Van lineair naar iteratief naar Agile

De evolutie van systeemontwikkelmethoden

Sander Hoogendoorn

Sinds mensenheugenis worden systeemontwikkelprojecten op eenzelfde, lineaire wijze uitgevoerd. Vier van iedere vijf projecten zijn echter niet succesvol. Projecten raken in tijdnood, boven budget, leveren verouderde functionaliteit, of worden helemaal niet opgeleverd. Dit is dramatisch. Het vertrouwen in automatisering is dan ook bedroevend laag. Gelukkig kan aan de wijze waarop systeemontwikkelprojecten worden uitgevoerd veel worden verbeterd. Een introductie in dertig jaar systeemontwikkelmethoden.

Jarenlang vond nagenoeg alle systeemontwikkeling plaats volgens een van de lineaire methoden daarvoor. Ondanks duidelijke uiterlijke verschillen vertonen deze watervalmethoden -zoals ze ook wel worden genoemd- veel gemeenschappelijke kenmerken. Een daarvan is het opstellen van een 'mijlpaalproduct' per fase (zie verder kader *Lineaire systeemontwikkelmethoden)*.

FUNCTIONELE DECOMPOSITIE

Met deze generatie methoden zijn vooral grote projecten uitgevoerd, variërend van enkele tot tientallen en zelfs honderden manjaren. Om ontwerp en bouw van dergelijke projecten behapbaar te houden wordt een systeem opgedeeld in onafhankleijk functionerende subsystemen, die op hun beurt verder worden onderverdeeld. Deze verdeling, die functionele decompositie wordt genoemd, is gebaseerd op de samenhang tussen de gegevens waarvan subsystemen gebruik maken. Op deze wijze ontworpen en gebouwde systemen zijn vaak snel herkenbaar. De gegevensgerichte aanpak vertaalt zich in deelsystemen als 'Debiteurenadminstratie' of 'Facturering'. Meer dan automatisering van bedrijfsprocessen, zijn dergelijke subsystemen verantwoordelijk voor het beheer van verzamelingen bij elkaar horende gegevens.

NATTIGHEID MET WATERVAL

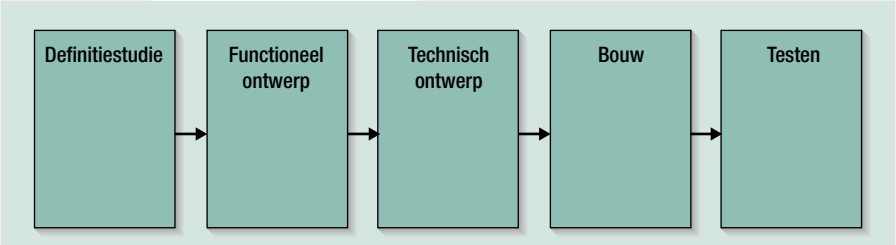
Lange tijd leek de gevolgde projectfasering dé manier om projecten uit te voeren. Toch gelden vier van iedere vijf systeemontwikkelprojecten als niet succesvol. Wat gaat er nu eigenlijk mis met lineaire projecten?

Veel problemen ontstaan juist door de gehanteerde fasering. Een functioneel ontwerper begint niet voordat de definitiestudie van een handtekening is voorzien. En als hij eenmaal van start is gegaan, is er geen weg terug. Nieuwe kennis, opgedaan tijdens het functioneel ontwerp, wordt niet meer verwerkt.

Deze werkwijze heeft belangrijke consequenties. Het is van essentieel belang dat de analist álle *requirements* (vereisten) voor het systeem boven water heeft voordat de definitiestudie is afgerond. Ieder mijlpaalproduct moet volledig zijn om afwijkingen te voorkomen. De eis volledig te zijn legt veel druk op projectmedewerkers, waardoor zij behoedzaam werken en rigide gebruik maken van geboden technieken. Dit komt de doorlooptijd van pro-

Lineaire systeemontwikkeling

Lineaire systeemontwikkelmethoden kennen vaste fasen en activiteiten, vaak in hoge mate van detail beschreven. SDM, de in Nederland bekendste lineaire methode, kent de fasen definitiestudie, functioneel ontwerp, technisch ontwerp, programmeren en testen (zie figuur 1). Elk van deze fasen, en vaak ook de activiteiten daarbinnen, wordt afgesloten met het opstellen van een mijlpaalproduct. Dit wordt gebruikt om het verder verloop van het project te bepalen. Iedere volgende fase start pas nadat het mijlpaalproduct is goedgekeurd en bevroren.
Rollen in deze methoden zijn daarnaast rechtstreeks gerelateerd aan de fasen van het proces en het opleveren van de bijbehorende mijlpaalproducten. Een definitiestudie wordt uitgevoerd door een analist, eventueel met gebruikers. Het functioneel ontwerp wordt vervolgens opgesteld door een functioneel ontwerper, het technisch ontwerp door een technisch ontwerper. Daarna bouwen ontwikkelaars het systeem en mogen testers de voltooide applicatie testen, voordat deze aan de gebruikers wordt opgeleverd.



Figuur 1: De basale fasen van een lineaire systeemontwikkelmethode.

jecten uiteraard niet ten goede. Een tweede consequentie is de onmogelijkheid van anticipatie op voortschrijdend inzicht. Naarmate systeemontwikkelprojecten vorderen ontstaat altijd meer begrip over de functionaliteit. Niet zelden ontdekt men gaandeweg nieuwe of vergeten functionaliteit. Maar dit inzicht is niet opgenomen in het mijlpaalproduct van eerdere fasen. En omdat die mijlpaalproducten als contract gelden voor de huidige fase, gaat voortschrijdend inzicht verloren.

UITSTROOM VAN KENNIS

Een ander probleem in watervalprojecten betreft de verschillende rollen. Zodra het functioneel ontwerp is opgeleverd, stroomt de functioneel ontwerper uit het project, zoals eerder de analist is uitgestroomd na het afronden van de definitiestudie. Niet in het mijlpaalproduct vastgelegde kennis over domein en systeem gaat nu verloren. Projectmedewerkers die een volgende fase invullen, moeten eerst worden ingewerkt in het project. Niet alleen kost dit tijd, het gaat ook ten koste van de kwaliteit. Alle input die een technisch ontwerper heeft om zijn mijlpaalproduct op te leveren, is schriftelijke documentatie. Hierop moet de technisch ontwerper een transformatie naar pseudocode baseren. En ondanks alle goede wil van zowel de functioneel als de technisch ontwerper, ontstaan hier inconsistenties en afwijkingen van het oorspronkelijke functioneel ontwerp.

Uiteindelijk wordt het opgeleverde systeem getest. De rol van testers is niet eenvoudig. Een project is hetzij zo uitgelopen dat voor testen nauwelijks tijd is of dat tests worden uitgevoerd als het systeem al gereed is. Indien testers in dergelijke gevallen nog fouten aan het systeem ontdekken, wordt ze dat niet in dank afgenomen.

ITERATIEVE SYSTEEM-ONTWIKKELMETHODEN

Eind jaren tachtig komen de tekortkomingen van lineaire methoden aan het licht. Vooral het gebrek aan anticipatie op voortschrijdend inzicht zet methodologisten aan tot het ontwikkelen van nieuwe systeemontwikkelmethoden. Belangrijk kenmerk van deze methoden is het veranderde proces. De fasen worden herhaald uitgevoerd in cycli, meestal drie tot zes. Tijdens iedere cyclus passeren alle fasen van het traditionele systeemontwikkelproces. Het systeem evolueert gedurende deze cycli tot het uiteindelijke resultaat. Voordeel van dergelijke methoden is dat veel beter wordt geanticipeerd op voortschrijdend inzicht. Voorts worden technieken geïntroduceerd als timeboxing en prototyping, waarmee systeemontwikkelprojecten beter te beheersen zijn. Bovendien kennen deze methoden meer aandacht voor samenwerking tussen de verschillende rollen. Dit leidt tot vereenvoudigde overdracht tussen gebruikers, ontwerpers en ontwikkelaars, hetgeen bijvoorbeeld tot uitdrukking komt in het toepassen van multidisciplinaire workshops.

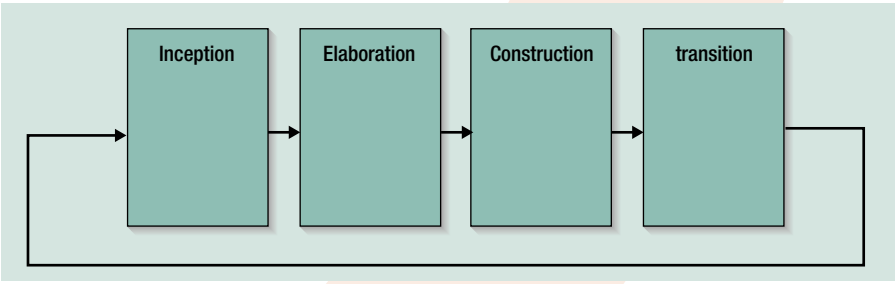
Iteratieve systeemontwikkelprojecten bieden diverse voordelen. Wie in iteraties werkt, kan beginnen met de belangrijkste functionaliteit. Knelpunten worden zo sneller geïnventariseerd en aangepakt. Vooral omdat ook testen integraal deel uitmaakt van iedere iteratie. In plaats van een *big bang* aan het eind van projecten, garanderen iteraties regelmatige en tijdige opleveringen, waardoor het vertrouwen groeit en het eindresultaat gemakkelijker wordt geaccepteerd. Iteratieve projecten gaan beter om met veranderende en nieuwe functionaliteit. Bij iedere nieuwe iteratie wordt steeds vastgesteld wat de belangrijkste nog resterende functionaliteit is. Nieuwe en veranderende functionaliteit wordt hierbij eveneens meegenomen.

Belangrijkste voordeel is wellicht het faciliteren in omgaan met veranderingen

IAD EN RUP

Tot deze generatie horen methoden als Iterative Application Development (IAD) en Rational Unfied Process (RUP). IAD is gericht op het opleveren van pilots en kent drie fasen, die iteratief worden doorlopen. In de definitiestudie wordt het pilotplan opgesteld op basis van doelen, beperkingen en eisen. Vervolgens worden de in het plan gedefinieerde pilots ontwikkeld. De resulterende pilot wordt ten slotte ingevoerd.

RUP is een samentrekking van verschillende objectgeoriënteerde methoden, verzameld door de *amigo's* Rumbaugh, Booch en Jacobson. Deze laatste leverde vanuit zijn methode Objectory de meeste input. RUP kent cycli die worden uitgevoerd in vier fasen (zie figuur 2), beginnend met de uitwerking van een idee tijdens *inception* en eindigend met de oplevering van een nieuwe versie van het sys-



Figuur 2: Fasering Rational Unified Process.

teem. Hierin spelen de modelleertechnieken van UML een belangrijke rol.

Opvallend aan beide methoden is de beschrijving van vele taken en activiteiten. Met de methoden is gepoogd een grote variëteit aan systeemontwikkelprojecten te bedienen. Dit heeft ze topzwaar gemaakt en in de praktijk lastig implementeerbaar. De methoden moeten voor ieder nieuw project of organisatie worden gestript. IAD biedt al vier strategiën om projecten uit te voeren. Van RUP moet een *development case* worden ontwikkeld, een deelverzameling van activiteiten en op te leveren producten. Het samenstellen van zo'n development case is verre van eenvoudig. Reden voor veel organisaties om RUP in zijn geheel te implementeren. Voorwaar geen sinecure.

Het Agile Manifesto

Het Agile Manifesto beschrijft de kenmerken van de nieuwe generatie methoden voor systeemontwikkeling. In het kort de waarden en bijbehorende principes van dit manifest:

- *Individuals and interactions over processes and tools*. Projecten draaien om mensen. De projectomgeving wordt zo ingericht dat projectmedewerkers optimaal kunnen functioneren, in een aanvaardbaar tempo.
- *Working software over comprehensive documentation*. Oplevering van waardevolle software is dé driver van projecten. Hoewel ook de documentatie van belang blijft, is die ondergeschikt aan het opleveren van de applicatie. Face-to-face communicatie is effectiever dan het over de muur gooien van mijlpaalproducten. Begrijpen is belangrijker dan vastleggen!
- *Customer collaboration over contract negotiation*. De hoogste prioriteit in projecten krijgt het tijdig en frequent opleveren van voor de klant waardevolle software, bij voorkeur in korte iteraties. Nauwe samenwerking tussen klant en ontwikkelaars is essentieel.
- *Responding to change over following a plan*. Agile-processen faciliteren het omgaan met veranderende eisen en proberen niet deze te voorkomen.

AGILE-METHODEN

Eind jaren negentig raakt het vakgebied opnieuw in een stroomversnelling wanneer massaal webapplicaties worden ontwikkeld. Vaak geldt hierbij een ultrakorte time-to-market. De bestaande systeemontwikkelmethoden, ook de iteratieve, blijken niet geschikt om dergelijke kleine, snelle projecten uit te voeren. Overal ontstaan nieuwe initiatieven, zoals DSDM, Extreme Programming en Ordina Smart. Uit onvrede met de manier waarop systeemontwikkeling plaatsvindt, schrijft een groep experts begin 2001 het Agile Manifesto, dat kenmerken beschrijft van de nieuwe generatie.

Kenmerken voor deze nieuwe methoden is bovenal dat zij iteratief zijn. Projecten doorlopen twee of meer iteraties, waarbij na iedere iteratie een nieuwe versie van het eindproduct wordt opgeleverd. Hetzij doordat nieuwe functionaliteit is ontwikkeld of doordat bestaande functionaliteit verder is gedetailleerd. Bij de start van een iteratie worden steeds de belangrijkste resterende eisen bepaald. Deze worden in de komende iteratie geïmplementeerd.

De methoden maken gebruik van *timeboxing*. Een timebox is een vaste periode waarin wordt gewerkt aan van te voren vastgestelde doelen. Timeboxen duren zo'n twee tot zes weken. Als de doelstellingen in traditionele systeemontwikkeling niet worden gehaald wordt meer tijd uitgetrokken of worden extra projectmedewerkers aangetrokken. In Agile-methoden betekent het niet halen van doelstellingen vooral dat er functionaliteit over de rand valt. Doordat tijdens elke iteratie de belangrijkste resterende functionaliteit wordt gerealiseerd, valt aan het eind van een project per definitie slechts de onbelangrijkste functionaliteit af.

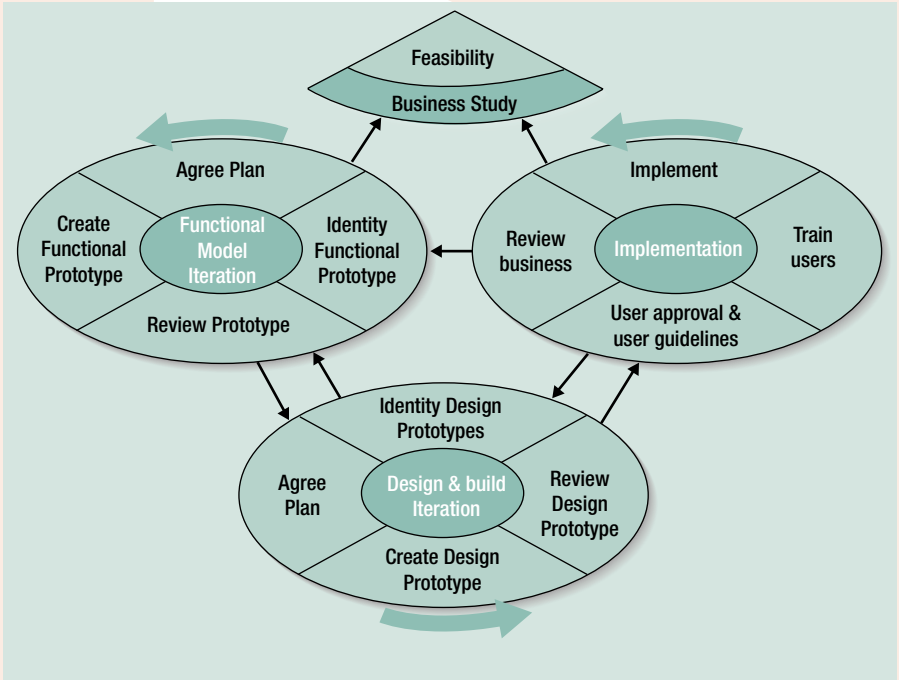
Gebruikersparticipatie is essentieel. Gebruikers maken deel uit van het projectteam en stellen bijvoorbeeld de prioriteiten. Dit vergroot de acceptatie van het eindresultaat. Terwijl vooral projectmanagers zich hierover blijven verbazen, blijkt dat gebruikers nauwelijks moeite hebben met het wegvallen van functionaliteit. Timeboxing garandeert ze de oplossing die zoveel mogelijk aan hun -wellicht veranderende- wensen tegemoet komt. En op tijd!

De beperkte scope van een timebox bevordert de betrokkenheid van de projectmedewerkers. Beslissingen worden genomen door het multidisciplinaire projectteam. Timeboxing zorgt voor een behapbaar tempo. Opvallend is dat dergelijke projecten nauwelijks overwerk kennen. Het verschil met traditionele projecten zit in wat wordt opgeleverd. In traditionele projecten moet altijd *alles* worden opgeleverd, in Agile-systeemontwikkelprojecten valt eventueel functionaliteit over de rand.

Veel werk in dit soort methoden wordt uitgevoerd in multidisciplinaire workshops. Op te leveren producten, zoals use case-diagrammen, worden *tijdens* workshops opgesteld. Communicatie gaat immers boven documentatie. Doordat alle betrokkenen deelnemen aan de workshops is afstemming achteraf overbodig. Iedereen krijgt zo bovendien eenzelfde referentiekader. Deze samenwerking verbetert zodoende de kwaliteit van de producten.

EXTREME PROGRAMMING

Ten slotte is er een belangrijk verschil met traditionelere iteratieve methoden als RUP en IAD. Agile-methoden zijn zonder uitzondering uitermate concreet en pragmatisch. Extreme Programming (XP) is het beste voorbeeld. XP is eind jaren negentig ontwikkeld door goeroe Kent Beck. XP onderscheidt slechts twee rollen: de klant en de ontwikkelaar. De klant schrijft *user stories*, korte verhaaltjes, waarin zijn wensen worden vastgelegd. Tijdens de *planning game* worden de prioriteiten vastgesteld, waarna de ontwikkelaars aan de slag gaan. In ultrakorte cycli wordt de applica-



Figuur 3: Het proces van DSDM.

tie ontwikkeld, waarbij het testen van de functionaliteit vaak geautomatiseerd gebeurd. XP kent nauwelijks ontwerpactiviteiten en hanteert het ontwikkelen van tests eigenlijk als ontwerp. Dit principe wordt *test first design* genoemd.

XP is van de nieuwe generatie het meest recht voor zijn raap. De ver doorgevoerde pragmatiek maakt de methode in grotere projecten echter lastiger toepasbaar dan bijvoorbeeld DSDM.

DSDM

De Dynamic Systems Development Methode (DSDM) is ontwikkeld voor kortcyclische projecten door een consortium van bedrijven. Ontwikkeling van applicaties vindt plaats in één team, waarin technische en bedrijfskennis is gecombineerd. DSDM kent twee eenmalige, inleidende fasen: haalbaarheidsstudie en bedrijfsstudie. Tijdens deze fasen wordt gekeken naar het bedrijfsmodel en de toepasbaarheid van de methode. Vervolgens kent de methode drie iteratieve fasen. De *functional model*-iteratie verfijnt de requirements. Tijdens *design & build* worden de requirements naar functionaliteit vertaald en gerealiseerd (zie figuur 3).

De overdracht van de gerealiseerde functionaliteit vindt plaats tijdens de laatste iteratie. DSDM geldt als onafhankelijke marktstandaard, en de belangstelling voor de methode neemt nog steeds toe.

ORDINA SMART

DSDM is echter een raamwerk. Dit betekent dat het raamwerk steeds opnieuw wordt ingevuld, afhankelijk van het project. De methode Ordina Smart is ontstaan vanuit de invulling van DSDM voor objectgeoriënteerde component based development projecten en webprojecten. Ordina Smart implementeert DSDM op pragmatische wijze. Zo wordt invulling gegeven aan het modelleren van eisen en functionaliteit, vooral op basis van UML. Use cases fungeren als rode draad door het project en vormen de basis voor functionele eisen, schattingen, ontwerp en tests. De methode biedt ook een generieke applicatie-architectuur, die de vertaling van eisen naar functionaliteit en code vergemakkelijkt. Veel van de *best practices* van DSDM en XP, zoals workshops, pair programming en test first design, maken deel uit de methode. De methode is gebruikt bij de allereerste Microsoft .NET-projecten in Nederland en is inmiddels bij diverse organisaties in gebruik, waaronder zelfs enkele concurrenten van Ordina.

De meeste organisaties in Nederland zijn nog altijd ingericht op het uitvoeren van traditionele systeemontwikkelprojecten. Het meest zichtbaar is dit aan functienamen als analist, functioneel ontwerper en technisch ontwerper. Implementatie van een nieuwe, Agile-werkwijze heeft meer voeten in de aarde dan het kiezen voor een methode en een ontwikkelomgeving. Zo kennen Agile-methoden heel andere rolverdelingen. Een deel van de verantwoordelijkheden van projectleiders is gedelegeerd aan het projectteam.

DE AGILE-ORGANISATIE

Ontwerpers, ontwikkelaars en ook testers -traditioneel vaak sluitpost- hebben meer verantwoordelijkheden en worden steevast eerder betrokken. Als ook het gebruik van UML in de methode wordt gepropageerd heeft dit consequenties voor het beheer van applicaties. De hiervoor verantwoordelijke afdelingen moeten kennis van de methode en van UML vergaren.

De culturele en organisatorische consequenties van het kiezen voor iteratieve methoden zijn groter dan wordt gerealiseerd. Agile-systeemontwikkeling kent gelukkig belangrijke voordelen, zoals vroegtijdig, geregeld en op tijd opleveren, grote betrokkenheid en verregaande samenwerking tussen business en IT, de mogelijkheden om projecten gecontroleerd uit te voeren in slechts enkele weken of maanden en de integatie van testen. Wellicht het belangrijkste voordeel is dat Agile-methoden, evenals iteratieve methoden, niet proberen veranderingen koste wat kost te voorkomen, maar faciliteren in het omgaan ermee. Meer en meer organisaties kiezen daarom voor nieuwe manieren van systeemontwikkeling. En da's maar goed ook, want de kwaliteit van systeemontwikkelprojecten is niet altijd om over naar huis te schrijven. ●

Sander Hoogendoorn (sander.hoogendoorn@ordina.nl) is Partner bij Ordina, gespecialiseerd in moderne systeemontwikkeling, en chief architect van de systeemontwikkelmethode Ordina Smart.