Werkboek

Software Life Cycle

Open Universiteit
Faculteit Management, Science & Technology

Cursusteam
Dr. ir. K.A.M. Lemmen, *cursusverantwoordelijke en auteur*
Ir. P. Oord, *auteur*

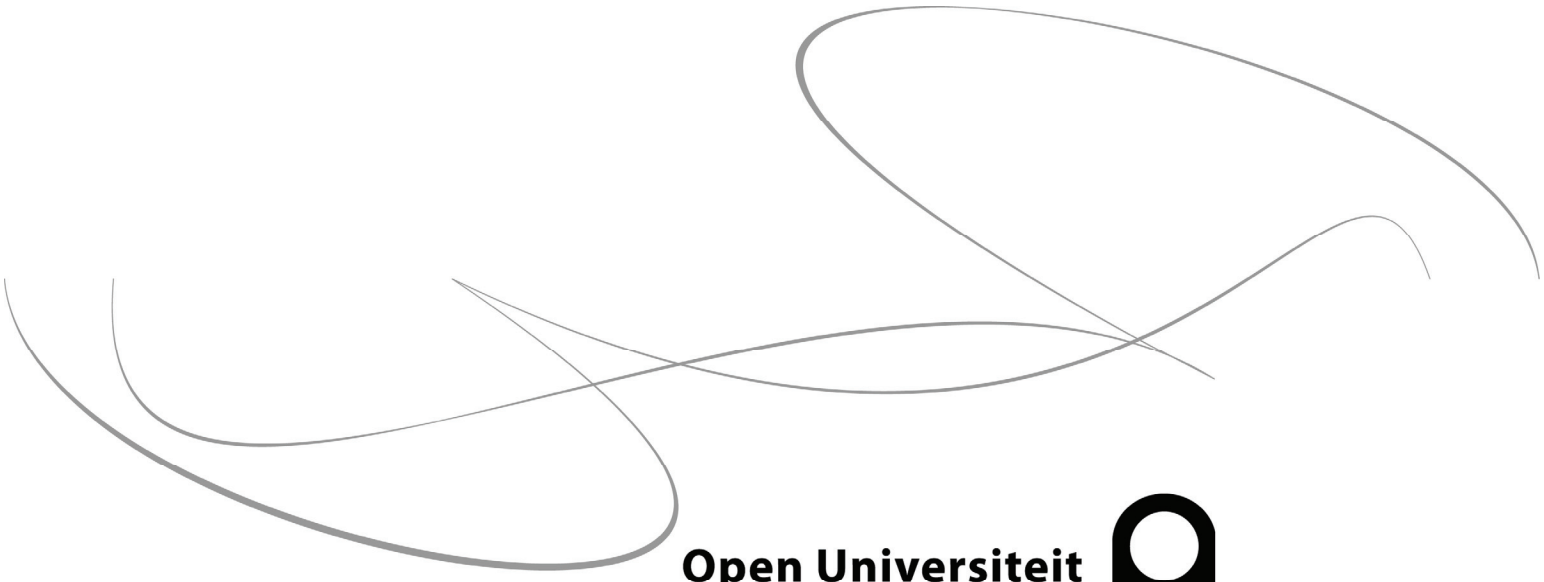
Deze cursus is grotendeels samengesteld uit delen van de cursussen Softwaremanagement (T24331) en Requirements engineering (T65311), waaraan de volgende auteurs hebben meegewerkt:
drs. M.J. te Boekhorst
ir. P. Oord
drs. Th.F. de Ridder
drs. H.J. Sint
ir. S. Stuurman
dr. W. Westera

Extern referenten
Prof. dr. A. van Lamsweerde
Dr. ir. D.M. van Solingen

Programmaleiding
Prof. dr. M.C.J.D. van Eekelen

WERKBOEK

Software Life Cycle



Open Universiteit
www.ou.nl

Productie
Open Universiteit

Redactie
John Arkenbout

Lay-out en illustraties
Maria Wienbröker-Kampermann

Omslag
Team Visuele communicatie, Open Universiteit

Druk– en bindwerk
OCÉ Business Services

Eerste druk: 2015

IM0303_50107_22012015

ISBN 978 94 91825 95 8 (werkboek)
ISBN 978 94 91825 92 7 (serie)

Cursuscode IM0303

Introductie tot de cursus

Introductie tot de cursus

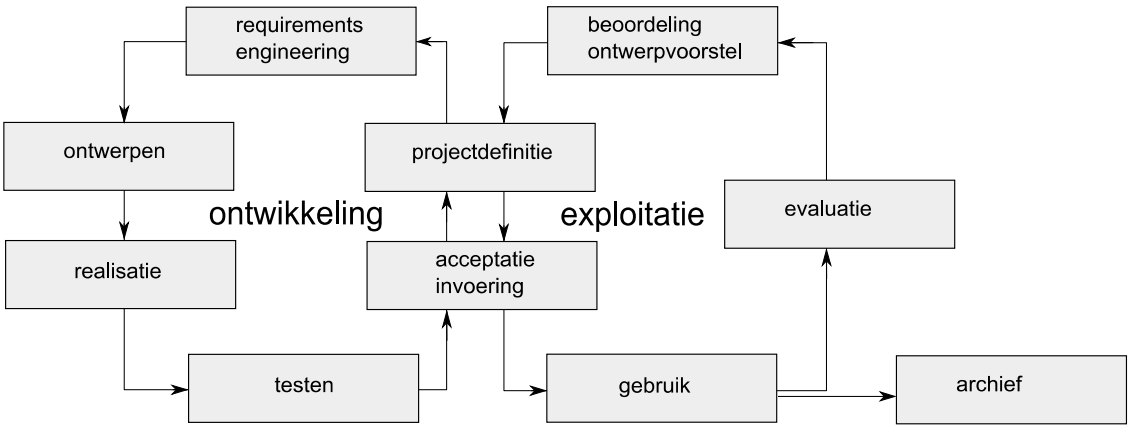
1 Inleiding

In deze cursus wordt een overzicht gegeven van enkele aspecten die van belang zijn voor het succesvol uitvoeren van softwareprojecten.
Elk softwareproduct kent een eigen levenscyclus, de Software Life Cycle (SLC) of vaak ook Software Development Life Cycle (SDLC) genoemd.
Een definitie van een SLC is (IEEE Standard Glossary of Software Engineering Terminology, 1983):

'The period of time that starts when a software product is conceived and ends when the product is no longer available for use. The software life-cycle typically includes a requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase and sometimes, retirement phase'.

Voor het uitvoeren van softwareprojecten zijn in de loop van de tijd verschillende modellen voor SLC's ontwikkeld, de software life cycle modellen (SLCM). Deze beschrijven de volgorde en de wijze waarop een aantal activiteiten uitgevoerd zou moeten worden om te komen tot een succesvol softwareproduct.
Elk SLC-model heeft eigen uitgangspunten en bij toepassing in de praktijk specifieke voor- en nadelen. Er zijn vele SLC-modellen in gebruik, al is in vrijwel elke situatie sprake van een mengvorm van een of meer modellen. Elk project, product en organisatie 'vragen' om een eigen invulling van een SLC-model. Kennis over belangrijke aspecten van SLC-modellen is dus belangrijk om in de praktijk bij een concreet product een geschikt model te kunnen kiezen.
Het is echter ondoenlijk om alle modellen, aspecten en fasen die van belang kunnen zijn in een cursus als deze te behandelen[i].

Wat wij in deze cursus aan de orde stellen is echter meer dan wat normaliter met SLC-modellen wordt bedoeld. We bekijken niet alleen de modellen zelf maar gaan ook verder in op de processen die binnen en buiten het softwareproces werkzaam zijn.
Hiervoor kijken we naar een algemeen model voor de interne processen bij de ontwikkeling en exploitatie van ict-systemen. Zie figuur 1.



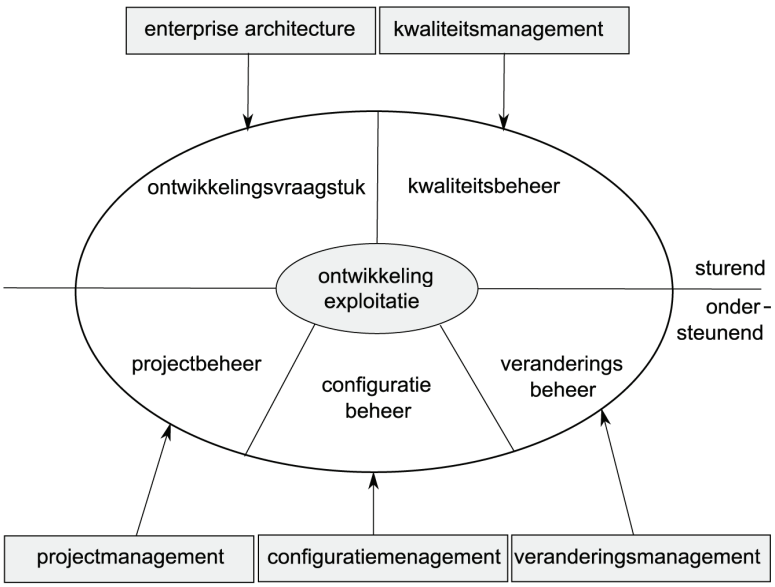
FIGUUR 1 Bicyclusmodel, interne processen

Dit model kenmerkt zich door twee in elkaar grijpende cycli (de ontwikkelingscyclus en de exploitatiecyclus) en wordt daarom ook wel het bicyclusmodel genoemd. De cycli bestaan uit processen die, wanneer ze sequentieel worden uitgevoerd overeenkomen met fasen in de ontwikkeling.

De ontwikkelingscyclus start met de projectdefinitie. Op grond hiervan volgen de fasen van requirements engineering, het ontwerpen, de realisatie en het testen van het systeem. Hierna volgt de acceptatie van het systeem en gaat dit in exploitatie. Het kan voorkomen dat een systeem niet geaccepteerd wordt. Dan volgt er, als er genoeg middelen beschikbaar zijn, een hernieuwde ontwikkelingscyclus om het systeem te verbeteren.

In de exploitatiecyclus zal het ict-systeem mogelijk na verloop van tijd geëvalueerd worden en komt er een verbetervoorstel (soms ook van IST naar SOLL-situatie genoemd) en een nieuwe projectdefinitie. Dit kan leiden tot een nieuwe ontwikkelingscyclus voor het systeem.
Het kan natuurlijk ook zijn dat het systeem om diverse redenen niet meer voldoet en dat er een compleet nieuw systeem ontwikkeld moet worden. Het huidige systeem raakt dan in onbruik (archief).

De bicyclus wordt aangestuurd door een aantal externe processen zoals weergegeven in figuur 2. Een intern proces kunnen we zien als de manifestatie van een extern proces bij een specifiek project.
De externe processen kunnen nog verdeeld worden in inhoudelijk sturende processen en projectondersteunende processen. In figuur 2 is dit schematisch weergegeven door de horizontale lijn.



FIGUUR 2 Bicyclusmodel, interne en externe processen

In deze cursus kijken we in leereenheid 1 eerst naar een aantal SLC-modellen in het algemeen.
Daarna benaderen we het softwareproces vanuit de projectondersteunende processen. Eerst behandelen we in leereenheid 2 aspecten die te maken hebben met het leiden van ict-projecten, dus projectmanagement. Een actueel onderwerp op dit gebied is SCRUM, een moderne 'agile' projectmanagementmethode. Dit komt in leereenheid 3 aan de orde.

Een logisch vervolg hierop is leereenheid 4, waarin we kijken naar hoe deze (projectmanagement)processen verbeterd kunnen worden. Een organisatie kan hierdoor komen tot een hoger kwaliteitsniveau van producten en diensten, en zich zo profileren als een 'professionele organisatie'. In figuur 2 valt dit onder veranderingsmanagement.

Daarna benaderen we het softwareproces vanuit de inhoudelijk sturende processen. In leereenheid 5 komt kwaliteitsmanagement aan de orde.
Het is daarbij belangrijk een goed beeld te hebben van softwarekwaliteit. Certificering op kwaliteit van een bedrijf kan een belangrijke reden zijn voor klanten om met een bedrijf in zee te gaan.

De leereenheden 1 tot en met 5 van de cursus zijn gebaseerd op belangrijke originele publicaties in het vakgebied. Hierin worden vooral de algemene principes beschreven die in de praktijk van industriële softwareprojecten daadwerkelijk tot verbeteringen hebben geleid. De cursus beoogt dat u door zelfstandig zoeken en verwerken van wetenschappelijke literatuur kennis en inzicht in het vakgebied verwerft. Het gaat er daarbij vooral om dat u onderscheid maakt tussen hoofd- en bijzaken, tussen sterke en zwakke uitspraken, tussen bereikt succes en verbloemd falen. Ook het vormen van een persoonlijke visie wordt van groot belang geacht.

Vanuit de optiek van inhoudelijk sturende processen komen we bij het bepalen van de enterprise architectuur bij het interne proces van eisenspecificatie, onderdeel van het gehele ontwikkelingsvraagstuk. Het opstellen en beheer van de eisen valt onder 'requirements engineering'. Dit proces komt uitgebreid aan de orde in deze cursus in leereenheid 6 en verder.

Uit een recent literatuuronderzoek[ii] naar karakteristieken voor het kiezen van het juiste SLCM blijkt dat de invloed van een aantal voor de hand liggende aspecten (zoals requirements) van het softwareproces eigenlijk nauwelijks onderzocht is. De aard van de requirements en de mate waarin requirements door ontwikkelaars worden begrepen kan bepalend zijn bij de keuze voor een SLCM. Hieruit mag blijken dat alle genoemde processen, gezichtspunten en de onderlinge relaties van belang zijn voor het begrijpen van de interne en externe processen tijdens de SLC.

De leereenheden 1 tot en met 5 worden afgesloten met het maken van een kort verslag waarin u een verband weet te leggen tussen de eigen werksituatie en de verworven kennis omtrent softwaremanagement uit deze leereenheden. Zie hiervoor ook de informatie hieronder bij de paragraaf *Tentaminering*.

De leereenheden 6 t/m 11 over Requirements Engineering zijn gebaseerd op het boek 'Requirements Engineering, From System goals to UML models to Software specifications' van Axel van Lamsweerde, John Wiley, 2009.
De leereenheden 6 en 7 geven aan de hand van het tekstboek een overzicht van het vakgebied en van de courante methoden en technieken die gebruikt worden op het terrein van requirements elicitatie Aansluitend volgt een hoofdstuk (leereenheid 8) over de rol die doelen (goals) kunnen spelen in requirements engineering. Deze leereenheid vormt een belangrijke brug naar leereenheid 9; de daar behandelde methode is goal-oriented. Deze hoofdstukken geven vooral een breed overzicht van bestaande methoden en technieken; elk daarvan wordt kort en daardoor noodzakelijkerwijs oppervlakkig behandeld. Het doel van dit deel is om een overzicht te geven van de stand van het vakgebied. Er wordt van u verwacht dat u de behandelde methoden en technieken kunt plaatsen en dat u de zwakke en sterke kanten ervan weet. Er wordt echter niet verwacht dat u deze methoden en technieken volledig zelfstandig kunt toepassen. De nadruk ligt dus op weten, niet op kunnen.

Leereenheid 9 en 10 behandelen enkele essentiële onderdelen van de KAOS-methode voor het opstellen van requirements. Er wordt een leereenheid besteed aan:
– het opstellen van een goal model
– het maken van een risicoanalyse en het uitbreiden van het goal model met behulp van obstacle analyse.

Leereenheid 11 bevat nog een oefencasus die in de online sessies aan bod komt. Dit ter ondersteuning van de eindopdracht waar zelfstandig een casus uitgewerkt moet worden.

Bij de opgaven en opdrachten gebruikt u de tool Objectiver om modellen te maken. Zie ook bij de leermiddelen.

U gaat diep in op één methode en op de bijbehorende technieken. De stof die behandeld wordt in deze leereenheden, dient u aan het eind van de cursus echt te beheersen; de nadruk ligt dus op kunnen.

Ten behoeve van het tentamen werkt u voor het onderdeel Requirements Engineering (leereenheden 6 tot en met 11) zelfstandig aan een casus met de bedoeling om zelfstandig de gewenste modellen uit te werken. Op de uitwerking wordt een keer feedback gegeven. Zie ook de informatie hieronder bij de paragraaf *Tentaminering*.

De verschillende opdrachten kunt u terugvinden op Studienet of opvragen bij de docent.

2 Leerdoelen

Na het bestuderen van de cursus wordt verwacht dat u kennis en inzicht heeft verworven over
– algemene principes die een rol spelen bij softwarelifecycles
– een aantal procesmodellen voor softwareontwikkeling
– een aantal methoden voor procesverbetering
– factoren die van belang zijn voor goed softwaremanagement
– aspecten die een rol spelen voor de kwaliteit van software.

Daarnaast wordt van u verwacht dat u
– de verworven inzichten kunt toepassen op een concrete situatie en daarvan verslag kunt doen
– de rol van requirements engineering in de lifecycle van een systeem kent en inzicht heeft in de belangrijkste obstakels in dit proces
– de verschillende technieken kent voor risicobeheersing en weet wanneer welke techniek geëigend is
– de meest gebruikte technieken kent voor de specificatie van requirements en inzicht heeft in de sterke en zwakke punten van deze technieken
– weet wat binnen de KAOS-methode bedoeld wordt met goals, objects, agents en operations, welke eigenschappen deze hebben en in welke typen ze zijn onder te verdelen

– in staat bent om voor een te ontwikkelen systeem, gebruikmakend van de tool Objectiver, de volgende modellen te construeren:
  – een goal model ter representatie van wat met behulp van het systeem bereikt moet worden
  – een obstacle model ten behoeve van risicoanalyse en als uitbreiding op het goal model
–kunt uitleggen hoe u bij het opstellen van een goal en obstacle model gebruik kunt maken van scenario's.
op grond van een casus zelf een goal model kunt opstellen met mogelijk bijbehorende obstacles. Dit model wordt beoordeeld en is onderdeel van het tentamen.

3 Voorkennis

Er wordt uitgegaan van het kennisniveau van een afgeronde bacheloropleiding Informatica.

4 Cursusmateriaal

Het cursusmateriaal bestaat uit:
– een werkboek: leidraad voor de cursus met leereenheden, een aantal artikelen, opgaven en aanvullende informatie bij de artikelen (onder meer een korte beschrijving in het Nederlands)
– het tekstboek 'Requirements Engineering, From System goals to UML models to Software specifications' van Axel van Lamsweerde, John Wiley, 2009.
– een cursussite op Studienet met actuele informatie over begeleiding en tentamen, aanvullingen en errata, artikelen, weblinks en de ingang tot de discussiegroep.

De cursus maakt gebruik van de KAOS-tool Objectiver. Dit is een professionele tool die u uitsluitend mag gebruiken voor het bestuderen van de cursus. Om de tool na installatie te kunnen gebruiken, hebt u een sleutel nodig die u kunt aanvragen bij de examinator. De geldigheidsduur van die sleutel is veertien maanden; u moet de sleutel daarom pas aanvragen als u daadwerkelijk begint met het bestuderen van de cursus. Details over de installatie van de tool en het verkrijgen van de sleutel vindt u op Studienet.

5 Studeeraanwijzingen

Een onderdeel van dit werkboek vormen de opgaven en oefenopdracht. Deze zijn toegevoegd als hulpmiddel om zowel het leesproces te verlevendigen als toe te spitsen op de essentie.
Het uitwerken van opgaven en oefenopdracht vormt een verplichting voor deelname aan het tentamen. U stuurt de antwoorden aan de begeleider en krijgt van deze feedback. Een en ander wordt ook besproken in de online sessies. Dit kan individueel zijn of groepsgewijs tijdens de begeleidingsbijeenkomsten. Precieze gegevens over inleveren van de opgaven en opdrachten vindt u op Studienet, met een selectie van de opgaven waarvan verwacht wordt dat u deze uitwerkt.

6 Begeleiding en tentaminering

Begeleiding
Bij deze cursus wordt een begeleide cyclus aangeboden met een aantal online begeleidingsbijeenkomsten. Hier worden opgaven besproken en toelichting gegeven op de leerstof, inclusief uitleg bij de eindopdrachten. De begeleidingscyclus heeft een duur van tien weken en een aansluitende tentamenweek.
Een volledig studieplan kunt u vinden op Studienet met per week aangegeven wat verwacht wordt dat u bestudeert en hoeveel tijd dit gaat kosten.
Voor verdere informatie kunt u de cursussite op Studienet raadplegen.

Tentaminering
De vorm van het tentamen is mondeling. Vertrekpunt daarbij is het eerdergenoemde korte verslag en de uitwerking van de opdracht voor Requirements Engineering, die ten minste een week voor het tentamen ingeleverd moeten zijn. De opdrachten zijn te vinden op Studienet of te verkrijgen via de docent.
Daarnaast zullen tijdens de tentamensessie ook andere onderwerpen uit de cursus besproken worden. Het tentamen kan dus over de gehele leerstof gaan. Voorwaarde voor deelname aan het tentamen is dat tijdig aan alle verplichtingen, opdrachten en verslag, is voldaan. Duur van het tentamen is 75 minuten.
Het cijfer voor de cursus bestaat uit het gewogen gemiddelde van de mondelinge bevraging over de gehele stof (gewicht 0,7) en de opdracht Requirements Engineering (gewicht 0,3).
Voor tentamendata kunt u de cursussite op Studienet raadplegen.

---

[i] zie daarvoor bijvoorbeeld T. Bhuvaneswari et al, A Survey on Software Development Life Cycle Models, in *International Journal of Computer Science and Mobile Computing Vol.2* Issue. 5, May- 2013, pg. 262-267, 2013.

[ii] K. Kumar and S. Kumar, A rule-based recommendation system for selection of software development life cycle models; In *Proceedings of ACM SIGSOFT Software Engineering Notes*. 2013, 1-6.

Life Cycle Models

Leereenheid 1

Life Cycle Models

INTRODUCTIE

Er wordt in deze leereenheid aandacht besteed aan de verschillende soorten software-lifecyclemodellen (SLCM). Het gaat daarbij vooral om de mogelijkheid van het versneld produceren van software en het produceren van nieuwe soorten software.

LEERDOELEN
Na het bestuderen van deze leereenheid wordt verwacht dat u
- de inhoud en het belang van het spiraalmodel kunt aangeven
- de uitgangspunten en betekenis van 'agile' methodologieën voor softwareontwikkeling kunt aangeven
- kunt aangeven wat de term 'software-economie' inhoudt en welke rol de aspecten complexiteit, proces, team en tools daarin spelen.

Materiaal
– Boehm, B.W., 'A spiral model of software development and enhancement'
– McMahon, P.E., 'Bridging agile and traditional development methods: a project management perspective'
– Royce, W., 'Improving Software Economics'.

Studeeraanwijzingen
De studieduur van deze leereenheid bedraagt circa 10 uur.

LEERKERN

A spiral model of software development and enhancement (Barry W. Boehm, 1988)

Als alternatief voor het traditionele watervalmodel wordt in dit artikel het spiraalmodel gepresenteerd door de bedenker Barry W. Boehm. Het spiraalmodel geeft sturing aan een iteratief ontwikkelproces op basis van het continu evalueren van risicofactoren.

Bridging agile and traditional development methods: a project management perspective (Paul E. McMahon, 2004)

McMahon schetst in dit artikel problemen die kunnen ontstaan bij samenwerking tussen organisaties met traditionele en 'agile' aanpak. Hij geeft daarvoor vijf aanbevelingen, die voor een groot deel terug te voeren zijn op het optimaliseren van de communicatie tussen diverse stakeholders.

Improving Software Economics (Walker Royce, 2009)

Een tamelijk uitgebreid artikel dat de verschillende aspecten van softwareontwikkeling bespreekt vanuit het perspectief van de software-economie. Er worden vier aspecten genoemd die hierbij een rol spelen:
– reductie van de productomvang en -complexiteit
– verbetering van het ontwikkelproces
– verhoging van de teamexpertise
– toename van procesautomatisering door inzet van geïntegreerde tools.
Dit artikel is in feite een aanbeveling voor het toepassen van (de uitgangspunten van) RUP.

OPGAVEN

OPGAVE 1.1
Hoe zal een spiraal worden gestart respectievelijk gestopt als alleen de eerste drie van de genoemde top-10- risicofactoren in ogenschouw worden genomen?

OPGAVE 1.2
Wat zijn de overeenkomsten en verschillen tussen 'agile' methodologieën en het spiraalmodel?
In hoeverre zijn de aanbevelingen van McMahon enerzijds gebaseerd op (het gebrek aan) communicatie en anderzijds op verschillen tussen de gebruikte methoden?

OPGAVE 1.3
Onder welke omstandigheden zou u de voorkeur geven aan het uitvoeren van een project met een 'agile' methode, en wanneer prefereert u de watervalaanpak?

OPGAVE 1.4
Waarom wijst Royce de meest gangbare wijze van transitie naar nieuwe technieken en technologie af, en welke argumenten heeft hij voor zijn eigen voorkeur?

Projectmanagement

Leereenheid 2

Projectmanagement

INTRODUCTIE

In deze leereenheid komen de vereisten van goed projectmanagement vooral indirect naar voren door het vermelden van factoren die gegarandeerd tot falen leiden.

LEERDOELEN
Na het bestuderen van deze leereenheid wordt verwacht dat u
- kunt onderbouwen dat het veronachtzamen van tijdsoverschrijdingen in een vroeg stadium catastrofale gevolgen heeft die later niet door het inzetten van extra menskracht te herstellen zijn
- kunt aangeven wat voor, tijdens en na een project de maatregelen zijn waarmee de belangrijkste faalfactoren kunnen worden voorkomen.

Materiaal
– Brooks, F.P., 'The mythical man-month after 20 years'
– Peter J. Denning, Chris Gunderson, and Rick Hayes-Roth, 'The Profession of IT, Evolutionary System Development'
– Verner, J.M. en W.M. Evanco, 'In-house software development: What project management practices lead to success?

Studeeraanwijzingen
De studieduur van deze leereenheid bedraagt circa 10 uur.

LEERKERN

The mythical man-month after 20 years (Frederick P. Brooks, 1995)

De wet van Brooks luidt dat het toevoegen van menskracht aan een uitlopend project dat project nog verder zal laten uitlopen. In het beroemde boek (niet dit artikel) 'The mythical man-month' (1975) wordt blootgelegd waarom tijdsoverschrijding in softwareprojecten zo'n hardnekkig en nauwelijks te repareren fenomeen is.
Dit artikel, een samenvatting van hoofdstuk 19 uit de editie van 1995, reflecteert op de editie van 1975. Brooks is nu overtuigd van de voordelen van objectgeoriënteerd programmeren. Verder benadrukt hij de sociologische aspecten van het softwareproces.

The Profession of IT, Evolutionary System Development (Peter J. Denning, Chris Gunderson, and Rick Hayes-Roth, 2008)

In dit artikel worden evolutionaire en agile methoden gepropageerd, speciaal ook voor grote systemen.

In-house software development: What project management practices lead to success? (June M. Verner. en William M. Evanco, 2005)

Dit artikel geeft een mooi beeld van de praktijk van projectmanagement. Aan de hand van een uitgebreide ondervraging bij honderd betrokkenen komen de onderzoekers tot verrassende resultaten en aanbevelingen.

OPGAVEN

OPGAVE 2.1

Bedenk verschillende argumenten waarom – nadat de eerste mijlpalen van een project niet op tijd gehaald zijn – het toevoegen van extra mensen als enige maatregel een uiteindelijk fiasco in de regel niet zal voorkomen.

OPGAVE 2.2

Aan het eind van het artikel van Verner en Evanco (op pagina 92 van het artikel, na 'Analysis of our survey...') werpen de schrijvers vier vragen op. Bedenk antwoorden op deze vragen op basis van uw eigen kennis en ervaring.

OPGAVE 2.3

Ten gevolge van de steeds sneller gaande technische ontwikkelingen hebben (vaak grote) organisaties steeds meer moeite met het managen van langdurige ICT-projecten. Vaak betekent dit een noodzakelijke overgang naar evolutionaire softwareontwikkelmethoden en andere managementmethoden. Wat betekent dit voor de aanbevelingen uit het artikel van Verner en Evanco?

Agile Life Cycles

Leereenheid 3

Agile Life Cycles

INTRODUCTIE

In deze leereenheid wordt aandacht besteed aan agile lifecyclemodellen. 'Agile' wordt vaak gezien als een logische derde stap in software-ontwikkeling. Na de 'staged' modellen zoals het watervalmodel en het iteratieve model is er een ontwikkeling naar 'continuous delivery', waarbij er dus voortdurend een werkend product wordt opgeleverd.

In de praktijk worden er veel agile modellen gebruikt, zowel voor softwareontwikkeling als voor het managen van softwareprojecten. In de loop van de tijd is er ten gevolge van bevindingen in de praktijk een breed scala aan mengvormen ontstaan.
Hier beperken we ons tot de modellen voor het managen van projecten. Dus specifieke agile softwareontwikkelmethoden zoals eXtreme Programming komen niet aan de orde.
Centraal staat in deze leereenheid de Scrum Life cycle. Deze kan goed als referentie gebruikt worden om andere modellen mee te vergelijken.

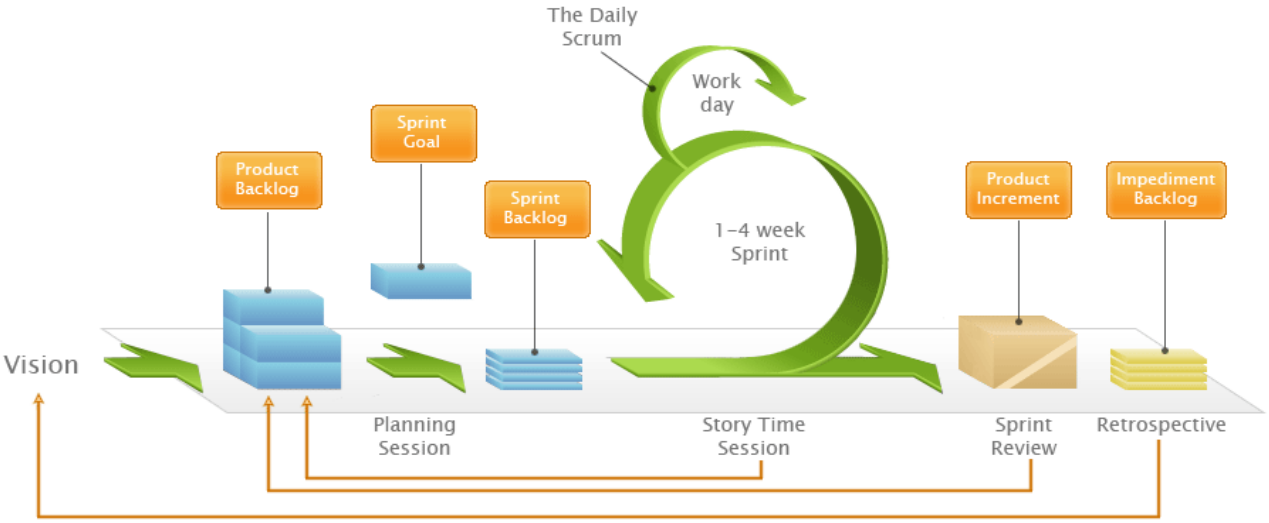
LEERDOELEN

Na het bestuderen van deze leereenheid wordt verwacht dat u
- weet hoe de Scrum methode kan worden gehanteerd in een softwareproject
- weet welke rollen en verantwoordelijkheden er zijn bij Scrum
- kunt argumenteren wat de voor- en nadelen van de Scrum-methode zijn, vergeleken met andere methoden
- kunt aangeven wat in de praktijk de gevolgen kunnen zijn van het afwijken van de Scrum-methode.

Materiaal
– Schwaber, K. & J. Sutherland. De Scrum Guide, (http://www.scrum.org/scrumguides/) (2010)
– Veli-Pekka Eloranta, Kai Koskimies, Tommi Mikkonen en Jyri Vuorinen, Scrum Anti-patterns – An Empirical Study (2013)
– The Scrum Papers, Scrum, Inc. (2012). Dit document kunt u op de cursussite vinden.

Studeeraanwijzingen
De studieduur van deze leereenheid bedraagt circa 10 uur.

LEERKERN



FIGUUR 3.1 De Scrum Life cycle
(bron: http://www.eclectic.eu/diensten/solutions/oplossingen/scrum/)

De Scrum Guide. (Ken Schwaber, Jeff Sutherland, 1991-2013)
(http://www.scrum.org/scrumguides/)

Voor het krijgen van een compleet beeld van Scrum is 'De Scrum Guide' van Schwaber en Sutherland, de bedenkers van Scrum, een goed uitgangspunt. Deze is onlangs ook in het Nederlands verschenen.

Leestip

Een uitgebreidere uitleg over Scrum geven Deemer, Benefield en Larman in Chapter 1: Introduction to Scrum van The Scrum Papers: Nuts, Bolts, and Origins of an Agile Framework (2012). Dit document (ruim tweehonderd pagina's) kunt u vinden op de cursussite op Studienet.

Deze beide bovengenoemde documenten laten zien dat Scrum op papier een heel simpele methode voor het managen van projecten is. In de praktijk zitten er nogal wat haken en ogen aan die zeker onderkend moeten worden alvorens dit in een bestaande organisatie te introduceren. Vaak gaan organisaties over op Scrum of een andere methode zonder duidelijk voor ogen te hebben wat daarvan het effect zal zijn, en welke problemen dit kan oplossen.

Scrum Development Proces (Ken Schwaber, 1995)

Schwaber beschrijft hier hoe, anders dan bij andere methoden waarbij alleen gereageerd wordt op onzekerheid en onvoorspelbaarheid, bij Scrum de onzekerheden juist gemanaged worden.
Schwaber geeft in verband daarmee argumenten om het software-ontwikkelproces een empirisch proces te noemen. Dat heeft gevolgen voor de wijze waarop het gemanaged moet worden.

Scrum Anti-patterns – An Empirical Study (Veli-Pekka Eloranta, Kai Koskimies, Tommi Mikkonen en Jyri Vuorinen, 2013)

Nu deze methode al ongeveer tien jaar intensief wordt toegepast beginnen ook de eerste wetenschappelijke artikelen hierover te verschijnen. Scrum wordt vaak gezien als een raamwerk of pattern waarbinnen verschillende methoden, practices gebruikt kunnen worden, zolang maar aan de randvoorwaarden uit de Scrum Guide is voldaan. Vanzelfsprekend kan het afwijken hiervan leiden tot onverwachte resultaten, zowel positief als negatief. Een, zij het enigszins beperkt onderzoek van Eloranta e.a. laat dit zien in de vorm van 'anti patterns'.

OPGAVEN

OPGAVE 3.1
In 'De Scumgids' wordt gesteld dat Scrum wordt gebruikt om product-ontwikkeling te managen, maar dat het geen proces of techniek is voor het bouwen van producten. Wat betekent dit voor softwareontwikkeling?

OPGAVE 3.2
In 'Scrum Development Proces' (pagina 62) stelt Schwaber 'The closer the development team operates to the edge of chaos, while still maintaining order, the more competitive and useful the resulting system will be.' Welke argumenten kunt u voor deze stelling aanvoeren?

OPGAVE 3.3
De Scrum-methode is door Schwaber gebaseerd op de constatering dat het softwareontwikkelproces 'empirisch' is. Hoe komt dat tot uiting in de methode?

OPGAVE 3.4
Eloranta e.a. stellen vast dat 'The […] observation from the result is that companies having more Scrum experience tend to use anti-patterns more.' (pagina 508). Zij geven een mogelijke verklaring maar wijzen op verder onderzoek. Wat zouden volgens u mogelijke verklaringen kunnen zijn?

Interessante links

http://www.scrum-institute.org

http://www.pmdocuments.com/2012/09/15/agile-scrum-roles-and-responsibilities/

Hoofdstuk 5.9. Scrum uit Fritjof Brave, Erwin Baardman, Ariane Moussault, Wegwijzer voor methoden bij Projectmanagement, 2e geheel herziene druk.

Process Improvement

Leereenheid 4

Process Improvement

INTRODUCTIE

Er wordt in deze leereenheid vooral aandacht besteed aan ervaringen om bij grootschalige softwareprojecten tot procesverbetering te komen op basis van het Capability Maturity Model Integration (afgekort CMMI, maar vaak ook nog als CMM, de voorloper van CMMI). Daarnaast wordt ook aandacht besteed aan het fundamentele verschil tussen een procescultuur en een projectcultuur om meer inzicht te krijgen in de conflicten die met procesverbetering gepaard gaan.

LEERDOELEN
Na het bestuderen van deze leereenheid wordt verwacht dat u
- het bedrijfsmatige belang van procesverbetering kunt aangeven
- de inhoud en betekenis van CMMI kunt aangeven
- inzicht hebt in de huidige en toekomstige trends op het gebied van procesverbetering
- relativerende kanttekeningen bij de CMMI-benadering kunt plaatsen
- inzicht hebt in hoe een CMMI-niveau wordt gemeten
- het wezenlijke conflict tussen een proces- en een projectcultuur kunt aangeven.

Materiaal
Om de artikelen in deze leereenheid goed te kunnen begrijpen is kennis over CMM(I) nodig. CMMI is een product van het Carnegie Mellon® Software Engineering Institute (SEI). We raden u aan eerst te kijken op de site van het CMMI Institute. Het artikel 'Process improvement for small organizations' van Kelly, D.P. en B. Culleton is ook een goede introductie op CMM.

Links
– CMMI Institute: http://whatis.cmmiinstitute.com/
– CMMI Version 1.2 Overview presentation: cmmi-overview07.pdf. Deze is op de cursussite te vinden.

Artikelen
– Kelly, D.P. en B. Culleton, 'Process improvement for small organizations'
– Börjesson, A. en L. Mathiassen, 'Successful process implementation'
– Solingen, R. van, 'Measuring the ROI of Software Process Improvement'
– Thomsett, R.,'A tale of two cultures'

Studeeraanwijzingen
De studieduur van deze leereenheid bedraagt circa 10 uur.

LEERKERN

Process improvement for small organizations
(Declan P. Kelly, Bill Culleton, 1999)

In een casus wordt beschreven hoe een klein bedrijf niveau 4 van CMM bereikt. 'Klein' was in 1999 toch altijd nog een bedrijf met 150 software engineers! Er wordt gewezen op de fundamentele verschillen in cultuur tussen grote en kleine organisaties, en het invoeren van een procesverbeteringstraject zou daarbij moeten aansluiten: maak gebruik van de sterke kanten van die cultuur. Bij kleine organisaties gaat het met name om het gebruiken en waarborgen van de persoonlijke creativiteit. Een van de succesfactoren blijkt te zijn dat men zich een hoge product-kwaliteit in plaats van een bepaald CMM-niveau ten doel stelt.

Successful process implementation
(Anne Börjessen, Lars Mathiassen, 2004)

Dit artikel bespreekt de ervaring met procesverbetering bij Ericsson. Het maakt onderscheid tussen de 'process engineers' (verantwoordelijk voor de invoering van procesverbeteringen) en de 'software practitioners' (die deze in de praktijk moeten brengen). Verschillen van inzicht en ook communicatieproblemen tussen deze twee groepen spelen een cruciale rol bij het succes van de verbeteringen. Verder geven de schrijvers wat (globale) aanbevelingen.

Measuring the ROI of Software Process Improvement
(Rini van Solingen, 2004)

Dit artikel laat zien dat het, in tegenstelling tot wat vaak wordt gedacht, heel goed mogelijk is een bruikbare kosten-batenanalyse van procesverbetering te maken. Van Solingen legt de nadruk op het betrekken van meerdere stakeholders bij het bepalen van de baten, en op bijvoorbeeld het belang van het stellen van de 'what-if-not'-vraag. Enkele concrete voorbeelden illustreren de resultaten van een pragmatische kosten-batenanalyse. Ook al zijn de uitkomsten slechts bij benadering te geven, de waarde van de analyse is dat het voor managers en beleidsmakers in een organisatie een zeer bruikbaar sturingsinstrument is.

A tale of two cultures (Rob Thomsett, 2011)

In dit artikel komt scherp naar voren waarom een proces- en project-cultuur wezenlijk met elkaar in conflict zijn. Omdat de voortdurende en snelle technologische en maatschappelijke veranderingen een vorm van dynamiek en flexibiliteit vergen die strijdig is met een procescultuur, is de projectcultuur al vele decennia lang in opmars.
Het artikel is een 'update' van zijn artikel 'The clash of two cultures: project management versus process management' uit 1994.

OPGAVEN

OPGAVE 4.1
Geef in het kort de karakteristieken van de vijf CMMI-niveaus. Waarom bevat niveau 1 geen KPA's (Key Process Areas)?

OPGAVE 4.2
Noem de belangrijkste succesfactoren voor het bereiken van CMMI-niveau 4 door het bedrijf S3.

OPGAVE 4.3
Beschrijf de relatie tussen de SPI initiatieven 9, 10 en 12 op het gebied van requirements management in Table 1 op pagina 38 van het artikel van Börjesson en Mathiassen, en verklaar waarom de laatste (12) volgens u als 'Highway initiative' is gepositioneerd (Figure 1, pagina 40).

OPGAVE 4.4
Volgens Van Solingen is een van de methoden om 'onzichtbare' baten van SPI te kwantificeren het stellen van de vraag aan managers welke uitgaven gerechtvaardigd zouden zijn om te komen tot het bereikte of gewenste resultaat. Geef uw mening over de betrouwbaarheid van deze methode. Schets daarbij twee duidelijk van elkaar verschillende scenario's.

Softwarekwaliteit

Leereenheid 5

Softwarekwaliteit

INTRODUCTIE

In deze leereenheid wordt aandacht besteed aan het begrip softwarekwaliteit. Hoewel in de vorige leereenheden gesproken is over procesverbetering moet eigenlijk eerst duidelijk zijn wat er verstaan kan worden onder softwarekwaliteit. Er kan ten eerste op zeer verschillende wijzen tegen kwaliteit worden aangekeken. Vervolgens blijkt in de praktijk dat ook menselijke factoren zeker niet onderschat mogen worden, waardoor het 'meten' van softwarekwaliteit veel complexer is dan het tellen van het aantal 'bugs' in de software.

LEERDOELEN
Na het bestuderen van deze leereenheid wordt verwacht dat u
- een bruikbare definitie kunt geven van softwarekwaliteit
- een aantal kenmerken kunt noemen van softwarekwaliteit
- kunt aangeven welke rol certificering speelt bij softwarekwaliteit
- kunt aangeven hoe softwarekwaliteit afhangt van menselijke factoren en welke problemen er in de praktijk van het meten van kwaliteit kunnen ontstaan.

Materiaal
Uit het boek van Heemstra, Kusters en Trienekens, 'Softwarekwaliteit' (1e druk):
– Hoofdstuk 6 Softwarekwaliteit, wat is dat?
– Hoofdstuk 18 Het realiseren, handhaven en garanderen van een goed softwareproces (paragrafen 18.4 en 18.5)
– Hoofdstuk 22 Invloed van de factor mens op softwarekwaliteit
en het artikel
– Hoffman, D., 'The Darker Side of Metrics'.

Studeeraanwijzingen
De studieduur van deze leereenheid bedraagt circa 10 uur.

LEERKERN

Hoofdstuk 6 Softwarekwaliteit, wat is dat?

Dit hoofdstuk is een mooie inleiding over het begrip softwarekwaliteit. Het behandelt de kenmerken van softwarekwaliteit vanuit verschillende (historische) gezichtspunten, met als laatste de ISO/IEC 9126 standaard.

Hoofdstuk 18, paragrafen 18.4 en 18.5 Het realiseren, handhaven en garanderen van een goed softwareproces

Deze paragrafen gaan over de rol van procescertificering bij softwareontwikkeling. Er worden twee vormen beschreven. De eerste komt voort uit het toepassen van de ISO-standaard. De andere is de procescertificering op basis van CMM. (CMM zelf komt in volgende leereenheden uitgebreider aan de orde).

Hoofdstuk 22 Invloed van de factor mens op softwarekwaliteit

Dit hoofdstuk beschrijft de relatie tussen zaken als persoonlijke motivatie, betrokkenheid, gedrag en softwarekwaliteit. Daar horen natuurlijk ook opleiding en professionalisering van de ontwikkelaars bij.

Opmerking
De conclusies over het onderzoek van Tracy Hall zijn enigszins tendentieus geformuleerd: bekijk hiervoor de samenvatting van dit onderzoek op de cursussite ('No quality without equality', Tracy Hall, 1995, s2101.pdf of zie de link op de cursussite). In het boek *Softwarekwaliteit* staat: 'vrouwen zijn minder tevreden over de kwaliteit van eigen werk', en in de samenvatting staat: 'women developers have higher standards than men'; een duidelijk andere benadering.

'The darker side of metrics' (Douglas Hoffman, 2000)

Dit artikel gaat over de invloed van het gedrag van ontwikkelaars, testers en het management van een organisatie op de resultaten van metrieken ten behoeve van het managen van softwareprojecten. Metrieken komen in volgende leereenheden aan de orde, maar dit artikel geeft in aansluiting op het vorige hoofdstuk uit *Softwarekwaliteit* goed aan hoe theorie en praktijk flink uiteen kunnen lopen ten gevolge van het gedrag van mensen.

OPGAVEN

OPGAVE 5.1
Beschrijf in het kort welke cruciale accentverschuivingen in het verleden hebben plaatsgevonden in de beschrijvingen van kwaliteitskenmerken van software.

OPGAVE 5.2
Certificering is geen garantie voor productkwaliteit. Beschrijf wat er volgens u aan de hand kan zijn met een gecertificeerd bedrijf dat een slechte prijs-prestatieverhouding levert.

OPGAVE 5.3
Wat is de meerwaarde van het bestaan van de SWEBOK voor het testen in een 'agile' ontwikkelteam? Bekijk het hoofdstuk 'Software testing' op http://www.swebok.org/ch5.html.

OPGAVE 5.4

Hoffman concludeert dat door observatie van gedrag en het her-overwegen van de meetmethoden de negatieve effecten in de toekomst vermeden zouden kunnen worden. Welke gevolgen zal dat volgens u voor de interne en externe softwarekwaliteit van de organisatie kunnen hebben?

Requirements Engineering: Setting the Scene

Leereenheid 6

Requirements Engineering: Setting the Scene

INTRODUCTIE

Bij deze leereenheid hoort hoofdstuk 1 van het tekstboek, dat het terrein afbakent.

Aan het eind van deze leereenheid geven we bovendien een korte inleiding in Systems Engineering, een onderwerp dat in dit boek op de achtergrond steeds aanwezig is.

LEERDOELEN

Na het bestuderen van deze leereenheid wordt verwacht dat u

- de betekenis kent van de termen WHY-, WHAT- en WHO-dimension
- kunt aangeven uit welke componenten een te ontwikkelen systeem (het system-to-be) is opgebouwd
- het verschil begrijpt tussen beschrijvende (descriptive) en voorschrijvende (prescriptive) uitspraken
- de betekenis begrijpt van de termen system requirement, software requirement, domain property, assumption en definition en het verband tussen deze typen uitspraken begrijpt
- weet wat monitored variables, controlled variables, input variables en output variables zijn en wat hun verband is met system en software requirements
- het verschil kunt aangeven tussen functionele en niet-functionele requirements en vier typen niet-functionele requirements kunt noemen
- enkele voorbeelden kunt geven van quality requirements
- het (spiraalvormige) verloop van het requirements proces kunt schetsen
- voorbeelden kunt noemen van eisen die aan een goed requirements document gesteld kunnen worden en gebreken die vermeden moeten worden
- kunt uitleggen waarom requirements engineering noodzakelijk is en welke obstakels er zijn voor een goed requirements engineering proces
- kunt aangeven waarin het systems engineering proces in bredere zin verschilt van softwareontwikkeling
- de verschillende stadia kunt noemen in het systems engineering process en kunt uitleggen wat hun functie is
- de betekenis kunt geven van de volgende termen: environmental phenomena, shared phenomena, software phenomena, four variable model, quality requirements, compliance requirements, architectural requirements, development requirements, systems engineering

Studeeraanwijzingen
Bestudeer hoofdstuk 1 van het tekstboek.

De studielast van deze leereenheid bedraagt circa 15 uur.

LEERKERN

1 Aanvullingen en opmerkingen

Figure 6.6 op pagina 34 van het tekstboek toont het requirements-proces als een herhaald doorlopen spiraal. Realiseer u dat deze spiraal de structuur van het eerste deel van het boek bepaalt:
– hoofdstuk 2 gaat over Domain Understanding and Elicitation
– hoofdstuk 3 gaat over Requirements Evaluation
– hoofdstuk 4 gaat over Specification and Documentation
– hoofdstuk 5 gaat over Quality Assurance
– hoofdstuk 6 gaat over Requirements Evolution, ofwel de verdere doorgangen door de spiraal

Hoofdstuk 7 is een overgang tussen het eerste en het tweede deel en valt hier buiten.

We behandelen overigens, zoals uiteengezet in de introductie-leereenheid, niet al deze hoofdstukken.

2 Opgaven

OPGAVE 6.1
Uit welke componenten is het *system-to-be* opgebouwd?

OPGAVE 6.2
Teken een wereld-machinediagram (als in figure 1.3 op pagina 18 van het tekstboek) voor een geldautomaat. Toon verschijnselen (phenomena) die te maken hebben met het uitnemen van de pas en het uittellen van het geld.

OPGAVE 6.3
De volgende uitspraken zijn afkomstig uit een systeem voor de toewijzing van ambulances (welke ambulance gaat naar een net gebeurd ongeluk). Geef voor elke uitspraak aan of het een system requirement, een software requirement, een domeineigenschap, een aanname of een definitie is.
a Binnen veertien minuten na binnenkomst van de melding van een incident bij de alarmcentrale moet er een ambulance ter plaatse zijn.
b Een ambulance kan slechts één gewonde tegelijk vervoeren.
c Een incident is een gebeurtenis waarbij minimaal één menselijke gewonde is gevallen.
d Na de invoer van een locatie van een incident zal het systeem de nummers en de gemeten locaties tonen van de vijf ambulances die volgens het systeem het dichtst bij de plaats van het incident zijn en beschikbaar zijn.
e Een incident wordt altijd binnen tien minuten gemeld bij de alarmcentrale.
f Voor elke ambulance en op elk moment wijkt de gemeten locatie niet meer dan 100 meter af van de werkelijke locatie.

OPGAVE 6.4
Wat is de categorie van de requirements uit de vorige opgave?

OPGAVE 6.5

Tekstboek pagina 59

Maak de eerste opgave op pagina 59 van het tekstboek ('Consider the case study description of the train control system….'). NB De verantwoordelijkheid voor het gesloten houden van de deuren bij de passagiers leggen, betekent dat er (zoals dat lang geleden het geval was) geen deurbeveiliging is.

OPGAVE 6.6

Tweede opgave op pagina 59 van het tekstboek (aangepast)

Noem voor het systeem voor het plannen van vergaderingen beschreven op pagina 9-12 van het tekstboek:

a twee strategische doelen (strategic objectives) van de *kopers* van dit systeem (het is ontwikkeld door een softwarehuis, zie pagina 9 van het tekstboek).

b twee functionele diensten (functional services)

c twee aannamen over de omgeving (environment assumptions).

OPGAVE 6.7

Zesde opgave op pagina 59 van het tekstboek (aangepast)

Bekijk een eenvoudig systeem met verkeerslichten om voetgangers een drukke weg te laten oversteken. Gegeven zijn een systeemeis en drie software-eisen als volgt:

(*SysReq:)* Het verkeerslicht zal voetgangers in staat stellen om veilig de weg over te steken door auto's te laten stoppen

(*Sofreq1:)* De schakelaar van het voetgangerslicht wordt op groen gezet binnen 30 seconden nadat er op de voetgangersknop is gedrukt

(*SofReq2:*) De schakelaar van het verkeerslicht voor auto's wordt op rood gezet ten minste 5 seconden voordat het voetgangerslicht groen wordt.

(*Sofreq3:*) Bij het starten van het systeem wordt de schakelaar van het voetgangerslicht op rood en die van het verkeerslicht op groen gezet.

a Welke domeineigenschappen en welke aannamen over de omgeving zijn nodig om het volgende te kunnen bewijzen:
(*SofReq1, SofReq2, SofReq3,* aannamen?, domeineigenschappen?) |= *SysReq*

b Toon dit ook aan.

c Zijn de aannamen over de omgeving realistisch?

OPGAVE 6.8

Noem minimaal vijf gewenste en vijf te vermijden eigenschappen van een requirements-document.

3 Systems engineering

In het tekstboek wordt, zoals ook al blijkt uit dit eerste hoofdstuk, het opstellen van software requirements in een veel breder kader geplaatst: het gaat bij de voorbeelden niet uitsluitend om 'zuivere' software-systemen. In de treincasus, beschreven op pagina 8-9 van het tekstboek, gaat het om een systeem waarin besturingsssoftware samenwerkt en afhankelijk is van sensoren en actuatoren in de trein en op de perrons. Agents in alle te specificeren systemen zijn niet alleen stukken software, maar mogelijk ook mensen en/of apparaten (devices) (zie pagina 16 van het tekstboek). Hetgeen waarvoor in deze cursus requirements worden opgesteld, is dus een *systeem* en niet uitsluitend een stuk software.

Er zijn enkele belangrijke verschillen tussen software ontwikkeling en systeemontwikkeling in bredere zin. Ten eerste is het nog moeilijker en duurder dan bij een zuiver softwaresysteem om de structuur van het systeem achteraf nog ingrijpend te wijzigen: hardwarecomponenten zijn nu eenmaal niet flexibel. Ten tweede zijn er bij het ontwikkelen van een hybride systeem allerlei verschillende ingenieursdisciplines betrokken, wat de kans op misverstanden en communicatiestoornissen vergroot.

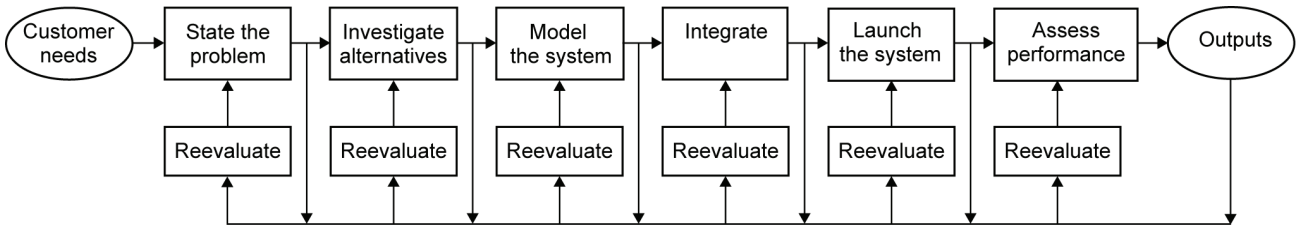
Om u bewust te maken van het feit dat de blik van een software engineer breder moet zijn dan alleen software, geven we hier een korte inleiding in het onderwerp systems engineering. De tekst is overgenomen van de website van INCOSE, The International Council on Systems Engineering (http://www.incose.org/practice/fellowsconsensus.aspx).

Definition of a system

A system is a construct or collection of different elements that together produce results not obtainable by the elements alone. The elements, or parts, can include people, hardware, software, facilities, policies, and documents; that is, all things required to produce systems-level results. The results include system level qualities, properties, characteristics, functions, behavior and performance. The value added by the system as a whole, beyond that contributed independently by the parts, is primarily created by the relationship among the parts; that is, how they are interconnected (Rechtin, 2000).

Systems Engineering

Systems Engineering is an engineering discipline whose responsibility is creating and executing an interdisciplinary process to ensure that the customer and stakeholder's needs are satisfied in a high quality, trustworthy, cost efficient and schedule compliant manner throughout a system's entire life cycle. This process is usually comprised of the following seven tasks: **S**tate the problem, **I**nvestigate alternatives, **M**odel the system, **I**ntegrate, **L**aunch the system, **A**ssess performance, and **R**e-evaluate. These functions can be summarized with the acronym SIMILAR: **S**tate, **I**nvestigate, **M**odel, **I**ntegrate, **L**aunch, **A**ssess and **R**e-evaluate. This Systems Engineering Process is shown in Figure 6.1. It is important to note that the Systems Engineering Process is not sequential. The functions are performed in a parallel and iterative manner.



FIGUUR 6.1 The Systems Engineering Process from A. T. Bahill and B. Gissing, Re-evaluating systems engineering concepts using systems thinking, *IEEE Transaction on Systems, Man and Cybernetics, Part C: Applications and Reviews*, 28 (4), 516-527, 1998.

State the problem

The problem statement starts with a description of the top-level functions that the system must perform: this might be in the form of a mission statement, a concept of operations or a description of the deficiency that must be ameliorated. Most mandatory and preference requirements should be traceable to this problem statement. Acceptable systems must satisfy all the mandatory requirements. The preference requirements are traded-off to find the preferred alternatives. The problem statement should be in terms of *what* must be done, not *how* to do it. The problem statement should express the customer requirements in functional or behavioral terms. It might be composed in words or as a model. Inputs come from end users, operators, maintainers, suppliers, acquirers, owners, regulatory agencies, victims, sponsors, manufacturers and other stakeholders.

Investigate Alternatives

Alternative designs are created and are evaluated based on performance, schedule, cost and risk figures of merit. No design is likely to be best on all figures of merit, so multicriteria decision-aiding techniques should be used to reveal the preferred alternatives. This analysis should be redone whenever more data are available. For example, figures of merit should be computed initially based on estimates by the design engineers. Then, concurrently, models should be constructed and evaluated; simulation data should be derived; and prototypes should be built and measured. Finally, tests should be run on the real system. Alternatives should be judged for compliance of capability against requirements. For the design of complex systems, alternative designs reduce project risk. Investigating innovative alternatives helps clarify the problem statement.

Model the system

Models will be developed for most alternative designs. The model for the preferred alternative will be expanded and used to help manage the system throughout its entire life cycle. Many types of system models are used, such as physical analogs, analytic equations, state machines, block diagrams, functional flow diagrams, object-oriented models, computer simulations and mental models. Systems Engineering is responsible for creating a product and also a process for producing it. So, models should be constructed for both the product and the process. *Process* models allow us, for example, to study scheduling changes, create dynamic PERT charts and perform sensitivity analyses to show the effects of delaying or accelerating certain subprojects. Running the process models reveals bottlenecks and fragmented activities, reduces cost and exposes duplication of effort. *Product* models help explain the system. These models are also used in tradeoff studies and risk management.
As previously stated, the Systems Engineering Process is not sequential: it is parallel and iterative. This is another example: models must be created before alternatives can be investigated.

Integrate

No man is an island. Systems, businesses and people must be integrated so that they interact with one another. Integration means bringing things together so they work as a whole. Interfaces between subsystems must be designed. Subsystems should be defined along natural boundaries. Subsystems should be defined to minimize the amount of information to be exchanged between the subsystems. Well-designed subsystems send finished products to other subsystems. Feedback loops around individual subsystems are easier to manage than feedback loops around interconnected subsystems. Processes of co-evolving systems also need to be integrated. The consequence of integration is a system that is built and operated using efficient processes.

Launch the system

Launching the system means running the system and producing outputs. In a manufacturing environment this might mean buying commercial off the shelf hardware or software, or it might mean actually making things. Launching the system means allowing the system do what it was intended to do. This also includes the system engineering of deploying multi-site, multi-cultural systems.
This is the phase where the preferred alternative is designed in detail; the parts are built or bought (COTS), the parts are integrated and tested at various levels leading to the certified product. In parallel, the processes necessary for this are developed – where necessary - and applied so that the product can be produced. In designing and producing the product, due consideration is given to its interfaces with operators (humans, who will need to be trained) and other systems with which the product will interface. In some instances, this will cause interfaced systems to co-evolve. The process of designing and producing the system is iterative as new knowledge developed along the way can cause a re-consideration and modification of earlier steps.
The systems engineers' products are a mission statement, a requirements document including verification and validation, a description of functions and objects, figures of merit, a test plan, a drawing of system boundaries, an interface control document, a listing of deliverables, models, a sensitivity analysis, a tradeoff study, a risk analysis, a life cycle analysis and a description of the physical architecture. The requirements should be validated (Are we building the right system?) and verified (Are we building the system right?). The system functions should be mapped to the physical components. The mapping of functions to physical components can be one to one or many to one. But if one function is assigned to two or more physical components, then a mistake might have been made and it should be investigated. One valid reason for assigning a function to more than one component would be that the function is performed by one component in a certain mode and by another component in another mode. Another would be deliberate redundancy to enhance reliability, allowing one portion of the system to take on a function if another portion fails to do so.

Assess performance

Figures of merit, technical performance measures and metrics are all used to assess performance. Figures of merit are used to quantify requirements in the tradeoff studies. They usually focus on the product. Technical performance measures are used to mitigate risk during design and manufacturing. Metrics (including customer satisfaction comments, productivity, number of problem reports, or whatever you feel is critical to your business) are used to help manage a company's processes. Measurement is the key. If you cannot measure it, you cannot control it. If you cannot control it, you cannot improve it. Important resources such as weight, volume, price, communications bandwidth and power consumption should be managed. Each subsystem is allocated a portion of the total budget and the project manger is allocated a reserve. These resource budgets are managed throughout the system life cycle.

Re-evaluate

Re-evaluate is arguably the most important of these functions. For a century, engineers have used feedback to help control systems and improve performance. It is one of the most fundamental engineering tools. Re-evaluation should be a continual process with many parallel loops. Re-evaluate means observing outputs and using this information to modify the system, the inputs, the product or the process. Figure 1 summarizes the Systems Engineering Process. This figure clearly shows the distributed nature of the Re-evaluate function in the feedback loops. However, all of these loops will not always be used. The particular loops that are used depend on the particular problem being solved.

Variations

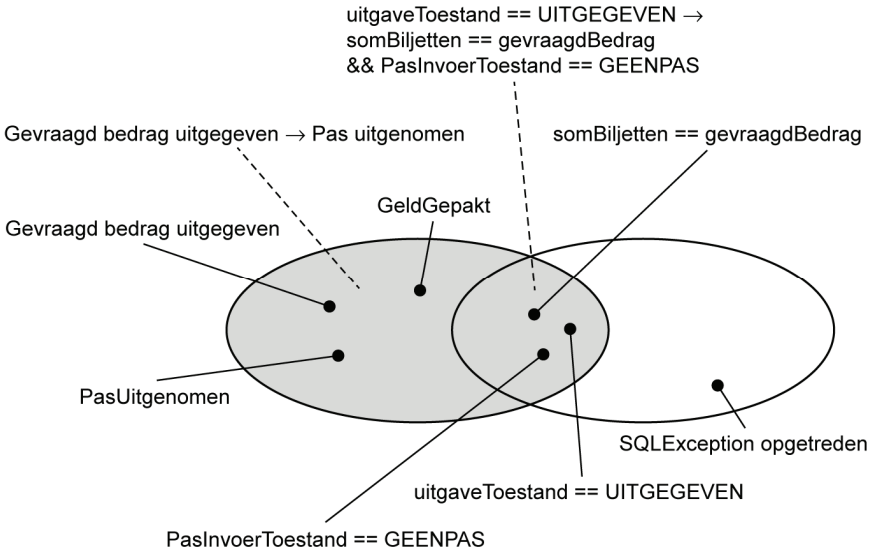
Like all processes, the Systems Engineering process at any company should be documented, measurable, stable, of low variability, used the same way by all, adaptive, and tailorable! This may seem like a contradiction. And perhaps it is. But one size does not fit all. The above description of the Systems Engineering process is just one of many that have been proposed. Some are bigger, some are smaller. But most are similar to this one.

TERUGKOPPELING

Uitwerking van de opgaven

6.1 Het *system-to-be* is opgebouwd uit de *software-to-be* (het te bouwen softwaresysteem) en componenten die samen de omgeving van die software vormen. Tot die omgeving behoren mensen, fysieke apparaten als sensoren en actuatoren en tot slot bestaande software waarmee het systeem moet communiceren.

6.2 Figuur 6.2 toont een mogelijke uitwerking.



FIGUUR 6.2 Wereld-machinediagram voor gelduitgifte door geldautomaat

Het belangrijke punt is, dat de software geen directe manier heeft om vast te stellen of de pas nog de automaat zit en of het gevraagde bedrag naar de gelduitvoer is getransporteerd. De software moet daarvoor afgaan op informatie afkomstig van sensoren. De eis "Gevraagd bedrag uitgegeven → Pas uitgenomen" is daarom een systeemeis, terwijl de eis "uitgaveToestand == UITGEGEVEN → somBiljetten == gevraagdBedrag & PasInvoerToestand == GEENPAS" software requirements zijn.

6.3 a Dit is een system requirement, een eis aan het systeem als geheel en niet alleen aan de toewijzingssoftware.
b Dit is een domeineigenschap (de waarheid ervan hangt niet van het systeem af maar alleen van de omgeving).
c Dit is een definitie (er is geen waarheidswaarde).
d Dit is een software requirement; het geeft een relatie tussen invoer (melding van de locatie van een incident) en uitvoer (de locaties van ambulances). Merk op dat het hier gaat om locaties en beschikbaarheid zoals bepaald door het systeem; dat dit klopt met de werkelijkheid moet apart gespecificeerd worden.
e Dit is een aanname, een eis waarvan we verwachten dat de omgeving er aan voldoet.

f Vanuit het systeem voor de toewijzing van ambulances gezien, is dit een aanname en wel een accuracy statement. Deze aanname is overigens verre van triviaal. In de totaal mislukte invoering van het London Ambulance System in 1992 werden de locaties van de ambulances bijgehouden door een autovolgsysteem, dat volstrekt onvoldoende functioneerde. De gemeten locatie week daardoor vaak kilometers af van de werkelijke locatie.

6.4 De system requirement uit onderdeel a is een performance requirement. De software requirement uit onderdeel d is een functionele requirement.

6.5 Zie onderstaande tabel.

	voordelen	*risico's*
bestuurder	Duidelijk wie verantwoordelijk is	Bestuurder ziet deuren niet en kan dus niet zeker weten of ze dicht zijn Bestuurder heeft ook andere taken, wat het risico van menselijk falen verhoogt
conducteur	Duidelijk wie verantwoordelijk is. Conducteur kan alle deuren langsgaan en ze van buiten sluiten.	Risico van menselijk falen. Het sluiten van de deuren neemt veel tijd in beslag.
passagiers	Eenvoudig en goedkoop.	Passagiers met haast openen de deur terwijl de trein nog (of al) rijdt, wat tot ernstige ongelukken kan leiden, met de bijbehorende hoge claims voor de vervoersmaatschappij.
software	Indien correct, zeer betrouwbaar	Softwarefout Fout in detectie of besturing

6.6 a Strategische doelen voor de kopers van het vergaderplannings-systeem zijn bijvoorbeeld:
– De deelname aan vergaderingen vergroten.
– Organisatorische overhead bij het plannen van vergaderingen terugdringen.
De andere doelen genoemd op pagina 10 en 11 zijn daarvan afgeleid. Het belang van een snelle scheduling dient bijvoorbeeld vooral om het risico terug te dringen dat mensen inmiddels al weer bezet zijn, en het terugbrengen van het aantal benodigde berichten is een zaak van efficiency.
b Enkele voorbeelden van services:
– Vergaderverzoek sturen aan alle potentiële deelnemers.
– Feschikte vergaderlocatie bepalen op grond van de deelnemers en hun standplaatsen.
– Gegeven een verzameling constraints, een tijdvak waarbinnen de vergadering moet vallen en de duur van de vergadering, een mogelijk tijdstip bepalen voor de vergadering.
– Deelnemers de tijd en locatie van de vergadering meedelen.
c Enkele voorbeelden van environment assumptions:
– Deelnemers reageren binnen n dagen op een vergaderverzoek.
– Een tijd die is opgegeven als beschikbaar, blijft minimaal m dagen (of uren) beschikbaar.
– Deelnemers kunnen op een dag niet meer dan x km reizen.

6.7 a Een mogelijke uitwerking is de volgende.

Domeineigenschappen:
– D1: De schakelaar van een verkeerslicht heeft twee standen: rood en groen.
– D2: Een oversteek van een weg is veilig als er geen auto's op de weg rijden.
– D3: Een auto die stilstaat, rijdt niet op de weg.
Aannamen over de omgeving:
– O1 (O2): Een verkeerslicht is rood (groen) dan en slechts dan als schakelaar van het licht op rood (groen) staat.
– O3: Voetgangers die willen oversteken op een moment dat het voetgangerslicht rood is, drukken op de voetgangersknop.
– O3: Voetgangers steken over als het voetgangerslicht groen is en steken niet over als het rood is.
– O4: Automobilisten rijden door als het verkeerslicht groen is.
– O5: Automobilisten stoppen als ze de oversteekplaats naderen terwijl het verkeerslicht al vijf seconden of langer rood is.

b Bewijs:
– Volgens Sofreq3 staat de schakelaar van het voetgangerslicht in de beginsituatie op rood. Volgens O1 is dan ook het voetgangerslicht rood en volgens O3 steken voetgangers dan niet over. Deze situatie is dus veilig.
– Volgens Sofreq1 wordt de schakelaar van het voetgangerslicht op groen gezet hoogstens 30 seconden nadat er op de knop is gedrukt; het licht wordt dan groen (O2) en de voetgangers steken over (O3).
– Volgens Sofreq2 is de schakelaar van het verkeerslicht al 5 seconden eerder op rood gezet en dus is volgens O1 ook het verkeerslicht al 5 seconden eerder op rood gegaan.
– Volgens O5 stoppen alle auto's dan bij het verkeerslicht.
– Volgens D3 rijden ze dan niet op de weg.
– Volgens D2 is de oversteek dan dus veilig.

Merk op, dat volgens deze specificatie het verkeerslicht mogelijk altijd rood blijft; daar spreken de software requirements zich niet over uit.
In het boek ontbreekt de derde requirement die de beginsituatie beschrijft. Dat zou echter kunnen betekenen dat beide lichten op groen staan bij de start van het systeem en dat is niet veilig.
Ook zeggen de eisen niets over andere weggebruikers; misschien is het realistischer om het overal over weggebruikers te hebben in plaats van over automobilisten.

c Uit de praktijk weten we natuurlijk dat niet alle voetgangers wachten tot het licht groen is en niet alle automobilisten stoppen voor een rood licht.

6.8 Zie de opsommingen op pagina 35-36 respectievelijk 37 van het tekstboek.

Domain Understanding and Requirements Elicitation

Leereenheid 7

Domain Understanding and Requirements Elicitation

INTRODUCTIE

Bij deze leereenheid hoort hoofdstuk 2 van het tekstboek, waarin een (kort) overzicht wordt gegeven van elicitatietechnieken.

LEERDOELEN
Na het bestuderen van deze leereenheid wordt verwacht dat u
- drie gebieden kunt noemen waarover kennis verzameld moet worden
- minstens vijf problemen kunt noemen bij knowledge elicitation
- van de volgende elicitatietechnieken kunt aangeven wat de techniek inhoudt, welk type kennis ermee verworven kan worden en wat de sterke en zwakke punten zijn:
  - background study
  - data collection
  - questionnaires
  - concept driven acquisition
  - narrative techniques (storyboards and scenarios)
  - mock-ups and prototypes
  - interviews
  - observation techniques
  - group sessions
- de betekenis kunt geven van de volgende begrippen: artefact-driven techniques, stakeholder-drive techniques, repertory grid, card sort, conceptual laddering, functional prototype, user interface prototype, mock-up prototype, throwaway prototype, evolutionary prototype, protocol analysis, ethnographics studies

Studeeraanwijzingen
Bestudeer hoofdstuk 2 van het tekstboek.

De studielast van deze leereenheid bedraagt circa 8 uur.

LEERKERN

1 Opgaven

OPGAVE 7.1
Lees case study 1 op pagina 6 van het tekstboek.
a Noem minstens drie verschillende groepen stakeholders van dit systeem en noem voor elke stakeholder een belang.
b Welke van de problemen bij knowledge elicitation genoemd op pagina 62 en 63 van het tekstboek zouden hier van toepassing kunnen zijn? Licht uw antwoord toe, eventueel met een verzonnen voorbeeld.

OPGAVE 7.2

a Wat is het verschil tussen artefact-driven en stakeholder-driven elicitatietechnieken?
b Noem twee artefact-driven technieken en twee stakeholder-driven technieken.
c Vindt u dit een scherp onderscheid? Zo nee, noem technieken die kenmerken van beiden hebben. Licht uw antwoord kort toe.

OPGAVE 7.3

Geef aan in hoeverre de volgende technieken gebruikt kunnen worden bij het opstellen van functionele requirements voor het system-to-be. Licht uw antwoord toe.
a Background study
b Het in samenwerking met een groep stakeholders opstellen van scenario's
c Card sorts

OPGAVE 7.4

Geef aan in hoeverre de volgende technieken gebruikt kunnen worden bij het opstellen van niet-functionele requirements voor het system-to-be. Licht uw antwoord toe en maak daarbij ook een onderscheid naar het type niet-functionele requirements.
a Data collection
b Questionnaires
c Throwaway prototypes

OPGAVE 7.5

Deze opdracht is gebaseerd op een opdracht op pagina 86 van het tekstboek en gaat over het hergebruik van kennis.
Gegeven een volledige en goede set van requirements voor een systeem dat bij de melding van een ongeluk automatisch de dichtstbijzijnde ambulance alarmeert en naar de plaats van het ongeluk dirigeert.
Het softwarehuis dat dit systeem gerealiseerd heeft, krijgt een opdracht voor het realiseren van een systeem waarmee via een webinterface taxi's besteld kunnen worden. Het systeem moet zelf een geschikte taxi oproepen en naar de klant sturen.
a Wat zijn belangrijke overeenkomsten tussen deze twee systemen?
b Wat zijn belangrijke verschillen?
c Schets een strategie voor hergebruik van kennis in dit geval.

TERUGKOPPELING

Uitwerking van de opgaven

7.1 a We noemen vijf voorbeelden van groepen stakeholders en hun mogelijke belangen.
– Het bestuur van de Universiteit wil betere toegang tot literatuur tegen lagere kosten.
– De medewerkers van de bibliotheken van de verschillende faculteiten willen dat hun werk optimaal ondersteund wordt. Ze willen ook hun baan houden; een gemeenschappelijk systeem is daarom voor hen mogelijk ook een bedreiging.
– De gebruikers van de bibliotheken willen snelle en makkelijke toegang tot literatuur.
– Degenen die de huidige facultaire systemen hebben gebouwd en deze nu onderhouden zijn mogelijk trots op hun systeem en vinden het niet prettig dat het overbodig wordt.
– De medewerkers van de faculteiten die nu over de aankopen gaan, willen graag zeggenschap daarover houden.

b Alle genoemde problemen kunnen optreden, zij het niet in gelijke mate.
– *Distributed and conflicting knowledge sources:* Iedere faculteit heeft nu zijn eigen bibliotheek met eigen procedures voor de aanschaf van boeken, de registratie van leden, de uitleen van materialen enzovoort (zie onderaan pagina 6 van het tekstboek). Die procedures vertonen onderling allerlei verschillen.
– *Difficult access to sources*: Het zou kunnen dat de medewerkers van de facultaire bibliotheken en/of de makers van de huidige systemen wantrouwend staan tegenover een centraal systeem. Hun medewerking is dan niet bij voorbaat gegarandeerd, zeker als hun dat veel tijd kost.
– *Obstacles to good communication*: De kennisverwervers zullen zich moeten inwerken in de manier waarop wetenschappers met bronnen omgaan om hun behoeften te kunnen begrijpen, maar in het algemeen is kennis over bibliotheekprocedures vrij expliciet en ook niet helemaal vreemd. Dit is naar verwachting dan ook niet het grootste probleem.
– *Tacit knowledge and hidden needs*: Het bibliotheekpersoneel kan bijvoorbeeld het feit dat alle leden geregistreerd moeten zijn, vergeten te vermelden omdat het voor hen triviaal is. Het voor het systeem belangrijke onderscheid tussen een boek (een titel) en een bijbehorend exemplaar is voor het personeel soms verwarrend.
Ook dit lijkt bij een bibliotheek niet het grootste probleem.
– *Sociopolitical factors*. Die kunnen er zeker zijn. Het bibliotheekpersoneel kan het gecentraliseerde systeem als een bedreiging zien. De faculteitsbesturen hadden veel invloed op de facultaire systemen; die invloed raken ze grotendeels kwijt als er een centraal systeem komt. Er kan wantrouwen bestaan tussen de verschillende faculteiten (historici, natuurwetenschappers en rechtsgeleerden gaan allemaal heel anders met bronnen om).
– *Unstable conditions*: Ook dit kan altijd optreden. Misschien was er één lid van het college van bestuur dé grote voorstander van dit nieuwe systeem, terwijl anderen er minder de noodzaak van inzien. Als dat ene lid een andere baan krijgt en vertrekt, besluiten de anderen mogelijk het project financieel te korten.

7.2 a Artefact-driven technieken berusten primair op het gebruik van specifieke artefacten. Stakeholder-driven technieken berusten primair op een bepaalde vorm van interactie met de stakeholders.
b Background study, data collection, questionnaires, concept driven acquisition, narrative techniques (storyboards and scenarios) en mock-ups / prototypes worden genoemd als artifact-driven technieken; interviews, observation techniques en group sessions als stakeholder-driven technieken.
c Het onderscheid lijkt ons eerder gradueel dan zwart/wit. Storyboards en scenario's bijvoorbeeld leveren specifieke artefacten op met een voorgeschreven formaat, maar zijn voor de totstandkoming volledig afhankelijk van de interactie met stakeholders. Tussen questionnaires en interviews zijn ook mengvormen denkbaar, bijvoorbeeld sterk gestructureerde interviews die wel direct worden afgenomen, of questionnaires met veel open vragen.

7.3 a Background studies zijn vooral van belang om kennis te verwerven over de organisatie, het domein en het system-as-is. Die kennis is nodig om met behulp van andere technieken de (functionele) requirements boven tafel te krijgen (anders begrijpt de kennisverwerver niet wat de stakeholders willen), maar die bijdrage is dus indirect.
b Deze techniek levert voorbeelden van gewenste interacties met het systeem en is dus vooral bedoeld voor het achterhalen van functionele requirements.
c Card sorts kunnen gebruikt worden om inzicht te krijgen in het domein maar ook, zoals de voorbeelden onderaan pagina 66 van het tekstboek aangeven, voor het achterhalen van domeineigenschappen en requirements.

7.4 a Data collection kan indirect bijdragen tot het boven tafel krijgen van niet-functionele requirements (zie ook paragraaf 2.2.2 op pagina 65 van het tekstboek). Zo laat het verzamelen van gegevens over het gebruik van het system-as-is (usage statistics) zien welke functies in dat systeem het meest gebruikt worden; het ligt voor de hand om daar ook de hoogste usability-eisen aan te stellen. Statistieken over het raadplegen van FAQ's en helpfuncties kunnen aanwijzingen geven voor usability-problemen. Cijfers over de performance kunnen een eerste indicatie geven voor performance-eisen aan het nieuwe systeem.
b Questionnaires behoren tot de technieken die kunnen worden ingezet bij het vaststellen van niet-functionele requirements. Denk bijvoorbeeld aan vragen als "Geef met een cijfer tussen 1 (weinig schade) en 5 (heel veel schade) aan hoe schadelijk het zou zijn voor het bedrijf als elk van de volgende gegevensverzamelingen verloren zouden gaan" (beveiliging), "Welke uitvalfrequenties van het systeem vindt u nog net acceptabel, indien de uitval niet langer dan een half uur duurt?" (availability). Vragen naar acceptabele responstijden zijn lastiger; pas als iemand achter de computer zit, is het duidelijk hoe lang bijvoorbeeld een responstijd van vijf seconden voelt.
c Throwaway prototypes zijn zeer geschikt voor het verwerven en vooral het valideren van de usability van het systeem, tenminste als het prototype dezelfde user interface heeft. Ze zijn minder geschikt voor andere niet-functionele requirements (zie pagina 71 van het tekstboek).

7.5 a De overeenkomsten zijn vrij groot. In beide gevallen moet het systeem
– de locatie van een aantal voertuigen volgen
– bij een aanvraag (een ongeluk of een taxioproep) het voertuig vinden dat het snelst ter plaatse kan zijn
– een oproep versturen naar dat voertuig
– verifiëren dat het voertuig de oproep ontvangen heeft en zich naar de betreffende plaats begeeft.
b Er zijn echter ook enkele niet onbelangrijke verschillen.
– Het ambulancesysteem is veel kritischer: lukt het een keer niet om binnen een acceptabele tijd een taxi te sturen, dan is dat vervelend; lukt het niet om binnen een acceptabele tijd een ambulance te sturen, dan kunnen er onnodig doden vallen.
– Zowel het aantal voertuigen als het aantal ritten is voor taxi's zo veel groter dan voor ambulances dat er een echt schaalverschil is; het is daardoor niet op voorhand zeker dat een strategie die goed werkt voor het ene systeem ook goed werkt voor het andere.
c Het domeinmodel van het ambulancesysteem kan vrij gemakkelijk geabstraheerd worden: het concept *ambulance* wordt vervangen door *voertuig*; *melding van ongeluk* door *aanvraag*, terwijl zaken die alleen relevant zijn voor het ambulancesysteem (aantal gewonden, aard van de verwondingen) niet worden meegenomen. Dat model kan (gebruikmakend van elicitatietechnieken) geconcretiseerd worden voor het taxisysteem, waarbij zaken die in het ambulancesysteem niet voorkwamen, worden toegevoegd. Een zelfde type techniek kan ook worden toegepast op de requirements zelf.

Goal Orientation in Requirements Engineering

Leereenheid 8

Goal Orientation in Requirements Engineering

INTRODUCTIE

Bij deze leereenheid hoort hoofdstuk 7 van het tekstboek. De contouren van deze goal-oriented methode worden in dit hoofdstuk al enigszins zichtbaar.

LEERDOELEN

Na het bestuderen van deze leereenheid wordt verwacht dat u

- kunt bepalen of een bepaalde uitspraak opgevat kan worden als een doel (goal) van een systeem
- de volgende begrippen kunt omschrijven en aan elkaar kunt relateren: multi-agent goal, single-agent goal, domain property, domain hypothesis, requirement, expectation
- de volgende typen doelen kunt onderscheiden: achieve, cease, maintain, avoid, soft goal, optimization goal
- kunt omschrijven wat bedoeld wordt met de goal-categorieën satisfaction, information, stimulus-response en accuracy
- goals kunt indelen op grond van hun type (of specification pattern) en op grond van hun categorie
- tenminste vijf redenen kunt noemen voor het belang van goals in requirements engineering
- de goal-oriented benadering kunt vergelijken met benaderingen gebaseerd op respectievelijk scenario's, use cases en agents
- de betekenis kunt geven van de volgende begrippen: doel (goal), agent, goal refinement, goal abstraction, behavioural goal, probabilistic goal en goal specification pattern.

Studeeraanwijzingen

De studielast van deze leereenheid bedraagt circa 7 uur. Dit hoofdstuk maakt in beperkte mate gebruik van lineaire temporele logica (LTL). Dit kunt u nalezen in hoofdstuk 4 van het boek. Dit behoort echter niet tot de verplichte leerstof.

LEERKERN

1 Opgaven bij paragrafen 7.1 en 7.2

OPGAVE 8.1

a Wat wordt in dit boek verstaan onder een doel (goal)?
b Wat wordt bedoeld met een agent?
c Wanneer is een doel ook een requirement? Wanneer is het een verwachting (expectation)?
d Wat is de overeenkomst en wat is het verschil tussen een domain property en een domain hypothesis?

OPGAVE 8.2

Tekstboek pagina 283

Maak de eerste zes onderdelen (a t/m f) van de eerste opgave bij hoofdstuk 7 uit het tekstboek ('Classify the following statements about a bank ATM system…'). We rekenen de passen daarbij tot de omgeving van het systeem en niet tot het systeem zelf, omdat de specificatie daarvan al vastligt.

2 Aanvullingen bij paragraaf 7.3

Verderop in het tekstboek (zie Goal specification patterns op pagina 589 en figure 17.1 op pagina 590) wordt een iets andere taxonomie gegeven die ook is overgenomen in Objectiver, de tool die u verderop in de cursus gaat gebruiken.

Goal specification pattern

Volgens deze taxonomie worden doelen langs twee dimensies ingedeeld, namelijk hun *specification pattern* (patroon) en hun categorie. Het patroon heeft betrekking op het temporele gedrag. Er worden vier patronen onderscheiden, namelijk Achieve, Cease, Maintain en Avoid. Figuur 8.1 toont voorbeelden van temporele formules die met elk van dit type doelen geassocieerd kunnen worden.

Achieve [TargetCondition]:
$\text{CurrentCondition} \Rightarrow \Diamond\ \text{TargetCondition}$
$\text{CurrentCondition} \Rightarrow \Diamond_{\leq d}\ \text{TargetCondition}$
$\text{CurrentCondition} \Rightarrow \bigcirc\ \text{TargetCondition}$

Cease [TargetCondition]:
$\text{CurrentCondition} \Rightarrow \Diamond\ \neg\text{TargetCondition}$
$\text{CurrentCondition} \Rightarrow \Diamond_{\leq d}\ \neg\text{TargetCondition}$
$\text{CurrentCondition} \Rightarrow \bigcirc\ \neg\text{TargetCondition}$

Maintain[GoodCondition]:
$\text{CurrentCondition} \Rightarrow \text{GoodCondition}$
$\text{CurrentCondition} \Rightarrow \Box\ \text{GoodCondition}$
$\text{CurrentCondition} \Rightarrow \text{GoodCondition}\ \boldsymbol{W}\ \text{NewCondition}$

Avoid [BadCondition]:
$\text{CurrentCondition} \Rightarrow \neg\text{BadCondition}$
$\text{CurrentCondition} \Rightarrow \Box\ \neg\text{BadCondition}$
$\text{CurrentCondition} \Rightarrow \neg\text{BadCondition}\ \boldsymbol{W}\ \text{NewCondition}$

FIGUUR 8.1 Temporele formules geassocieerd met de vier goal specification patterns

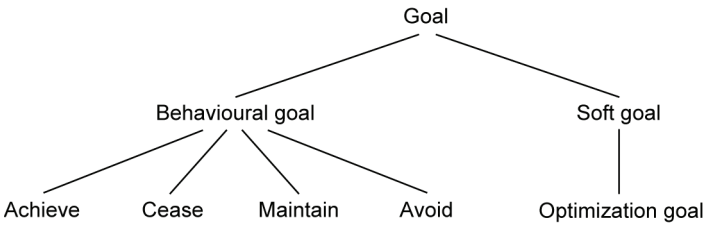
In Objectiver worden verder ook nog *optimization goals* genoemd als expliciete subcategorie van soft goals. De volgende definitie van dit begrip is ontleend aan het al eerder genoemde proefschrift van Emmanuel Letier:

Optimization goal

Optimization goals are soft goals that enable one to compare alternative system designs through minimize/maximize operators.

Een voorbeeld is dus een doel als 'het aantal interacties tussen de beoogde deelnemers aan een geplande vergadering moet zo klein mogelijk zijn'.

Figuur 8.2 toont de volledige indeling gebruikt in Objectiver. We houden deze indeling verder aan.



FIGUUR 8.2 Goal patterns

OPGAVE 8.3
a Is een soft goal hetzelfde als een non-functional goal?
b Wat wordt bedoeld met een probabilistic goal? Is dit per definitie een soft goal?

OPGAVE 8.4
Tekstboek pagina 284.
Maak de tweede opgave bij hoofdstuk 7 uit het tekstboek ('Identify the type and the category of each of the following goals…'). Gebruik voor het type figuur 8.2 hierboven.

3 Opgaven bij paragrafen 7.4-7.6

OPGAVE 8.5
Noem tenminste vier redenen waarom doelen belangrijk zijn in het requirements engineering-proces.

OPGAVE 8.6
a Hoe verhoudt een goal-oriented benadering zich tot een specificatie die uitgaat van agent operaties?
b Hoe verhoudt een goal-oriented benadering zich tot een scenario-oriented benadering?

OPGAVE 8.7
Uit andere cursussen bent u mogelijk op de hoogte van het specificeren van requirements met behulp van use-casebeschrijvingen (eventueel aangevuld met een use casediagram). Hoe verhoudt die benadering zich tot goal orientation?

TERUGKOPPELING

Uitwerking van de opgaven

8.1 a Een doel is een voorschrijvende uitspraak over het systeem waaraan dit moet voldoen door de samenwerking van de agents waaruit het systeem is opgebouwd (in het Engels staat er 'a prescriptive statement of intent that the system should satisfy through cooperation of its agents'.
b Een agent is een actieve systeemcomponent die een rol speelt bij het verwezenlijken van de doelen van het systeem. Een agent kan een mens zijn, of een stuk software, of een apparaat (met name sensoren en actuatoren zijn agents).
c Een requirement is een doel onder de verantwoordelijkheid van één agent van de software onder constructie (de 'software-to-be'). Een expectation is een doel onder de verantwoordelijkheid van één agent in de omgeving software onder constructie.
d De overeenkomst is dat beide beschrijvende uitspraken (descriptive statements) zijn, waarvan de waarheidswaarde niet beïnvloed wordt door het gedrag van het systeem. Het verschil is dat een domain property onveranderlijk is en altijd waar (bijvoorbeeld een gevolg van natuurwetten) terwijl een domain hypothesis wel kan veranderen.

8.2 a (ATM is an acronym for Automatic Teller Machine). Dit is een definitie.
b (Ubiquitous cash services shall be provided). Dit is een hoog niveau (strategisch) doel waar de realisatie van het ATM systeem aan bijdraagt.
c (ATM passwords are 4-digit numbers). Als de passen niet tot het systeem behoren, dan is dit een beschrijvende uitspraak over de omgeving van het systeem. Het is echter geen natuurwet; de geldwereld kan besluiten om over te gaan op een ander type wachtwoord. Het is dus een domain hypothesis.
d (The likelihood of forgetting ATM cards at ATM counters shall be reduced as much as possible). Dit is een doel, maar geen requirement of expectation. Bij de verwezenlijking ervan zijn immers verschillende agents betrokken, zoals de automaat zelf, de gebruiker en de software die de automaat bestuurt.
e (Users will take their ATM card back if it is returned with a beep before cash delivery). Dit schrijft gedrag voor van één agent (de gebruiker) in de omgeving van de software-to-be; het is dus een expectation.
f (Any ATM card shall be kept after three wrong password entries). Dit is een doel onder verantwoordelijkheid van de software-to-be en dus een requirement.

8.3 a Nee, een soft goal kan zowel functional als non-functional zijn. Zie pagina 271 van het tekstboek (tweede alinea onder het subkopje Difference between goal types and goal categories).
b Een probabilistic goal stelt dat een bepaald doel gehaald moet worden in (minimaal) een bepaald percentage van de gevallen, bijvoorbeeld "bij 95% van de noodmeldingen moet er binnen veertien minuten een ambulance ter plaatse zijn". Omdat precies vastgesteld kan worden of het uiteindelijke systeem aan een dergelijk doel voldoet, is het geen soft goal maar een behavioural goal.

8.4 a (The red button on the ambulance panel shall be pushed when the ambulance arrives at the incident scene). Dit is een behavioural goal van type Achieve en categorie Stimulus-response
b (The stress inherent to working conditions of ambulance crews should be reduced): type Soft goal, categorie Usability. Omdat er 'reduced' staat en niet 'minimized', benoemen we dit niet als optimization goal.
c (An ambulance is displayed by the software as being available if and only if it is actually ready for intervention at its ambulance station): type Maintain, categorie Accuracy.
d (The nearest available ambulance shall be mobilized for the incident): type Achieve, categorie Satisfaction.
e (An ambulance may not be mobilized for an incident if it has not been allocated to that incident): type Avoid, categorie Satisfaction.
f (Upon mobilization the selected ambulance shall receive a mobilization order on its mobile terminal): type Achieve, categorie Information.

8.5 Paragraaf 7.4 noemt vijftien redenen voor het belang van doelen. We halen er enkele uit die in onze ogen zeer relevant zijn (maar andere antwoorden uit de lijst zijn uiteraard ook goed).
– Door te werken met doelen is het verband tussen de strategische doelen van de organisatie en de uiteindelijke requirements voor het systeem gegarandeerd.
– Doelen geven richting aan het requirements elicitation proces.
– Het steeds verder verfijnen van doelen geeft (a) een natuurlijke manier om complexe specificaties te structureren en (b) biedt daardoor uiteindelijk ook een structuur voor het requirements-document.
– Doelen bieden een aanknopingspunt voor risicoanalyse (risico's kunnen geformuleerd worden als omstandigheden die de verwezenlijking van een bepaald doel verhinderen).
– Door te werken vanuit globale doelen worden de systeemgrenzen duidelijk, zodat er geen nutteloze requirements worden opgenomen.

8.6 a Zie paragraaf 7.6.5. Deze benaderingen liggen in elkaars verlengde. In goal orientation zijn de doelen van het systeem de drijvende krachten achter het requirements engineering-proces, maar bij het specificeren van de uiteindelijke requirements (de bladeren in de boom van doelen met hun verfijningen) spelen de agents een belangrijke rol. En als wordt uitgegaan van agents, dan worden die benoemd en geanalyseerd op grond van de doelen die ze moeten verwezenlijken.
b Zie paragraaf 7.6.1. Deze benaderingen zijn complementair. Het sterke punt van goal orientation is het expliciet vastleggen van de bedoeling van het systeem; een mogelijk zwak punt is dat het precieze systeemgedrag impliciet blijft. Bij scenario's is het precies omgekeerd.

8.7 Use-casebeschrijvingen geven een uitwerking in woorden van specifieke scenario's (met nog wat alternatieven). Wat in paragraaf 7.6.1 wordt gezegd over scenario's, geldt dus ook voor use-casebeschrijvingen. Paragraaf 7.6.3 gaat niet over zulke beschrijvingen, maar over use-case-diagrammen die niet veel meer tonen dan de use cases zelf en de agents die erbij betrokken zijn.

Modelling System Objectives with Goal Diagrams

Leereenheid 9

Modelling System Objectives with Goal Diagrams

INTRODUCTIE

Bij deze leereenheid horen de inleiding bij deel II en hoofdstuk 8 van het tekstboek.

LEERDOELEN

Na het bestuderen van deze leereenheid wordt verwacht dat u

- kunt aangeven wat bedoeld wordt met achtereenvolgens de intentional, structural, responsibility, functional en behavioural view van een te ontwikkelen systeem en welke KAOS-modellen daarmee overeenstemmen
- goals kunt definiëren en daarbij de volgende goal features kunt gebruiken: name, def, type, category, priority en issue
- een gegeven goal kunt verfijnen met behulp van AND- en OR-splitsing en het verschil daartussen weet
- kunt beslissen of een goal een requirement is en zo ja een verantwoordelijke agent kunt benoemen
- kunt beslissen of een agent tot de environment of tot de software behoort
- parent goals kunt vinden bij een gegeven goal
- conflicten tussen goals kunt identificeren en deze in de goal tree kunt opnemen
- een AND/OR goal graph kunt construeren bestaand uit goals, soft goals, optimization goals en requirements en daarbij gebruik kunt maken van de heuristieken en patronen beschreven in het tekstboek
- de betekenis kunt geven van de volgende begrippen: complete refinement, consistent refinement, minimal refinement, milestone drive refinement pattern, decomposition-by-case pattern, guard introduction pattern, unmonitorability-driven refinement pattern, uncontrollability-driven refinement pattern.

Studeeraanwijzingen

De inleiding bij deel II, op pagina 287-291, geeft een overzicht van de komende hoofdstukken. Deze inleiding behoort tot de studiestof.

De studielast van deze leereenheid bedraagt circa 10 uur.

LEERKERN

1 Opgave bij inleiding blok II

OPGAVE 9.1

a Wat wordt bedoel met de intentional view van een te ontwikkelen systeem en welk KAOS-model stemt daarmee overeen?

b Idem voor de responsibility view.

2 Opgaven bij hoofdstuk 8

OPGAVE 9.2
Wat wordt verstaan onder achtereenvolgens
a complete refinement
b consistent refinement
c minimal refinement?

OPGAVE 9.3
Wat is het verschil tussen AND- en OR-refinement?

OPGAVE 9.4
Geef in enkele zinnen aan wat de essentie is van de volgende patronen:
a milestone-driven refinement pattern
b decomposition-by-case pattern
c guard-introduction pattern
d divide-and-conquer pattern
e unmonitorability-driven refinement pattern
f uncontrollability-driven refinement pattern

OPGAVE 9.5
Tekstboek pagina 331
Maak de tweede opgave op pagina 331 van het tekstboek ('In a simple patient-monitoring system…').

OPGAVE 9.6
Welke bladeren in de goal tree uit figuur 9.1 in de uitwerking van opgave 9.5 zijn requirements en welke zijn expectations?

OPGAVE 9.7
De goal van een systeem voor het toewijzen van resources aan processen kan worden geformuleerd als Achieve[ResourceAllocated if ResourceRequested].
a Splits deze goal in subgoals, gebruikmakend van het guard-introduction pattern, met als guard condition ResourceAvailable.
b Wat is de betekenis van het laatste van de drie subgoals?
Aanwijzing
Unless heeft bijna dezelfde betekenis als *until* (het enige verschil is dat P until Q eist dat Q ooit gaat gelden, terwijl P unless Q ook waar is als Q nooit gaat gelden, mits P dan altijd waar blijft).
c Zal de derde subgoal in de praktijk altijd verwezenlijkt worden? Wat kunt u daar uit opmaken?
d Leg uit waarom in dit geval het milestone-driven pattern niet gebruikt kan worden, met Achieve[ResourceAvailable] als milestone.

OPGAVE 9.8
Tekstboek pagina 332
Maak de eerste opgave op pagina 332 ('Consider the following portion…').

OPGAVE 9.9
Tekstboek pagina 333
Maak de derde opgave op pagina 333 ('Consider the following goal model fragment from an air traffic control system…').

TERUGKOPPELING

Uitwerking van de opgaven

9.1 a De intentional view modelleert de functionele en niet-functionele doelen van het systeem. In de KAOS-methode wordt deze view gemodelleerd in het goal model.
b In de responsibility view worden de agents vastgelegd die samen het systeem vormen evenals de systeemdoelen waarvoor ze verantwoordelijk zijn. In de KAOS-methode wordt deze view gemodelleerd in het agent model.

9.2 Een verfijning van een goal is
a compleet als de verwezenlijking van de subgoals samen met de geldigheid van domeineigenschappen en hypothesen garandeert dat ook de oorspronkelijke goal verwezenlijkt is
b consistent als er in het geheel van subgoals, hypothesen en domeineigenschappen geen tegenspraak zit
c minimaal als de verfijning compleet is terwijl het weglaten van willekeurig welke enkele subgoal de verfijning incompleet maakt.

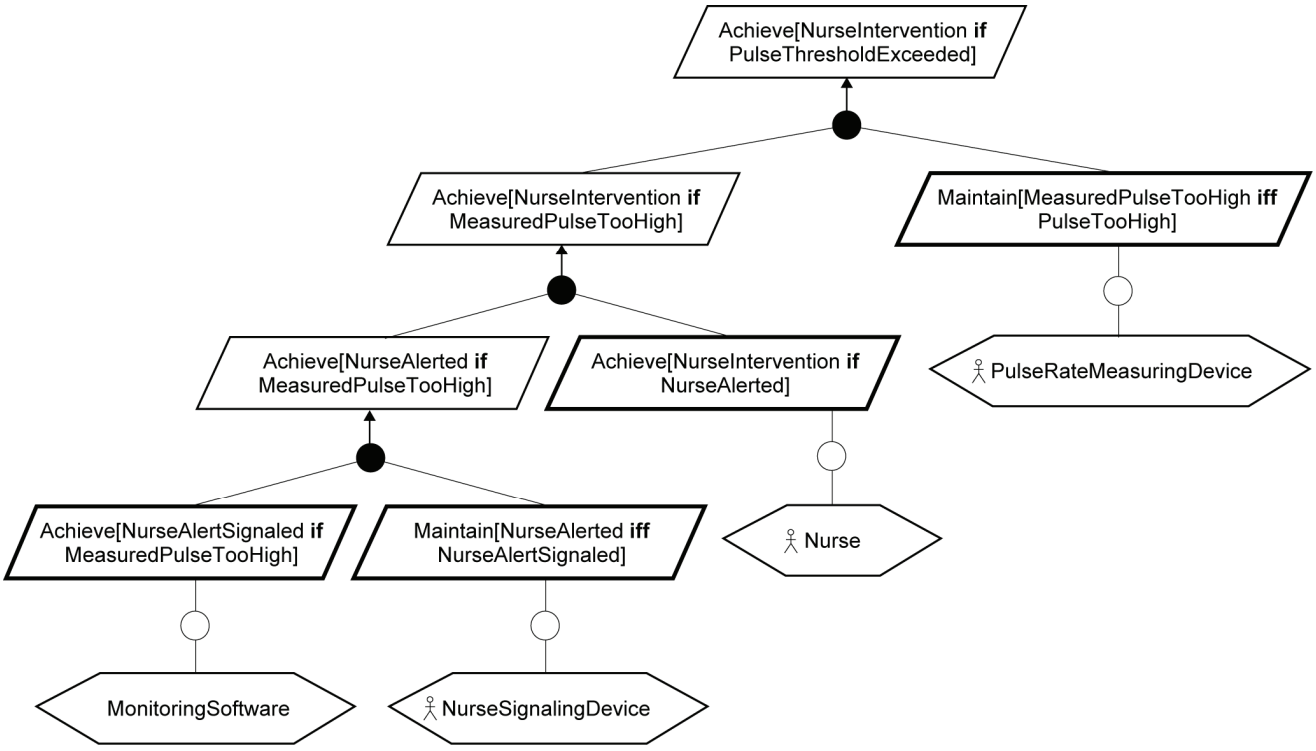
9.3 Bij een AND-splitsing moet alle subgoals bereikt zijn om ook de gesplitste goal te bereiken. Een OR-splitsing geeft alternatieve manieren om de goal te bereiken. Bij de uiteindelijke realisering van het systeem moet voor één van de alternatieven worden gekozen. Het is verrassend makkelijk om die twee door elkaar te halen, speciaal bij het splitsen van een goal in verschillende gevallen. Zie valkuil H14 op pagina 318 van het tekstboek.

9.4 a Dit patroon splitst een behavioural Achieve goal op in subgoals die na elkaar bereikt worden. Het bereiken van de oorspronkelijke goal moet dan onmogelijk zijn zonder ook de subgoals te bereiken; het is niet de bedoeling dat je bij het gebruiken van dit patroon vast gaat ontwerpen of programmeren! Er is ook een variant voor een Maintain goal.
b Dit patroon splitst een behavioural Achieve goal op in verschillende subgoals voor verschillende, elkaar uitsluitende situaties. De opsplitsing moet compleet zijn. Let op dat het hier om een AND-splitsing gaat (zie ook opgave 9.3).
c Dit patroon is toepasbaar als het systeem moet wachten tot een andere conditie gaat gelden (de guard) om de goal te kunnen bereiken. De oorspronkelijke conditie moet ook blijven gelden.
Zie opgave 9.7 voor een voorbeeld.
d Dit patroon is bedoeld voor Maintain goals, waarbij de te handhaven conditie gesplitst kan worden in twee of meer deelcondities die allemaal gehandhaafd moeten blijven. Voor elke deelconditie wordt een subgoal toegevoegd, eveneens van het type Maintain.
e Dit patroon is van toepassing als een goal waarbij een grootheid gemonitord moet worden, niet aan één agent toegewezen kan worden. Kijk in dit verband ook naar figure 1.4 op pagina 21 van het tekstboek, waarin te zien is dat een monitored variable niet direct door de software-to-be gemonitord wordt maar dat daar een input device (bijvoorbeeld een sensor) bij betrokken is. Dit patroon is nuttig omdat het laat zien dat er nog verder verfijnd moet worden. Het geeft vaak aanleiding tot de introductie van accuracy goals, om te garanderen dat de gemeten waarde door de sensor ook klopt met de echte waarde.

f Dit patroon is vergelijkbaar met het vorige, maar nu gaat het niet om monitored maar om controlled variables en niet om een input maar om een output device.

9.5 a In deze boom wordt PulseThresholdExceeded opgevoerd als een subgoal van NurseAlerted. Maar dat een patiënt een te hoge polsslag heeft, is geen doel van het systeem. Het is een gebeurtenis waar het systeem op moet reageren. Een overeenkomstige fout in de LAS-casus is om de goal AmbulanceInterventionOnCarAccident een subgoal CarAccident te geven. (Beide fouten zijn in Leuven echt door studenten gemaakt).

b Het probleem is, dat de software noch direct de polsslag meet, noch direct de verpleging waarschuwt. Het eerste wordt gedaan door een sensor; het tweede door een actuator. De software kan ook niet ingrijpen; dat moet een verpleger doen. Er zijn dus naast de software nog drie actoren: de polsmeter (die een gemeten polsslag doorgeeft), het alarm (dat afgaat op een teken van de software) en de verpleger zelf. Figuur 9.1 toont een mogelijke verfijning van deze goal.

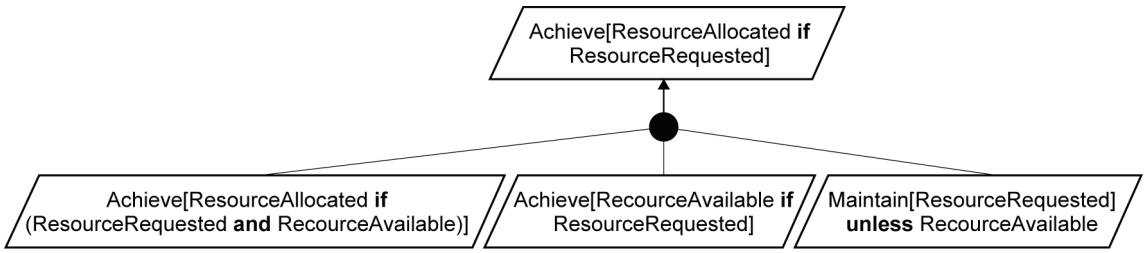


FIGUUR 9.1 Verfijning op grond van monitorability en controllability

De requirements Maintain[MeasuredPulseTooHigh iff PulseTooHigh] en Maintain[NurseAlerted iff NurseAltertSigneled] zijn accuracy requirements die een verband leggen tussen een grootheid in de software en de overeenkomstige grootheid in de omgeving. Bij deze requirements staat iff (if and only if), terwijl bij de andere if staat. Er zijn immers waarschijnlijk nog andere condities dan een te hoge polsslag die leiden tot het alarmeren van de verpleegster. Merk verder op dat al deze verfijningen volledig zijn.

9.6 Een requirement is een single agent goal onder verantwoording van de software-to-be; een expectation is een single agent goal onder verantwoording van de omgeving. Van de vier agents uit figuur 9.1 horen er drie tot de omgeving, namelijk NurseSignalingDevice (dat is een pieper of een alarmbel), Nurse (de menselijke verpleger) en PulseRateMeasuringDevice. (In figuur 9.1 is dit weergegeven door de poppetjes bij deze agents). Alleen de goal Achieve[NurseAlertedIfMeasuredPulseTooHigh] is dus een requirement; de andere drie zijn expectations.

9.7 a Zie figuur 9.2.
b De derde subgoal geeft aan dat de goal alleen wordt bereikt als de vraag naar de resource gehandhaafd blijft totdat deze beschikbaar is.
c In de praktijk kan het voorkomen dat de vraag voor een resource wordt ingetrokken voor die resource beschikbaar komt. De oorspronkelijke goal stelt echter dat als een resource gevraagd wordt, die ooit gealloceerd moet worden. Eigenlijk is die goal dus te streng. Dit is typisch een punt dat bij obstacle analysis duidelijk kan worden (zie volgende leereenheid).
d Het milestone-driven pattern identificeert een tussentoestand: eerst geldt de current condition, dan de milestone condition en als laatste de target condition. Het is daarbij niet relevant of in de tussentoestand (bij het bereiken van de milestone condition) de oorspronkelijke conditie ook nog geldt.In dit geval wordt de oorspronkelijke goal niet bereikt als de resource beschikbaar komt terwijl de vraag is vervallen.

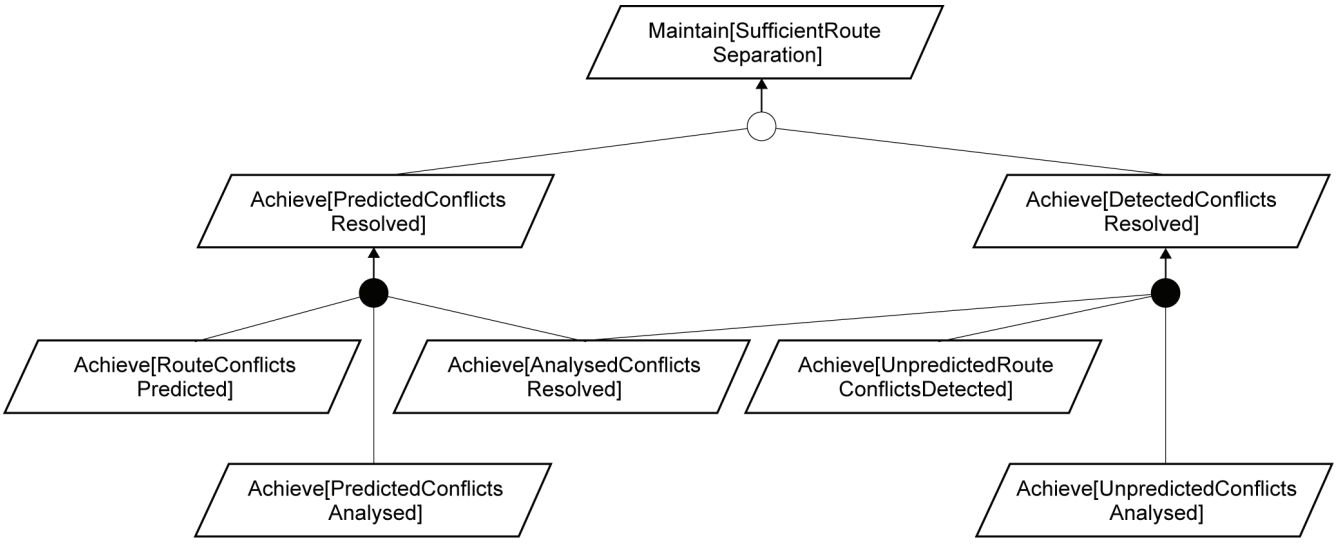


FIGUUR 9.2 Splitsing met behulp van het guard-introduction pattern

9.8 Deze fout staat vermeld op pagina 318 van het tekstboek als H14; 'Do not confuse AND-refinement into multiple cases with OR-refinement into multiple alternatives'. In dit voorbeeld had een AND-verfijning gebruikt moeten worden in plaats van een OR-verfijning. Er is immers geen sprake van twee verschillende manieren om de goal Maintain[LightOn if RoomAttended] te verwezenlijken die tot twee systemen leiden; het gaat om twee gevallen die door hetzelfde systeem gedekt moeten worden.

9.9 a De parent goal is makkelijk te achterhalen met behulp van een WHY-vraag en kan bijvoorbeeld omschreven worden als Avoid[AircraftCollision]. Een parent goal daarvan is bijvoorbeeld Achieve[SafePassengerTransportation].
b, c Er zijn kennelijk twee soorten RouteConflicts: degenen die vooraf voorspeld kunnen worden (PredictedRouteConflicts) en degenen die pas op het moment zelf gedetecteerd worden (DetectedRouteConflicts). Voor het eerste type bevat de boom uit de opgave twee mijlpalen (milestones): voor ze opgelost kunnen worden, moeten ze eerst voorspeld en vervolgens geanalyseerd worden.

Voor de tweede soort conflicten ontbreken de goals voor analyse en oplossing (dit is de onvolledigheid waarnaar gevraagd wordt in onderdeel c). Uit de opgave is niet echt op te maken of voorspelde en gedetecteerde conflicten op dezelfde manier kunnen worden geanalyseerd en opgelost. Wij hebben aangenomen dat de analyse kan verschillen maar de oplossing niet, zodat een graaf ontstaat (deze aanname wordt gesuggereerd door de verwoording van de goal AnalysedConflictsResolved, waarin het type van de conflicten niet vermeld staat). Deze graaf is weergegeven in figuur 9.3. Merk op dat deze graaf geen requirements bevat; elke goal kan nog verder verfijnd worden.



FIGUUR 9.3 Graaf voor SufficientRouteSeparation

Merk op dat we de eerste opsplitsing niet getekend hebben als een complete verfijning. Uiteindelijk zal moeten worden aangetoond dat de subgoal Achieve[UnpredictedRouteConflictsDetected] ook werkelijk alle niet voorspelde conflicten detecteert. Pas dan mag de verfijning als volledig worden aangemerkt.

Risk Analysis on Goal Models

Leereenheid 10

Risk Analysis on Goal Models

INTRODUCTIE

Bij deze leereenheid hoort hoofdstuk 9 plus een klein deel van hoofdstuk 13 van het tekstboek.

Voor bestudering van deze leereenheid is het van belang dat u paragraaf 3.2 en vooral paragraaf 3.2.2 uit het tekstboek goed doorneemt.

Risicoanalyse is uiteraard belangrijk om falen van de software-to-be te voorkomen. In de casus die u zelf gaat uitwerken in leereenheid 11, zult u zien dat risicoanalyse ook gebruikt wordt om het opstellen van de goal graph in stappen op te splitsen. In eerste instantie wordt het feit genegeerd dat sommige goals niet onder alle omstandigheden haalbaar zijn. In tweede instantie worden die goals onder de loep genomen, en wordt met behulp van risk analysis een obstacle diagram opgesteld waarin te zien is welke mechanismen het behalen van de goals in de weg kunnen zitten. Vervolgens kan dan bekeken worden welke maatregelen er nodig zijn om dat te voorkomen. Dit leidt tot een verdere verfijning van de goal graph. Deze aanpak is enigszins vergelijkbaar met de ontwikkeling van use-casebeschrijvingen in het Unified Process, waarbij eerst de gang van zaken wordt beschreven bij een normale afhandeling van een use case en pas later ook de uitzonderingen worden bekeken.

Aan het eind van deze leereenheid kijken kort naar de rol van scenario's bij het opstellen van goal graphs en het uitvoeren van een obstacle analysis.

LEERDOELEN

Na het bestuderen van deze leereenheid wordt verwacht dat u

- kunt omschrijven wat in de KAOS-methode verstaan wordt onder een obstakel (obstacle)
- weet wat bedoeld wordt met volledigheid (completeness) voor een doel of hypothese G van een verzameling obstakels
- de rol begrijpt van domain properties bij het identificeren van een (uiteindelijk) complete verzameling obstakels
- obstakels kunt definiëren en daarbij de volgende features kunt gebruiken: name, def, category, likelihood, criticality, issue
- een gegeven obstakel kunt verfijnen met behulp van AND- en OR-splitsing en het verschil daartussen weet
- kunt beslissen of een knoop in een obstacle graph een blad is en er vervolgens een resolution link mee kunt verbinden
- een AND/OR obstacle graph kunt construeren voor een assertie G
- de volgende technieken kunt toepassen voor het oplossen van obstakels: goal substitution, agent substitution, obstacle prevention, goal weakening, obstacle reduction, obstacle mitigation

- begrijpt hoe scenario's gebruikt kunnen worden als ondersteuning voor de analyse van goals en obstacles
- een obstakelanalyse kunt uitvoeren op een goal graph, bestaande uit selectie van asserties en vervolgens het identificeren, verfijnen, evalueren en oplossen van obstakels
- de betekenis kunt geven van de volgende begrippen: hazard, threat, inaccuracy, misinformation, resolution link en complete refinement.

Studeeraanwijzingen
De studielast van deze leereenheid bedraagt circa 8 uur.

LEERKERN

1 Opgaven bij hoofdstuk 9

OPGAVE 10.1
Omschrijf wat in de KAOS-methode wordt verstaan onder
a een obstakel voor het bereiken van een goal of hypothese G
b volledigheid (completeness) van een verzameling obstakels.

OPGAVE 10.2
a Leg uit wat het verband is tussen AND-verfijning van goals en OR-verfijning van obstacles.
b Is er ook een overeenkomstig verband tussen OR-verfijning van goals en AND-verfijning van obstacles? Licht dit toe.

OPGAVE 10.3
a Schets twee op logica gebaseerde technieken voor obstacle refinement.
b Is het gebruik van deze technieken uitsluitend een kwestie van formele logica?

OPGAVE 10.4
Pas de tweede techniek uit de vorige opgave (zie pagina 347 van het tekstboek en figure 9.10 op pagina 348) toe op de volgende goals. Bedenk voor beide zelf een geschikte domeineigenschap. Druk in woorden uit welk resultaat u hiermee heeft bereikt.
a Achieve $_{\leq 2 \text{ minutes}}$ [NurseIntervention if PulseThresholdExceeded]
b Maintain [ElectricPumpsOn]

OPGAVE 10.5
Beschrijf kort de volgende obstacle resolution tactics:
a goal substitution
b agent substitution
c obstacle prevention
d goal weakening
e obstacle reduction
f goal restoration
g obstacle mitigation

OPGAVE 10.6
Tekstboek pagina 356
Maak de eerste opgave op pagina 356 van het tekstboek (‘Consider a simplistic engine-control system…’)

OPGAVE 10.7
Tekstboek pagina 356
a Maak de vierde opgave op pagina 356 van het tekstboek (‘Consider the accuracy goal IncidentFormAccuratelyEncoded….’).
b Bedenk enkele voor de hand liggende tegenmaatregelen en bekijk daarna of u die onder kunt brengen in de lijst met tactieken uit paragraaf 9.3.3.

OPGAVE 10.8
Tekstboek pagina 356
Maak de vijfde opgave op pagina 356 van het tekstboek (‘Consider the goal AllocatedAmbulanceMobilized….’).

2 Scenario’s

Het opstellen van een goal model en het uitbreiden daarvan met behulp van obstacle analysis wordt door sommige mensen ervaren als erg abstract en daarom vrij lastig. Dit kan leiden tot wonderlijke goal trees (denk aan het benoemen van subgoals als PulseThresholdExceeded of CarAccident, zoals vermeld in opgave 6.5 uit de vorige leereenheid). Het kan dan nuttig zijn om de analyse te concretiseren door ook gebruik te maken van scenario’s. Dit wordt toegelicht in hoofdstuk 13, waarvan we u nu vragen een klein deel te bestuderen.

Studeeraanwijzing

Bestudeer uit hoofdstuk 13 van het tekstboek paragraaf 13.1 (pagina 450-454) en paragraaf 13.3 tot aan 13.3.2 (pagina 463-467). In paragraaf 13.3 kunt u de opmerkingen over state machines overslaan.

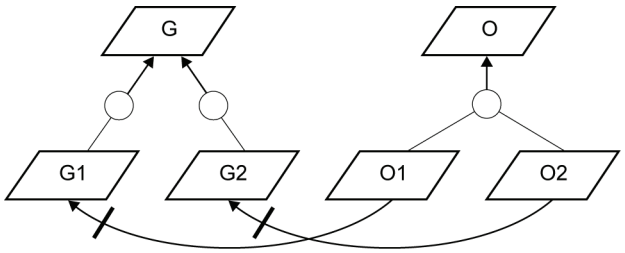
OPGAVE 10.9
Bekijk het normale scenario voor het plannen van een vergadering in figure 4.10 op pagina 137 van het tekstboek.
Welke exception cases kunnen optreden in dit normale scenario en hoe wordt het scenario dan voortgezet? (U mag zich beperken tot een beschrijving en hoeft de bijbehorende abnormale scenario’s niet te tekenen).

TERUGKOPPELING

Uitwerking van de opgaven

10.1 a Een conditie O is een obstacle voor een goal G als tegelijkertijd geldt dat
– O en de domeineigenschappen samen G onhaalbaar maken
– O consistent is met de domeineigenschappen
– O ook werkelijk kan optreden.
b Een verzameling obstacles O_1, …, O_n is volledig voor een goal G wanneer G gegarandeerd bereikt is wanneer geen van de obstacles geldt (en de domeineigenschappen wel gelden).

10.2 a Als
– een goal G kan worden verfijnd tot G1 **and** G2, en
– O1 is een obstakel is voor G1 en
– O2 is een obstakel is voor G2
dan is O1 **or** O2 een obstakel voor G.
Zie ook figure 9.5 op pagina 342 van het tekstboek.
Omdat obstakel analyse bij voorkeur begint bij de bladeren van de goal tree (zie paragraaf 9.3.1 op pagina 344 van het tekstboek), geeft dit verband een manier voor het naar boven propageren van obstakels.
b Het is niet moeilijk om een overeenkomstige figuur te maken voor het duale geval; zie figuur 10.1.



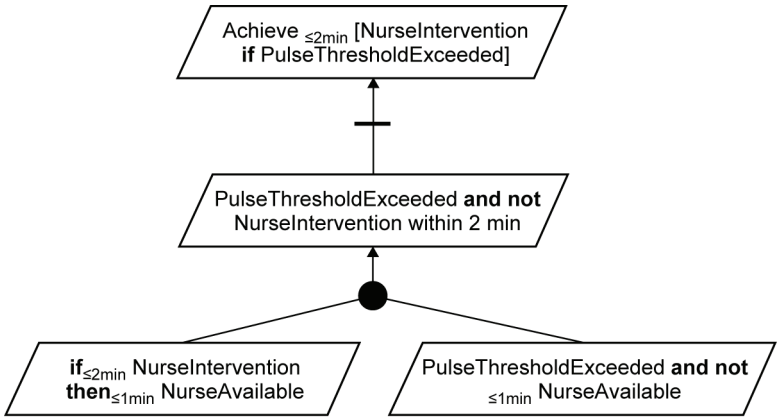
FIGUUR 10.1 Goal OR refinement en obstacle AND refinement

Maar betekent dit nu ook dat O1 **and** O2 een obstakel is voor G? De OR-verfijning van G betekent dat er twee alternatieve systemen denkbaar zijn, één waarin G gerealiseerd is door G1 te realiseren, en één waarin G gerealiseerd is door G2 te realiseren. Bij de obstakelanalyse moeten deze alternatieven afzonderlijk worden onderzocht. Als O1 een obstakel is voor G1, dan is O1 ook een obstakel voor G in het systeem dat ontstaat als G verfijnd wordt tot G1. Het optreden van O2 heeft in dat systeem geen invloed op G (er van uitgaand dat O1 en O2 onafhankelijk zijn, dus dat niet bijvoorbeeld geldt O2 → O1). De conditie O = O1 **and** O2 is weliswaar in beide systemen een obstakel voor G, maar geen *minimaal* obstakel. Voor propagatie naar boven hebben we hier dus niet veel aan.

10.3 a Tautology-based refinement en identifying necessary conditions for the obstructed target; zie pagina 347 en verder van het tekstboek.
b Voor tautology-based refinement geldt dit wel maar bij het toepassen van het obstacle refinement pattern beschreven op pagina 347 en 348 is ook domeinkennis noodzakelijk.

10.4 a Er kan alleen worden ingegrepen door een verpleger als deze beschikbaar is. Een geschikte domeineigenschap is dan bijvoorbeeld
if$_{\leq 2min}$ NurseIntervention
then$_{\leq 1min}$ NurseAvailable

De bijbehorende obstacle refinement is getoond in figuur 10.2.



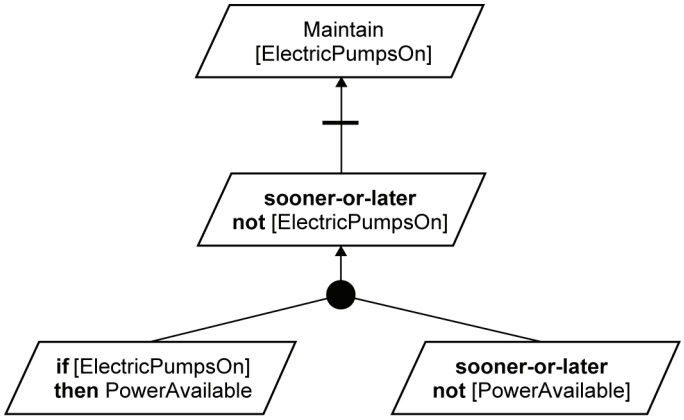
FIGUUR 10.2 Obstacle verfijning van achieve goal met behulp van een domeineigenschap

Door deze verfijning is duidelijk geworden dat snelle beschikbaarheid van een verpleger essentieel is voor het bereiken van het doel.

b Elektrische pompen werken alleen als er elektriciteit beschikbaar is. Een geschikte domeineigenschap is dus

if [ElectricPumpsOn] **then** PowerAvailable

De bijbehorende obstacle refinement is getoond in figuur 10.3



FIGUUR 10.3 Obstacle verfijning van maintain goal met behulp van een domeineigenschap

Hierdoor wordt duidelijk dat de elektriciteitsvoorziening gegarandeerd moet worden (bijvoorbeeld met noodaggregaten).

10.5 a Goal substitution: zoek een andere verfijning van de ouder van G waarin G en O niet meer voorkomen.
b Agent substitution: ken G aan een andere agent toe, resulterend in systeemgedrag waarbij O niet langer kan optreden.
c Obstacle prevention: neem Avoid[O] op als extra doel. Dit is vaak een goede tactiek voor obstakels bij safety en security goals.
d Goal weakening: verzwak het doel dusdanig dat het obstakel verdwijnt. Als het doel de vorm van een 'if c then g'-uitspraak heeft, kan dat door een extra conditie aan c toe te voegen of een or-clausule aan g.
e Obstacle reduction: reduceer de kans dat het obstakel optreedt door een ad hoc tegenmaatregel.
f Goal restoration: als O niet te vermijden is, voeg dan een goal 'if O then sooner-or-later TargetCondition toe'. Dat betekent dat er voor uitzonderingssituaties andere tactieken worden geformuleerd.
g Obstacle mitigation: ook hier wordt O getolereerd maar word teen nieuwe goal toegevoegd die de consequenties verzwakt.

10.6 Informeel: In de startmodus (StartMode) geldt, volgens de domain property, dat de druk dan in orde is (**not** PressureTooLow). In dat geval hoeft er dus ook geen alarm te zijn. De conditie (StartMode **and not** AlarmRaised) is dus geen probleem.

We kunnen het ook formeel beredeneren, als volgt.

O is een obstakel voor een goal G als onder andere geldt dat {O, Dom} ⊨ **not** G**.** In dit voorbeeld zou dus moeten gelden dat

{StartMode **and not** AlarmRaised, **if** StartMode **then not** AlarmRaised} ⊨
not (**if** PressureTooLow **then** AlarmRaised)

Om in te zien dat dit niet klopt, herschrijven we bovenstaande formule eerst als volgt (gebruikmakend van het feit dat (**if** A **then** B) gelijk is aan (**not** A **or** B):

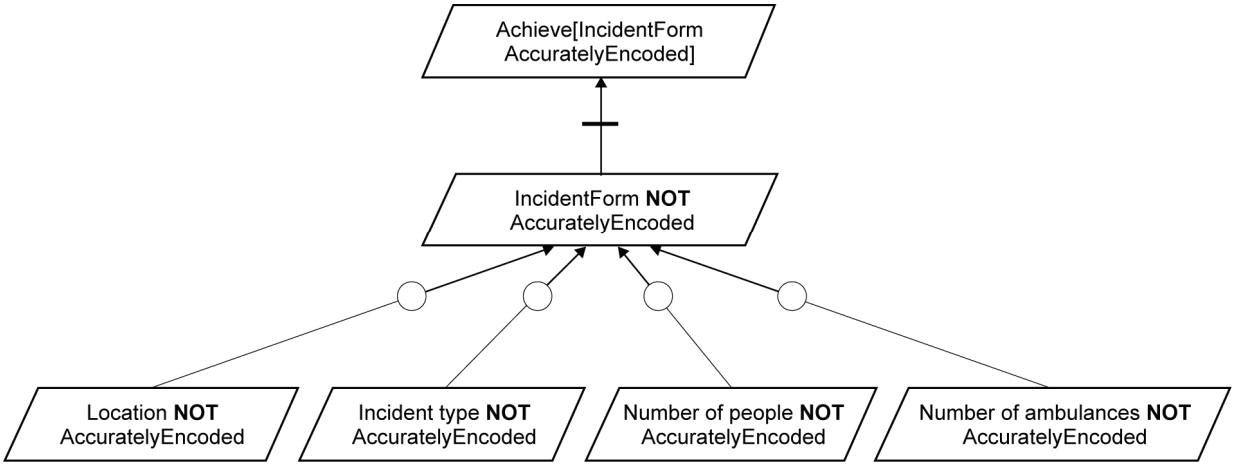
{StartMode **and not** AlarmRaised, **not** StartMode **or not** AlarmRaised} ⊨
not (**not** PressureTooLow **or** AlarmRaised)

en dan, gebruikmakend van De Morgan, tot

{StartMode **and not** AlarmRaised, **not** StartMode **or not** AlarmRaised} ⊨
PressureTooLow **and not** AlarmRaised)

Maar stel nu dat geldt StartMode, **not** AlarmRaised, **not** PressureTooLow. Dan zijn beide condities in de linkerkant waar, maar de conditie rechts niet. Deze formule is dus niet geldig.

10.7 a Figuur 10.4 toont het eerste niveau van de obstacle tree.



FIGUUR 10.4 Deel van obstacle tree voor IncidentFormAccuratelyEncoded

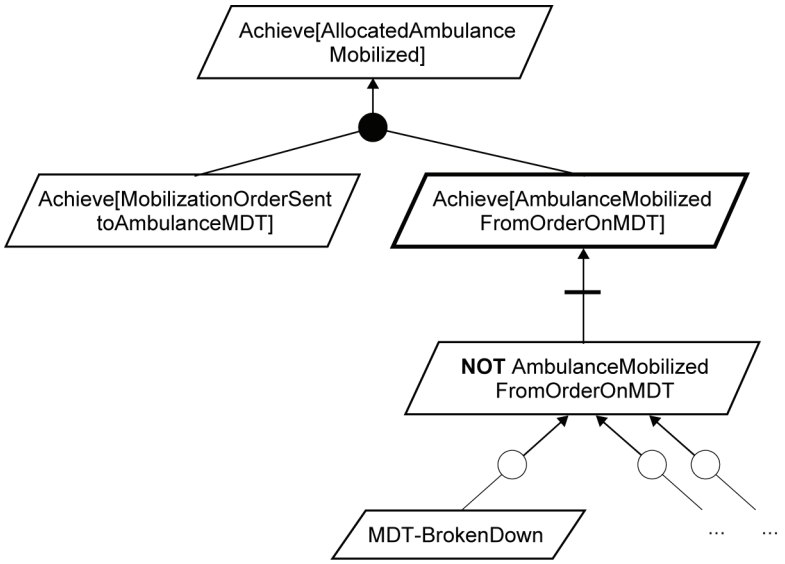
Tabel 10.1 toont een verdere verfijning van de obstakels in figuur 10.4; deze paste niet meer in de figuur. Vanwege de begrijpelijkheid hebben we in plaats van de naam meteen de omschrijving van de obstakels opgenomen. De opsplitsing van de eerste drie obstakels lijkt veel op elkaar; de laatste wijkt af omdat de schatting van het aantal ambulances gemaakt wordt door degenen die het formulier invult en niet door degene die het incident meldt.

TABEL 10.1 Verdere verfijning van de obstakels uit figuur 10.4

Location not accurately encoded	Location not accurately reported Location misheard Error in writing down location
Incident type not accurately encoded	Incident type not clear at moment of reporting Incident type not accurately reported Incident type misheard Error in writing down incident type
Number of people needing assistance not accurately encoded	Number of people not clear at moment of reporting Number of people not accurately reported Number of people misheard Error in writing down number of people
Number of ambulances needed not accurately encoded	Number of ambulances not clear at moment of receiving call Wrong estimate of number of ambulances needed Error in writing down the estimated number

b Alle obstakels uit tabel 10.1 kunnen voorkomen (ze zijn 'feasible'). De kans op communicatiefouten en schrijffouten kan verkleind worden door een nauwkeurig protocol voor degenen die de melding aannemen, bijvoorbeeld
– Probeer de betrouwbaarheid van de melder te verhogen door zelf rustig te blijven en het belang van juiste informatie te benadrukken.
– Herhaal elk gegeven dat de melder noemt en vraag dan om een bevestiging.
– Schrijf de gegevens na die bevestiging meteen op.
– Herhaal aan het eind nogmaals alles wat er is opgeschreven.
Dit valt onder obstacle prevention (zie pagina 351 van het tekstboek). Fouten in de melding kunnen mogelijk worden opgespoord door latere meldingen van hetzelfde incident te vergelijken met de eerdere meldingen en bij afwijkingen maatregelen te nemen. Dit valt onder goal restoration (zie pagina 352 van het tekstboek).

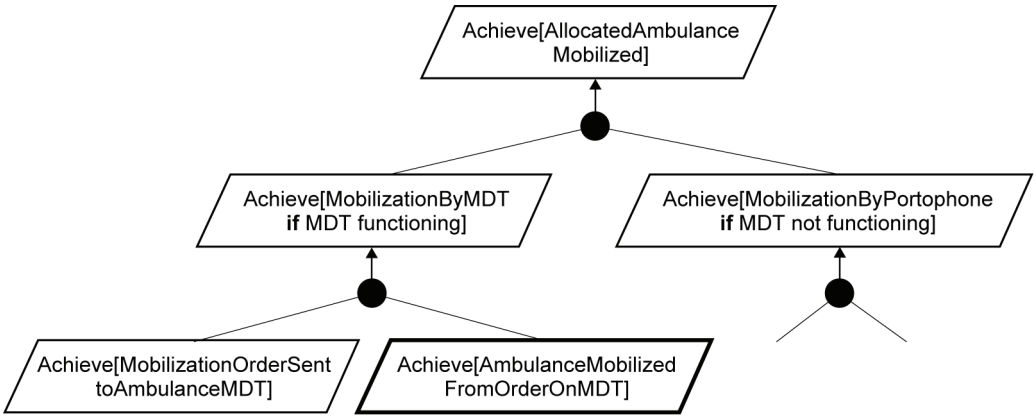
10.8 Figuur 10.5 toont voor de duidelijkheid het deel van de goal tree en het obstakel vermeld in de opgave.



FIGUUR 10.5 Goal tree met obstakel voor AllocatedAmbulanceMobilized

We bekijken een voor een de maatregelen uit paragraaf 9.9.3.
– Goal substitution: er kan een ander en robuuster communicatiekanaal gekozen worden dan de Mobile Data Terminal (bijvoorbeeld een portofoon).
– Agent substitution: lijkt niet van toepassing.
– Obstacle prevention: lijkt niet van toepassing; er is geen denkbare tegenmaatregel die *garandeert* dat de MDT nooit stuk gaat.

– Goal weakening is theoretisch mogelijk; de goal kan bijvoorbeeld vervangen worden door
Achieve[AmbulanceMobilizedFromOrderOnMDT **if** MDT Functioning].
Zonder verdere aanpassingen zou dit echter leiden tot het niet halen van het hoofddoel van het ambulancesysteem: snel een ambulance ter plaatse hebben.
– Obstacle reduction: de kans dat de MDT stuk gaat kan mogelijk gereduceerd worden door een goed preventief onderhoudsprogramma.
– Goal restoration lijkt niet mogelijk (zie ook het volgende punt).
– Obstacle mitigation is daarentegen wel goed mogelijk als er een alternatief communicatiekanaal is, bijvoorbeeld een portofoon. Als de ambulancestaf de oproep via de MDT niet binnen een minuut (of zo) bevestigt, dan worden ze alsnog via de portofoon benaderd. Figuur 10.6 toont de nieuwe goal tree. NB Dit is obstacle mitigation en geen goal restoration omdat er een ouder van AmbulanceMobilizedFromOrderOnMDT hersteld wordt, en niet die goal zelf.
– Niets doen is geen optie want resulteert in het niet halen van het hoofddoel van het ambulancesysteem: snel een ambulance ter plaatse hebben.



FIGUUR 10.6 Goal tree na obstacle mitigation

Er zijn drie redelijke opties
– Zie af van het gebruik van de kwetsbare MDT.
– Verminder de kwetsbaarheid door een onderhoudsprogramma.
– Zet er een ander communicatiesysteem naast als back-up.

Het nadeel van eerste optie is vermoedelijk dat de MDT een formulier uitprint terwijl de communicatie via de portofoon mondeling is en dus eerder tot fouten leidt. De derde optie lijkt dan beter; alleen in geval van een storing wordt teruggevallen op de portofoon. In hoeverre de tweede optie het probleem verhelpt, is niet te beoordelen.

10.9 In de eerste interactie, waarbij de scheduler het vergaderverzoek bevestigt aan de initiator, kan niets mis gaan.

Bij de tweede interactie, het vragen van de constraints aan de deelnemers, kan een exception case optreden als een deelnemer binnen een bepaalde tijd (bijvoorbeeld twee dagen) geen constraints verstuurt. Mogelijke voortzettingen zijn:
– De scheduler vraagt nogmaals om constraints en die worden dan wel op tijd verstuurd. Na de afwijking gaat het oorspronkelijke scenario dan verder.
– De scheduler vraagt nogmaals om constraints maar die worden nu ook niet op tijd gestuurd. De scheduler kan dan bijvoorbeeld de initiator hier van op de hoogte stellen en dan verder gaan met het plannen van de vergadering alsof de betreffende deelnemer altijd kan.
Dit zijn dus twee abnormale scenario's.

Een volgend punt dat mis kan gaan, is het vaststellen van een vergadertijdstip. Mogelijk zijn de constraints dusdanig, dat er geen geschikt tijdstip te vinden is. Wat er dan moet gebeuren, moet overlegd worden met de stakeholders. Enkele alternatieven:
– De initiator wordt in kennis gesteld van het feit dat het plannen van de vergadering mislukt is, inclusief de botsende constraints. Het wordt aan de initiator overgelaten om een nieuw verzoek te formuleren.
– De scheduler suggereert zelf aanpassingen van het verzoek en legt die ter goedkeuring voor aan de initiator (bv. zonder mevrouw Y kan de vergadering wel doorgaan).

A Goal-Oriented Model-Building Method in Action

Leereenheid 11

A Goal-Oriented Model-Building Method in Action

INTRODUCTIE

Bij deze leereenheid horen delen van hoofdstuk 15 in het tekstboek, waarin de KAOS-methode geïllustreerd wordt door het modelleren van een systeem voor veiligheidsbeheersing in een mijn. Vooral dan die delen die over het goalmodel en obstacles gaan is voor ons belangrijk. Paragraaf 1 van deze leereenheid betreft dit deel van het tekstboek.

In paragraaf 2 van deze leereenheid werkt u zelf een casus uit, waarin u een KAOS-model opbouwt voor LAS, de London Ambulance Service System. Deze casus is afkomstig uit het proefschrift van Emmanuel Letier, *Reasoning about Agents in Goal-Oriented Requirements Engineering* (Université Catholique de Louvain, 2001). De opdrachten over deze casus moeten uitgevoerd worden met behulp van Objectiver.

Deze leereenheid bereidt u dus in twee stappen voor op de eind-opdracht. Eerst bestudeert u een gegeven casus, waarbij u en passant ook de verschillende stappen in de KAOS-methode herhaalt. De opgaven bij dit deel zijn gericht op die herhaling. Vervolgens past u de methode zelf toe op een tweede casus, waarbij u uw uitwerkingen steeds kunt vergelijken met een gegeven terugkoppeling.

LEERDOELEN

Na het bestuderen van deze leereenheid wordt verwacht dat u

- de gang van zaken kunt schetsen bij het ontwikkelen van een KAOS-model, dat wil zeggen de onderdelen goal model en obstacles
- zelf met enige sturing een KAOS-model kunt construeren
- bij de constructie van de modellen de tool Objectiver kunt gebruiken, waarbij alle elementen in het model via de properties uitgebreid gedocumenteerd zijn.

Studeeraanwijzingen

Bestudeer in hoofdstuk 15 alleen de stof die hoort bij die onderdelen van het KAOS-model die we behandeld hebben in de cursus.
Bij het bestuderen van deze leereenheid dient u de tool Objectiver te gebruiken. Informatie over de tool vindt u in de introductieleereenheid en op Studienet.

Ook de uitwerking in Objectiver van de casus uit paragraaf 2 staat op Studienet, onder de naam LAS.ob

De studielast van deze leereenheid bedraagt circa 6 uur.

LEERKERN

1 Aanvullingen en opgaven bij hoofdstuk 15 uit het tekstboek

Studeeraanwijzing

Bestudeer hoofdstuk 15 uit het tekstboek, alleen die stukken die gaan over het goal model en obstacles. In deze paragrafen worden alle door ons behandelde stappen van het opstellen van een model doorlopen. Dit deel fungeert daardoor ook als een herhaling van de stappen in de KAOS-methode.

In de casusbeschrijving Mine safety control op dezelfde pagina is sprake van een *sump*. Dit is een opvangbassin; een laag gelegen plek waar vloeistoffen zich kunnen verzamelen.

Belangrijke opmerking

Let goed op de redelijk systematische manier waarop in deze casus modellen worden opgebouwd. Het is verleidelijk om bij het opstellen van de verschillende modellen louter intuïtief te werk te gaan en alle modellen los van elkaar te beschouwen; dit leidt echter tot deelmodellen zonder samenhang.

OPGAVE 11.1

a Van welk type zijn de bladeren WaterPumpedOut if PumpOn, SuffcientPumpCapacity en NoExcessiveWaterFlow in figure 15.4 op pagina 505 van het tekstboek?
b Zijn ze alle drie nodig? Licht dit toe en stel desgewenst een alternatief voor.
c Er wordt opgemerkt dat deze hypothesen voortkomen uit ‘satisfaction arguments’. Wat wordt hier in dit geval precies mee bedoeld?

OPGAVE 11.2

In figure 15.6 op pagina 508 van het tekstboek is de goal PumpOn**If**HighWaterDetected opgedeeld in twee subdoelen. Leg uit waarom dat nodig is en ook waarom het in figure 15.4 niet nodig was.

OPGAVE 11.3

Welke van de goals uit het lijstje bovenaan pagina 510 van het tekstboek zijn accuracy goals?

OPGAVE 11.4

Wat is strong mitigation en waarom voldoet de oplossing uit figure 15.9 op pagina 513 van het tekstboek daaraan?

2 De LAS-casus

In deze paragraaf wordt een deel van een casus uitgewerkt over het London Ambulance System. De casusbeschrijving vindt u in de bijlage . Deze casus biedt tevens oefenstof voor de uitwerking van de eindopdracht. U wordt geacht om bij het uitwerken van deze casus (net als bij de eindopdracht), de tool Objectiver te gebruiken. Op Studienet vindt u een document met aanwijzingen voor het gebruik van Objectiver.

Studeeraanwijzing

Op Studienet vindt u het Objectiver-project LAS.ob, dat uitwerkingen bevat van alle opdrachten. Er staan meer diagrammen dan we nodig hebben. Belangrijk zijn natuurlijk de modellen met betrekking tot goals en obstacles. De diagrammen zijn in de terugkoppeling niet opnieuw getekend; u moet als onderdeel van de terugkoppeling dus dit project raadplegen. In dit werkboek geven we alleen wat commentaar op deze uitwerking.

Niet alleen de casus zelf maar ook een deel van de uitwerkingen van de opdrachten zijn ontleend aan het proefschrift van Emmanuel Letier, al hebben we soms voor andere verfijningen gekozen en hebben we de system-to-be gemoderniseerd. Wij houden in de uitwerkingen het Engels uit dit proefschrift aan. Uiteraard zijn onze uitwerkingen niet de enige mogelijke.

Net als bij het systeem voor de mijn uit paragraaf 1, wordt begonnen met de modellering van het bestaande systeem. Als eerste goal voor het systeem kunt u gebruiken: Achieve[AmbulanceIntervention].

OPDRACHT 11.5

In deze opdracht gaat u in Objectiver een eerste (preliminary) goal diagram opstellen. Vul daarbij steeds ook de properties van alle goals in. Geef elke goal het juiste type (goal, soft goal, requirement of expectation) en definieer agents voor eventuele requirements en expectations. Laat ook bij elke verfijningstap zien welk pattern u heeft gebruikt. HOW en WHY zijn ook mogelijke patterns.

a Lees de casusbeschrijving in de bijlage.

b Identificeer een parent goal (stel WHY-vraag).

c Verfijn deze parent goal door het stellen van HOW-vragen (de oorspronkelijke goal wordt dan één subgoal van deze parent).
*Aanwijzing: g*ebruik het uncontrollability pattern in combinatie met het milestone pattern.

d Verfijn ook de oorspronkelijke goal nog één stap verder, waarbij aan nog een ORCON-standaard moet worden voldaan.
Aanwijzing: De goal tree is hiermee nog niet af; niet alle kinderen zijn dus requirements of expectations.

e Ken environment agents toe aan expectations.

Het is interessant om op te merken dat Letier in zijn diagrammen soms doelen opneemt die enigszins onrealistisch zijn; dit komt dan later weer goed wanneer, bij het analyseren van de obstakels nieuwe subdoelen worden toegevoegd. Het eerste goal diagram heeft dus dezelfde functie als het hoofdsuccesscenario bij use case analyse: in eerste instantie gaan we ervan uit dat alles normaal verloopt en pas later worden uitzonderingen geanalyseerd. Het is wel van belang om het feit dat een doel niet realistisch is, meteen te documenteren.

De volgende stap wordt het verfijnen van de goal Achieve[AmbulanceMobilization]. De verfijning van zo'n goal kan op verschillende manieren aangepakt worden. Eén mogelijkheid is het stellen van HOW-vragen en die op grond van de casusbeschrijving beantwoorden. Letier pakt het iets anders aan: hij gaat uit van de agents en hun capabilities. Hij benoemt daartoe als nieuwe agent CACagent, een verzamelnaam voor alle medewerkers van de Central Ambulance

Control room, die later nog verfijnd kan worden. Vervolgens constateert hij dat de goal Achieve[AmbulanceMobilization] om (onder meer) de volgende redenen onrealiseerbaar is voor de CAC agents:
– CAC agents kunnen de attributen location van Incident en Ambulance niet monitoren
– Bovendien kan het doel helemaal niet gerealiseerd worden als er geen ambulance beschikbaar is die binnen 11 minuten op de plaats van het incident kan zijn.

OPDRACHT 11.6
a Splits de goal Achieve[AmbulanceMobilization] in subgoals door het stellen van HOW-vragen naar aanleiding van de casusbeschrijving.
b Verfijn deze subgoals zo nodig verder om de problemen met agent capabilities aan te pakken.

Om de casus niet te groot te maken, verfijnen we alleen de goals Maintain [AccurateAmbulanceAvailabilityInfo] en Maintain [AccurateAmbulanceLocationInfo] verder. We gaan daarbij uit van een system-to-be waarin dit vergaand geautomatiseerd is.

Anders dan in 1992, kan nu gebruik gemaakt worden van (openbare) mobiele netwerken en van GPS. Alle ambulances worden uitgerust met een GPS-systeem, dat via een mobiel netwerk informatie verstuurt naar een softwaresysteem AVLS (Automatic Vehicle Location System), dat op een centrale server draait. Dit systeem gaat zorgen voor het bijhouden van die informatie. De ambulancestaf heeft hier geen rol in.

Een voertuig is niet altijd zichtbaar voor GPS; in dat geval kan de positie echter wel berekend worden door de software. Als de ambulance een basis binnenrijdt en daar blijft, is de laatste positie die van de basis; als de ambulance door een tunnel rijdt, kan de positie bij benadering berekend worden. Neem één goal op voor het geval het voertuig onzichtbaar is, en ken die toe aan het AVLS. U hoeft die goal hier niet verder te verfijnen (gebruik in Objectiver de *assignment* in plaats van de *responsibility* link).

Bij het bepalen van de status van een ambulance heeft de staf wel een functie: wanneer die status verandert, moeten zij dat kenbaar maken. Ook deze informatie kan via het mobiele netwerk naar het AVLS worden verstuurd.

OPDRACHT 11.7

Ontwerp volgens de aanwijzingen hierboven een verfijning voor de goals Maintain [AccurateAmbulanceAvailabilityInfo] en Maintain [AccurateAmbulancePositionInfo]. Ken agents toe aan requirements en expectations.

OPGAVE 11.8

De goal tree uit opdracht 11.6 bevat een goal Maintain[AmbulanceAvailability]. Kunt u, zonder een feitelijke obstacle analyse uit te voeren, voorspellen wat de uiteindelijke tegenmaatregel zal zijn (zie tekstboek pagina 350 en verder voor de mogelijkheden).

OPDRACHT 11.9

Doe een obstacle analyse voor de expectation Maintain[AccurateAvailabilityInfoSignaled]. Als u nieuwe goals toevoegt, hoeven die niet verfijnd te worden. Geef ook aan welke typen tegenmaatregelen u heeft gebruikt.

TERUGKOPPELING

1 Uitwerking van de opgaven

11.1 a Op grond van de vorm zijn dit domain hypotheses of domain properties (zie ook pagina 298 en 299 van het tekstboek). Van deze drie zijn de laatste twee hypotheses maar is de eerste (WaterPumpedOut if PumpOn) een property: een beschrijving van de werking van de pomp. (Met PumpOn wordt bedoeld dat de pomp zijn werk doet en niet dat er op het knopje 'aan' is gedrukt; dit wordt overigens pas echt duidelijk bij de obstacle analysis in paragraaf 15.2).
b De hypothesen NoExcessiveWaterFlow en SufficientPumpCapacity zijn min of meer complementair; we kunnen ons voorstellen dat we die samen zouden nemen in WaterFlowDoesntExceedPumpCapacity. En daarvan kunnen we ons afvragen of het een domeinhypothese is, of een goal. Als het als goal benoemd wordt, dan valt het berekenen van een veilige pompcapaciteit binnen het systeem.
c Het doel Maintain[SumpPumpedOut**If**HighWater] wordt in eerste instantie gereduceerd to het subdoel PumpOn**If**HighWater. Maar dit is onvoldoende om te bewijzen dat het bassin dan ook echt wordt leeggepompt: daarvoor zijn de extra hypothesen nodig. Als u zo'n model opstelt, moet u zich dus steeds afvragen of de subdoelen wel voldoende zijn om het oorspronkelijke doel te bewijzen en zo niet, wat u moet toevoegen.

11.2 Op pagina 326 in hoofdstuk 8 van het tekstboek wordt het uncontrollability-patroon uitgelegd. De agent Safety controller bestuurt niet de pomp zelf, maar alleen de aan/uit-knop; de conditie PumpOn kan door deze agent daarom niet gegarandeerd worden. De uncontrollable conditie PumpOn wordt daarom vervangen door de controllable conditie PumpSwitchOn en er wordt een subdoel PumpOn **iff** PumpSwitchOn toegevoegd. (Misschien vraagt u zich nu meteen af of dat wel klopt: gaat de pomp wel altijd aan als er op de knop wordt gedrukt? Dat is een terechte vraag, zoals uit de obstacle analysis in paragraaf 15.2.3 zal blijken).
In figure 15.4 was deze opdeling niet nodig omdat een menselijke operator zowel de knop in kan drukken als controleren of de pomp dan ook aanslaat. De goal PumpOn**If**HighWaterDetected is in die context een goede requirement, want realiseerbaar door één agent.

11.3 Op pagina 270 van het tekstboek staat wat accuracy goals zijn: 'non-functional goals requiring the state of variables controlled by the software to reflect accurately the state of corresponding quantities controlled by environment agents'. In dit lijstje staan dus twee accuracy goals, namelijk Maintain[PumpOn**Iff**SwitchOn] en Maintain[PumpOff**Iff**SwitchOff].

11.4 Zie pagina 352-353 van het tekstboek. Obstacle mitigation betekent het dulden van een obstacle terwijl tegelijkertijd de consequenties worden verzacht (weak mitigation) of volledig weggenomen (strong mitigation). In het laatste geval wordt een goal toegevoegd die garandeert dat bij het optreden van de obstacle toch aan een goal hoger in de boom kan worden voldaan. In dit geval gaat het om het toevoegen van een extra subgoal (MineEvacuated**If**CriticalWater) helemaal boven in de boom.

11.5 De uitwerking van deze opdracht in Objectiver vindt u als onderdeel van het Objectiver-project LAS.ob, in de package Opdracht11.5. Kijk niet alleen naar de goal graph zelf, maar ook naar de properties van alle doelen. We geven in dit werkboek nog wat commentaar op de uitwerking.

a Geen terugkoppeling.

b We willen interventie van een ambulance omdat er een ongeluk is gebeurd. Letier formuleert hiervoor de parent goal Achieve[IncidentResolved] en tekent hierbij aan dat in het midden wordt gelaten wanneer een incident precies is opgelost. Het betekent in elk geval niet dat er geen levens verloren mogen gaan (helaas sterven mensen soms voor de ambulance ter plaatse is of in de ambulance op weg naar het ziekenhuis; eisen dat dit niet gebeurt is onrealistisch).

c Bij de verfijning van deze goal zijn zowel het uncontrollability als het milestone-driven refinement pattern gebruikt. Het systeem kan de afhandeling van een incident niet garanderen. Het moet daarvoor eerst gemeld worden (milestone) en de interventie door de ambulance (milestone) moet het incident ook daadwerkelijk oplossen. Beide zijn expectations: er moet aan voldaan worden door de omgeving van het systeem. Wat 'oplossen' precies betekent, is expres vaag gehouden (zie onder b).

d Ook het bereiken van de goal Achieve[AmbulanceIntervention] kan door het systeem niet gegarandeerd worden. Deze wordt daarom verder verfijnd volgens hetzelfde patroon als onder c. Letier formuleert als aanname dat een incident afgehandeld wordt door één ambulance maar tekent daar meteen bij aan dat die verwachting feitelijk niet realistisch is. Dit wordt later via obstacle analysis opgelost.
Het subdoel Achieve[AmbulanceMobilization] is geen requirement maar een gewone goal.

e Het melden van het incident gebeurt door het publiek; de feitelijke interventie door de ambulancemedewerkers.

11.6 De uitwerking is bevat in de package Opdracht 11.6 van LAS.ob. Denk eraan dat de wortel van deze boom (Achieve[AmbulanceMobilization]) al deel uitmaakt van een ander diagram; deze moet niet opnieuw gedefinieerd worden maar aan het diagram worden toegevoegd als bestaand concept. We geven een toelichting bij deze uitwerking.

a Als we, kijkend naar de casusbeschrijving, HOW-vragen stellen, dan zien we drie stappen: call taking, resource identification en resource mobilization. Het ligt voor de hand om voor elk een subdoel te benoemen.

– Een goal als Achieve[CallTaken] zegt heel weinig; het gaat er ook om dat de details kloppen (in eerste instantie alleen de locatie; zie de casusbeschrijving). We benoemen de goal daarom als Achieve[AccurateIncidentInfo] en houden in gedachten dat we een informatieobject nodig hebben voor het incident. We nemen deze goal op als expectation onder de verantwoordelijkheid van een CACagent. Bij een volledige uitwerking moet hier verder naar gekeken worden; een fout bij het noteren of invoeren van de details is snel gemaakt en het systeem moet daar een procedure voor hebben (richtlijnen voor de telefonisten om altijd te controleren of ze de locatie goed hebben verstaan; het opnemen van alle gesprekken zodat ze later nog ene keer beluisterd kunnen worden; noteren van telefoonnummer van de beller…). We houden dat buiten deze beperkte casus.

– De tweede subgoal is dan Achieve[AmbulanceAllocation].
– De derde subgoal is Achieve[MobilizationOfAllocatedAmbulance].
b Zijn er verdere verfijningen nodig op grond van de unrealizability problemen?
– De introductie van het doel Achieve[AccurateIncidentInfo] biedt meteen een oplossing voor het feit dat de CACagent de locatie van het incident niet kan monitoren: deze goal is ook een accuracy goal die stelt dat de details van het incident overeenkomen met hetgeen wordt opgenomen in het informatieobject. Dit object kan wel worden gemonitord.
– Met de tweede subgoal zijn twee problemen. Om een ambulance te kunnen alloceren is de locatie nodig, maar die locatie zelf kan door de CACagents niet worden gemonitord. Ook hier is dus een informatieobject en een accuracy goal nodig: AmbulanceInfo en Maintain[AccurateAmbulanceInfo]. Verder is deze subgoal onrealiseerbaar als er geen ambulance beschikbaar is die op tijd bij het incident kan zijn. We splitsen deze daarom in twee nieuwe subgoals: één die voortdurende beschikbaarheid moet garanderen en één die een ambulance alloceert als er één beschikbaar is. Bij de latere obstacle analyse kan dan gekeken worden naar de obstakels voor de eerste subgoal.
We splitsen meteen de goal Maintain[AccurateAmbulanceInfo] nog verder op in twee subgoals voor elk van de attributen: Maintain[AccurateAmbulanceAvailabilityInfo] en Maintain[AccurateAmbulancePositionInfo].

11.7 Een mogelijke uitwerking van deze opdracht staat in de package Opdracht 11.7 van LAS.ob.
Merk op, dat we de signaling device voor het versturen van de availibility information en de GPS device aanmerken als environment agents. Dat betekent dat we ervan uitgaan, dat dergelijke devices kant en klaar op de markt zijn en dat die niet in het kader van dit project ontwikkeld hoeven te worden. Voor de GPS device (een apparaat dat op gezette tijden informatie over de locatie van een voertuig naar een server stuurt) is deze aanname zonder meer juist; voor de device die de availability signals moet versturen is dat iets minder duidelijk. We laten dit verder zitten.

11.8 Volgens de specificaties moet er altijd een ambulance beschikbaar zijn die binnen 11 minuten na mobilisatie op de plaats van het incident kan zijn.
Ten eerste zijn er omstandigheden denkbaar waaronder dat niet mogelijk is, bijvoorbeeld als er bij rellen zoveel straten zijn gebarricadeerd dat de locatie van het incident heel slecht te bereiken is.

Ten tweede zou het kunnen dat in alle gevallen vasthouden aan deze eis betekent dat er een financieel onhaalbare overcapaciteit aan ambulances nodig is. Je moet dan immers rekening houden met de mogelijkheid dat er toevallig binnen een uur tien ongelukken binnen een cirkel van een kilometer plaatsvinden, zelfs al komt dat in de praktijk nooit voor.

Uiteindelijk zal dit uitdraaien op goal weakening, bijvoorbeeld door de goal te herformuleren als For every location, there should in 95% of the cases be an available ambulance able to arrive at that location within 11 minutes. Zorgvuldige statistische analyse kan dan laten zien hoeveel ambulances daarvoor per basis nodig zijn.

11.9 De obstacle analysis begint altijd met het ontkennen van de goal; in dit geval levert dat dus een obstacle Not AccurateAvailabilityInfoSignaled. Dit obstacle wordt vervolgens verfijnd. Wij bedachten drie mogelijke oorzaken: het ambulance personeel vergeet signalen te sturen, het ambulancepersoneel stuurt een verkeerd signaal en het ambulancepersoneel kan niets sturen omdat de signaling device niet goed werkt.
Voor alle drie kunnen we een tegenmaatregel bedenken die neerkomt op obstacle prevention: het personeel kan regelmatig herinnerd worden aan het feit dat ze de signalen moeten versturen; om fouten te voorkomen kan een bevestiging voor elk signaal gevraagd worden ('De ambulance is nu beschikbaar. Raak 'yes' aan om dit te bevestigen of 'no' als u de waarde wilt wijzigen') en in het geval van storingen kan overgeschakeld worden op een backup.
Deze tegenmaatregelen overziend, kunnen we ons echter afvragen of dit nog goed functioneert: wordt het niet heel irritant voor het personeel dat ze die signalen steeds moeten geven (en dan ook nog moeten bevestigen)? Terwijl, bij nader inzien, een groot deel van die informatie al beschikbaar is. Het softwaresysteem weet wanneer een ambulance gemobiliseerd is en uit de GPS-gegevens kan worden bepaald wanneer een gemobiliseerde ambulance bij het ongeluk is. Alleen het einde van de interventie moet door het personeel worden aangegeven (als de ambulance de locatie van het incident verlaat, dan kan dat het einde zijn maar die kan ook op weg zijn naar het ziekenhuis). We introdcueren daarom hoger in de boom een nieuw doel Maintain[DerivedAvailabilityInfo]. Dit is goal substitution.

Het diagram vindt u in de package Opdracht 11.9 van LAS.ob.

BIJLAGE

Bron: Emmanuel Letier, *Reasoning about Agents in Goal-Oriented Requirements Engineering*, Thèse presentée en vue de l'obtention du grade de Docteur en Sciences Appliquées, Université Catholique de Louvain, 2001

The London Ambulance Service (LAS) has two main functions: responding to emergency calls requiring the rapid intervention of an ambulance, and dealing with nonurgent patient journeys. The case study is only concerned with the handling of urgent calls.

The UK Government imposes performance standards (called ORCON) for accident and emergency calls upon ambulance services. In 1992, when the first automated system for LAS was put into use, the performance standard for the LAS was:

An ambulance must arrive at the scene within 14 minutes for 95% of the calls.

It was the difficulty of meeting that standard that motivated the need for a new system.

Our elaboration of the goal model for the LAS is mostly based on a section of the Inquiry Report that describes the rationale for a Computer Aided Despatch (CAD) system.

Excerpts from that section are reproduced below:

In order to understand the rationale behind the development of the CAD system it is essential to understand the manual system that it would replace and its shortcomings.

The Manual system operates as follows:

Call Taking
When a 999 or urgent call is received in the Central Ambulance Control (CAC) room, the Control Assistant (CA) writes down the call details on a pre-printed form. The incident location is identified from a map book, together with the map reference co-ordinates. On completion of the call the incident form is placed into a conveyor belt system with other forms from fellow CA's. The conveyor belt then transports the forms to a central collection point within CAC.

Resource Identification
Another CAC staff member collects the form from the central collection point and, through reviewing the details on the form, decides which resource allocator should deal with it (based on the three London Division - North East, North West, and South). At this point, potential duplicate calls are also identified.
The resource allocator then examines the form for his/her sector and, using status and location information provided through the radio operator and noted on forms maintained in the "activation box" for each vehicle, decides which resource should be mobilized. This resource is then also recorded on the form which is passed to a despatcher.

Resource Mobilisation
The despatcher will telephone the relevant ambulance station (if that is where the resource is) or will pass mobilisation instructions to the radio operator if the ambulance is already away from the station.
According to the ORCON standards this whole process should take no more than 3 minutes.

There are some clear deficiencies with a totally automated manual system including:
a identification of the precise location can be time consuming due to often incomplete or inaccurate details from the caller and the consequent need to explore a number of alternatives through the map books;
b the physical movement of paper forms around the Control Room is inefficient;
c maintaining up to date vehicle status and location from allocators' intuition and reports from ambulances as relayed to and through the radio operators is a slow and laborious process;
d communicating with ambulances via voice is time consuming and, at peak times, can lead to mobilization queues;
e identifying duplicate calls relies on human judgment and memory. This is error prone;
f dealing with call backs is a labour intensive process as it often involves CA's leaving their posts to talk to the allocators;
g identification of special incidents needing a Rapid Response Unit or the helicopter (or a major incident team) relies totally on human judgment.

A computer aided despatch system is intended to overcome most of these deficiencies through such features as:
a a computer based gazetteer with public telephone box identification;
b elimination of the need to move paper around the control room;
c timely and (in the case of location information) automated update of resource availability information;
d computer based intelligence to help identify duplicates and major incidents;
e direct mobilization to the ambulance on the completion of the call thus potentially, in simple cases, achieving mobilization inside one minute.