaat nemen. Hiervoor voegen we aan de activiteiten het volgende attribuut toe:

A_i.employee (Employee E_k ε E): de werknemer die taak A_i gaat uitvoeren

In paragraaf 3.4 hebben we voor alle taken impliciet verzamelingen met potentiële uitvoerders vastgelegd ($E(A_i)$). We moeten nu voor iedere taak bepalen welke werknemer de taak feitelijk gaat uitvoeren. We kiezen er voor om deze beslissing bij de gebruiker te leggen. We willen dat het model hierbij ondersteuning geeft maar de feitelijke beslissing niet neemt. Dit realiseren we door elke kandidaat een *geschiktheids score* te geven voor de voor de uitvoering van de betreffende taak. Hiervoor definiëren we de volgende relatie:

score(E_i,A_j) (percentage): de *geschiktheidscore* van werknemer E_i voor de uitvoering van activiteit A_j

Bij het bepalen van deze score kunnen we rekening houden met een aantal factoren. Ten eerste hebben we te maken met het vaardigheidsniveau van de werknemers voor de voor de taak benodigde competenties. Hiervoor definiëren we de volgende relatie:

avg_skill(E_k,A_l) (percentage): het gemiddelde vaardigheidsniveau van werknemer E_k over alle competenties benodigd voor de uitvoering van activiteit A_l.

Dit wordt alsvolgt berekent:

avg_skill(E_k,A_l) = (SUM(C_x in *C*(A_l.role)): skill(E_k, C_x)) / #(*C*(A_l.role))

Idealiter houden we niet alleen rekening met de vaardigheden van de werknemers maar ook met hun agenda in de periode waarin het project plaatsvindt. Efficiency Online heeft echter aangegeven dat het uitwerken van dit *scoringsmechanisme* lage prioriteit heeft. Daarom is er besloten het model niet uit te breiden met een berekening voor het bepalen van de geschiktheid van werknemers op basis van hun agenda. We houden het binnen ons model dus bij de score op basis van de vaardigheidsniveaus:

score(E_i,A_j) = avg_skill(E_k,A_l)

Toch willen we de gedachtegang achter het scoren van werknemers op basis van hun tijdsindeling in dit verslag meenemen. Daarom zullen we in de sectie *Scoringsmechanisme*, aan het eind van deze paragraaf, een voorbeeld geven van hoe dit mechanisme in zijn werk zou kunnen gaan. Binnen ons model beschouwen we het toekennen van de werknemers aan activiteiten als gebruikers invoer. Voor elke activiteit dient door de gebruiker dus het attribuut A_i.employee ingevuld te worden, alvorens er begonnen kan worden met het inroosteren van de activiteiten:

Invoereis 5:
Alvorens er begonnen kan worden met het inroosteren van activiteiten dient voor alle taken aangegeven te zijn welke werknemer de uitvoering van de taak op zich gaat nemen.

Wanneer er aan deze eis voldaan is kunnen de activiteiten ingeroosterd worden. Onder het inroosteren van een activiteit verstaan we het vastleggen van één of meerdere perioden waarin de activiteit uitgevoerd wordt door de toegekende werknemer. Deze combinatie van activiteit, werknemer en periode nemen we in ons model op als *events*:

Events: EV_i ε *EV*

Aan een event kennen we de volgende attributen toe:

- EV_i.activity (Activity A_k ε *TASKS*(A_p)): de activiteit waar tijdens event EV_i aan gewerkt wordt.
- EV_i.employee (Employee E_k ε *E*): de werknemer voor wie het event EV_i ingeroosterd is.
- EV_i.plan_date (date): de datum wwaarop het event EV_i ingeroosterd is.
- EV_i.duration (integer in (1,24)): het aantal ingeroosterde uren.

Een event vindt dus per definitie op één dag plaats. Dit geldt uiteraard niet voor een taak: de uitvoering hiervan zal in de meeste gevallen over meerdere dagen verspreid plaatsvinden. Het inroosteren van taken komt binnen ons model neer op het aanmaken van events, die samen minimaal de totale duur van de taak omvatten. Hierbij gaan we er vanuit dat we de taak zo spoedig mogelijk af willen hebben. Bij het aanmaken van de events proberen we deze dus zo vroeg mogelijk in de tijd te plaatsen. De start en einddatum van een activiteit worden dus impliciet vastgelegd door de events. Om deze datums binnen het model toegankelijker te maken voegen we hiervoor ook attributen toe aan de activiteiten:

- A_i.plan_startdate (date): de geplande startdatum van activiteit, gebaseerd op de ingeroosterde events
- A_i.plan_enddate (date): de geplande einddatum van activiteit, gebaseerd op de ingeroosterde events

Om binnen het model op een handige wijze met de events om te kunnen gaan definiëren we een aantal afgeleide verzamelingen:

- *EV*(E_i): Alle events van werknemer E_i.
- *EV*(A_i): Alle events voor activiteit A_i.
- *EV*(E_i, A_j): Alle events van werknemer E_i voor activiteit A_j.
- *EV*(E_i, *date*): Alle events EV_k van werknemer E_i gepland op datum *date*.

We gaan de taken uit *TASKS*(A_p) één voor één inroosteren. Dit doen we op volgorde van *latest start date*, zoals de bepaald is in de kritieke pad analyse. Hiermee voorkomen we proberen een activiteit in te roosteren waarvoor geldt dat één van de vereiste voorgaande activiteiten nog niet ingeroosterd is. Het algoritme voor het inplannen van een activiteit A_k is te vinden in Appendix A2 onder Algoritme 5a. We zullen ons hier beperken tot een globale beschrijving van de werking van dit algoritme.

We gaan de taak A_k inroosteren voor werknemer A_k.employee. De duur van de taak is gegeven door A_k.duration. We gaan als volgt tewerk.

1. Om te beginnen bepalen we de vroegst mogelijke startdatum. Deze datum wordt vastgelegd door het maximum van:
    a. De huidige datum + 1 dag.
    b. De theoretische *earliest start* zoals berekend bij het bepalen van het kritieke pad.

c. De minimale startdatum die opgegeven is door de gebruiker (A_k.minsdate)
d. Het maximum van de ingeplande einddata van de voorgaande taken + 1 dag.

2. Vervolgens gaan we, met de bij 1 bepaalde datum als uitgangspunt, net zolang zoeken naar beschikbare uren in het rooster van de werknemer totdat de volledige duur van de activiteit gedekt is door events.

3. Ten slotte bepalen we op basis van de ingeroosterde events de geplande start en einddatum van de activiteit.

Wanneer we dit voor alle taken gedaan hebben, hebben we een rooster, en impliciet een planning voor het project. In deze eerste versie van het rooster en de planning kunnen vervolgens wijzigingen aangebracht worden. In de volgende paragraaf beschrijven we de verschillende mogelijke aanpassingen in de planning, en breiden we ons model uit om met deze aanpassingen om te kunnen gaan. Eerst zullen we echter in de volgende secties het roosteren van de taken uitwerken voor het voorbeeldproject en het voorheen beschreven scoringsmechanisme op basis van beschikbare tijd bespreken.

3.6.1 Het voorbeeldproject: inroosteren van de taken

We zullen nu de taken van het voorbeeldproject inroosteren. We beginnen met het toekennen van werknemers aan de verschillende taken. In de onderstaande tabel staan voor iedere taak zowel de potentiële als de toegekende werknemers.

Activity A_i	$E(A_i)$	A_i.employee
A_2	Sjoerd Geraedts Daniel van der Wallen	Sjoerd Geraedts
A_7	Pascal Cramer Wouter Ewalds Sjoerd Geraedts Tom Hendrix Wouter Radder	Wouter Ewalds
A_8	Pascal Cramer Wouter Ewalds Sjoerd Geraedts Tom Hendrix Wouter Radder	Wouter Ewalds
A_9	Pascal Cramer Wouter Ewalds Sjoerd Geraedts Tom Hendrix Wouter Radder	Tom Hendrix
A_{11}	Wouter Ewalds Jeroen de Hertog Otto Moerbeek Daniel van der Wallen	Daniel van der Wallen
A_{12}	Pascal Cramer Wouter Ewalds Sjoerd Geraedts Tom Hendrix Wouter Radder	Tom Hendrix
A_5	Pascal Cramer Wouter Ewalds Sjoerd Geraedts Tom Hendrix Wouter Radder	Tom Hendrix
A_6	Sjoerd Geraedts Daniel van der Wallen	Sjoerd Geraedts

Tabel 8: toekenning van werknemers aan de taken van het voorbeeldproject.

Nu we de verschillende werknemers aan de taken toegekend hebben, gaan we de taken inroosteren met behulp van algoritme 5a. We zullen niet de volledige lijst met events geven maar slechts de geplande start en einddata van de verschillende activiteiten. Deze zijn weergegeven in tabel 9.

Activiteit A_i	A_i.plan_startdate	A_i.plan_enddate	A_i.employee
A_2	2-1-2006	2-1-2006	Sjoerd Geraedts
A_7	3-1-2006	4-1-2006	Wouter Ewalds
A_8	5-1-2006	5-1-2006	Wouter Ewalds
A_9	6-1-2006	10-1-2006	Tom Hendrix
A_{11}	11-1-2006	17-1-2006	Daniel van der Wallen
A_{12}	11-1-2006	13-1-2006	Tom Hendrix
A_5	18-1-2006	3-2-2006	Tom Hendrix
A_6	6-2-2006	6-2-2006	Sjoerd Geraedts
A_{13}	6-2-2006	6-2-2006	

Tabel 9: De planning van het voorbeelproject.

Wanneer we deze planning vergelijken met de optimale planning die we in de vorige paragraaf bepaald hebben, zien we dat de einddatum van het project in deze planning één werkdag later valt dan in de theoretische planning. Dit is enigszins tegen de verwachting in aangezien we geen andere projecten ingepland hebben en ook de keuze van de werknemers geen belemmering in de uitvoer van de taken is. Dit verschil is te verklaren uit het feit dat het roostering algoritme een de uitvoering taak altijd op een nieuwe dag laat beginnen. Wanneer een taak volgens de planning halverwege een bepaalde dag voltooid is, worden de opvolgende taken pas vanaf de volgende dag ingeroosterd. Het roostering algoritme maakt dus altijd een bepaalde bufferruimte aan tussen twee taken, terwijl het kritieke pad algoritme dit niet doet.

3.6.2 Scoringsmechanisme

In het begin van deze paragraaf hebben we een score geïntroduceerd voor het beoordelen van de geschiktheid van de verschillende werknemers voor het uitvoeren van een bepaalde taak.

We hebben ervoor gekozen om bij het bepalen van deze score geen rekening te houden met te agenda van de verschillende werknemers. Efficiency Online is echter wel van mening dat een dergelijke uitbreiding van het model toegevoegde waarde kan leveren. Daarom geven we hier een omschrijving van hoe een relatief eenvoudige uitwerking van dit mechanisme eruit zou kunnen zien.

Volgens de in deze paragraaf beschreven werkwijze dient er aan alle taken een werknemer toegekend te zijn alvorens de taken ingeroosterd kunnen worden. Op deze manier is het niet mogelijk om voor alle taken uit het project een bruikbare score op basis van tijd te bepalen. We kunnen namelijk pas iets zeggen over een taak als we alle voorgaande taken ingeroosterd hebben en weten in welke periode de taak idealiter

uitgevoerd dient te worden. Dit is precies het idee achter de werking van onze uitwerking van het tijd scoringsmechanisme.

In eerste instantie worden alleen de taken beschouwd die niet afhankelijk zijn van andere taken. We bepalen voor elk van deze taken, voor iedere potentiële werknemer op welke datum deze de taak op zijn vroegst af zou kunnen hebben. Vervolgens geven we de werknemer die de taak het eerste af kan hebben een *tijdscore* van 100. De werknemer die de taak in theorie als laatste af heeft krijgt een *tijdscore* van 0. Voor de overige werknemers wordt de score relatief aan de "eerste" en "laatste" werknemer bepaald. Vervolgens dient er door de gebruiker aan deze taken een werknemer toegekend te worden, en worden de taken ingeroosterd. Dit gebeurt op dezelfde wijze als in de eerder beschreven methode: de taken worden verdeelt over events die zo vroeg mogelijk ingeroosterd worden.

Nu we deze eerste groep taken ingeroosterd hebben, is er een verzameling taken ontstaan (met daarin minimaal één taak) waar voor elke taak uit deze verzameling geldt dat alle voorgaande taken reeds ingeroosterd zijn. Voor alle taken uit deze verzameling kunnen we voor elke potentiële werknemer bepalen wanneer deze de taak op zijn vroegst af kan hebben. Zo kunnen we op dezelfde wijze als voorheen een score bepalen voor alle kandidaat werknemers. Vervolgens wordt er aan ieder van deze taken een werknemer toegekend en worden ze ingeroosterd. Op deze manier kunnen we doorgaan totdat alle taken van het project ingeroosterd zijn. De planner heeft bij het toekennen van werknemers aan taken met deze methode constant informatie beschikbaar over welke werknemer de taak in theorie als eerste af zal hebben.

3.7 Wijzigingen in de Planning

Ons model is nu dus in staat een rooster te genereren voor alle uit te voeren activiteiten binnen een project. Vervolgens kan er met behulp van dit rooster een planning doorberekend worden. Bij een gepland project hebben we voor elke activiteit een geplande start en einddatum. In de praktijk is al bij vele projecten gebleken dat, naarmate het project vordert de werkelijkheid steeds meer afwijkt van deze eerste versie van de planning. Hiervoor zijn verscheidene oorzaken te identificeren, zoals bijvoorbeeld het veranderen van klantwensen of het uitlopen van één of meerdere activiteiten.
We willen voorkomen dat we éénmaal een planning maken, welke vervolgens in de kast verdwijnt en nooit meer naar gekeken wordt omdat hij teveel afwijkt van de praktijk. Daarom beschouwen we goede omgang met deze wijzigingen als cruciaal voor de toepasbaarheid van het model in de praktijk. We willen dat de planning op ieder willekeurig moment tijdens het project aangepast kan worden aan de huidige stand van zaken. In deze paragraaf zullen we het model uitbreiden om dit te realiseren. Hiervoor zullen we een aantal verschillende categorieën van wijzigingen introduceren en het model aanpassen om met de wijzigingen om te gaan.

3.7.1 Voortgang van het project

De eerste categorie van wijzigingen zijn de wijzigingen die voortkomen uit het feit dat het project voortgang maakt. Wanneer de voortgang exact overeenkomt met de huidige planning hoeft er in principe niets te gebeuren. In de praktijk zal dit echter niet het geval zijn. Daarom willen we de projectplanning kunnen aanpassen aan de voortgang van het project. Dit doen we door alle stappen van het model opnieuw te doorlopen, maar dan rekening houdend met de voortgang van de taken. We voegen aan de activiteiten het volgende attribuut toe:

A_i.remaining_duration (integer): het geschatte aantal uren dat er nog openstaat voor het voltooien van activiteit A_i.

De uitbreidingen die we aan het model maken zijn relatief eenvoudig. Ten eerste moeten we de remaining_duration een startwaarde geven. Hiervoor nemen we de door de gebruiker ingevoerde duur van de activiteit. Dit realiseren we door het algoritme voor het toevoegen van afhankelijkheden naar het projectresultaat (Algoritme 1a) uit te breiden. Dit resulteert in Algoritme 1b (appendix A2), waarin de toevoegingen in het blauw weergegeven zijn.
Ook bij het inroosteren van de taken veranderd er iets in ons model. We gaan nu namelijk voordat er nieuwe events aangemaakt worden, eerst alle huidige events voor het betreffende project verwijderen. Hiervoor definiëren we eerst een afgeleide verzameling met daarin alle events die bij een bepaald project horen:

$EV(A_p) = \{EV_k \in EV \mid EV_k.activity \in PROJECT(A_p)\}$

Vervolgens voegen we aan het algoritme voor het inroosteren van de taken (algoritme 5a) een functie voor het verwijderen van de events toe. Verder zetten we in het algoritme een extra controle die ervoor zorgt dat alleen de taken die nog niet voltooid zijn ingeroosterd worden. Ook roosteren we een taak nu niet voor de volledige duur in, maar alleen voor resterende duur. Het resulterende algoritme is te vinden in appendix A2 onder algoritme 5b, hierin zijn de toevoegingen/wijzigingen ten opzichte van algoritme 5a blauw gedrukt.

3.7.2 Wijzigingen in de projectdefinitie

Gedurende een project kunnen er wijzigingen optreden in de project definitie. Een taak kan bijvoorbeeld langer duren dan verwacht, of er kunnen activiteiten nodig zijn die in eerste instantie niet voorzien waren. We willen dat het model hiermee om kan gaan, daarom leggen we vast dat we de volgende acties gedurende een project moeten kunnen uitvoeren:

- toevoegen/verwijderen activiteiten
- toevoegen/verwijderen afhankelijkheden
- wijzigen van (de middelen schatting van) activiteiten

In termen van ons model komt dit neer op het aanpassen van de volgende (in paragraaf 4.4) gedefinieerde verzamelingen, relaties en attributen:

- *A*

- *SUB*
- A_i.duration voor alle Ai ε *A*
- A_i.minsdate voor alle Ai ε *A*
- A_i.maxedate voor alle Ai ε *A*
- A_i.minsuchours voor alle Ai ε *A*
- A_i.role voor alle Ai ε *A*
- *RP*

Het model is momenteel zodanig opgezet dat het bijna volledig kan omgaan met deze aanpassingen. Het toevoegen en wijzigen van activiteiten vormt geen probleem. Ook het toevoegen en verwijderen van afhankelijkheden vereist geen uitbreidingen van het model. Dankzij de in de vorige sectie beschreven uitbreiding kan het project namelijk ten aller tijden volledig opnieuw gepland worden. Wanneer dit gebeurt, en het kritieke pad netwerk opnieuw opgesteld en "opgelost" wordt, wordt de huidige projectdefinitie als basis gebruikt, en dus zullen alle wijzigingen in deze projectdefinitie dan meegenomen worden. Bij het vervolgens opnieuw inroosteren van de taken vormt dit herberekende kritieke pad de basis, en ook hier worden dus alle wijzigingen in de projectdefinitie meegenomen.

De enige uitbreidingen die we het model moeten geven heeft betrekking op het verwijderen van activiteiten. We moeten nu namelijk specificeren wat we doen met relaties en attributen doen die naar de te verwijderen activiteit verwijzen. Bij het verwijderen van activiteiten hebben we te maken met afhankelijkheden, ingeroosterde events en de boomstructuur van de WBS.
We beschouwen een gerelateerde afhankelijkheid als irrelevant wanneer we een activiteit willen verwijderen, en dus kiezen we ervoor de afhankelijkheid mee te verwijderen, In termen van ons model:

$del(A_k, dep(A_k, A_1)) = del(A_k, dep(A_j, A_k)) = \text{"delete"}$

Hetzelfde geldt voor de events; wanneer een activiteit verwijderd wordt, en dus niet meer uitgevoerd hoeft te worden, hoeven er uiteraard ook geen events voor ingeroosterd te zijn, dus:

$del(A_k, E_1.activity)) = \text{"delete"}$

De vraag is nu hoe we omgaan met de onderliggende activiteiten bij het verwijderen van een activiteit. We kiezen ervoor om bij het verwijderen van een zekere activiteit A_k ook alle onderliggende activiteiten te verwijderen. Dit doen we door bij het verwijderen van een activiteit A_k ook alle subrelaties $sub(A_k,A_l)$ te verwijderen, en vervolgens bij het verwijderen van een subrelatie $sub(A_k,A_l)$ ook activiteit A_l te verwijderen. middel van de volgende drie relaties:

- $del(A_k, sub(A_k,A_1)) = \text{"delete"}$
- $del(sub(A_k,A_1),A_1) = \text{"delete"}$

Hiermee hebben we ons model dusdanig aangepast dat we om kunnen gaan met wijzigingen in de projectdefinitie.

3.7.3 Wijzigingen in de toekenning van werknemers

Gedurende een project kan, om allerlei verschillende redenen, besloten worden om de (verdere) uitvoering van een taak door een andere werknemer te laten plaatsvinden. Hiervoor hoeven we ons model niet uit te breiden. We kunnen namelijk, dankzij de gemaakte wijzigingen in het rooster algoritme, voor alle taken van het project de toegekende werknemer wijzigen, en vervolgens simpelweg de taken opnieuw inroosteren. Alle bestaande events worden namelijk verwijderd, en vervolgens worden er nieuwe events aangemaakt voor de op dat moment toegekende werknemers.

3.7.4 Wijzigingen in het rooster

Het rooster dat door het model gegenereerd wordt kan nog niet voldoende beïnvloed worden om in de praktijk gebruikt te worden. Daarom breiden we het model uit met de mogelijkheid om op twee verschillende manieren invloed uit te oefenen op het gegenereerde rooster.

1. *Blokkeren van dagdelen*
   We willen de mogelijkheid toevoegen om aan te geven dat medewerkers op bepaalde tijden niet beschikbaar zijn voor het werk aan projecten. Om dit in het model te realiseren maken we gebruik van de reeds gedefinieerde entiteit *events*. In het rooster zitten vooralsnog alleen *events* die betrekking hebben op activiteiten die bij een project horen. We geven de gebruiker de mogelijkheid om *ongespecificeerde* events toe te voegen en te beheren. Hiermee bedoelen we events die geen betrekking op een activiteit binnen ons model hebben. Behalve de activiteit (EV_i.activity) worden alle attributen van het event (dwz: EV_i.employee, EV_i.plan_date, EV_i.duration) door de gebruiker ingevuld. Voor deze toevoeging hoeft het model in technische zin dus niet verder uitgebreid te worden.

2. *Opsplitsen van activiteiten*
   De tweede mogelijkheid die we toevoegen is de mogelijkheid om een gedeelte van een activiteit (dwz één of meerdere achtereenvolgende events) aan een andere medewerker toe te wijzen. Dit is binnen ons model als equivalent aan het opsplitsen van de activiteit in subactiviteiten.
   Om deze toevoeging in het model op te nemen voegen we algoritme 6 toe aan het model. In appendix A2 is dit algoritme uitgewerkt in de gehanteerde pseudocode.
   Als invoer van het algoritme dienen een activiteit (A_k), een startdatum, een einddatum en een medewerker opgegeven te worden. Vervolgens wordt de activiteit zodanig in subactiviteiten opgesplits dat:

   - er één nieuwe activiteit aangemaakt is die alle events binnen de opgegeven periode dekt, binnen deze periode uitgevoerd dient te worden en toegewezen is aan opgegeven werknemer.

- er voor alle overige events één of twee nieuwe activiteiten aangemaakt zijn, die deze events dekken, voor (danwel na) de opgegeven periode uitgevoerd dienen te worden en aan de oorspronkelijke medewerker toegewezen zijn.

Wanneer een activiteit op deze wijze opgesplits is dient het project volledig opnieuw gepland te worden.

Door middel van deze toevoegingen kunnen we het rooster aanpassen aan onregelmatigheden en onvoorziene omstandigheden.

3.8 Conclusie

We hebben nu een wiskundig model opgesteld voor het genereren van een rooster en het doorberekenen van een planning. Het plannen van een project met dit model gaat in een aantal stappen:

1. *Projectdefinitie*: de gebruiker breekt het project op in activiteiten en relaties, en maakt voor iedere activiteit een schatting van de benodigde middelen.
2. *Theoretisch optimale planning*: Op basis van de projectdefinitie wordt berekend wanner de activiteiten uit het project op zijn vroegst af kunnen zijn
3. *Toewijzen werknemers*: de gebruiker geeft aan welke werknemers de verschillende activiteit uit zullen voeren.
4. *Inroosteren activiteiten*: de activiteiten worden ingeroosterd in het schema van de uitvoerende werknemer.
5. *Bepalen hoger niveau planning*: Uit dit rooster wordt vervolgens een hoger niveau planning doorberekend.

Gedurende de stage hebben we dit model grotendeels geïmplementeerd binnen de software generator van Efficiency Online. Het hieruit resulterende prototype wordt in het volgende hoofdstuk gepresenteerd.

Hoofdstuk 4: Het Prototype

In dit hoofdstuk presenteren we het tijdens de stage gemaakte prototype[1]. Dit doen we, net als het model, aan de hand van ons voorbeeld project. De nadruk zal hierbij liggen op de functionaliteit, en niet op de technische uitwerking hiervan.

Het prototype is gemaakt binnen de software generator van Efficiency Online en bestaat uit twee *modules* die elk onderverdeeld zijn in een aantal *pagina's*.

In paragraaf 4.1 geven we enige uitleg over de werking van de generator, en zullen we kort beschrijven hoe de verschillende onderdelen van het model terugkomen in het prototype. Vervolgens behandelen we in de paragrafen 4.2 en 4.3 de verschillende modules en pagina's van het prototype. In paragraaf 4.4 bespreken we hoe de in paragraaf 3.7 besproken wijzigingen in de planning terugkomen in het prototype, en in paragraaf 4.5 kijken we kort terug op het prototype.

4.1 Softwaregenerator en prototype

Het idee achter de generator is dat bepaalde, veel voorkomende structuren in datamodellen in applicaties vaak tot vergelijkbare functionaliteit leiden. De generator biedt een aantal standaard componenten, die *behaviours* genoemd worden, waar delen van het datamodel 'ingegoten' kunnen worden. Deze behaviours kunnen door middel van configuratie aangepast worden naar de specifieke wensen van de klant. Een simpel voorbeeld van een behaviour is de zogenaamde *list*, waarin (een gedeelte van) de inhoud van een tabel grafisch weergegeven wordt. De behaviours worden door middel van modules in pagina's weergegeven. Met behulp van templates kan de lay-out van deze componenten aangepast worden. Verder bestaat er de mogelijkheid om met behulp van zogenaamde *usercode* op bepaalde 'plekken' binnen de applicatie bestaande functies te overschrijven, om zo een specifiek gedrag te realiseren. Een voorbeeld hiervan is de functie *onLogin*, die gebruikt kan worden om bij het inloggen een bepaalde actie uit te voeren. Het merendeel van de wensen van klanten kan met behulp van de bestaande componenten, de templates en de usercode opgevangen worden, maar toch blijkt dat bij ieder project weer functionaliteit vereist wordt waarvoor de generator aangepast dient te worden. De generator is dus constant in ontwikkeling.

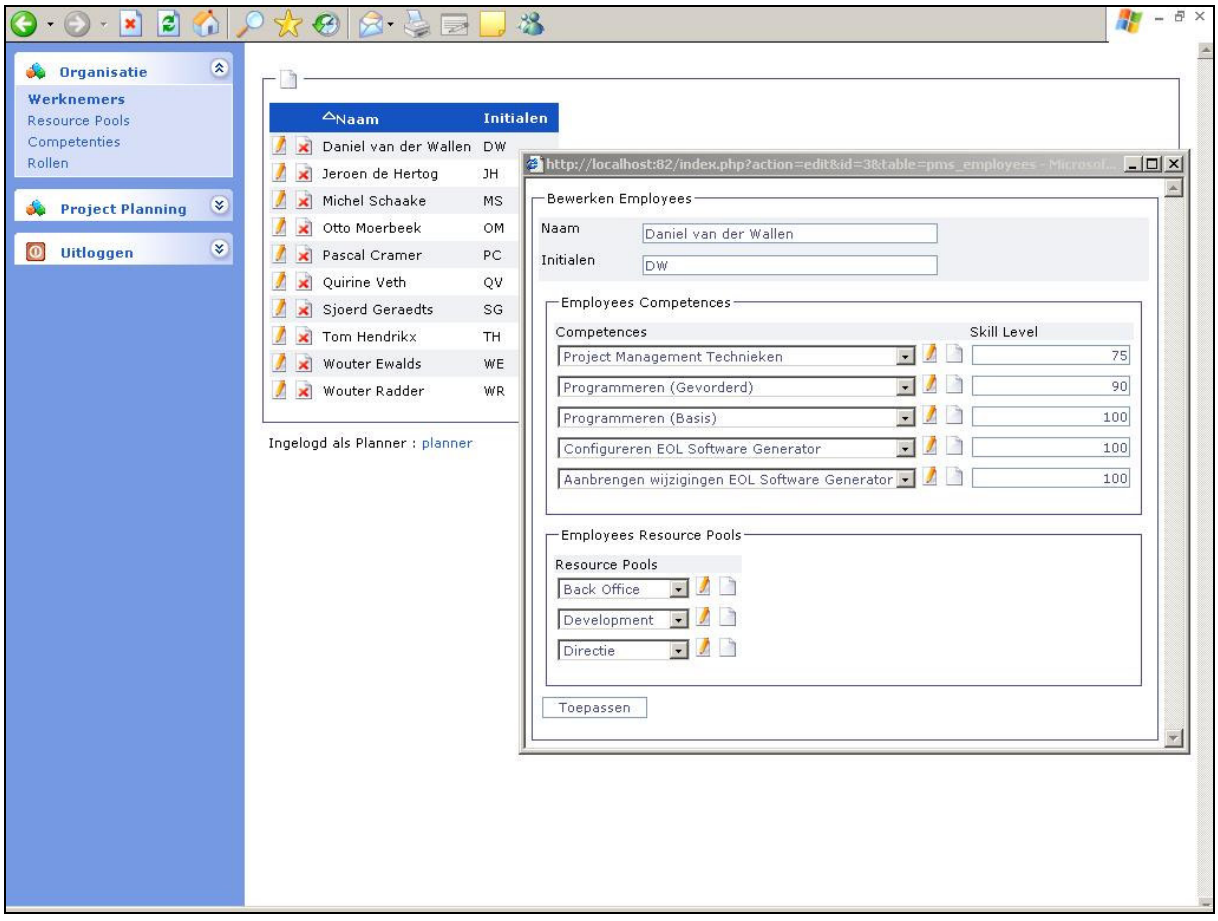
We hebben het prototype dus ontwikkeld binnen deze generator. Het prototype is opgebouwd uit een database, een nieuw behaviour, configuratie en usercode.
Het datamodel komt voort uit een directe vertaling van het model. De entiteitenverzamelingen binnen het model komen overeen met tabellen in het datamodel.

[1] We willen nogmaals benadrukken dat het gaat om een prototype, en niet om een volledig voltooide applicatie. Zoals vermeld is het model hierin slechts gedeeltelijk geïmplementeerd. Naast het feit dat de functionaliteit niet volledig is, is ook de layout nog niet definitief, er missen bijvoorbeeld icoontjes en vertalingen.

De attributen zijn velden van deze tabellen, en ook de relaties binnen het model worden vertaald in tabellen.

4.2 Organisatie module

In deze module worden de organisatorische gegevens, zoals besproken in paragraaf 3.3, beheerd. Het gaat dus om het beheren van werknemers, competenties, resource pools, rollen en de relaties tussen deze entiteiten. Elk van deze vier entiteiten (tabellen in termen van ons datamodel) hebben binnen de module een eigen pagina waarop de inhoud van de tabel beheerd kan worden. Achter elk van deze pagina's zit een *list behaviour*. Alle pagina's in deze module in elkaar gezet met behulp van configuratie opties die de software generator biedt.
In de onderstaande screenshot zien we hoe de lijst van werknemers eruit ziet.



Screenshot 1: de pagina "Werknemers" binnen de organisatie module en een venster waarin de gegeven van een werknemer bewerkt kunnen worden.

Naast de lijst met de werknemers uit ons voorbeeldproject zien we een popup venster waarin we de gegevens van een werknemer kunnen bewerken. Bovenaan dit venster staan de gegevens die directe bij de werknemer horen, te weten de velden *naam* en *initialen*.

Onderaan staan twee lijstjes waarin de relaties met de andere entiteiten weergegeven zijn. De bovenste lijst geeft alle competenties die de werknemer bezit met bijbehorend vaardigheidsniveau weer. In de onderste lijst staan alle resource pools waar de werknemer deel van uitmaakt. De overige pagina's van deze module zien er vergelijkbaar uit en bestaan uit lijsten van resource pools, competenties en rollen. We zullen deze pagina's daarom niet stuk voor stuk behandelen maar houden het voor deze module op de gegeven uitleg.

4.3 Planning module

Nu we de algemene gegevens van de organisatie vastgelegd hebben kunnen we beginnen met het plannen van projecten. Dit gebeurt in de module *Project Planning*. Deze module bestaat uit de pagina's *Project Definitie*, *Inroosteren Activiteiten*, *Planning*, *Kalender*, *Overzicht* en *Rooster*. In de onderstaande secties zullen we deze pagina's stuk voor stuk bespreken.

4.3.1 Projectdefinitie

Op deze pagina wordt de volledige projectdefinitie (zoals beschreven in [paragraaf 4.4]) opgesteld. De pagina is opgedeeld in twee gedeelten. Aan de linker kant van de pagina bevindt zich een selectie lijst waar een project geselecteerd kan worden. Door op het "<<"-linkje te klikken kan dit gedeelte van de pagina tijdelijk verborgen worden.
Aan de rechterkant worden de activiteiten van het geselecteerde project als een boom weergegeven. Bovenaan het rechter gedeelte van de pagina bevindt zich een paneel met daarop een aantal acties die gebruikt kunnen worden voor het opbouwen van de WBS[2]. We zullen nu voor alle acties een korte omschrijving van de achterliggende functionaliteit geven.

Nieuw Project
Maakt een nieuw project aan en opent een venster waarin de gegevens van het project kunnen worden ingevuld.

Activiteit Toevoegen
Voegt een nieuwe activiteit toe onder de geselecteerde activiteit.

Activiteit Bewerken
Opent een venster waarin de gegevens van de geselecteerde activiteit aangepast kunnen worden.

Activiteit Verwijderen
Verwijderd de geselecteerde activiteit, alle onderliggende activiteiten en alle bijbehorende afhankelijkheden.

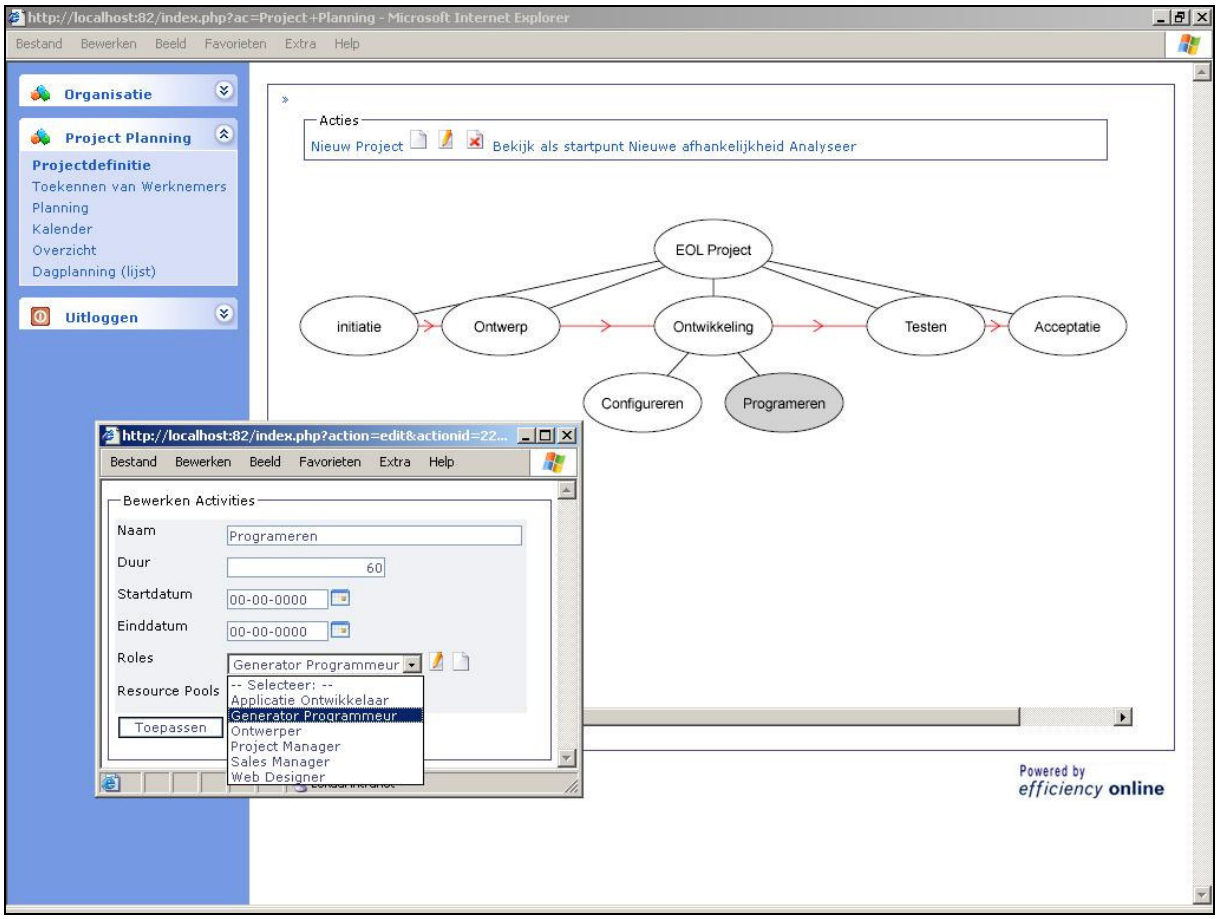
Bekijk als Startpunt
Geeft de activiteitenboom weer met de geselecteerde activiteit als wortel.

[2] Merk op dat nog niet alle acties een icoontje hebben

Nieuwe Afhankelijkheid
Maakt een nieuwe afhankelijkheid aan en opent een venster waarin de twee betrokken afhankelijkheden geselecteerd kunnen worden.

Analyseren
Controleert of aan alle invoereisen voldaan is. Indien dit het geval is wordt de kritieke pad analyse uitgevoerd en wordt voor elke taak uit het project de verzamelingen met potentiële uitvoerders bepaald.

Met de acties *Activiteit Toevoegen*, *Activiteit Verwijderen* en *Nieuwe Afhankelijkheid* kan de WBS opgebouwd worden. Met behulp van de actie *Activiteit Bewerken* kunnen de gegevens betreffende de geselecteerde activiteit opgegeven danwel gewijzigd worden.
In de onderstaande screenshot is de projectdefinitie pagina van ons voorbeeldproject te zien.



Screenshot 2: de pagina "Projectdefinitie" waarop een gedeelte van de WBS van ons voorbeeldproject zichtbaar is.

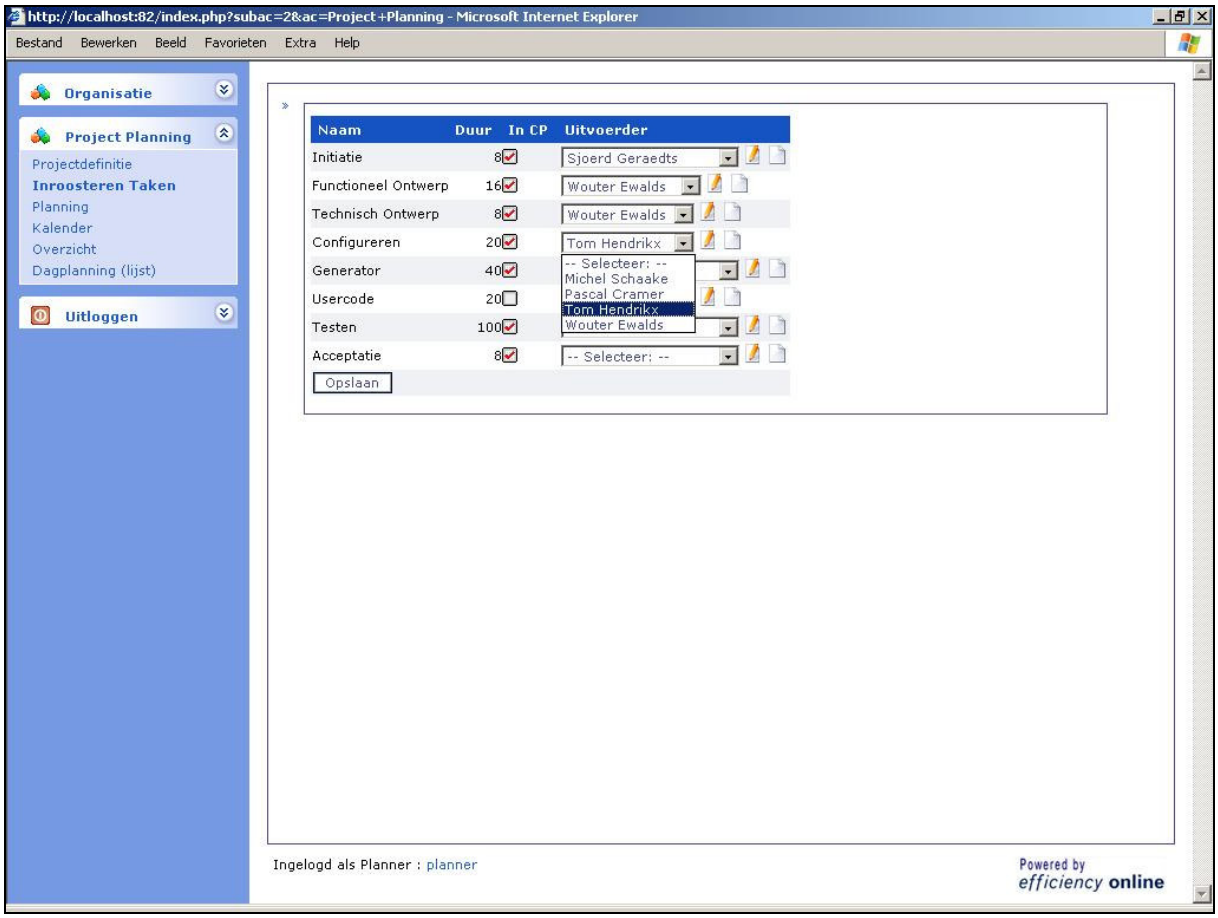
Merk op dat we in deze screenshot de linkerkant van het scherm (de lijst met projecten) verborgen hebben om een beter beeld van de WBS te kunnen geven. Merk tevens op dat

de WBS in deze screenshot niet de volledige WBS van ons voorbeeld project is. Verder zien we een tweede venster waarin de gegevens van de activiteit *Programmeren* bewerkt kunnen worden.
Wanneer de WBS volledig opgebouwd is en alle benodigde gegevens ingevoerd zijn kan er (door middel van de actie *Analyseer*) begonnen worden met het analyseren van de invoer. Dit gebeurt in een aantal stappen. Ten eerste wordt er gecontroleerd of er aan alle relevante invoereisen voldaan is.[3] Indien dit niet het geval is worden er verder geen acties ondernomen.
Als er wel aan de invoereisen voldaan is, wordt voor elke taak uit het project de lijst met kandidaat uitvoerders bepaald en opgeslagen in de database.[4] Vervolgens kan er begonnen worden met het bepalen van de theoretische optimale planning. Dit gebeurd exact volgens het in paragraaf 3.5 beschreven proces: het in deze paragraaf gedefinieerde gedeelte van het model is letterlijk geïmplementeerd in het prototype.
Wanneer de actie *Analyseer* succesvol voltooid is, hebben we de theoretisch optimale planning bepaald. De volgende stap betreft het daadwerkelijk plannen van het project. Dit gebeurt op de pagina *Inroosteren Activiteiten*.

4.3.2 Inroosteren activiteiten

Deze pagina bestaat wederom uit twee gedeelten. De linkerkant is gelijk aan de linkerkant van de projectdefinitie pagina en bevat dus een selectielijst met daarop de verschillende projecten. De rechterkant bevat een lijst met daarin de nog niet voltooide taken uit het geselecteerde project. De lijst is gesorteerd op volgorde van oplopende *latest start date* zoals deze berekend is tijdens het kritieke pad algoritme. In screenshot 3 zien we deze lijst voor ons voorbeeldproject.

[3] Het betreft de in [paragraaf 5.3] gedefinieerde invoereisen 1, 2, 3 en 4.
[4] Zie het laatste gedeelte van de sectie *Specificeren van Activiteiten* uit [paragraaf 4.4].



Screenshot 3: de pagina "Inroosteren Taken".

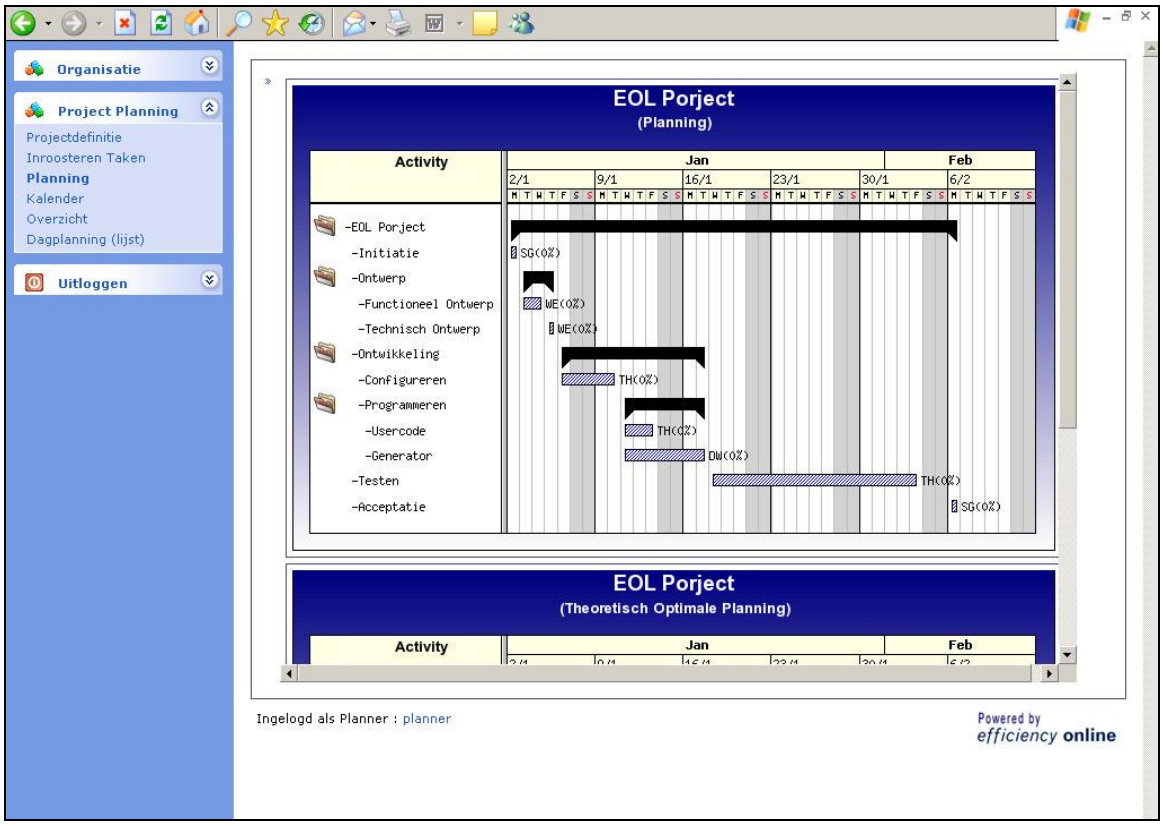
Zoals we in de screenshot zien, kunnen we in deze lijst voor iedere taak een uitvoerder selecteren. Er kan alleen gekozen worden uit de kandidaat uitvoerders die voor iedere taak impliciet vastgelegd zijn door de ingevoerde rol en resource pool. In paragraaf 3.6 hebben we een mechanisme beschreven waarbij de planner door middel van geschiktheidscores ondersteund wordt bij het uitkiezen van de verschillende werknemers. We hebben er in verband met de beperkte tijd voor gekozen dit mechanisme nog niet te implementeren in het prototype. Het idee is dat wanneer dit mechanisme geïmplementeerd de score in de selectielijstjes achter iedere werknemer weergegeven wordt.

Wanneer er voor alle taken uit de lijst een werknemer geselecteerd is, kunnen deze ingeroosterd worden. Door op de *opslaan*[5] knop te drukken wordt het in paragraaf 3.6 beschreven algoritme voor het inroosteren van de taken uitgevoerd.
Zodra dit voltooid is, is de eerste versie van de planning klaar. In de overige pagina's van deze module wordt de planning op verschillende niveaus en methoden weergegeven.

[5] Het opschrift: "Opslaan" is de standaard vertaling van de generator, hier ontbreekt dus een specifieke vertaling.

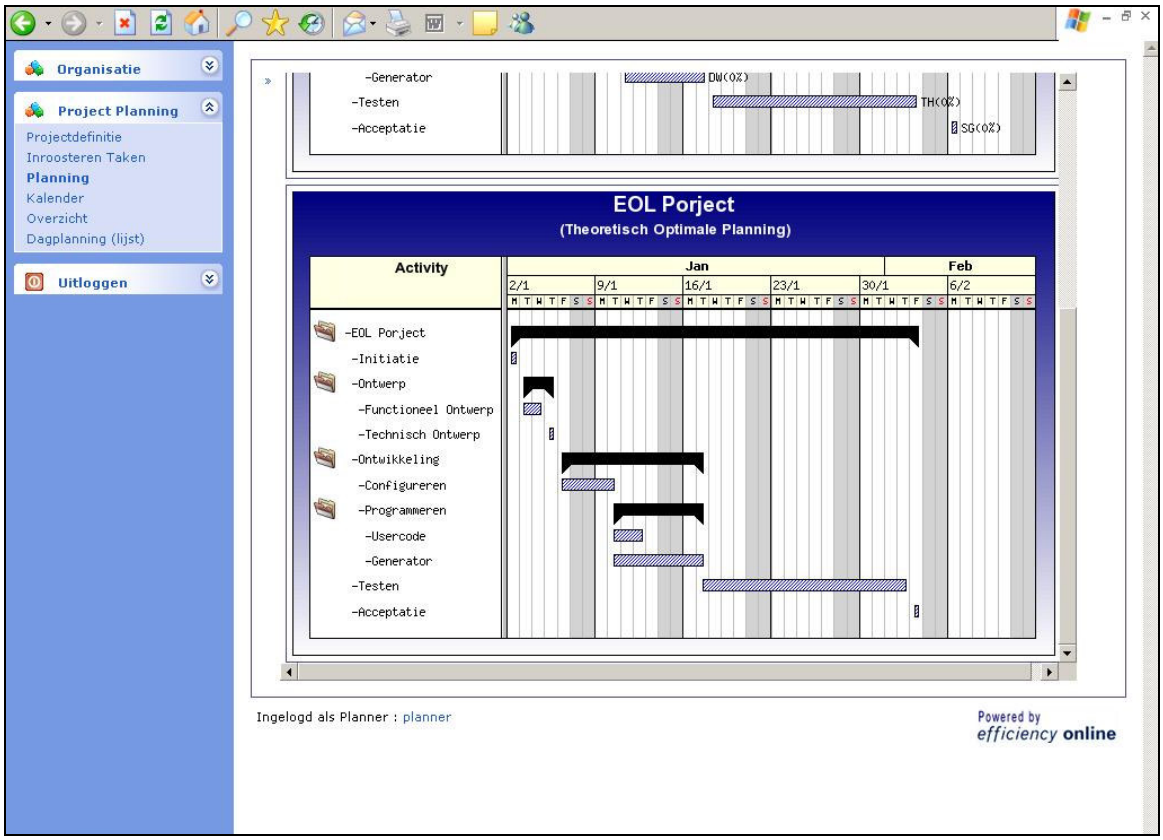
4.3.3 Planning

Op deze pagina wordt de planning van een project weergegeven in de vorm van een zogenaamde Gantt Chart. Net als de voorheen besproken pagina's is ook deze pagina opgedeeld in twee gedeelten, waarbij aan de linkerkant dezelfde selectie lijst met projecten bevat. Aan de rechter kant zijn twee Gantt Charts te zien; één voor de actuele planning, en één voor de theoretisch optimale planning. In screenshot 4 zien we hoe (een gedeelte van) de pagina eruit ziet.



Screenshot 4: de pagina "Planning", met daarop zichtbaar de Gantt Chart voor de actuele planning.

In de screenshot zien we een Gantt Chart van de planning van ons voorbeeldproject. Zowel de abstracte activiteiten als de taken zijn in dit schema weergegeven. De Gantt Chart is opgedeeld in twee delen. Aan de linkerkant zien we een lijst met alle activiteiten van het project. In de lijst zijn de verschillende niveaus van activiteiten zichtbaar door middel van het inspringen van de items in de lijst. Aan de rechter kant zien we een tijdslijn, waarin de periode van uitvoering van alle activiteiten uit het lijstje weergegeven is. De abstracte activiteiten zijn weergegeven als zwarte balkjes en de taken als blauw geruite balkjes. Achter elke taak zijn de initialen van de van de uitvoerende werknemers weergegeven, en daarachter zien we het percentage van de taak dat reeds voltooid is.
Wanneer we in de pagina omlaag scrollen zien we de Gantt Chart voor de theoretisch optimale planning (zie screenshot 5). De opbouw van dit schema is gelijk aan die van het schema van de actuele planning, met als verschil dat er in de theoretische planning geen werknemers en voortgang weergegeven worden.

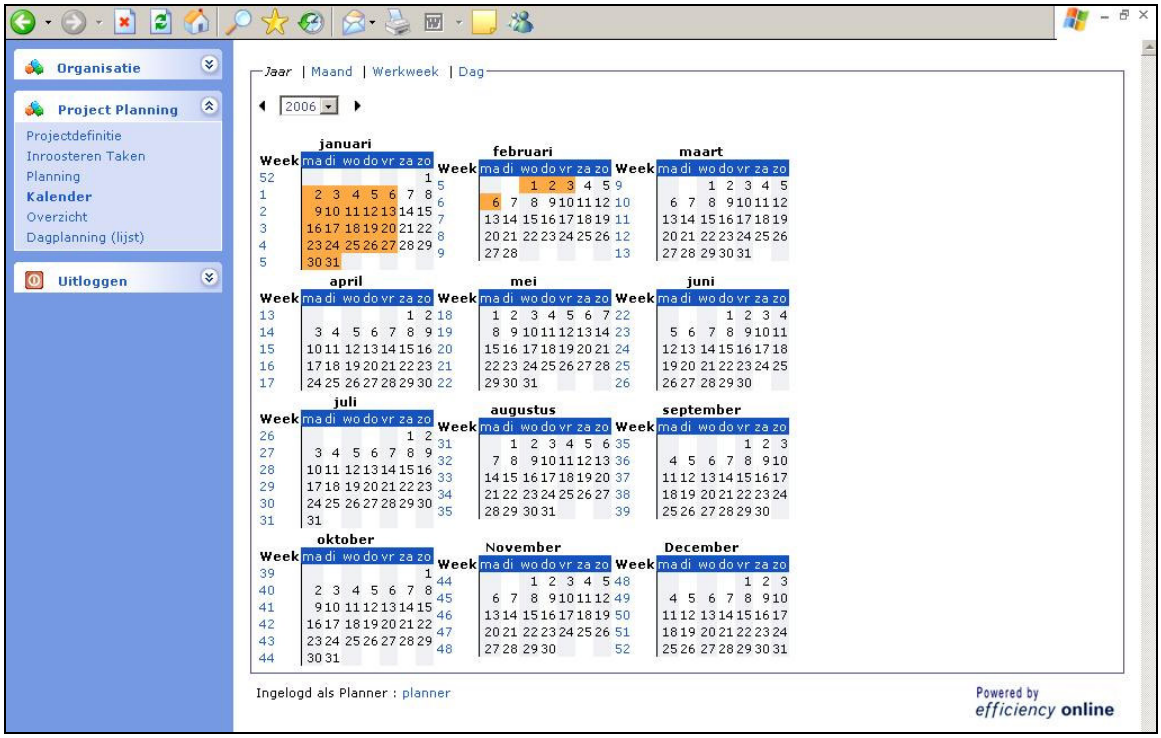


Screenshot 5: de pagina "Planning", met daarop zichtbaar de Gantt Chart voor de theoretisch optimale planning.

Door deze twee schema's met elkaar te vergelijken kunnen we een goed beeld krijgen van de kwaliteit van de actuele planning. Merk op dat afwijkingen van de theoretisch optimale planning in de praktijk lang niet altijd per definitie slecht nieuws zijn. In veel gevallen is niet nodig, of zelfs niet wenselijk, om een project zo snel mogelijk af te hebben.

4.3.4 Kalender

Op deze pagina vinden we een kalender waarin de ingeroosterde events weergegeven worden. Het achterliggende behaviour van deze kalender is momenteel nog in ontwikkeling. De kalender functioneert in ons prototype dus nog niet naar behoren. We zullen ons in dit verslag dan ook beperken tot het geven van een screenshot en het uitleggen van het idee achter de kalender met betrekking tot ons model.

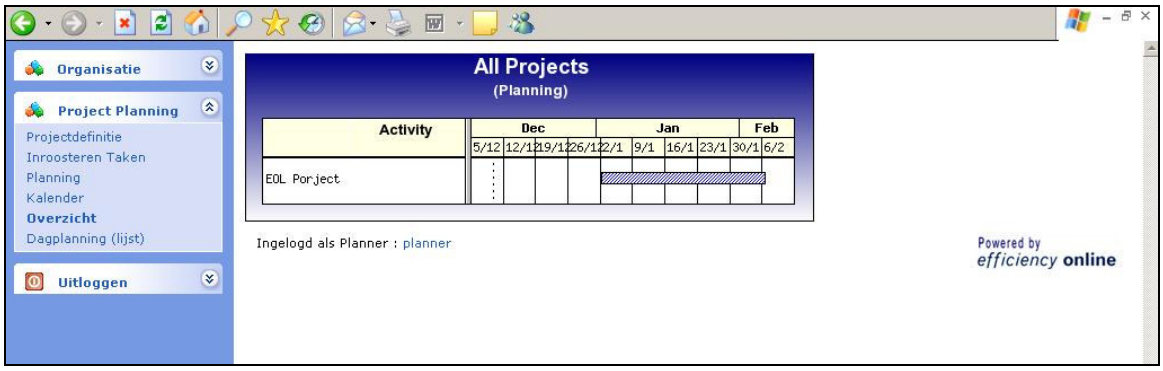


Screenshot 6: een eerste versie van de pagina "kalender".

Het idee is dat we een aantal verschillende views op de kalender gaan creëren. Naast een view met daarin alle events voor alle projecten denken we bijvoorbeeld aan een view waarbij we alleen de events van een specifiek project zien. Een ander voorbeeld is een view waarbij we events van de momenteel ingelogde werknemer te zien krijgen.

4.3.5 Overzicht

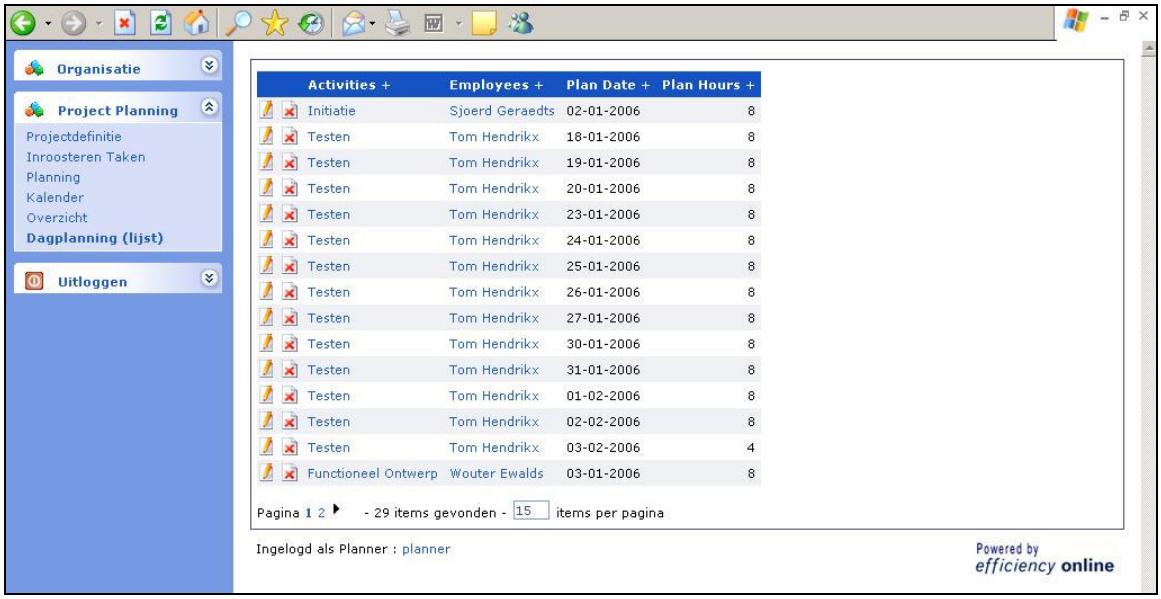
Op deze pagina vinden we een overzicht van de planning van alle projecten binnen de betreffende organisatie. Wederom is er gekozen voor het gebruik van een Gantt Chart. In plaats van de activiteiten worden hier de verschillende projecten weergegeven. In screenshot 7 zien we hoe dit eruit ziet.



Screenshot 7: de pagina "overzicht", met daarop de Gantt Chart waarop alle projecten weergegeven worden.

4.3.5 Dagplanning

Op deze pagina vinden we een lijst met alle ingeroosterde events in het systeem (zie de onderstaande screenshot).



Screenshot 8: de pagina "dagplanning", met daarop de lijst van alle ingeroosterde events uit het systeem.

We zien in de lijst voor ieder event de betreffende activiteit, de werknemer voor wie het event ingeroosterd is, de datum waarop het event ingeroosterd is en het aantal uren die op die dag ingepland zijn. Met behulp van filters (een standaard functie van het list behaviour in de generator) kunnen we de weergegeven events beperken. Zo kunnen we bijvoorbeeld alle events voor een bepaalde activiteit of voor een bepaalde werknemer opvragen[6]. We willen deze pagina uiteindelijk gebruiken voor het aanbrengen van

[6] Merk op dat wat nog ontbreekt in het prototype is de mogelijkheid om alle events van een bepaald project weer te geven. Dit zal in een latere uitbreiding van het systeem toegevoegd moeten worden.

wijzigingen in het rooster. Hierover vinden we meer terug in de volgende paragraaf die gaat over het wijzigen van een gemaakte planning in het prototype.

4.4 Wijzigingen in de planning

In paragraaf 3.7 hebben we een aantal mogelijkheden aan ons model toegevoegd om een gemaakte planning te wijzigen. In deze paragraaf zullen we bespreken welke van deze wijzigingen binnen het prototype reeds geïmplementeerd zijn, hoe de geïmplementeerde wijzigingen in zijn werk gaan, en hoe het prototype opgezet is om in de toekomst met de overige wijzigingen om te gaan. Hierbij behandelen we één voor één de in paragraaf 3.7 besproken categorieën van wijzigingen, te beginnen met de wijzigingen naar aanleiding van de voortgang van projecten.

4.4.1 Voortgang van het project

Het model is zo opgezet dat het om kan gaan met de voortgang van projecten. Samengevat komen deze omgang erop neer dat er:

1. alleen de niet-voltooide activiteiten beschouwd worden
2. er met de resterende tijd gerekend wordt in plaats van met de volledige duur
3. bij het inroosteren van de taken eerst alle bestaande events van het betreffende het project verwijderd worden.

Deze functionaliteit is volledig opgenomen in de implementatie van de betreffende algoritmen. De interface van ons prototype biedt echter nog niet de mogelijkheid om de resterende duur van een activiteit via de applicatie in te voeren. Het idee is dat deze resterende duur ofwel vanuit de projectdefinitie pagina met de hand ingevoerd wordt, ofwel via de urenregistratie automatisch berekend wordt. Beide mogelijkheden zijn dankzij de mogelijkheden die de generator biedt relatief eenvoudig in te bouwen via configuratie dan wel usercode.
Wanneer we de planning willen aanpassen aan de voortgang, moeten we de projectdefinitie opnieuw laten analyseren (door middel van de actie *analyseer* op de *projectdefinitie* pagina) en vervolgens opnieuw de taken inroosteren (door middel van de *Opslaan* knop op de pagina *inroosteren taken*).

4.4.2 Wijzigingen in de projectdefinitie

Ook deze categorie van wijzigingen is volledig geïmplementeerd binnen het prototype. Op de projectdefinitie pagina kunnen activiteiten en afhankelijkheden toegevoegd en verwijderd worden en kunnen ingevoerde gegevens bij de activiteiten gewijzigd worden.
Om de planning bij te werken dient, na het aanbrengen van wijzigingen in de projectdefinitie, deze opnieuw geanalyseerd te worden, en moeten vervolgens de taken opnieuw ingeroosterd worden. Indien er bij een activiteit wijzigingen in de vereiste rol of resource pool zijn gemaakt, kan het voorkomen dat de huidige toegekende werknemer niet meer in de lijst met potentiële zit. In dit geval dient er eerst opnieuw een werknemer aan de activiteit toegekend te worden alvorens deze ingeroosterd kan worden.

Merk op dat wanneer er wijzigingen in de projectdefinitie gemaakt zijn, maar deze niet opnieuw geanalyseerd en/of de taken niet opnieuw ingeroosterd zijn, de huidige planning op achterhaalde informatie gebaseerd is. In het prototype wordt hier nog niets mee gedaan, maar wanneer het prototype uitgebreid wordt tot een volledig systeem willen we een methode toevoegen om hiermee om te gaan.

4.2.3 Wijzigingen in de toekenning van werknemers

We kunnen op elk moment op de pagina *Inroosteren Taken* de uitvoering van een taak aan een andere werknemer toekennen. Om de planning hierop aan te passen moeten we dan uiteraard wel de taken opnieuw inroosteren. Doordat een wijziging in de toekenning van een werknemer pas opgeslagen wordt bij het drukken op de *Opslaan* knop, en deze ook direct de taken opnieuw inroostert, kunnen er naar aanleiding van wijzigingen in de toekenning van werknemers geen synchronisatie probleem ontstaan zoals bij het maken van wijzigingen in de projectdefinitie.

Wijzigingen in het rooster

De mogelijkheid om wijzigingen in het rooster te maken, is momenteel nog niet geïmplementeerd in het prototype. Het idee is dat er vanuit de pagina *Dagplanning* events gewijzigd, toegevoegd en verwijderd kunnen worden. Hiervoor moeten echter eerst de in paragraaf 3.7 beschreven modeluitbreidingen geïmplementeerd worden.

4.5 Conclusie

We hebben een prototype ontwikkeld waarin het in hoofdstuk 3 beschreven model gedeeltelijk geïmplementeerd is. Het prototype bestaat uit twee modules, die ieder onderverdeeld zijn in een aantal pagina's.
In de *Organisatie Module* kunnen de belangrijke organisatorische gegevens beheerd worden. In de *Planning Module* kunnen projecten gepland worden. De projectdefinitie kan volledig opgesteld worden. Vervolgens kan de theoretische optimale planning bepaald worden, kunnen er werknemers aan de activiteiten toegekend worden, en kan er een rooster gegenereerd worden. Op basis van dit rooster wordt de planning doorberekend.
Deze planning kan vervolgens bekeken worden in de vorm van een Gantt Chart, en vergeleken worden met de theoretisch optimale planning. Ook kan de planning van meerdere projecten tegelijk kan in een Gantt Chart weergegeven worden. De projectdefinitie en toekenning van werknemers kunnen aangepast worden, waarna het rooster en de planning opnieuw gegenereerd kunnen worden, om zo de voortgang van het project te volgen.
Het gegenereerde rooster kan in de vorm een lijst met events bekeken worden. Ook bevindt zich binnen het prototype een kalender waar deze events op terug te vinden zijn. Deze kalender is echter nog niet volledig en is nog in ontwikkeling. De opzet is dat het rooster zowel vanuit de kalender als vanuit de lijst met de hand aangepast kan worden.

Alvorens het prototype in de praktijk inzetbaar is, zal het eerst verder ontwikkeld moeten worden. Omdat het prototype binnen de software generator van Efficiency Online

ontwikkeld is, kunnen veel aanpassingen en uitbreidingen relatief eenvoudig gemaakt worden. In het volgende hoofdstuk zullen we hier nader op in gaan.

Hoofdstuk 5: Conclusies

In paragraaf 1.3 hebben we de doelstelling van de stage gedefinieerd:

> *Het **doel** van de stage is het opstellen van een model en het maken van een (werkend) prototype dat de basis vormt voor een zogenaamd decision support system (DSS) voor projectplanning. Dit systeem dient uiteindelijk zowel gebruikt te worden voor de interne planning van Efficiency Online, als aangeboden worden aan de klanten van Efficiency Online. Dit prototype dient ingebouwd te worden binnen de software generator van Efficiency Online.*

Deze doelstelling is behaald: er is een model opgesteld waarmee in een aantal stappen eenvoudig een planning gegenereerd kan worden. De kracht van dit model is dat de gegenereerde planning mee kan groeien met de realiteit.

Het model is geïmplementeerd binnen de softwaregenerator van Efficiency Online in de vorm van een werkend prototype. Dit prototype biedt een webinterface

In de volgende paragraaf zullen we een meer uitgebreide evaluatie van de doelstelling geven. Vervolgens wordt in paragraaf 5.2 beschreven welke stappen er ondernomen dienen te worden om van het prototype een volwaardige applicatie te maken.

5.1 Evaluatie van de doelstelling

In paragraaf 1.3 hebben we een aantal voorwaarden opgesteld, waaraan het resulterende model dient te voldoen. We bespreken nu hoe en in hoeverre ons model en prototype aan deze voorwaarden voldoet.

Transparantie
We hebben gesteld dat het bij het gebruik van het systeem voor de gebruiker te allen tijde duidelijk moet zijn wat het precies doet. We zijn van mening dat het model op een manier is opgezet waarbij dit goed gerealiseerd kan worden.
Met name de aanname dat alle laagste niveau activiteiten “één persoon taken” zijn draagt hieraan bij. Deze aanname zorgt ervoor dat de invoer van de duur van een taak slechts op één manier geïnterpreteerd kan worden.
Wanneer we echter naar het prototype kijken moeten we toegeven dat de transparantie hier nog niet van het gewenste niveau is. Dit wordt grotendeels veroorzaakt door een gebrek aan tekstuele feedback: er is binnen het prototype nog geen systeem aanwezig voor het geven van meldingen over wat er precies gebeurd. Ook het ontbreken van een helpfile speelt hierbij een rol. Dit zijn dus duidelijk punten waarop het systeem nog verbeterd kan worden.

Beperkte complexiteit
We hebben ervoor gekozen om de complexiteit van het model te beperken. Dit wordt met name gerealiseerd door de keuze voor deterministische schattingen voor de duur van

activiteiten. Dit heeft als nadeel dat we de onzekerheid in duur van de activiteiten niet meenemen bij het inroosteren van deze activiteiten. We zijn echter van mening dat deze beperking in complexiteit in combinatie met de mogelijkheid om de planning “mee te laten groeien” met de uitvoering een grotere toegevoegde waarde levert dan het invoeren van een stochastische duur van de activiteiten.

Hanteerbaarheid
We zijn van mening dat voor zowel het model als het prototype de hanteerbaarheid hoog is. De hoeveelheid benodigde invoer is goed te overzien, en ook het aantal stappen dat doorlopen dient te worden voor het genereren van een planning/rooster is beperkt.

Flexibiliteit
Doordat het model op een dusdanig wijze is opgezet dat de planning van een project ten aller tijden opnieuw bepaald kan worden, hebben we in ons model een hoge mate van flexibiliteit bereikt. Ook de mogelijkheid om het gegenereerde rooster met de hand aan te passen draagt hier in grote mate aan bij.

Al met al moeten we concluderen dat we in behoorlijke mate aan de voorwaarden, en dus de doelstelling voldaan hebben. Met name het punt transparantie met betrekking op de implementatie vereist nog de nodige aandacht. Verder is het implementeren van de mogelijkheid om met de hand wijzigingen in het rooster te maken een vereiste uitbreiding voordat het prototype in gebruik genomen kan worden voor de interne projectplanning van Efficiency Online.

5.2 Van Prototype naar volwaardige applicatie

In deze paragraaf bespreken we een aantal potentiële uitbreidingen van het model/prototype. Een aantal van deze uitbreidingen beschouwen we als cruciaal voordat het prototype in gebruik genomen kan worden als een volwaardig systeem. Andere van de uitbreidingen die we noemen zijn slechts een opzet en vereisen nog het nodige onderzoek om de toegevoegde waarde en haalbaarheid vast te leggen.

Wijzigen rooster

In ons model hebben we de mogelijkheid opgenomen om wijzigingen aan te brengen in het rooster. Deze wijzigingen hebben we echter nog niet geïmplementeerd. We beschouwen dit als een cruciale uitbreiding voor het in gebruik nemen van het systeem.

Verbetering van de interface

Deze uitbreiding heeft betrekking op het prototype. Met de huidige interface van het prototype is het model geïmplementeerd, maar de werkbaarheid is niet heel erg hoog. Ook vanuit een cosmetisch oogpunt is de huidige interface niet optimaal. We beschouwen de huidige interface als voldoende voor het gebruik van de interne projectplanning binnen Efficiency Online, maar wanneer we het systeem ook aan klanten willen aanbieden zullen er de nodige verbeteringen in de interface aangebracht moeten worden. We beschouwen dit dus als een belangrijke uitbreiding/verbetering van het prototype.

Helpfile en feedback

Een andere uitbreiding die de werkbaarheid het systeem aanzienlijk zou vergroten is het toevoegen van een helpfile, en nog belangrijker, een degelijk mechanisme van meldingen die de gebruiker ten aller tijden overzicht geeft over de huidige status van de planning van het project waar hij op dat ogenblik mee bezig is.

Stochastiek en verzamelen van data

We hebben ervoor gekozen in de eerste versie van het model geen stochastiek te gebruiken. Deze keuze is bewust gemaakt om de complexiteit van het model binnen de perken te houden en de werkbaarheid te vergroten. Het schatten van kansverdelingen voor de duur van activiteiten is namelijk geen eenvoudige bezigheid, en verlaagt de werkbaarheid van het model in zekere mate. Dit is dan ook de voornaamste reden dat we gekozen hebben voor een deterministische duur van de activiteiten.
We zijn echter van mening dat wanneer we te maken hebben met projecten met een herhalend karakter, het verzamelen van data over de verschillende *typen* activiteiten ons de nodige informatie kan geven over de verdelingen van de duur van de activiteiten. Deze informatie kunnen we vervolgens gebruiken om bij het inroosteren van de activiteiten rekening te houden met de risico's van de activiteit. Dit is echter een uitbreiding die voorlopig een lage prioriteit heeft binnen Efficiency Online.

Project templates

Een mogelijke uitbreiding aan het model zou de invoering van project templates zijn. Het idee hierachter is dat we bij het aanmaken van de projectdefinitie de mogelijkheid hebben

om uit een lijstje templates te kiezen, welke ieder een vooraf gedefinieerde WBS bevatten. Vooral bij organisaties die veel projecten met een herhalend karakter hebben (zoals Efficiency Online) kan dit toegevoegde waarde leveren.

Scoringsmechanisme
In paragraaf 3.6 hebben we een mechanisme omschreven voor het toekennen van scores aan werknemers, om op een dergelijke manier de planner te helpen bij het kiezen van werknemers voor de verschillende taken. Het idee is dat er bij het bepalen van de scores rekening gehouden wordt met de beschikbare tijd van de verschillende werknemers.

Appendix A: Het model

In deze bijlage vinden we het volledige model zoals we dit tijdens de stage opgesteld hebben terug. Dit model bestaat uit de definitie van verzamelingen, entiteiten, relaties, algoritmen en hun onderlinge samenhang.
In appendix A1 vinden we een overzicht van alle bij het model behorende verzamelingen, entiteiten en relaties. In Appendix A2 vinden we de algoritmen terug.

Appendix A1: Verzamelingen, Entiteiten en Relaties

In deze appendix vinden we alle in het model gedefinieerde entiteiten met de bijbehorende attributen en relaties.

Employees*: E_i ε *E

Attributen:

- E_i.hrs$_j$ (integer): het aantal uren dat werknemer E_i werkt op dag j van de week.

Resource Pools*: RP_i ε *RP

Relaties:

- emprp(E_i,RP_j) (binary) = 1 als werknemer E_i deel uitmaakt van resource pool RP_j, 0 anders

Afgeleide Verzamelingen:

- *E*(RP_q) = {E_p ε *E*| emprp(E_p, RP_q) = 1}

Competences*: C_i ε *C

Relaties:

- *skill*(E_i,C_j) ε (0, 100): "het *vaardigheidsniveau* van werknemer E_i in competentie C_j"

Afgeleide Verzamelingen:

- *E*(C_i) := {E_k ε *E* | *skill*(E_k,C_i)> 0}: "de verzameling van werknemers die competentie C_i bezitten".

Roles*: RL_i ε *RL

Relaties:

- crl(C_i, RL_j) (binary) = 1 als competentie C_i deel uitmaakt van rol RL_j, 0 anders

Afgeleide Verzamelingen:

- *C*(RL_q) = {C_p | $rlc_{p,q}$ = 1}

Activities*: A_i ε *A

Attributen:

- A_i.duration (integer): de duur van activiteit A_i in uren.
- A_i.minsdate (date): de minimale startdatum van activiteit A_i.

- A_i.maxedate (date): de maximale einddatum van activiteit A_i.
- A_i.minsuchours (integer): minimale achtereenvolgende werkduur van activiteit A_i.
- A_i.es (date-time): de *earliest start* van activiteit A_i.
- A_i.ef (date-time): de *earliest finish* van activiteit A_i.
- A_i.ls(date-time): de *latest start* van activiteit A_i.
- A_i.lf(date-time): de *latest finish* van activiteit A_i.
- A_i.slack (integer): de speelruimte voor de uitvoering van activiteit A_i.
- A_i.in_cp (binary) = {1 als A_i in het kritieke pad zit, 0 anders
- A_i.employee (Employee E_k ε *E*): de werknemer die taak A_i gaat uitvoeren
- A_i.plan_startdate (date): de geplande startdatum van activiteit, gebaseerd op de ingeroosterde events
- A_i.plan_enddate (date): de geplande einddatum van activiteit, gebaseerd op de ingeroosterde events
- A_i.remaining_duration (integer): het geschatte aantal uren dat er nog openstaat voor het voltooien van activiteit A_i.

Relaties:
- sub(A_i,A_j) ε *SUB*
- sub(A_i,A_j) (binary) = 1 als A_j een subactiviteit van A_i is, 0 anders.
- dep(A_i, A_j) (binary) = {1 als activiteit A_j afhankelijk is van de voltooiing van activiteit A_i, 0 anders
- cpdep(A_i, A_j) (binary) = {1 als activiteit A_j in het kritieke pad netwerk afhankelijk is van de voltooiing van activiteit A_i, 0 anders
- score(E_i,A_j) (percentage): de *geschiktheidscore* van werknemer E_i voor de uitvoering van activiteit A_j.
- del(A_k, dep(A_k, A_l)) = del(A_k, dep(A_j, A_k)) = "delete"
- del(A_k, E_l.activity)) = "delete"
- del(A_k, sub(A_k,A_l)) = "delete"
- del(sub(A_k,A_l),A_l) = "delete"

Afgeleide Verzamelingen:
- *Project Activities*: *PROJECT(A_p)* := "de verzameling van activiteiten binnen project A_p".
- *SUB(A_p)* = { sub(A_i,A_j) | A_i, A_j ε *PROJECT(A_p)* & sub(A_i,A_j) = 1}: de verzameling van subactiviteit-relaties binnen project A_p
- *CHILD(A_i)* := {A_k ε *A* | *sub(A_i,A_k)* = 1 OR (sub(A_i,A_q) = 1 && A_k ε CHILD(A_q))}: Alle onderliggende activiteiten van activiteit A_i
- *PROJECT*(A_p) = *CHILD*(A_p) v {A_p)
- *TASKS(A_i)* := {A_k ε *CHILD*(A_i)| *sub(A_i,A_k)*= 0 voor alle *k*}: de laagste-niveau activiteiten van project A_p, oftewel de taken
- *E*(A_i.role) := de verzameling van werknemers die de rol van activiteit A_i kunnen vervullen
- *RP*(A_i) ⊂ *RP*: de verzameling van resource pools die aan activiteit A_i toegekend zijn.
- *E*(*RP*(A_i)): de verzameling van werknemers die ontstaat uit de vereniging van alle resource pools van activiteit A_i.
- *E*(A_i) = *E*(A_i.role) ∩ *E*(*RP*(A_i))

- *DEP*(A_p) = {dep(A_i,A_j) | A_i, A_j ε *PROJECT*(A_p) & dep(A_i,A_j) = 1}: de verzameling van afhankelijkheden binnen project A_p.
- *CPNW*(A_p) = TASKS(A_p) v {A_r}: de verzameling van kritieke pad netwerk activiteiten
- *CPDEP*(A_p) = {dep(A_i,A_j) | A_i, A_j ε *PROJECT*(A_p) & cpdep(A_i,A_j) = 1}: de verzameling van afhankelijkheden binnen het kirtieke pad netwerk van project A_p.
- avg_skill(E_k,A_l) (percentage): het gemiddelde vaardigheidsniveau van werknemer E_k over alle competenties benodigd voor de uitvoering van activiteit A_l.
- avg_skill(E_k,A_l) = (SUM(Cx in *C*(A_l.role)): skill(Ek, Cx)) / #(*C*(A_l.role))
- score(E_i,A_j) = avg_skill(E_k,A_l)

Events: EV_i ε *EV*

Attributen:

- EV_i.activity (Activity A_k ε *TASKS*(A_p)): de activiteit waar tijdens event EV_i aan gewerkt wordt.
- EV_i.employee (Employee E_k ε *E*): de werknemer voor wie het event EV_i ingeroosterd is.
- EV_i.plan_date (date): de datum wwaarop het event EV_i ingeroosterd is.
- EV_i.duration (integer in (1,24)): het aantal ingeroosterde uren.

Afgeleide Verzamelingen:

- *EV*(E_i): Alle events van werknemer E_i.
- *EV*(A_i): Alle events voor activiteit A_i.
- *EV*(E_i, A_j): Alle events van werknemer E_i voor activiteit A_j.
- *EV*(E_i, *date*): Alle events EV_k van werknemer E_i gepland op datum *date*.
- EV(A_p) = {EV_k ε *EV* | EV_k.activity ε *PROJECT*(A_p)}

Appendix A2: Algoritmen

In deze appendix vinden we alle algoritmen van het model terug. De algoritmen zijn genummerd van algoritme 1 tot en met algoritme 6. Een aantal van deze algoritmen zijn gedurende het opstellen van het model één of meerdere malen uitgebreid. Van deze algoritmen zijn alle versies gegeven, waarbij het

Algoritme 1a:
Algoritme voor het toevoegen van afhankelijkheden *naar* het project resultaat voor alle "hoogste niveau" activiteiten die nog geen "uitgaande" afhankelijkheden hebben.

```
run:
foreach (Ak in PROJECT(Ap)){
   if (sub(Ap,Ak)= 1 && Ak NOT IS Ar){
      if (foreach Al PROJECT(Ap): dep(Ak, Al)= 0){
         dep(Ak,Ar) = 1
      } else {
         dep(Ak,Ar) = 0
            }
      }
}
```

Algoritme 1b:
Het algoritme voor het toevoegen van afhankelijkheden *naar* het project resultaat voor alle "hoogste niveau" activiteiten die nog geen "uitgaande" afhankelijkheden hebben, uitgebreid om voor iedere taak uit het project een startwaarde aan de resterende duur te geven.

```
run:
foreach (Ak in PROJECT(Ap)){
   if (Ak in TASKS(Ap)) {
      if (Ak.remaining_duration = UNDEFINED) {
         Ak.remaining_duration = Ak.duration
      }
   }
   if (sub(Ap,Ak)=1 && Ak NOT IS Ar){
      if (foreach Al PROJECT(Ap): dep(Ak, Al)=0){
         dep(Ak,Ar) = 1
      } else {
         dep(Ak,Ar) = 0
      }
   }
}
```

Algoritme 2:
Algoritme voor het bepalen van de afhankelijkheden in het kritieke pad netwerk.

```
init:
cpdep(Ai,Aj) = 0 for all Ai, Aj in PROJECT(Ap)

run:
foreach (dep(Ak,Al) in DEP(Ap)){
```

```
   if (Ak in CP(Ap) && Al in CP(Ap)){
      cpdep(Ak,Al) = 1
   } elseif (Ak in CP(Ap)){
      foreach(Aq in TASKS(Al)){
         if(!hasIncWBSDeps(Aq)       &&       !hasIncCPDeps(Aq)       &&
         !parentHasIncDepLowerLvl(Aq, dep(Ak,Al))) {
            cpdep(Ak,Aq) = 1
         }
      }
   } elseif (Al in CP(Ap)){
      foreach(As in TASKS(Ak)){
         if(!hasOutWBSDeps(As) && !hasOutCPDeps(As) &&
         !parentHasOutDepLowerLvl(As, dep(Ak,Al))) {
            cpdep(As,Al) = 1
         }
      }
   } else {
      foreach(As in TASKS(Ak)){
         foreach(Aq in TASKS(Al)){
            if(!hasOutWBSDeps(As) && !hasOutCPDeps(As)&&
            !parentHasOutDepLowerLvl(As, dep(Ak,Al))) {
               if(!hasIncWBSDeps(Aq) && !hasIncCPDeps(Aq)&&
               !parentHasIncDepLowerLvl(Aq, dep(Ak,Al))) {
                  cpdep(As,Aq) = 1
               }
            }
         }
      }
   }
}
```

functions:

```
hasIncWBSDeps (Activity As){
   //controlleert of activiteit As inkomende afhankelijkheden in de WBS
   heeft
}

hasIncCPDeps (Activity As){
   //controlleert of activiteit As inkomende afhankelijkheden in de het
   kritieke pad netwerk heeft
}

hasOutWBSDeps (Activity As){
   //controlleert of activiteit As uitgaande afhankelijkheden in de WBS
   heeft
}

hasOutCPDeps (Activity As){
   //controlleert of activiteit As uitgaande afhankelijkheden in de het
   kritieke pad netwerk heeft
}


parentHasIncDepLowerLvl(Activity As, Dependency org_dep){
   //controlleert of er een parent van As is, welke een inkomende
   afhankelijkheid heft die op een lager niveau zit dan org_dep.
```

```
}


parentHasOutDepLowerLvl(Activity As, Dependency org_dep){
   //controlleert of er een parent van As is, welke een uitgaande
   afhankelijkheid heft die op een lager niveau zit dan org_dep.
}
```

Algoritme 3:
Het kritieke Pad Algoritme. In dit algoritme worden voor alle activiteiten uit het kritieke pad netwerk de earliest en latest start/finish bepaald

```
init:
foreach (As in CPNW(Ap)){
   Ai.es = 00-00-0000
   Ai.ef = 00-00-0000
   Ai.ls = 00-00-0000
   Ai.lf = 00-00-0000
}

run:
fillESAndEFRecursive(Ar)
Ar.ls = Ar.es
Ar.lf = Ar.ef
fillLSAndLFForPredessesorsOf(Ar)
foreach (As in CPNW(Ap)) {
   if (As.ef == As.lf) {
      As.in_cp = 1
   } else {
      As.in_cp = 0
   }
}


functions:
fillESAndEFRecursive(Activity As) {
   if not(As.es == 00-00-0000 && As.ef == 00-00-0000){
      stop
   }
   if (preActivities(As) == {empty set} ) {
      As.es = max(Ap.minsdate, Ar.minsdate, today + 1 day)
      As.ef = As.es + As.duration
   } else {
      max_pre_ef = Ap.startdate
      foreach (Aq in preActivities(As)){
         fillESAndEFRecursive(Aq)
         if (Aq.ef > max_pre_ef){
            max_pre_ef = Aq.ef
         }
      }
      As.es = max(max_pre_ef, As.minsdate)
      As.ef = As.es + As.duration
   }
}

fillLSAndLFForPredessesorsOf(Activity As){
```

```
   foreach (Aq in preActivities(As)){
      if (Aq.lf == 00-00-0000 && As.ls == 00-00-0000){
         Aq.lf = As.ls
         Aq.ls = Aq.lf - Aq.duration
      } else {
         if (As.ls < Aq.lf) {
            Aq.lf = As.ls
            Aq.ls = Aq.lf - Aq.duration
         }
      }
      fillLSAndLFForPredessesorsOf(Aq)
   }
}

preActivities(As) {
   /*
      geeft alle activiteiten die binnen het kritieke pad network een
         uitgaande afhankelijkheid naar activiteit As hebben
   */
}
```

Algoritme 4:
Met dit algoritme wordt de earliest en latest start/finish bepaald voor alle activiteiten die niet in het kritieke pad netwerk zitten

```
run:
foreach (Ak in Project(Ap)) {
   if (Ak not in Tasks(Ap)) {
      Ak.es = minimum(Aq in Tasks(Ak)): Aq.es
      Ak.ef = maximum(Aq in Tasks(Ak)): Aq.ef
      Ak.ls = minimum(Aq in Tasks(Ak)): Aq.ls
      Ak.lf = maximum(Aq in Tasks(Ak)): Aq.lf
   }
}
```

Algoritme 5a:
Algoritme voor het inroosteren van de taken uit project A_p.

```
run:
foreach(Ak in TASKS(Ap) ORDER Ak.ls){
   date_pointer = determineMinStartdate(Ak)'
   duration_covered = 0
   first_event = true
   while(duration_covered < Ak.duration) {
      free_hours = getFreeHoursOnDate(Ak.employee, date_pointer)
      if (free_hrs > Ak.minsuchours){
         schedule_hrs     =     minimum(free_hrs,     Ak.duration     -
                        duration_covered)
         new: EVn in EV
         EVn.activity = Ak
         EVn.employee = Ak.employee
         EVn.plan_date = date_pointer
         EVn.duration = schedule_hrs
         if (first_event){
            Ak.plan_startdate = date_pointer
```

```
            first_event = false
         }
         duration_covered = duration_covered + schedule_hrs
      }
      date_pointer = date_pointer + 1 day
   }
   Ak.plan_enddate = date_pointer - 1 day
}

functions:
determineMinStartdate(Activity Ak){
   //returns the minimum of:
   -current date + 1 day
   -Ak.es
   -Ak.minsdate
   -maximum{Aj | cpdep(Aj,Ak) = 1}: Ak.plan_enddate + 1 day
}

getFreeHoursOnDate(Employee Ek, Date date){
   j = dayOFTheWeek(date)
   result = Ek.hrsj
   foreach(EVk in EV(Ek, date)) {
      result = result - EVk.duration
   }
   if (result < 0){
      result = 0
   }
   return result
}

minimum(Integer a, Integer b){
   //returns the minimum value of the set {a, b}
}

function dayOFTheWeek(Date date){
   //returns the date's day of the week as a number between 1 and 7
}
```

Algoritme 5b:
Algoritme voor het inroosteren van de taken uit project A_p, uitgebreid voor het herplannen van taken bij afwijking van de planning en wijzigingen in de WBS.

```
run:
foreach(Ak in TASKS(Ap) ORDER Ak.ls){
   if(Ak.remaining_duraton > 0) {
      duration_covered = 0;
      first_event = true
      deleteEvents(Ap)
      date_pointer = determineMinStartdate(Ak)
      while(duration_covered < Ak.remaining_duration) {
         free_hours = getFreeHoursOnDate(Ak.employee, date_pointer)
         if (free_hrs > Ak.minsuchours){
            schedule_hrs     =     minimum(free_hrs,     Ak.duration     -
                        duration_covered)
            new: EVn in EV
```

```
            EVn.activity = Ak
            EVn.employee = Ak.employee
            EVn.plan_date = date_pointer
            EVn.duration = schedule_hrs
            if (first_event){
               Ak.plan_startdate = date_pointer
               first_event = false
            }
            duration_covered = duration_covered + schedule_hrs
         }
         date_pointer = date_pointer + 1 day
      }
      Ak.plan_enddate = date_pointer - 1 day
   }
}

functions:
determineMinStartdate(Activity Ak){
   //returns the minimum of:
   -current date + 1 day
   -Ak.es
   -Ak.minsdate
   -maximum{Aj | cpdep(Aj,Ak) = 1}: Ak.plan_enddate + 1 day
}

getFreeHoursOnDate(Employee Ek, Date date){
   j = dayOFTheWeek(date)
   result = Ek.hrsj
   foreach(EVk in EV(Ek, date)) {
      result = result - EVk.duration
   }
   if (result < 0){
      result = 0
   }
   return result
}

minimum(Integer a, Integer b){
   //returns the minimum value of the set {a, b}
}

function dayOFTheWeek(Date date){
   //returns the date's day of the week as a number between 1 and 7
}

deleteEvents(Project Ap){
   foreach(EVk in EV(Ap)){
      remove: EVk from EV
   }
}
```

Algoritme 5c:
Algoritme voor het inroosteren van de taken uit project A_p, uitgebreid voor het herplannen van taken bij afwijking van de planning, wijzigingen in de WBS en met de hand aangebrachte wijzigingen in het rooster.

```
run:
foreach(Ak in TASKS(Ap) ORDER Ak.ls){
   if(Ak.remaining_duraton > 0) {
      duration_covered = 0;
      first_event = true
      deleteEvents(Ap)
      date_pointer = determineMinStartdate(Ak)
      while(duration_covered < Ak.remaining_duration) {
         free_hours = getFreeHoursOnDate(Ak.employee, date_pointer)
         if (free_hrs > Ak.minsuchours){
            schedule_hrs    =    minimum(free_hrs,    Ak.duration    -
                           duration_covered)
            new: EVn in EV
            EVn.activity = Ak
            EVn.employee = Ak.employee
            EVn.plan_date = date_pointer
            EVn.duration = schedule_hrs
            if (first_event){
               Ak.plan_startdate = date_pointer
               first_event = false
            }
            duration_covered = duration_covered + schedule_hrs
         }
         date_pointer = date_pointer + 1 day
      }
      Ak.plan_enddate = date_pointer - 1 day
   }
}

functions:
determineMinStartdate(Activity Ak){
   //returns the minimum of:
   -current date + 1 day
   -Ak.es
   -Ak.minsdate
   -maximum{Aj | cpdep(Aj,Ak) = 1}: Ak.plan_enddate + 1 day
}

getFreeHoursOnDate(Employee Ek, Date date){
   j = dayOFTheWeek(date)
   result = Ek.hrsj
   foreach(EVk in EV(Ek, date)) {
      result = result - EVk.duration
   }
   if (result < 0){
      result = 0
   }
   return result
}

minimum(Integer a, Integer b){
   //returns the minimum value of the set {a, b}
}

function dayOFTheWeek(Date date){
```

```
   //returns the date's day of the week as a number between 1 and 7
}

deleteEvents(Project Ap){
   foreach(EVk in EV(Ap)){
      remove: EVk from EV
   }
}
```

Algoritme 6:
Algoritme voor het opslitsen van activiteit A_k in subactiviteiten naar aanleiding van het wijzigen van de uitvoerende werknemer voor een deel van de activiteit.

```
input:
   -activity (Ak)
   -startdate (sd)
   -enddate (ed)
   -employee (E1)

run:
   pre_period = 0;
   in_period = 0;
   post_period = 0;

   foreach(EVn in EV(Ak) ORDER EVp.plan_date){
      if (EVn.plan_date < sd) {
         pre_period += EVn.duration;
      } elseif(EVn.plan_date > ed) {
         post_period += EVn.duration;
      } else {
         in_period += EVn.duration;
      }

      if(pre_period > 0) {
         new: An in A
         An.duration = pre_period;
         An.maxedate = sd;
         sub(Ak,An) = 1;
      }

      if(in_period > 0) {
         new: An in A
         An.duration = in_period;
         An.minsdate = sd;
         An.maxedate = ed;
         sub(Ak,An) = 1;
      }

      if (post_period) {
         new: An in A
         An.duration = post_period;
         An.minsdate = ed;
         sub(Ak,An) = 1;
      }
   }
```

Appendix B: Literatuurlijst

- *Optimization of Business Processes: An Introduction to Applied Stochastic Moddeling*, Koole (2005)
- *Computers in de Samenleving*, Koetsier (2001)
- *Prince 2*, http://www.prince2.com/whatisp2.html
- *Wikipedia*, http://en.wikipedia.org/wiki/Main_Page
- *PERT*, http://www.netmba.com/operations/project/pert/