

UML, Octopus of Hatley

Met de toename van de complexiteit van software-intensieve systemen groeit ook de behoefte aan goede ontwikkelmethodieken. Ordina en Task Switch onderzoeken een drie methodieken en merken dat die op grote lijnen elkaar niet veel ontlopen. Op detail komen de verschillen echter duidelijk naar voren.

RENÉ ZENDEN, ORDINA & GER SCHOEBER Task Switch

In steeds meer professionele- en consumentenproducten is het gebruik van software niet meer weg te denken. De ontwikkeling van software voor technische systemen heeft de afgelopen tien jaren een enorme vlucht genomen. Van de bepaling van de stand van een schuifdak van een auto tot aan het koffiezetapparaat in de keuken, steeds meer zien we het gebruik van software opduiken.

De bouw van software voor dergelijke technische systemen kent zijn eigen evolutie. Oorspronkelijk werden de systemen direct vanuit de hardware aangestuurd. Het aantal features is dan nog minimaal. Maar gaandeweg ontstaan wensen voor de uitbreiding van deze features. Enerzijds gedreven door vragen van de consument of klant (bijvoorbeeld extra functionaliteit), anderzijds vanuit de productenbouwers zelf (bijvoorbeeld het toevoegen van service-voorzieningen).

De eerste uitbreidingen vinden nog in hardware plaats. Tot het moment dat de eerste processor in het product wordt geïntroduceerd. Zo'n processor biedt veel flexibiliteit naar de toe-

komst, maar vormt wel een extra kostenpost voor het product. Uiteraard vanwege de processor zelf, maar ook door extra toevoegingen als rom- en eventueel ram-geheugen. Het opent de weg naar de implementatie van additionele functionaliteit in software. De eerste uitbreidingen in software zijn nog niet zo omvangrijk; de hardware-engineer is dan nog de aangewezen persoon om die te implementeren. Maar allengs wordt de software complexer van karakter en de inzet van software-specialisten noodzakelijk. Er ontstaat een behoefte om software-engineers in het productontwikkelteam op te nemen.

Vanaf dat moment gaat de evolutie in een versnelling. De mogelijkheden van de software zijn ontdekt en het bedenken van nieuwe ideeën krijgt een enorme boost. Zo ook het aantal software-engineers. Waar voorheen de features nog ontwikkeld werden door een enkele engineer, staan er nu opeens teams opgesteld waarin men nauw met elkaar samenwerken. Communicatie wordt het toverwoord en middelen om de technische aspecten, analyses en ontwerpen met elkaar te delen zijn onontbeerlijk geworden.

Procedures en methodieken

Om het ontwikkelproces in de hand te houden worden procedures en methodieken geïntroduceerd. Denk aan ISO-standaarden en het CMM (capability maturity model). In de loop der jaren hebben diverse ontwikkelmethodieken het licht gezien. Elke methodiek kent haar eigen accenten. Denk hierbij onder meer aan applicatiedomeinen,

technische aspecten, volledigheid, tool-ondersteuning en toegankelijkheid.

Het valt in de praktijk niet mee om op basis van rationele argumenten een methode te kiezen. Het liefst wil men uiteraard de beste keuze maken zonder dat er veel kennis vereist is van alle methodieken. Op dit moment kan dit alleen met vrij gedetailleerde kennis van methodieken en tooling. Middels een studie hebben we geprobeerd een aanpak te ontwikkelen die zo'n keuze toch mogelijk moet maken. De weg die we daarbij bewandeld hebben is als volgt:

- samenstellen van lijst van ontwikkelmethodieken en definiëren van evaluatiecriteria;
- waarden van de ontwikkelmethodieken aan de hand van de evaluatiecriteria;
- beschrijven van domeinaspecten en de relatie tussen de domeinaspecten en de evaluatiecriteria (zie kader Applicatiedomeinen).

Dit laatste punt is met name belangrijk om de juiste weegfactoren aan te kunnen brengen. Stap 2 geeft alleen een zo objectief mogelijke inschatting van het voldoen aan betreffende evaluatiecriteria. Voor elk domeinaspect geldt echter dat bepaalde evaluatiecriteria meer of minder van belang zijn. Door ook die relatie in kaart te brengen (stap 3) is het mogelijk de beste ontwikkelmethodiek voor een bepaald applicatiedomein te selecteren.

De methodieken

Hatley & Pirbhai

Derek J. Hatley en Imtiaz A. Pirbhai hebben in de jaren tachtig een ontwikkelmethodiek opgezet. Deze zogenoemde *Strategies for Real-Time System Specification* staat vooral bekend onder de naam *Hatley & Pirbhai*. In wezen bestaat de aanpak uit twee methoden voor het specificeren van enerzijds de vereisten, en anderzijds de ontwerpstructuur van op software gebaseerde systemen. Hatley & Pirbhai kenmerkt zich door de zogenoemde functionele aanpak. Uitgaande van de functionele vereisten, kijkt Hatley & Pirbhai naar verwerkings-, besturings- en timing-as-

Applicatiedomeinen

Het rapport wat ten grondslag lag aan dit artikel bevat een uitdieping van de relatie tussen evaluatiecriteria en applicatiedomein. Dit biedt de mogelijkheid redelijk eenvoudig een inschatting te maken in hoeverre een ontwikkelmethodiek geschikt is toe te passen voor de bouw van een systeem uit een bepaald applicatiedomein.

& Pirbhai?

pecten. Vanuit de meer fysieke aspecten kijkt deze aanpak juist naar architecturale aspecten waarbij rekening wordt gehouden met het iteratieve karakter van de ontwikkeling van systemen.

Unified modelling language

In de jaren negentig is door een gezamenlijke inspanning van Grady Booch, James Rumbaugh en Ivar Jacobsen (zie interview in PT Embedded Systems 2002, nummer 3) de *Unified modelling language* (UML) ontstaan. UML is een modelleertaal die met name geschikt is voor het beschrijven van systemen vanuit een objectgeoriënteerde aanpak. Uitgangspunt hierbij is een model te creëren dat de werkelijkheid zo goed mogelijk benadert. Daarnaast is het doel een vereenvoudiging van die werkelijkheid te maken. Dit moet er toe leiden dat we middels de UML-modellen beter in staat zijn de systemen te begrijpen die moeten worden ontwikkeld. Doelstellingen hierbij zijn: visualisatie, structuren, templates en documenteren van beslissingen. Het toepassen van de 'taal' UML wordt beschreven in het 'proces' RUP (Rational unified process).

Octopus

Octopus is een methode die de uitdagingen van realtime systemen combineert met de mogelijkheden van een objectgeoriënteerde aanpak. Maher Awad (zie interview in PT Embedded Systems 1999, nummer 4), Juha Kuusela en Jurgen Ziegler hebben hun ervaringen opgedaan in realtime objectgeoriënteerde projecten gebundeld en verwerkt tot een systematische aanpak van software-ontwikkeling en verwerkt in de Octopus-methode. De Octopus-methode is een uitbreiding op OMT en Fusion en focust volledig op embedded realtime systemen. Hierdoor worden belangrijke aspecten voor dit domein zoals interrupts, concurrency en synchronisatie expliciet geadresseerd. Het houdt rekening met de intensieve interactie die embedded systemen hebben met hun omgeving via sensors en actuatoren en de daarbij behorende timing-requirements. De me-

thode biedt houvast waar de meeste andere, meer algemene, methodes ophouden.

Evaluatiecriteria

In het kader *Evaluatiecriteria* is een overzicht weergegeven van de evaluatiecriteria. Het bevat in totaal twaalf criteria die naar onze mening relevant zijn bij het toetsen van ontwikkelmethodieken.

Het gaat in het kader van dit artikel te ver om alle criteria in detail te beschrijven. Om een idee te geven hoe we de criteria gebruikt hebben tijdens ons onderzoek zullen we er één wat verder uitdiepen, namelijk die van de dynamische en statische aspecten.

Dynamische en statische aspecten

Het beschrijven van een systeem kan plaats vinden op verschillende niveaus van abstractie. Typisch wordt hierbij gedacht aan conceptueel niveau, logisch niveau en fysiek niveau. Op conceptueel niveau wordt een systeem als een black box beschouwd. Hierin komen alleen elementen voor die het 'concept' van het systeem beschrijven. Op logisch niveau zien we het systeem als een white box. Op fysiek niveau volgen dan de verdere details als: hoe zien de structuren er uit, waar ligt de data of hoe stroomt de data, hoe werken processen en threads samen. Kortom, het systeem volgens verschillende abstractieniveaus beschrijven is in wezen een bepaalde systeemdoorsnede.

Een andere systeemdoorsnede wordt gevormd door te focuseren op alleen de statische en dynamische aspecten. In de voorbeelden uit de vorige alinea op het gebied van het fysieke abstractieniveaus zien we dit terug. Met andere woorden, op dit fysieke niveau is een onderscheid te maken tussen statische en dynamische aspecten: waar ligt data en waar stroomt data. Het onderscheid maken tussen dynamische en statische aspecten is uiteraard ook mogelijk op de andere (conceptuele en logische) abstractieniveaus. De dynamische aspecten op logisch niveau beschrijven bijvoorbeeld hoe de elementen binnen de white box view met elkaar samenwerken.

Evaluatiecriteria

Communicatie

Hoe goed is de ontwikkelmethode om aan mensen van verschillende disciplines systeemaspecten te verklaren. Kunnen zij de beschrijving zelfstandig lezen, begrijpen en gebruiken?

Toegankelijkheid

Werkt betreffende methode intuïtief?

Ondersteuning voor bekende architectuurpatronen

Kent de methode standaard architectuurpatronen zoals pipe/filter, layering, componenten, enzovoort?

Dynamische en statische aspecten beschrijven

Biedt de methode mogelijkheden om zowel statische systeemaspecten als dynamische systeemaspecten gescheiden ten opzichte van elkaar te beschrijven?

Mogelijkheid tot controleren van consistentie en van juistheid

Wanneer meerdere views of aspecten van het systeem gescheiden ten opzichte van elkaar zijn vast te leggen of te beschrijven, biedt betreffende methode dan mogelijkheden de consistentie van betreffende views op een eenvoudige wijze te verifiëren?

Information hiding

Laat de methode toe informatie op een bepaald abstractieniveau of in een bepaalde view al dan niet te verbergen?

Scheiding tussen abstractie en leveling van views

Kent de methode meerdere hiërarchische abstractieniveaus?

Ondersteuning voor zowel top-down als wel bottom-up benadering

Schrijft de methode al dan niet een vaste benadering van systeemontwerp voor?

Hardware/software-allocatie c.q. partitioning

Biedt de methode een mogelijkheid om functionaliteit toe te wijzen aan hardware en software, dit gedurende het ontwikkelproces?

Integratie van embedded en realtime aspecten

Heeft de methode kennis van timing of realtime aspecten, zoals bijvoorbeeld timing, synchronisatie?

Geschiktheid voor de complete lifecycle van een product

Kan betreffende architectuur-ontwikkelmethode worden ingezet gedurende de gehele ontwikkelfase van het betreffende product?

Tools

Wordt betreffende ontwikkelmethode ondersteund door automatische tooling?

De verschillende ontwikkelmethoden kennen elk de nodige diagrammen of mogelijkheden om allerlei dynamische en statische aspecten op verschillende abstractieniveaus weer te geven. Denk hierbij bijvoorbeeld aan state-diagrammen, bericht/volgorde-diagrammen, beschrijvingen van interrupts en events, use-cases, data-flow, control-flow, architecture-interconnect-diagrammen en diagrammen voor de >

weergave van de executiearchitectuur. Om de score van elke methode te bepalen op dit 'dynamische en statische aspecten'-criterium is voor elke methodiek onderzocht in hoeverre die al dan niet voldoet aan de aspecten zoals ze zojuist in detail zijn beschreven. Voor alle criteria is een detaillering als hierboven aangebracht. Dit teneinde een zo volledig mogelijke score te kunnen geven.

Vergelijking

In het vergelijkingsoverzicht zijn de door ons geëvalueerde methoden naast elkaar gezet. Per evaluatiecriterium is een score weergegeven. Dit is een score die door ons is toegekend op basis van het al dan niet voldoen aan specifieke punten vervat in een uitgebreide beschrijving van de evaluatiecriteria. Die zijn vastgelegd in het door ons opgestelde rapport dat aan dit artikel ten grondslag ligt. De score varieert tussen de 0 en 100 %.

Uit de vergelijking blijkt dat elke methode weliswaar haar eigen sterke en zwakke kanten kent, maar dat ze gemiddeld genomen over alle twaalf evaluatiecriteria elkaar niet zoveel ontlopen (figuur 1). In ons rapport zijn we verder gegaan om ook de relatie tussen de evaluatiecriteria en typische applicatiedomeinaspecten te leggen.

Als voorbeeld beschouwen we het audio/video-domein. Denk hierbij aan het consumentenproduct digitale tv. Voor dit domein zijn zowel realtime als omvang van datatransport redelijk van belang. Minder van belang is data-integriteit. Het af en toe haperen van een beeld doordat dit niet snel genoeg kan worden opgebouwd is niet acceptabel. Dat willekeurige beeldpunten in streaming-video niet altijd correct zijn, zal nagenoeg niet opvallen. Dit geeft het belang snelheid versus correctheid weer.

Bij realtime aspecten spelen onder andere de evaluatiecriteria *hardware/software-partitionering* en uiteraard *ondersteuning voor realtime aspecten* een belangrijke rol. Door deze criteria een zwaardere weegfactor, en de criteria die van belang zijn voor data-integriteit een lagere weegfactor mee te geven, kan de beste toepasbaarheid van een bepaalde ontwikkelmethodiek voor het audio/video-domein worden gegeven.

Conclusie

Als we naar de tabel kijken dan zien we dat de ontwikkelmethoden elk hun eigen positieve als minder positieve ac-

centen kennen. Gemiddeld over het totaal ontlopen ze elkaar niet veel. Zo is er is niet één methode die alle onderzochte aspecten volledig afdekt.

UML scoort in ons overzicht beter op gebied van consistentieverificatie dan Hatley & Pirbhai en Octopus. De toegekende waardering komt door de verscheidenheid van tools op de markt die UML als modelleertaal ondersteunen en daarin het met name goed doen op gebied van consistentieverificatie. Dit in tegenstelling tot Octopus. Toch geniet UML niet de topscore bij het aspect *ondersteuning door tools*. Dit omdat we zien dat er weliswaar redelijk wat tooling aanwezig is, maar deze over het algemeen slechts een deel van de methode afdekken (UML is als taal nog steeds in ontwikkeling). Op het laatste punt zien we Hatley & Pirbhai beter scoren.

Hatley & Pirbhai is naar onze mening duidelijk minder toepasbaar voor de gehele lifecycle van een product. Ze kan wel zeer sterk mee in de analysefase en redelijk in de ontwerpfase. De consistentie tussen beide fasen is wat lastig te verifiëren. Waar Hatley & Pirbhai weer sterk scoort is op het gebied van leveling van views. Dit wordt ook zeer goed door tools ondersteund.

Octopus kent met name een aantal eigenschappen die prettig zijn bij de ontwikkeling van systemen die een sterke hardware/software-koppeling hebben. Ze kent goede mogelijkheden voor het beschrijven van dynamische en statische aspecten, en het toekennen van functionaliteit aan hardware en software. Daarnaast onderscheidt Octopus

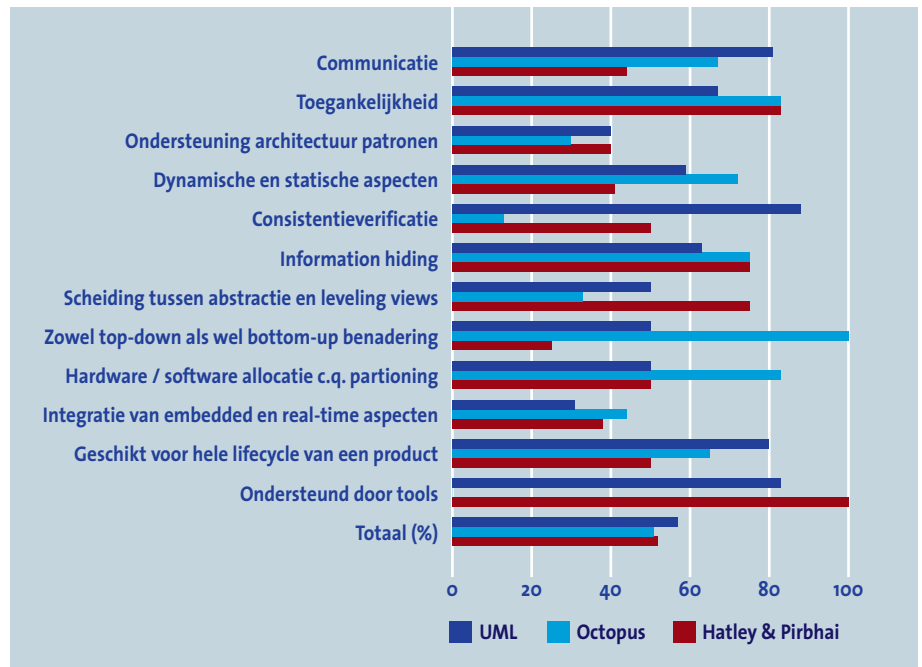
zich in positieve zin vanwege het feit dat zowel een topdown- als bottom-up-benadering prima te doen zijn waarbij ook gedurende de ontwikkeling makkelijk tussen kan worden gewisseld.

Op het gebied van integratie van 'embedded' en 'realtime' liggen de methoden niet dicht bij elkaar, maar kennen ze op detail weer elk hun eigen sterktes en zwaktes. Grosso modo zien we voor alle methoden verbetermogelijkheden. Denk hierbij aan ondersteuning van *Rate monotonic analysis* (RMA), beschrijving van timing, deadlines en duration, modelleren van queues. Octopus dekt op dit onderwerp het beste de lading. ■

Meer informatie

Deze evaluatie is tot stand gekomen uit een gezamenlijke inspanning van René Zenden, architect binnen Ordina Technical Automation en Ger Schoeber, eigenaar en consultant van Task Switch. Ordina Technical Automation is gespecialiseerd in het ontwikkelen van productsoftware, productiesoftware en technische software voor industriële toepassingen. Task Switch is een consultingbureau met diensten op gebied van coaching, projectmanagement en architectuur voor embedded software-systemen.

De hier gepubliceerde evaluatie is slechts een korte samenvatting. Ten grondslag aan deze publicatie ligt een rapport dat eigendom is van Ordina Technical Automation. We zijn benieuwd naar uw mening over deze evaluatie. Voor vragen of reacties kun u contact opnemen met René Zenden (rene.zenden@ordina.nl) of Ger Schoeber (ger.schoeber@task-switch.nl).



1. Uit een vergelijking blijkt dat elke methode weliswaar haar eigen sterke en zwakke kanten kent, maar dat ze gemiddeld genomen over twaalf evaluatiecriteria elkaar niet veel ontlopen.