'Er is in twintig jaar bijna niets gebeurd'

Softwareontwikkeling is een ideeënsoep

'Een ideeënsoep zonder duidelijke ingrediënten'. Zo omschrijft Ivar Jacobson, chief technology officer van het bedrijf Ivar Jacobson International en medeontwikkelaar van de Unified Modeling Language, de huidige situatie in de softwareontwikkeling.

"Alle software-ontwikkelprocessen – commercieel of niet, gedocumenteerd of niet, omgeven met de oude ontwikkeltalen of de modernste kretologie – zijn niets anders dan een combinatie van stukjes en beetjes uit eerder ontwikkelde processen, aangevuld met een klein beetje lokale kennis uit een eigen specifiek domein of branche. Iedere nieuwe methode mag dan wel worden gepresenteerd als volledig nieuw, maar in feite gaat het altijd (ook bij UML) om een combinatie van oude ideeën met een paar nieuwe inzichten."

Het resultaat is volgens Jacobson dat veel ideeën en kennis verloren gaan, omdat ze worden overspoeld door nieuwe trends die door nieuwe goeroes worden verkocht. Managers, die niet de laatste mode volgen komen in een lastig parket. Ze raken volgens hun jongere medewerkers 'out of touch'. Er worden pilotprojecten opgestart om aan te tonen dat de nieuwe methodes toch echt beter functioneren en de moderne gemotiveerde ontwikkelaar krijgt – op kleine schaal – die wel aan het werk. Het gevolg is dat de nieuwe benadering de oude verdringt en dat de oude overboord wordt gezet. Met alle goede elementen erbij. Pas later blijkt dat ook de nieuwe methode componenten bevat die niet met elkaar kunnen communiceren. Hier is dus geen sprake van survival of the fittest, maar van survival van de modernste, aldus Jacobson. Ondertussen wordt bar weinig vooruitgang geboekt in softwareontwikkeling. Net als in de mode-industrie is er 'much ado about next to nothing'. In zoiets triviaals als de mode is dat nog wel acceptabel, maar gezien de enorme investeringen in de softwarebranche is dit pure verspilling.

Als voorbeeld haalt Jacobson de agile-hype aan. "De laatste mode in onze sector is 'being agile'. Laat me duidelijk stellen dat de agile-beweging een duidelijke positieve bijdrage heeft geleverd aan de software-industrie. Het wijst ons erop dat mensen de belangrijkste factor zijn bij de ontwikkeling van software. Bij de meer technisch georiënteerde modes zoals objectoriëntatie en programmeren in Java wordt dat nog wel eens vergeten. Door op dit gebied een aantal waarden vast te leggen in het agile manifesto is iets gecreëerd dat een golfbeweging naar een nieuwe modetrend zal kunnen weerstaan."

"Het is jammer dat dit niet kan worden gezegd van de vele methodes, die worden verkocht als zijnde de enige juiste ondersteuning van de agile-filosofie. Voor een beweging die de mens boven processen en tools stelt, zijn er wel verdraaid veel 'nieuwe' processen en tools bij gekomen", stelt Jacobson vast.

Theorie

Als we in staat zijn om de ingrediënten in de soep te isoleren zouden we beter in staat zijn een werkmethode te ontwikkelen die overweg kan met de veranderende industrie. Daar kunnen nieuwe ideeën worden ingebracht, die de aanwezige kennis verder kunnen uitbreiden. We moeten de mensen helpen begrijpen hoe ze goede software kunnen bouwen, vindt Jacobson. Hij stelt dat daarvoor een deugdelijke theorie moet worden ontwikkeld.

Ivar Jacobsen: "De laatste mode in onze sector is 'being agile'."

Je kunt best succes hebben met het wiel opnieuw uitvinden, zoals in de softwarebranche maar al te vaak gebeurt. In de jaren 60-70 was Ericsson bijvoorbeeld heel succesvol met het ontwikkelen van componenten. Dat was sterk technologie-gebaseerd. Toen kwam de agile-beweging. Die heeft veel goede dingen geadopteerd. Nu gaat het bijna alleen nog om Scrum en dat is minder gelukkig. We vinden wel dingen uit, maar het overgrote deel komt neer op opnieuw uitvinden.

"Het wordt me steeds duidelijker dat we behoefte hebben aan een theorie voor softwareontwikkeling. Tussen de verschillende methodes zitten hele kleine verschillen. Het is nu zaak om een gezamenlijke kern, een basis te ontwikkelen en van daaruit de ontwikkeling voort te zetten. Onze hoop is er nu op gevestigd een fundament voor software-engineering vast te kunnen leggen. Om tot een standaardisering te komen. Dat is goed voor de ontwerpers van methoden. Zij kunnen aan die standaard nieuwe ideeën toevoegen. Voor de industrie is het ook beter, omdat het de ontwikkeling van software en methodes in een stroomversnelling kan brengen", vertelt Jacobson.

> ***'Het wordt steeds duidelijker dat we behoefte hebben aan een theorie voor software-ontwikkeling.'***

We hebben nu ook wel een standaard – UML – maar die is erg complex. Die kan beslist worden verbeterd. Maar eerst zouden we het eens moeten worden over de betekenis van verschillende begrippen. Wat bedoelen we met requirements, wat met testen, wat met releases? Ik heb het gevoel dat er veel steun is voor een standaard. De enige uitzondering wordt gevormd door de ontwikkelaars, die kiezen voor 'nieuwe mode'.

UML vereenvoudigen

UML heeft in het verleden de nodige tegenslagen gehad, maar de kritieken nemen af en het gebruik ervan neemt weer toe, onder meer omdat ook Microsoft – dat aanvankelijk werkte met Domain Specific Languages – nu ook UML ondersteunt. "UML moet worden verbeterd en daar zijn we nu mee bezig. Slechts 20 procent UML wordt regelmatig gebruikt. Dit betekent waarschijnlijk dat het sterk kan worden vereenvoudigd. UML is trouwens niet de enige methode. We gebruiken use cases, feature driven design. Dat is allemaal prima, maar we zouden dat in een kern moeten onderbrengen."

"Het klinkt misschien abstract, maar we hebben in mijn bedrijf zo'n 'kernel' ontwikkeld. Deze wordt door veel bedrijven met succes gebruikt. Ik denk niet dat onze kernel straks als standaard wordt geaccepteerd, maar we hebben wel laten zien dat het kan. De kernel bevat een soort taal, waarmee je de practices omschrijft. Je kunt onze vijftien practices afzonderlijk gebruiken, maar ook kiezen uit een combinatie van de verschillende practices. We hebben inmiddels laten zien dat het kan. Dat er een weg is om tot een algemene theorie te komen. Daar moeten we naartoe werken."

De bestaande platforms dienen rond deze theorie voornamelijk om het ontwerp aan te passen. Neem bijvoorbeeld use case driven development. Daarmee beschrijf je de requirements. Vervolgens kun je volgens Jacobson het design uitwerken met Java of welke ander platform dan ook.

Een ander waardevol aspect van een duidelijke theorie is een grotere eenvoud in het werk. Een groot bedrijf heeft al snel twintig verschillende methodes in gebruik. Enkele IT-ers weten daarmee om te gaan, veel anderen niet. Als je nieuwe medewerkers aanneemt hebben die minstens zes maanden nodig om zich in te werken. Als ze die kernel hebben, waar alle methodes in zijn ondergebracht als een soort unified language en als practices gaan gebruiken wordt dat veel eenvoudiger. Je gebruikt een practice om te testen, een andere om een architectuur te ontwikkelen. Je brengt dus veel meer duidelijkheid aan.

Social development

"Agile heeft ons wel veel goeds gebracht, Ik zie het als social development, waardoor veel ontwikkelaars meer plezier in hun werk hebben gekregen. Maar degenen, die het hebben uitgedacht spelen er nu geen grote rol meer in. Dat zijn nu de bedrijven, die met agile technieken hebben leren omgaan en er ervaring mee hebben opgebouwd. Die de ideeën erachter hebben overgenomen."

"Zo'n zelfde traject heeft SOA afgelegd. Toen het werd ontwikkeld zeiden de aanhangers 'vergeet wat je weet over software-ontwikkeling. We doen het nu alleen nog met SOA'. Het probleem is dat wanneer mensen te veel willen doen, ze ook veel fouten maken. Veel SOA-projecten zijn volledig mislukt."

Het grootste risico van SOA ontwikkeling is dat op enig moment onvoldoende services beschikbaar zijn, zoals wanneer een serviceleverancier platgaat. Daarbij komt nog dat dat des te erger is in het geval van een service die veel in applicaties wordt gebruikt.

De oplossing in drie stappen

Ivar Jacobson denkt in drie stappen tot de 'waarheid van software-engineering' te kunnen komen:

Stap 1: Zoek de kern; de moeder van alle methodes

Bij zijn zoektocht naar de kern gaat Jacobson er vanuit dat de bestaande methodes veel gemeen hebben. Ze worden tenslotte allemaal gebruikt om software te ontwikkelen. Iedereen is het erover eens dat je altijd bepaalde handelingen moet verrichten als je software wilt bouwen. Iedereen weet dat je code moet schrijven, dat je die moet testen, dat je moet denken aan de requirements, dat er een backlog moet zijn en dat je een plan moet hebben. Uit deze zaken kun je een 'DNA voor softwareontwikkeling' destilleren. Jacobson noemt dit de 'essential core' of 'kernel'.
Door ongeveer 50 methodes te bestuderen, inclusief XP en Scrum, heeft het team van Jacobson een kernel samengesteld met iets meer dan 20 elementen. Je kunt bijvoorbeeld requirements vastleggen met features of met use cases. Hoe dan ook hebben de methodes een gemeenschappelijke basis, die in de kernel wordt vastgelegd.

Stap 2: Kom tot een groter begrip van methodes en processen

Belangrijk om tot een groter begrip te komen is dat we leren inzien dat iedere methode bestaat uit een soep van practices. Soms worden die practices expliciet genoemd; soms ook niet. RUP bestaat bijvoorbeeld uit verschillende geïntegreerde practices, terwijl Scrum in essentie bestaat uit één practice (project management) en enkele agile werkpatronen.

Stap 3: Beschrijf iedere interessante methode met gebruik van de kernel

Wanneer de kernel is vastgesteld kun je van daaruit alle methodes uniform beschrijven als specialisaties of extensies van deze kernel. De impliciete practices uit de verschillende methodes of processen zoals architectuur, componenten en iteraties worden uit de kernel 'geoogst'.Sommige practices zullen elkaar overlappen, zoals de risk driven iteraties van RUP en de backlog driven sprints van Scrum. Sommige andere practices kunnen elkaar aanvullen, zoals use cases en project management.
De kernel neemt zo volgens Jacobson de 'cosmetische' verschillen tussen de methodes weg. RUP praat over iteraties en Scrum over sprints, maar deze doen in wezen hetzelfde. Het wordt hierdoor eenvoudiger de werkelijke verschillen tussen de methodes te onderscheiden.

Maar er zijn meer valkuilen. De harmonisatie van de gegevens is daar een van. SOA is geen Esperanto, dus de services moeten wel op elkaar zijn afgestemd. Het eigenaarschap vormt ook nog wel eens een punt van discussie. Van wie zijn de services? En dan is er nog de kwestie van vraag en aanbod. Er bestaat nu eenmaal geen gouden gids, waaruit je maar van alles kunt bestellen.

"Van de huidige agile-technieken zal niet één het in zijn huidige vorm overleven. Maar ze hebben wel allemaal invloed gehad en zullen dat blijven hebben. RUP heeft ons leren omgaan met use case driven development. Use cases worden nog veel gebruikt en zijn erg succesvol. XP en het idee van test driven development zal ook best overleven. Ik denk dat alle methodes wel iets hebben bijgedragen, maar als 'brand' zullen ze verdwijnen. Ze zijn allemaal onderdeel van de soep en waarom zou je namen geven aan al die soepen. Het zijn soepen vol practices. Daar hebben we geen namen voor nodig. We moeten namen geven aan de practices."

Soep en practice

Het verschil tussen een practice en een soep (methodologie) is dat de methodologie van alles in zich heeft, terwijl de practice een duidelijk doel heeft en de onderdelen van de practice met elkaar samenhangen. Je stelt de requirements op aan de hand van use cases, dan ga je de use cases ontwerpen, coderen, testen. Dat alles wordt in de practice samengebracht. Maar wanneer je architectuur en use cases neemt is dat een ander verhaal. Je kunt architectuur ontwikkelen zonder use cases. Architectural development kun je uitvoeren met use cases, features, user stories, dus met verschillende technologieën. Het zijn de bouwstenen voor de methode. Met practices heb je een duidelijke structuur, terwijl je bij de architectuur weer met die soep te maken hebt, waarvan je maar moet zien wat je daaruit gebruikt.

Ivar Jacobsen.

Je moet dus de features uit de methodologie identificeren en kijken of je daarmee kunt werken. Kun je multiple inheritance gebruiken. Soms kan dat, maar het is altijd de vraag welke practices je uit de methodologie kunt gebruiken en welke niet samengaan. Als we de ingrediënten uit de soep halen – laten we use case

Steeds meer bedrijven gaan outsourcen. De software draait bijvoorbeeld in India, maar er is geen benul hoe dat gebeurt. De les die we daaruit moeten trekken is dat we daar erg voorzichtig mee om moeten gaan, vindt Jacobsen.

development nemen – dan kun je daar ook op voortborduren. Dan kun je zeggen: we gaan die practice van use case development verbeteren. Maar het blijft dan een practice op zich.

> ***'We moeten erg voorzichtig omgaan met de activiteiten die we outsourcen.'***

Platformpolitiek

De theorie die we willen ontwikkelen heeft overigens geen effect op de grote verscheidenheid aan platforms , talen en applicaties die in gebruik zijn. Jacobson: "We zullen niet minder afhankelijk worden van de technologie, maar wel duidelijker practices krijgen, waarmee je de bestaande platforms kunt doorontwikkelen of verbeteren. De verscheidenheid aan gebruikte software is een heel ander probleem. Dat heeft te maken met politiek en met veel geld. Ik heb wel eens gezegd dat de grote software leveranciers hun klanten tot gevangenen maken. En dat is nog steeds zo. Vendor-lock-in komt nog altijd op grote schaal voor. Als je dit probleem wilt aanpakken ben je je leven niet meer zeker. Daar zijn zulke grote belangen mee gemoeid. Als dat probleem al kan worden opgelost, dan zal die oplossing toch van de grote leveranciers moeten komen. Van Microsoft, Oracle, SAP, IBM. Misschien gebeurt dat ooit, maar reken er niet op dat dit binnen de komende honderd jaar zal zijn."

"Je ziet nu dat hetzelfde gebeurt bij outsourcing. De business is aanvankelijk voorzichtig gaan outsourcen. Maar inmiddels is zo veel van de activiteit geoutsourced, dat je ook hier te maken hebt met een lock-in door outsourcingbedrijven. De software draait in India, het beheer vindt daar plaats, maar je hebt er geen idee van hoe dat gebeurt. We begrijpen niet hoe het daar werkt. We begrijpen niet wat ze er precies doen. En we begrijpen niet waarom we er zo veel voor moeten betalen. We zitten er aan vast en moeten maar hopen dat het allemaal goed gaat. De les die we hieruit kunnen leren is dat we erg voorzichtig met oursourcing om moeten gaan."

Robert de Ruiter is hoofdredacteur van Optimize.