

Een model is misschien wel het oudste hulpmiddel om vraagstukken inzichtelijk te maken. Nieuw bij softwareontwikkeling is de trend naar tools die vanuit abstracte modellen zelf de software genereren. En dan het liefst één tool voor alle toepassingen. Dat klinkt ideaal, maar in de praktijk blijkt de kloof tussen abstracte modellen en praktische software vaak te groot.

Modelgebaseerde softwareontwikkeling

Een model geeft een vereenvoudigde weergave van de werkelijkheid. Dat is niets nieuws. We gebruiken al modellen sinds mensenheugenis. Denk daarbij aan het planetarium met ons zonnestelsel van Eise Eisinga sinds 1781. Modellen zijn een handig communicatiemiddel. Een schetsje op papier kan een ingewikkeld probleem snel inzichtelijk maken. Specialisten uit verschillende disciplines kunnen makkelijker samenwerken als slechts de essentie van een probleem wordt gecommuniceerd. Ieder zorgt vervolgens voor de invulling op zijn eigen vakgebied. Waarom dan toch een artikel in Trends & hypes over modelgebaseerde softwareontwikkeling? Omdat er een trend is naar tools die vanuit abstracte modellen zelf de software genereren en de mogelijkheden van dit soort tools gehyped worden. Daarbij wordt het liefst één tool voor alle toepassingen ingezet. Dat klinkt ideaal: op een hoog abstractieniveau een model maken en je tools doen de rest. In de praktijk blijkt de kloof tussen abstracte modellen en bruikbare software vaak te groot.

De ontwerper is in een bepaald aspect geïnteresseerd en daarvoor laat hij alles weg wat niets toevoegt.

Invalshoek

Een model is een abstractie van iets concreets. De ontwerper is in een bepaald aspect geïnteresseerd en daarvoor laat hij alles weg wat niets toevoegt voor dat aspect. Wat overblijft, is de essentie voor die betreffende invalshoek. Bij het ontwerpen van een softwaresysteem worden twee soorten invalshoeken onderscheiden: modellen voor structuur en modellen voor gedrag. Voor een eenvoudig natuurkundig experiment kan een complex systeem als een auto worden gemodelleerd als een object met slechts een massa en een snelheid. Met een dergelijk model kan gerekend

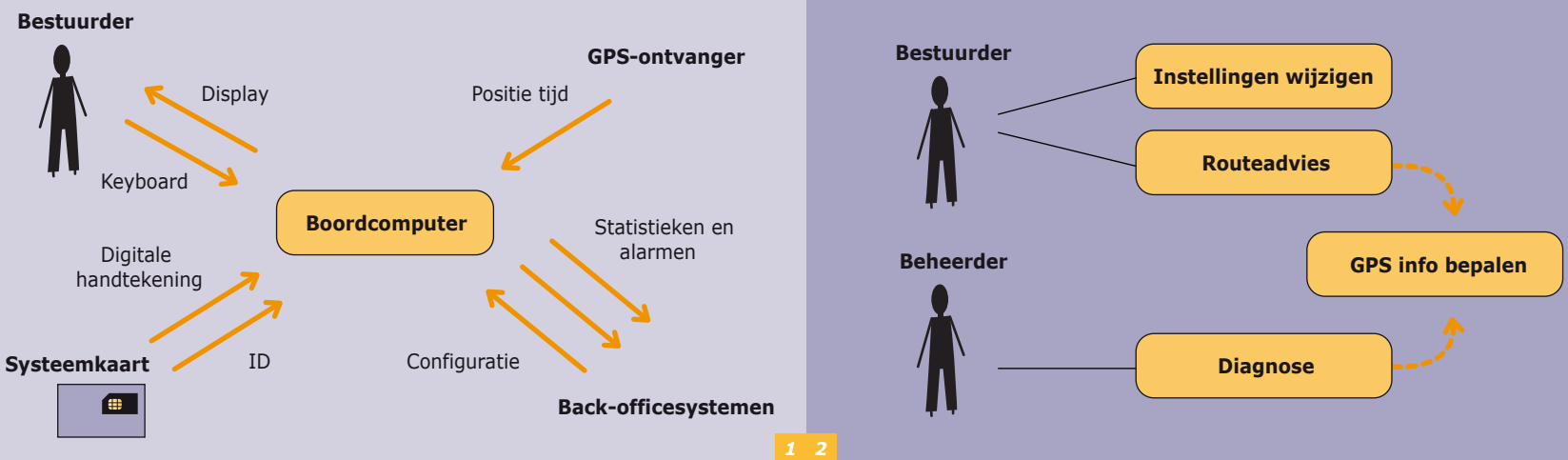
worden aan bijvoorbeeld de lengte van de remweg of de impact van een botsing.

Een aantal modellen is, vanwege hun focus en eenvoud, heel geschikt voor communicatie met de klant. Met name modellen die het domein van de klant weergeven. Voorbeelden van dergelijke modellen zijn: use case diagrammen (gedrag in termen van de klant) en het context diagram (systeem in relatie met externe systemen). Andere modellen zeggen meer over de implementatie van het systeem. Deze modellen ondersteunen primair het bouwproces en zijn voor de klant minder belangrijk.

UML

Voor softwareontwikkeling worden in het algemeen modellen gebruikt van de UML-standaard. UML staat voor Unified Modeling Language en is ontstaan uit verschillende eerdere stromingen met betrekking tot softwareontwikkeling. Hoewel UML zijn beperkingen heeft, is het een veel gebruikte standaard. Het idee van UML is dat de gebruiker op een hoog abstractieniveau zijn werk kan doen, in een vorm of taal die dicht bij zijn belevingswereld staat. Vaak in een grafische vorm, met figuren en diagrammen. UML is feitelijk een gereedschapskist met modelleergereedschappen. Het is geen methodiek die voorschrijft hoe je software moet ontwerpen. Er zijn diverse leveranciers die integrale UML-gebaseerde ontwerptools aanbieden.

Een trend is dat dergelijke tools buiten het UML-domein treden en extra toegevoegde waarde proberen te creëren op het vlak van softwareontwikkeling. In de negentiger jaren noemde men dit CASE-tools (Computer-aided Software/System Engineering). Tegenwoordig zien we vaak termen



1. Context diagram, systeem in relatie met externe systemen. 2. Use case diagram, gedrag in termen van de klant.

als "model-driven" en "model-based". Een voorbeeld hiervan is MDA (Model-Driven Architecture) waarbij men in verschillende transformatiestappen vanuit een model (bv. UML) en informatie over het platform, code voor een systeem probeert te genereren. De stap van UML naar bruikbare code van een implementatie is echter erg groot. Naast de ontwerpmodellen moet nog veel informatie worden toegevoegd aan dit proces voordat bruikbare applicaties gegenereerd kunnen worden.

Bij Technolution hebben we de ervaring dat het telkens toevoegen van meer detailinformatie aan een ontwerp (top-down) vaak ontspoord. Het gebruik van UML is slechts efficiënt tot een bepaalde diepgang (een bepaald abstractieniveau). Ook is in de praktijk de diversiteit aan platformen in ons werkdomein groot. Een ontwikkelaar zal hierdoor ook in een bottom-up richting vanuit het platform naar een design toe moeten werken. Hierbij kunnen andere modellen worden toegepast die relevant zijn voor de specifieke applicatie.

Men dient zich te realiseren dat UML geen programmeertaal is, maar een modelleertaal, met het bijbehorende hoge abstractieniveau. En het is zo algemeen mogelijk gehouden (general-purpose). Het is een gereedschapskist voor een doorsnee ontwikkeltraject. De architect neemt er die dingen uit die hem op dat moment van pas komen. Maar bij "niet-doorsnee" (specialistische) ontwikkelingen schiet UML tekort. Zodra specifieke eisen niet meer binnen UML zijn in te passen, zal hij het palet aanvullen met gespecialiseerde gereedschappen. Dit geldt bijvoorbeeld voor problematiek gerelateerd aan realtime gedrag, multi-threading en systeem performance. Om een goed werkend product te krijgen, gebruiken ontwerper en ontwikkelaar kennis en kunde uit vele domeinen.

Model als testomgeving

Een andere inzet van modellen zien we bij een testomgeving. Functioneert het product in de omgeving van de klant, met alle specifieke omstandigheden van dien? Om daar achter te komen wordt ook de omgeving van het te ontwikkelen product gemodelleerd. De in ontwikkeling zijnde software moet samenwerken met verschillende andere apparatuur zoals: sensoren, actuatoren, netwerken, back- en front-end systemen enzovoort. Hoe ga je dat testen? Het is niet nodig om deze apparatuur direct binnen bereik te hebben om de software te testen. Daarvan wordt een simulator gemaakt. Dat is meestal een computer die alle inputs en outputs simuleert. Het te testen softwareproduct wordt daaraan gekoppeld. Naast een ontwerpmodel voor het product is er ook een model van de omgeving nodig bij de productontwikkeling. De ontwikkelaar kan hierdoor al in een zeer vroeg stadium van het project en zeer frequent testen uitvoeren binnen de gesimuleerde omgeving. Hij hoeft daardoor niet te wachten tot de systeemomgeving bij de klant gereed en beschikbaar is, of naar een specifieke locatie te reizen om daar te testen met speciale apparatuur.

Conclusie

Modellen zijn onmisbaar, en dat weten we al eeuwen. Modellen in het ontwerpproces maken complexe systemen begrijpbaar en bespreekbaar. De inzet van simulatoren (externe systemen) voor testdoeleinden ondersteunt de software-ontwikkeling gedurende het gehele ontwikkeltraject. Het genereren van applicaties uit ontwerpmodellen met behulp van tools is nog een brug te ver, zeker voor de specifieke en complexe projecten die Technolution uitvoert. Kennis van architecten en ontwikkelaars is nog steeds de onmisbare en bepalende factor voor de kwaliteit en toepasbaarheid van de applicatie.

Om een goed werkend product te krijgen, gebruiken ontwerper en ontwikkelaar kennis en kunde uit vele domeinen.