Informatiesysteem ondersteunt ontwerpproces

Systeemontwerp en requirementsbeheer

Veel van de huidige ingebedde toepassingen zijn zo complex dat ze slechts te ontwikkelen zijn met behulp van een systematische ontwerpmethode. Ook het testen van deze applicaties is zo moeilijk geworden dat hiervoor een speciale strategie nodig is waarmee al in het ontwerpproces rekening moet worden gehouden. Systemen worden niet meer ontwikkeld door een klein team op een vaste locatie, maar door soms honderden ontwerpers die zich zelfs in verschillende landen kunnen bevinden. Bovendien eisen de gebruikers van dergelijke systemen steeds vaker dat het ontwerp wordt uitgevoerd en gedocumenteerd conform een gestandaardiseerd kwaliteitsmodel. Het wordt daarom steeds belangrijker om de eisen waar het ingebedde systeem aan moet voldoen, de requirements, in elke fase van het ontwerp naar voren te kunnen halen. Vooral bij ontwerpers van telecommunicatiesystemen is dan ook belangstelling ontstaan voor requirementsbeheer en bijbehorende tools.

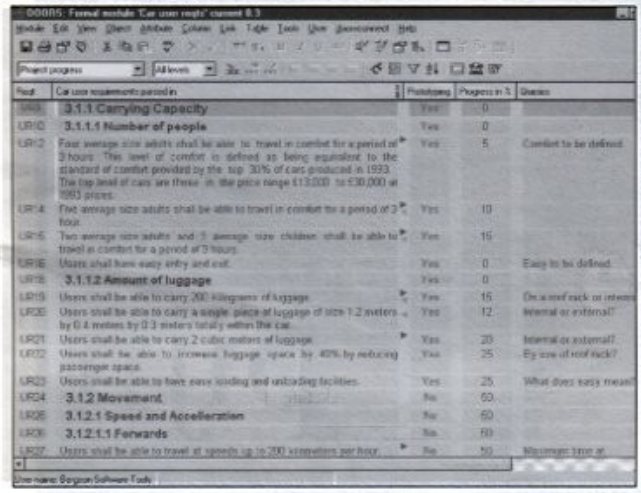
Hans van Thiel

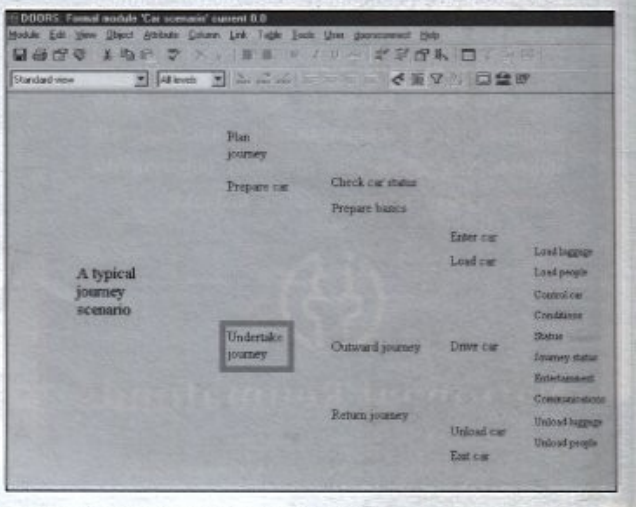
Voor relatief eenvoudige ingebedde systemen die zijn gebaseerd op een gering aantal OTS (off the shelf) componenten zal in het algemeen geen uitgebreid ontwerpmodel nodig zijn. De taken die het systeem moet uitvoeren zijn overzichtelijk, evenals de randvoorwaarden die de werkomgeving met zich meebrengt. De traditionele manier is om op basis van deze eisen de meest geschikte hardware te selecteren en vervolgens met de daarvoor beschikbare tools en ontwikkelomgevingen de software te schrijven en te testen. Eenvoudige systemen kunnen in de praktijk natuurlijk onaangenaam complex blijken, maar de moeilijkheid ligt meestal bij de interactie met een externe omgeving en de real-time voorwaarden. De eerste slag van het ontwerp, het vaststellen van wat het systeem moet gaan doen, hoeft meestal weinig bijgesteld te worden en een eenmaal geschreven document met deze eisen (requirementsdocument) blijft het globale referentiepunt van de ontwerper(s). De relatie tussen de kenmerken van systeem, deelsystemen en componenten en de uiteindelijke werking van het geheel is voor eenvoudige ingebedde toepassingen ook redelijk te overzien.

Het hergebruik van gedeelten van het ontwerp vormt geen belangrijke factor. De hardware is standaard en een vernieuwde versie van het ingebedde systeem zal min of meer vanzelfsprekend de nieuwe versies van dezelfde OTS componenten benutten. Hergebruik van software vindt slechts plaats op ad hoc basis. Alleen de ervaring van de ontwerper(s) met het eerste systeem zal een belangrijke rol spelen bij de bouw van verbeterde versies maar die ervaring is impliciet, persoonsgebonden en niet of nauwelijks traceerbaar.

Complexiteit

De situatie is bij de huidige generatie ingebedde toepassingen totaal verschillend. Niet alleen de software maar ook de hardware zal vaak toepassingsspecifiek zijn, wat inhoudt dat hardwareontwerp deel uitmaakt van het totale ontwerp. Bovendien zijn ingebedde systemen dikwijls zeer heterogeen, met componenten van verschillende soort. Integratie op IC-niveau dient onder meer om de complexiteit terug te dringen, maar toch zal het aantal componenten zo groot blijven dat het overzien van de onderlinge relaties een probleem is. Omdat het aantal interacties veel groter is dan bij eenvoudige toepassingen is ook het aantal mogelijke foutbronnen weer groter. De ingebedde software zal meer functionaliteit moeten bieden en omvangrijker, diverser en daarmee in principe onoverzichtelijker zijn. Al deze fac-





Twee schermen van een demonstratieprogramma voor requirementsbeheer. Hier worden de eisen gedocumenteerd voor een autorit.

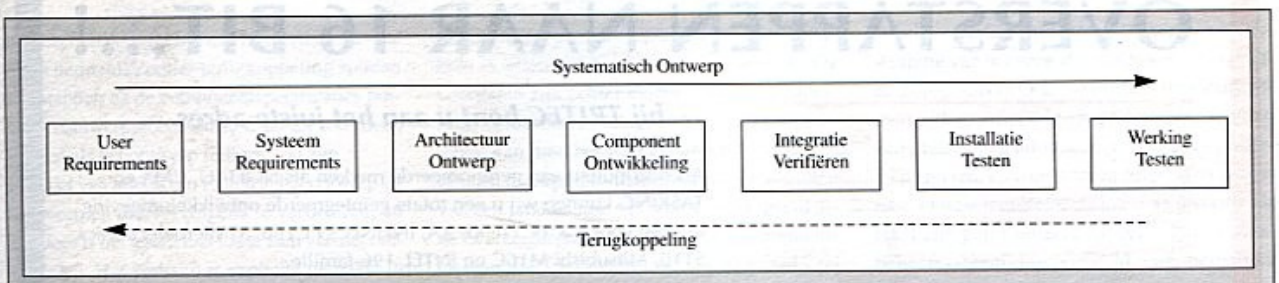


Fig. 1. Het basis lifecycle model (naar Stevens et al.).

toren vergroten niet alleen de kans op fouten maar bemoeilijken ook het testen op die fouten. Verbeteringen aan een zeer ingewikkeld product zullen niet sprongsgewijs en totaal maar juist geleidelijk, incrementeel, aangebracht worden. Dit betekent dat bestaande hardware en software dit proces moeten ondersteunen. Bovendien zal door de omvang van het ontwerp (en de kosten) de behoefte aan expliciet hergebruik in andere toepassingen alleen maar toenemen. Zeer complexe applicaties kunnen ook niet meer door een klein team ontwikkeld worden. Gedistribueerd ontwerpen, door diverse groepen op diverse locaties, vergt gestructureerde procedures met bijbehorende documentatie.

Bij een toenemende complexiteit van zowel het product als de ontwikkeling van het product moet het ontwerpproces expliciet, persoonsonafhankelijk en traceerbaar worden. Het concept van een systematische levenscyclus (lifecycle) kan dit ideaal enigszins binnen bereik brengen.

Systemen

Een systeem is een verzameling van elkaar verschillende onderdelen die door op elkaar en de omgeving in te werken een functie vervullen (de synergie) die ze elk apart niet vervullen. De onderdelen kunnen zelf ook weer systemen zijn, waarbij de synergie van zo'n subsysteem een interactie is met een ander subsysteem of met de omgeving van het systeem. De karakterisering (modellering) van een systeem in subsystemen met hun onderlinge relaties is de systeemarchitectuur. Het doel van het ontwerpproces is eigenlijk om de best mogelijke systeemarchitectuur te bepalen met de beste componenten.

Door de abstractie van het concept systeem is het geenszins triviaal om vast te stellen wat nu een systeem is en welke architectuur daar dan bij hoort. Een kilo aardappelen, bijvoorbeeld, is geen systeem, omdat de verzameling aardappelen geen andere functie heeft dan elke aardappel op zich. Een zak aardappelen daarentegen is wel een systeem omdat de aardappelen en de verpakking samen een vervoerbare eenheid vormen. Er is synergie. Vanuit een ander standpunt is een aardappel echter zelf ook een systeem omdat de schil een andere functie heeft dan de aardappel zelf.

Dat systemen voor velerlei uitleg vatbaar zijn biedt ook wel voordelen. Als het mogelijk blijkt om met systems-engineering verschillende systeemarchitecturen op te stellen en te vergelijken kunnen de duurste en meest ingrijpende ontwerpbeslissingen tot het laatst mogelijke moment worden uitgesteld. Een systeemarchitectuur van een ontwerp is tenslotte maar een beschrijving in een document of databank.

Lifecycle

Een ontwerpproces heeft een begin en een eind met daartussenin verschillende fasen. Omdat er altijd terugkoppeling plaats heeft spreekt men ook bij het eenvoudigste, lineaire model, van een ontwikkelingslevenscyclus (development lifecycle). Dit basismodel staat aangegeven in figuur 1.

Voorafgaand aan het ontwerp van de architectuur worden twee fasen onderscheiden, het bepalen van de gebruikerseisen (user-requirements) en daaropvolgend de systeemeisen (system-requirements). De user-requirements stellen wat de gebruiker van het system verwacht; de system-requirements zijn de eisen waaraan het systeem moet voldoen om op zijn beurt aan de user-requirements te kunnen voldoen. De user-requirements vormen met elkaar een operationeel model van het systeem, de system-requirements een functioneel model. De laatste worden daarom ook wel functionele specificaties genoemd.

De user-requirements voor een zak aardappelen zullen datgene zijn wat de boer, de winkelier, de transporteur, de consument er van verwacht. De system-requirements zijn de kenmerken die dat mogelijk maken. De eis dat de aardappelen een week houdbaar moeten blijven is een user-requirement, de eis dat er een bepaalde hoeveelheid lucht bij de aardappelen moet kunnen is een systeem-requirement. De user-requirement bepaalt de systeem-requirement en in het algemeen zijn er meer mogelijkheden om aan de eerste te voldoen. Soms moet echter een user-requirement worden aangepast omdat er geen systeemeisen te vinden zijn waar aan kan worden voldaan (terugkoppeling). Het kan blijken (en het zál meestal ook blijken) dat een keuze moet worden gemaakt tussen requirements. Als alle verbanden tussen requirements expliciet zijn, gaat dat natuurlijk makkelijker dan wanneer er geen structuur is aangegeven. Traceerbaarheid van requirements is dus een belangrijk hulpmiddel bij het systematisch ontwerpen.

Het architectuurontwerp vloeit voort uit de systeemeisen en de systeemarchitectuur bevat de componentspecificaties die de individuele componenten vastleggen. Die bouwstenen moeten vervolgens worden geproduceerd, getest, dan samengevoegd tot subsystemen en in hun integratie getest. Daarna moet het gehele systeem worden getest en tenslotte wordt gecontrolerd of

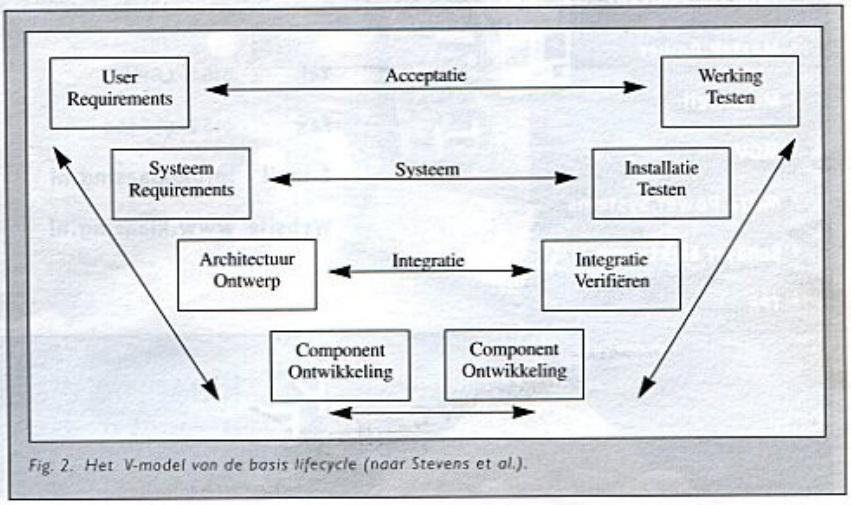


Fig. 2. Het V-model van de basis lifecycle (naar Stevens et al.).

het geheel ook werkelijk kan uitvoeren waarvoor het is bedoeld. Zonder terugkoppeling zouden alle stappen na de componentspecificaties buiten het ontwerpproces vallen maar dit zou alleen kunnen als het ontwerp feilloos zou zijn.
Het doel van tests is om fouten in het ontwerp op te sporen en die vervolgens te verbeteren, en daardoor is het testen niet los te zien van het ontwikkelen. Het verband is expliciet gemaakt in figuur 2, waarin de lifecycle is weergegeven in een V - model (lagenmodel). De acceptatietests, waaraan het systeem moet voldoen om door de afnemer te worden aanvaard, zijn verbonden met de user-requirements. Systeemtests controleren of het systeem aan de vastgestelde noodzakelijke kenmerken voldoet. Integratietests onderzoeken of de componenten met elkaar samenwerken zoals bedoeld en elke component wordt uiteraard individueel getest. Het lagenmodel volgens figuur 2 geeft enkele mogelijkheden aan die het lineaire model niet toont. Een ontwerpteam kan, in principe, samen met de afnemer de user-requirements formuleren met alle bijbehorende acceptatietests zonder dat nog maar iets verder is vastgelegd. Testontwerp is volgens dit model een volwaardig deel van het totale ontwerp.
De geschetste levenscyclus is, door de aanwezigheid van terugkoppeling, uit te rollen tot diverse afgeleide modellen die beter toegesneden zijn op specifieke situaties en behoeften. Incrementeel ontwikkelen of ontwikkelen vanuit vastgelegde subsystemen en componenten kan met development lifecycles die zijn toegesneden op bepaalde bedrijven, sectoren of productgroepen. In al deze gevallen vormen de verschillende soorten requirements en hun verbanden een onmisbare ondersteuning van het ontwerp.

Requirementsbeheer

Het formuleren van requirements, het vaststellen van verbanden, en het gebruik ervan in architectuurontwerp van complexe ingebedde systemen is uiteraard buitengewoon moeilijk. De voordelen zijn echter evident. Als men onderlinge verbanden en relaties tot tests kan expliciteren, kan men het ontwerpproces ook ondersteunen met een informatiesysteem. De kwaliteit van een product, die wel wordt gedefinieerd als de overeenstemming tussen user-requirements en operationele performance, is uiteraard veel beter te beheersen met behulp van expliciete links tussen requirements en tests.
Documentatie van het ontwerpproces is gestructureerder en het ontwerpproces is daardoor beter te conformeren aan officiële kwaliteitsstandaarden. Als het ontwerpproces goed gedocumenteerd is kan het gedistribueerd gebeuren, door verschillende ontwerpteams die zich niet op een zelfde locatie hoeven te bevinden. Bovendien stellen afnemers steeds vaker eisen aan de documentatie van het ontwerp om de reproduceerbaarheid te verzekeren.

Vanuit deze optiek heeft een aantal bedrijven, waaronder Quality Systems and Software (QSS) en Rational Software Corporation, commerciële systemen gebouwd voor requirementsbeheer. De belangrijkste functie van deze tools is ongetwijfeld de vastlegging van de links tussen de requirements, maar uiteraard is er ook verdere structuur aanwezig. Requirements zullen zijn gerubriceerd (bijv. veiligheid, gebruikersbediening, requirements aan timing of aan geheugen) en een prioriteit hebben. Ze kunnen ook een hiërarchie vertonen die in een boomstructuur is aan te geven en ze kunnen op meer plaatsen in het ontwerp terugkeren. Anderzijds zijn er misschien modulen te onderscheiden met onderling zeer weinig verbanden, die zich dan uitstekend lenen voor parallelle ontwikkeling.
Requirementsbeheer is zeer dynamisch. In de loop van het ontwerp wordt voortduren van alles veranderd en dit vereist niet alleen een goed versiebeheer (history management) maar ook ondersteuning van het veranderingsproces zelf (change management). Ook risicobeheer (risk management) kan met de requirements worden verbonden. Alle informatie moet, liefst op allerlei manieren, kunnen worden getoond en aangepast. Omdat requirements in eerste instantie als tekst zijn geformuleerd zal een requirementsbeheersysteem bijvoorbeeld ook dergelijke bestanden kunnen inpassen.
Omdat het requirementsgegevensmodel zich toch bovenal kenmerkt door de vele links is een relationele databank (die gebaseerd is op de tabelvorm) minder geschikt en requirementsbeheersystemen gebruiken dan ook in het algemeen het netwerkmodel. De suite van QSS, Dynamic Object Oriented Requirements System (DOORS) is, zoals de naam al aangeeft, geïnspireerd door het objectmodel.
Requirementsbeheer is zeker niet specifiek voor ingebedde systemen of toepassing in softwareengineering. Het is oorspronkelijk vooral gebruikt in grote industriële projecten, vooral voor militaire systemen, maar thans hebben ook spelers in de telecommunicatiesector als Lucent, Motorala en Ericsson het ingepast in ontwerpprocessen. ■

Geraadpleegde bronnen

1. Stevens, R., Brook, P., Jackson, K., Arnold, S. „*Systems Engineering - Coping with Complexity*". Prentice Hall Europe (1998). ISBN 0 - 13-095085-8;
2. Stevens, R. „*The DOORS Dynamic Object Oriented Requirements System*". Te downloaden van www.qssinc.com;
3. de website van de International Council on Systems Engineering (Incose): www.incose.org;
4. de website van Quality Systems en Software: www.qssinc.com;
5. de website van Rational Software Corporation: www.rational.com.