**Systeemontwikkeling is niet makkelijk. Het niet halen van doelstellingen in projecten is eerder regel dan uitzondering. Ik vergelijk ons vak graag met het maken van films. Iedere nieuwe film is een uitermate complex en geraffineerd samenspel tussen producer, regisseur, acteurs en tientallen andere stakeholders als scriptschrijvers, kostuumontwerpers en castingbureau's. Uiteraard met uiteenlopende belangen. Het aantal factoren dat bijdraagt aan het slagen of falen van een film is enorm. En als de film dan eindelijk klaar is, is het altijd nog maar de vraag of de film aanslaat bij het publiek. Lights? Camera? Action!**



# La vita è bella

## *Een modern epos over systeemontwikkeling*

### Op hoop van zegen

*Guido Pieters (1986) - Jonge zeeman Danny de Munk verzet zich tegen reder die zijn schepen verwaarloost.*

Het is augustus 2002. Hartje zomer. In de grote vergaderruimte vindt de kick off plaats van project Picasso. Enthousiast presenteert de projectleider zijn planning. De analyse neemt drie maanden in beslag. In november gaat het ontwerp van start. Medio januari 2003 leveren de ontwerpers op en aansluitend start de ontwikkeling van de nieuwe applicatie. Omdat de functionaliteit complex is, neemt deze fase van Picasso zes maanden in beslag, zo is geschat. Dit betekent dat de applicatie, na uitvoerig testen, in september 2003 wordt opgeleverd. Iedereen, van opdrachtgever tot ontwikkelaars, knikt instemmend. Een haalbare planning. Zo lijkt het. Inmiddels is het augustus 2003. Het ontwerp is bijna gereed. "We zijn op negentig procent", zeggen de ontwerpers. Uit hun stem spreekt meer hoop dan realisme. Het initiële enthousiasme is omgeslagen in berusting. De projectleider extrapoleert de uitloop. In het huidige tempo wordt de applicatie in mei 2004 opgeleverd. Ruim een half jaar later dan gepland. En het project is nog niet halverwege. Nog meer uitloop is niet denkbeeldig.

Herkent u deze - uiteraard denkbeeldige - situatie? Inmiddels is het vakgebied systeemontwikkeling al bijna een halve eeuw oud. Kunnen we nog steeds niet schatten? Waarom lopen projecten keer op keer uit? En waarom zijn projecten vrijwel altijd veel duurder dan verwacht? Een gemiddeld systeemontwikkelproject kost zo'n negentig procent meer dan oorspronkelijk begroot. Wereldwijd is slechts een-op-vier van projecten succesvol. Op tijd, binnen budget en met de gewenste functionaliteit en kwaliteit. Hoe komt dit? En belangrijker, kan het ook anders?

### History of the world, part I

*Mel Brooks (1981) - Komedie die de wereldgeschiedenis op de hak neemt.*

Een schrijnend voorbeeld van lineaire systeemontwikkeling trof ik bij een grote verzekeraar. Hier werd een applicatie getest waaraan ruim tien jaar was ontworpen en ontwikkeld. Het had requirements van tien jaar oud. Tien jaar. Kunt u zich nog herinneren wat u tien jaar geleden deed? De wereld is inmiddels flink veranderd. Waar gaan dit soort projecten eigenlijk mis?

**Mijlpalen** Één van de kenmerken van lineaire systeemontwikkeling is dat elke fase eenmalig wordt uitgevoerd. Iedere fase wordt afgesloten met een mijlpaal. Wanneer deze is bereikt eenmaal is bereikt, geldt het mijlpaalproduct - de analyse of het functioneel ontwerp - als bindend. In principe betekent dit dat het nooit meer wordt herzien. Contracten tussen opdrachtgever en opdrachtnemer worden dichtgetimmerd rond dergelijke mijlpaalproducten. Functionaliteit vast. Prijs vast. Einddatum vast. Zo verdwijnt al snel alle flexibiliteit uit het project. Omdat mijlpalen zo definitief zijn, gaan informatie-analisten, ontwerpers en ontwikkelaars tergend behoedzaam te werk. Er is geen weg terug en dus

**Kick off**

moet het op te leveren werk volledig en correct zijn. Alle functionaliteit moet boven tafel. Als een ontwerper immers niet alle functionaliteit beschrijft, wordt deze ook nooit gebouwd. Deze volledigheid is de belangrijkste reden voor het uitlopen van fasen in projecten. Want wanneer ben je eigenlijk volledig?

**VOORTSCHRIJDEND INZICHT** Mijlpalen dienen om te voorkomen dat er nieuwe wensen en eisen ontstaan in een project, bijvoorbeeld doordat gebruikers deze poneren. Gebruikers weten toch immers nooit wat ze willen? En dat betekent immers schuivende planningen. Toch? Mis. Voortschrijdend inzicht is er altijd. De wereld draait immers door. Het voorkomen van voortschrijdend inzicht is een utopie. In lineaire systeemontwikkeling leiden nieuwe inzichten steevast tot change request forms en meerwerk. Maar is voortschrijdend inzicht wel altijd dodelijk voor projecten?

**ROLLEN EN KENNIS** Wanneer een functioneel ontwerper klaar is met zijn mijlpaalproduct, gaat een technisch ontwerper er mee aan de slag. In de meeste projecten is dit moment tevens het moment dat de functioneel ontwerper het project verlaat. En al is deze laatste nog zo grondig te werk gegaan, nooit is alles compleet. Veel kennis zit immers in het hoofd van de functioneel ontwerper. Ik heb ooit een project gezien waarin één ontwerper in anderhalf jaar tijd alle functionaliteit in zeventien dikke rode mappen noteerde. Al bij de allereerste vraag van de ontwikkelaars moest de ontwerper met het schaamrood op de kaken erkennen dat dat er nu net niet in stond.

**GEEN FEEDBACK** Noch een informatie-analist, noch een ontwerper ziet zo ooit het eindresultaat van zijn werk. Mijlpaalproducten worden traditioneel 'over de muur' gegooid. Er is geen feedback mogelijk. Dit betekent dat het onmogelijk is te leren van gemaakte fouten. Een voorbeeld? In het ontwerp van een webapplicatie trof ik nog dit jaar het gebruik van functietoetsen F1 tot en met F12 aan.

**GEBRUIKERSBETROKKENHEID** Wanneer worden gebruikers betrokken bij een systeemontwikkelproject? Meestal helemaal in het begin. In interviews mogen gebruikers hun wensen en eisen kenbaar maken. Vervolgens verdwijnt het project voor tijden uit zicht. En pas helemaal aan het eind, anderhalf jaar na dato, als de applicatie wordt opgeleverd, wordt het resultaat trots gepresenteerd aan de verraste gebruikers.

Gebruikers hebben vaak zeer bruikbare ideëen, waar maar mondjesmaat gebruik van wordt gemaakt. Recent antwoordde een informatie-analist mij op mijn vraag wat de gebruikers van het ontwerp vonden: "Ik weet precies wat de gebruikers willen." Wat denkt u?

**TESTEN ALS SLUITPOST** Nog eentje. Testen. Waarom gebeurt dit alleen aan het eind van een project? Pas als de applicatie door de ontwikkelaars is opgeleverd wordt er getest. Op zich lijkt dit niet onlogisch. Er kleven echter twee bezwaren aan deze fasering. Naamate een fout later in een project wordt ontdekt, nemen de kosten voor het oplossen van deze fout bijna exponentieel op. Het is dan ook belangrijk fouten zo vroeg mogelijk te traceren. Een tweede bezwaar betreft het uitlopen van het project. Als eerdere fasen in een project te laat opleveren, komt de einddatum van het project in gevaar. En waar kan dan tijd worden teruggewonnen? Juist. "We testen wel twee weken, in plaats van de oorspronkelijke twee maanden."

**MANOUVREREN** Al met al is het niet vreemd dat projecten meer geld kosten dan gepland, te laat opleveren en te weinig functionaliteit opleveren. Om over kwaliteit maar niet te spreken. Maar ook projectleiders zijn creatief. Er is een aantal assen waarlangs gemanouvreerd kan worden om een project alsnog binnen de perken te houden.

**TIJD** De meest logische. Er is meer tijd nodig. Maar zoals bekend kost tijd ook geld. Het project wordt duurder. Bovendien zijn einddata in projecten vaak zeer belangrijk. Een energiemaatschappij moest onlangs een marketing-campagne annuleren, omdat de bijbehorende webapplicatie niet op tijd werd opgeleverd. Tijd is meestal belangrijker dan functionaliteit.

**MENSEN** Tijd beperkt? Dan maar meer mensen inzetten. Maar meer mensen inzetten kost meer geld. Meer mensen inzetten geeft bovendien een interessante uitdaging, die vaak wordt onderschat. Nieuwe mensen moeten worden ingewerkt. En wie doet dat? Precies. De

mensen die toch al onder hoge tijdsdruk werkten. De productiviteit in dergelijke projecten komt dan meestal hortend en stotend tot stilstand, als een oude stoomlocomotief op roestige rails. Brooks' law treedt in werking. Adding more resources to a late project makes it even later. Vrij vertaald: je kunt niet met negen vrouwen in één maand een baby krijgen.

**OVERWERK** Als er geen nieuwe mensen kunnen worden ingezet, moeten de huidige ontwikkelaars maar langer werken. Bij een van de grotere .NET-projecten die in ons land zijn uitgevoerd, moesten de ontwikkelaars om tijd goed te maken vier maanden lang vijf avonden in de week overwerken. En, schrik niet, een tijd lang bovendien in het weekend. Overwerk gaat alleen goed op korte termijn. Langdurig overwerk leidt onherroepelijk tot verminderde motivatie en dus tot lagere productiviteit en kwaliteit.

**FUNCTIONALITEIT** Is functionaliteit een as om langs te manouvreren? Het lijkt van niet. Minder functionaliteit wordt gezien als een aderlating. Misschien zelfs gezichtsverlies. "Alles is belangrijk," zei een opdrachtgever onlangs. "Maar niet alles is even belangrijk," riposteerde één van mijn collega's terecht.

Wat nu? Geen van de assen manouvreert prettig genoeg om een project uit het slop te trekken. Wat zijn de gevolgen? Organisaties zoeken naar meer zekerheid. Het initiële vertrouwen tussen opdrachtgever en opdrachtnemer verdwijnt als sneeuw voor de zon. Contracten worden verder dichtgetimmerd. Alle flexibiliteit verdwijnt uit het project. Sla de vakbladen er maar op na. Wekelijks berichten ze over vastlopende projecten, ruzies en rechtzaken. Wat nu?

## THE TOWERING INFERNO

*Irwin Allen (1974) - Bij opening van slecht geconstrueerd kolossaal gebouw breekt brand uit. Steve McQueen blust ternauwernood.*

Tijd voor een analyse. Waarom zijn we met zijn allen niet in staat om in harmonie, op tijd en binnen budget de gewenste functionaliteit op te leveren? Voor een belangrijk deel komt dit door de manier waarop tegen systeemontwikkeling wordt aangekeken. Als bekendste metafoor voor het ontwikkelen van software geldt het bouwen van huizen. De architect ontwerpt, en de aannemer voert uit. De architect doet het creatieve werk, de aannemer het productiewerk. Productiewerk is steeds vergelijkbaar. En dus kunnen reglementen worden voorgeschreven die bepalen hoe er wordt gebouwd. De bouwmetafoor lijkt zeer aannemelijk. Ze is jarenlang gebruikt voor het opzetten van procedures en het inrichten ontwikkelstraten en software factories.

De bouwmetafoor is echter vooral fout. Software ontwikkeling is geen productiewerk. Tot op het moment dat de ontwikkelaar zijn code compileert, is het ontwikkelen van software creatief werk. Zeker in objectgeoriënteerde ontwikkelomgevingen zijn er voor ieder probleem tenminste tien goede oplossingen, echter elk met eigen voordelen, maar ook consequenties. Het ontwikkelen van software is een continu proces van het creatief afwegen van alternatieven. Om het anders te stellen: systeemontwikkeling is niet het bereiden van de maaltijd, systeemontwikkeling is het bedenken van het recept.

**BEHEERSBAARHEID** Wat betekent dit voor de beheersbaarheid van projecten? Er is één belangrijke constatering over systeemontwikkelprojecten: geen twee projecten in de wereld zijn gelijk. Geen twee projecten behandelen exact dezelfde materie. Projecten hanteren verschillende ontwikkelomgevingen, verschillende programmeertalen. De complexiteit van projecten loopt uiteen van het maken van een electronische telefoonlijst voor de receptioniste tot het landen van voertuigen op Mars.

Hoeveel mensen maken deel uit van het project? Een bank nodigde mij eens uit een audit uit te voeren op een omvangrijk project. Er werkte honderdtwintig mensen aan het project. Na anderhalf jaar kende iedereen elkaar nog steeds niet. Wat voor type mensen maken deel uit van een project? Geen twee projecten kennen exact dezelfde bezetting. En zelfs als een team intact wordt gehouden voor verschillende projecten; mensen verschillen van dag tot dag. Ze stappen met het verkeerde been uit bed, worden zwanger, ziek, zijn eigenwijs of zijn de avond te voor flink doorgezakt.

Maar we hadden het over werkwijzen van projecten. Als geen twee projecten hetzelfde zijn, wat betekent dat voor de te hanteren werkwijze in projecten? Simpel. Er is geen one-size-fits-all.

**Een gemiddeld functioneel ontwerp**

**Ontwerpers en ontwikkelaars ontwerpen samen**

## Metropolis

*Fritz Lang (1929) - Klassieke futuristische film.*

Ruim dertig jaar geleden was het vakgebied systeemontwikkeling nog nieuw. Er dus was er nog geen algemeen gebruikte werkwijze. Totdat, begin jaren zeventig, lineaire systeemontwikkeling zijn intrede deed. Voor het eerst kenden projecten een vaste, schijnbaar beheersbare fasering. Analyse, globaal ontwerp, gedetailleerd ontwerp, bouw, testen en documenteren. Nu, dertig jaar later, kennen en hanteren vrijwel alle projecten een dergelijke werkwijze. Helaas niet altijd met evenveel succes. Gelukkig is dit jonge vakgebied nog altijd in ontwikkeling. Met enige regelmaat van de klok ontstaan er nieuwe technieken, nieuwe methoden, nieuwe gereedschappen en zelfs nieuwe paradigma's. Steeds wordt het vakgebied iets volwassener. Ook de werkwijzen van projecten zijn aan ontwikkeling onderhevig. Gelukkig maar.

**Nauw samenwerkend team**

Vooral de laatste jaren is het vakgebied onder invloed van een groot aantal factoren in een stroomversnelling terecht gekomen. De wereld van systeemontwikkeling ziet er anders uit dan we tot nu toe aannamen. Lineaire systeemontwikkeling is niet zaligmakend. Zo ontstaat er eind jaren negentig een nieuwe golf aan systeemontwikkelmethoden. Tijd voor modernisering en professionalisering van onze systeemontwikkeling.

## Once upon a time in America

*Sergio Leone (1984) - Epische film met Robert De Niro.*

**Agile manifesto** In februari 2001 komt een groep van zeventien vertegenwoordigers van diverse moderne methoden bij elkaar in een sneeuwhut in Utah. Naast skiën en eten, nemen de zeventien de tijd om vast te stellen hoe moderne systeemontwikkeling er uit zou kunnen zien. Het resultaat is vastgelegd in het agile manifesto. Dit agile manifesto zet een viertal aspecten van systeemontwikkeling op scherp.

- *Mensen*: de mensen en hun interactie is belangrijker dan het volgen van een specifieke methode of het gebruik van een specifiek gereedschap.
- *Software*: het opleveren van de applicatie is belangrijk dan het schrijven van lijzige mijlpaalproducten.
- *Samenwerking*: de samenwerking met de klant is belangrijker dan het gedetailleerd uitonderhandelen en uitkauwen van contracten.
- *Verandering*: het is belangrijker dat projecten kunnen omgaan met veranderingen dan het plan gedetailleerd wordt gevolgd.

De brede generatie van systeemontwikkelmethoden die deze aspecten onderschrijven, worden de agile methoden genoemd. Agile als in vlug en lenig. Zoals een turner.

## Total recall

*Paul Verhoeven (1990) - Arnold Schwarzenegger krijgt zijn eigen herinneringen terug. De wereld ziet er heel anders uit dan hij dacht.*

Het agile manifesto beschrijft slechts de contouren van de werkwijze van projecten. Er zijn diversen methoden die de kenmerken van dergelijke werkwijzen beschrijven. Wat kenmerkt deze agile methoden nu eigenlijk?

**Betrokkenheid van gebruikers** Net als het dagelijks leven verbetert ook de kwaliteit van applicaties van feedback. Gebruikers zijn meer dan ooit betrokken bij projecten. Zelfs medeverantwoordelijk voor het resultaat. Gebruikers geven continu feedback. Dit verhoogt niet alleen de kwaliteit, maar ook de acceptatie door de gebruikers. Per definitie past de geboden functionaliteit beter bij hun wensen en eisen. Projecten

maken middels deze gebruikersbetrokkenheid bovendien gebruik van het altijd aanwezige voortschrijdend inzicht, in plaats van het uit te bannen.

**SAMENWERKING** Meer dan ooit is een systeemontwikkelproject een samenwerkingsverband. Tussen klant en team. Tussen opdrachtgever en opdrachtnemer. Testers en gebruikers. Tussen ontwerpers en ontwikkelaars. Ontwikkelaars en testers. Geen mijlpaalproducten meer over de muur. Een voorbeeld? Ontwerpers en ontwikkelaars stellen samen een ontwerp op. Zo is er altijd een soepele overdracht van kennis. Stel de mens in het project centraal, niet het product. Dit maakt de traditionele volledigheid van mijlpaalproducten veel minder belangrijk.

Samenwerking vindt ook zijn weerslag in contractonderhandelingen. Niet meer het dichttimmeren van specificaties, die daarna door alle partijen met voeten worden getreden. Creëer vertrouwen. Een opdrachtgever vertelde tijdens een evaluatie: "Dit is het eerste project dat ik hier meemaak dat een joint venture is tussen business en IT."

**OPEN COMMUNICATIE** Projecten verzanden regelmatig in politieke steekspellen. Zeker als er 'zwarte pieten' worden uitgedeeld. Agile methoden stimuleren open communicatie. Geen contractonderhandelingen tot de dood er op volgt. Geven en nemen. Vermijdt gekonkel.

**KORTCYCLISCH** Agile methoden kennen korte iteraties, variërend van twee tot zes weken. Tijdens een iteratie werkt het team aan de requirements met de hoogste prioriteit. Aan het eind van een iteratie wordt steeds een deel van de applicatie opgeleverd. Ontworpen, ontwikkeld, getest en geaccepteerd. Dit maakt maximale feedback mogelijk. Het steeds opnieuw prioriteren zorgt ervoor dat het team altijd aan de belangrijkste resterende requirements wrekt. Zo is er ruimte voor nieuwe requirements of het laten vervallen van bestaande. Voortschrijdend voorzicht in optima forma.

Het team levert iedere drie weken iets op. De klant ziet letterlijk de voortgang. "We zijn op tachtig procent van het functioneel ontwerp," vertelt een waterval-ontwerper. Maar hoeveel is dat dan, tachtig procent? Kort cyclisch werken is concreet. "Er zijn zesendertig van de vijfenzestig use cases af." Proeft u het verschil?

**MINIMALISEREN DOCUMENTATIE** Natuurlijk is documentatie belangrijk. Het is echter niet het voornaamste doel van een systeemontwikkelproject. Want dat is het opleveren van de applicatie. Agile methoden stellen dit dan ook centraal. Minimaliseer de documentatie. Stel documentatie alleen op als deze waarde toevoegt aan het project. Niet omdat de gehanteerde werkwijze dit voorschrijft.

**VOORDELEN** Wat zijn nu de voordelen van zo'n werkwijze? Er zijn er legio.

- *Pragmatiek*: Allereerst de pragmatiek. Projecten zijn afgeslankt en ontdaan van overtollig vet. Daar waar voor de meeste watervalprojecten een jaartje Weight Watchers geen overbodige luxe is. Er worden geen overbodige bijproducten gerealiseerd. Het doel van een project is het opleveren van software; niet het schrijven van allerhande documenten die ooit verplicht zijn gesteld.
- *Lage kosten*: Er wordt precies dat gerealiseerd, dat nodig is voor de klant en de gebruikers. Niets minder en niets meer. Dit maakt projecten goedkoper.
- *Op tijd*: Anders dan in traditionele projecten is de einddatum heilig. Projecten leveren liever iets minder functionaliteit op, dan dat ze de einddatum overschrijden. Het steeds opnieuw prioriteren garandeert dat als niet alle functionaliteit is gerealiseerd op de einddatum, alleen de franje resteert. Enkele jaren gelden coachte ik een project waar op deze manier wat functionaliteit overboord dreigde te gaan. Ik stelde de verontruste gebruikers gerust door te stellen dat er na oplevering een tweede project kon worden gestart, met als doel het opleveren van de resterende functionaliteit. Zo'n project is nooit gestart.
- *Hoge acceptatie*: De functionaliteit is iteratief optimaal afgestemd op de belanghebbenden. De acceptatie van de applicatie door de gebruikers is nagenoeg zeker. Waarom? Ze zijn voortdurend betrokken en bovendien medeverantwoordelijk voor het project.

**KEERZIJDE** Natuurlijk is er ook een keerzijde aan de medaille. Wat betekent agile systeemontwikkeling nu voor uw organisatie?

**Workshop waarin use cases worden gemodelleerd**

**Direct zichtbare voortgang**

- *Verandering*: Agile methoden kenmerken zich door samenwerking en het delen van verantwoordelijkheden. Projecten zijn weinig hiërarchisch. Er wordt in projecten meer beroep gedaan op de capaciteit van medewerkers. Samenwerken betekent van alle markten thuis zijn. Gespecialiseerde medewerkers worden generalistischer ingezet. Projectleiders hebben veel eerder een faciliterende rol, niet een voorschrijvende. Ontwikkelaars en testers uit het verdomhoekje! Voor veel organisaties is dat nog ver van hun bed. Mijn advies? Zet een pragmatisch verandertraject in en laat u ondersteunen door ervaren agilisten.
- *Contracten*: Vertrouwen is niet ieder project gegeven. Vertrouwen moet groeien. Opdrachtgever en opdrachtnemer moeten werken aan hun relatie. Start simpel. Selecteer een door u vertrouwde leverancier met parate kennis en doe samen uw eerste agile project.
- *Grote projecten*: Een vraag die veel gesteld wordt is of agile methoden ook in grote projecten toe te passen zijn. Het antwoord is ja. Stel uzelf niet de vraag of een agile methode wel past. Stel uzelf de vraag hoe agile u kunt zijn in uw projecten. Bovendien, zijn grote projecten nu echt nodig? Grote projecten zijn vaak vooral status. Houdt projecten zo klein mogelijk. Laat u eens afschrikken door de fantastische Chaos Chronicles van de Standish Group.

## The good, the bad and the ugly

*Sergio Leone (1966) - Drie revolverhelden op zoek naar een verborgen fortuin. Clint Eastwood is de good guy. Lee van Cleef en Eli Wallach leggen het af.*

Inmiddels is er een hele nieuwe generatie van systeemontwikkelmethoden. In de Verenigde Staten aangevoerd door extreme programming. In Nederland en Groot Brittanië door DSDM. Hier volgt een kort en bondig overzicht van de heden ten dagen gangbare methoden. Hoe agile zijn deze verschillende methoden eigenlijk? En hoe en wanneer zijn ze bruikbaar?

**Extreme programming (XP)** De agile methode extreme programming leunt op vier kernwaarden: communicatie, eenvoud, feedback en moed. Nagenoeg de enige rollen zijn die van klant en ontwikkelaar. De klant noteert requirements in user stories. Dit is de eenheid van werk in extreme programming. Bij de start van iedere iteratie stelt de klant vast welke user stories worden ontwikkeld. De ontwikkelaars realiseren de user stories. De klant test en accepteert.

De methode kent buitengewoon concrete best practices als pair programming, continue integratie van nieuwe functionaliteit en test first design. Extreme programming is het geesteskind van Kent Beck en voor het eerst uitgeprobeerd in een project bij Chrysler. In de Verenigde Staten neemt de populariteit van deze agile methode snel toe. Nederland en België vertonen een licht stijgende belangstelling.

Laat u niet afschrikken door de naam. Extreme programming werkt het best in teams met gedisciplineerde en volwassen ontwikkelaars. Vooral in zwang bij technici.

**DSDM** Nog een methode met een verkeerde naam. Nee, DSDM heeft niets van doen met SDM. DSDM is in Groot Brittanië ontwikkeld door het DSDM Consortium, onder leiding van Jennifer Stapleton. De methode is voortgekomen uit rapid application development, niet uit lineaire systeemontwikkeling.

DSDM stoelt op negen principes, waaronder gebruikersbetrokkenheid, empowered teams, het regelmatig opleveren van producten en geïntegreerd testen. Van de agile methoden kent DSDM de meeste rollen. Alles in DSDM is omkeerbaar. Dit vertaalt zich in een proces met twee eenmalige fasen, feasibility study en business study, en drie iteratieve fasen: functional model iteration, design & build iteration en implementatie. Vanuit ieder van deze fasen is terugkeren naar de vorige mogelijk. Bekend zijn ook de begrippen timeboxing en MoSCoW, een set richtlijnen om te prioriteren.

DSDM presenteert zich als een raamwerk voor systeemontwikkeling. Een brede focus. In de praktijk betekent dit dat steeds opnieuw invulling aan proces en producten wordt gegeven. Dit is flexibel, maar ook tijdrovend. Meer openheid vanuit het DSDM Consortium kan bovendien geen kwaad.

**Smart** De methode Smart is de enige methode in dit rijtje van puur Nederlandse makelij. Smart is ontwikkeld door Ordina en combineert aandacht voor mensen, modelleren, applicatiearchitectuur, ontwikkelen en testen in een pragmatisch aanpasbaar proces. Smart is vrij beschikbaar en kent een naadloze integra-

tie vanaf het modelleren van requirements en functionaliteit, via applicatiearchitectuur tot het ontwikkelen, testen en accepteren van de applicatie. Hierbij gelden use cases (uit de modelleertaal UML) als eenheid van werk, vergelijkbaar met user stories in extreme programming en prioritised requirements in DSDM. Smart kent slechts een beperkt aantal rollen.

**Andere agile methoden** Er zijn veel meer agile methoden, zoals Crystal, Scrum en feature driven development. Deze kennen gelijksoortige processen, rollen en best practices als de hierboven genoemde agile methoden. Geen van deze methode heeft echter een uitgebreide achterban in Europa. Naast de agile methoden is er een groeiende verzameling aan aansluitende ideëen in het vakgebied. Interessant zijn bijvoorbeeld lean development, agile modeling en de pragmatic programmers.

**Rational Unified Process (RUP)** Een eerste vreemde eend in de bijt. Voor de duidelijkheid: Rational Unified Process is geen agile methode. Grondlegger Ivar Jacobson beschouwt de methode als de grootste verzameling kennis over software engineering in de wereld. Het Rational Unified Process kent vier fasen: inception, elaboration, construction en transition. Dwars daarop kent de methode negen workflows, van requirements management tot beheer. De matrix van fasen en workflows omsluit een uitgebreid netwerk van vele activiteiten. De methode onderkent ruim veertig rollen. Het Rational Unified Process wordt geleverd door Rational, inmiddels onderdeel van IBM. De methode wordt in Nederland vooral toegepast door grote organisaties in langlopende, complexe projecten. Toen ik Ivar Jacobson ooit vroeg naar zijn mening over agile development, antwoordde de Zweed: "*RUP is agile enough.*" Dat kan zijn, maar om Rational Unified Process effectief toe te passen is tijdrovend kalibreren van de methode onontkoombaar.

**Prince2** Een tweede vreemde eend in de bijt: Prince2, de prof. dr. ir. Ackermans van deze opsomming. Prince2 is echter geen agile systeemontwikkelmethode. Het is zelfs geen systeemontwikkelmethode. Prince2 is een projectmanagementmethode. Geliefd onder projectmanagers, onbekend onder technici. Ideëen van Prince2, zoals het regelmatig herzien van de business case, zijn echter vaak te combineren met het agile gedachtengoed. Dat wil zeggen, mits Prince2 naar de geest wordt toegepast, en niet naar de letter. Stel documenten alleen op wanneer ze waarde toevoegen, niet omdat ze voorgeschreven zijn.

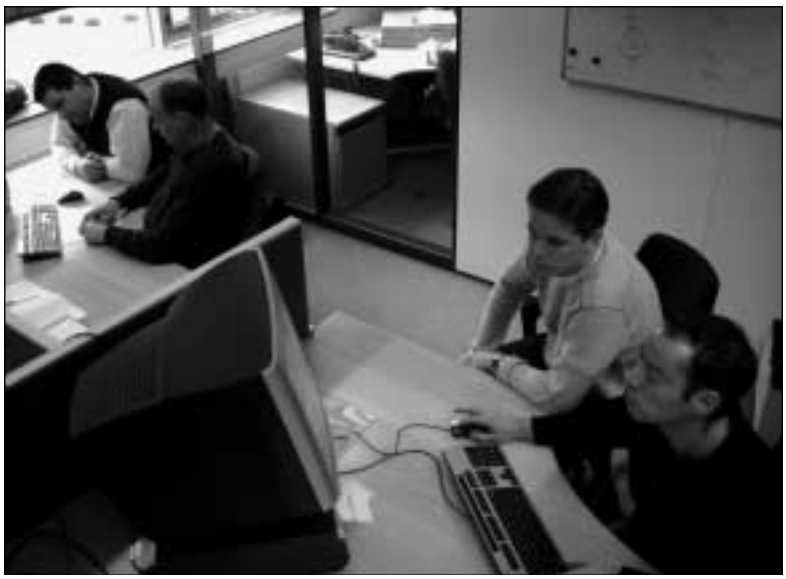
**Wie wat waar wanneer?** Er is geen one-size-fits-all. Geen van de bestaande methoden is de best passende voor al uw projecten. Een verlofadministratie is geen hypotheekaanvraagsysteem. Maar wanneer is welke methode toe te passen? Extreme programming is vooral toegepast in kleine projecten. Ruimere aandacht voor ontwerp en architectuur kan de methode echter breder toepasbaar maken. DSDM is - net zoals Smart - toegepast in zowel grote als kleinere projecten. Dat kan. DSDM laat veel ruimte, maar kent daardoor weinig handvatten. Rational Unified Process is vooral ingezet in grote, langlopende projecten. Prince2 tenslotte, wordt in systeemontwikkelprojecten meestal naast een van de systeemontwikkelmethoden gepositioneerd.

## The dirty dozen

*Robert Aldrich (1967) - Men on a mission.*

Hieronder volgt een dozijn best practices. Het zijn de best practices van Smart. Maar laat dat u niet van de wijs brengen. Dit dirty dozen geldt als een uitstekend hulpmiddel om ook andere projecten succesvol te laten zijn.

**Fit** Geen methode past altijd. Naarmate omvang en complexiteit van projecten toeneemt, neemt helaas ook de hoeveelheid overhead toe. Het is minder makkelijk communiceren. Alistair Cockburn noemt dit het toevoegen van ceremonie. Ook in agile systeemontwikkeling geldt dit. In eenvoudige projecten gebruik in minder modelleertechnieken dan in complexere projecten. Use cases, user interface diagram, activiteitendiagram - om te testen - en een klassediagram modelleer ik altijd, sequence diagrammen en componentendiagrammen alleen in omvangrijke en complexe projecten. Pas de werkwijze aan het project aan, niet het project aan de werkwijze. Wees schaars met het toevoegen van ceremonie. *Have just enough methodology.*



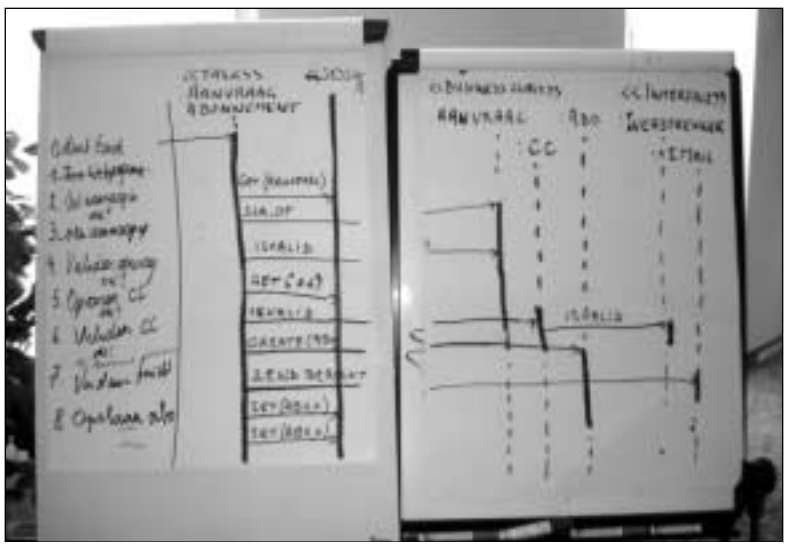
**Pair testing links, pair programming rechts**

**SAMENWERKING** Projecten slagen niet omdat de projectleider briljant is. Ook niet door de fantastische ontwikkelomgeving. Projecten slagen als er wordt samengewerkt. Ieder geslaagd project is een joint venture tussen opdrachtgever en opdrachtnemer. Zo'n project is herkenbaar aan uitstraling van het team. Zorg ervoor dat mensen zich thuis voelen in het project. Plaats het hele team in één ruimte. Zo dicht mogelijk bij opdrachtgever en gebruikers. Dit stimuleert samenwerken en communiceren. Lever regelmatig op. Bijvoorbeeld een use case per dag. Een regelmatig opleverend team voelt zich al snel succesvol.

**OPEN COMMUNICATIE** Open communicatie vergroot de betrokkenheid van de mensen in een project: vermijdt gekonkel, stimuleer overleg. Het is beter snel achter problemen te komen dan te laat. Organiseer bijvoorbeeld workshops. Kies een enkel doel. Het modelleren van requirements, het ontwerpen van de user interface of het evalueren van iteraties. Bereidt workshops goed voor. Bereik consensus. Dit voorkomt afstemming achteraf. Zorg voor dagelijks overleg in stand up meetings. Neem beslissingen gezamenlijk.

Open communicatie vergt een ander type projectleider. Geen autoriteit die het werk verdeelt, maar een facilitator. De projectleider organiseert de interactie tussen project en omgeving. De projectleider zorgt er voor dat het team aan de slag kan. Dat alle hardware werkt. De licenties geregeld zijn. De projectleider regelt afspraken, bijvoorbeeld met gebruikers of beheerders.

**GEDEELDE VERANTWOORDELIJKHEID** Verdeel de verantwoordelijkheden over het team. Maak gebruikers verantwoordelijk voor requirements en acceptatie. Houdt gebruikers en ontwikkelaars samen verantwoordelijk voor de opgeleverde use cases. Maak ontwikkelaars en testers verantwoordelijk voor het ontwerp. Verantwoording vergroot de betrokkenheid bij het project en maakt het team pro-actief. Niemand in het team is eigenaar van een deel van het ontwerp en de code. Ontwerp en code zijn gemeenschappelijk eigendom. Als iets beter of efficiënter kan, doet het. Refactor waar nodig.



**Ultiem pragmatisch modelleren: twee flipovers**

**FEEDBACK** Des te sneller het project feedback ontvangt, des te sneller het is bij te sturen. Zorg voor regelmatige feedback van gebruikers, liefst dagelijks. Organiseer iedere dag stand-up meetings met de gebruikers, kort en simpel. Wat is er gisteren gedaan? Wat doen we vandaag? Stand-up meetings voorkomen vaak alle andere vergaderingen in een project. Bijvoorbeeld de oersaaie wekelijkse voortgangsvergadering op maandag. "Wil iemand nog iets zeggen over pagina drie van de notulen? Nee? Dan gaan we nu door naar de actiepunten." Door iedere dag een klein, maar zichtbaar aspect van de applicatie op te leveren, zijn gebruikers en opdrachtgever optimaal in staat bij te sturen. Feedback creëert vertrouwen. Vertrouwen creëert acceptatie.

**ITERATIEF** Werk in iteraties. Des te korter de iteraties, des te groter het effect. Immers, bij de start van een iteratie wordt bepaald welke requirements tijdens deze iteratie worden gerealiseerd. Inclusief nieuwe en veranderde requirements. Korte iteraties betekent meer invloed van de klant op het project. Meer feedback. Ik raad iteraties van langer dan acht weken af. De invloed van de klant is dan miniem. Het duurt te lang voordat nieuwe of veranderde requirements ingepast kunnen worden. Definieer iteraties bij voorkeur allemaal even lang. Dit zorgt voor stabiliteit en zekerheid. Bovendien zijn afspraken voor kick offs en evaluaties zo simpel vast te leggen voor de gehele duur van het project. Ik plan vaak al bij de start van een project alle evaluaties.

**TIMEBOX** Een timebox is een periode met een vaste start- en einddatum. In deze periode tracht het team gestelde doelen - use cases - te realiseren. Als niet alle doelen zijn gerealiseerd, zijn deze resterende wellicht over te dragen naar volgende timeboxes. Verleng nooit een timebox. Voer uw projecten uit in timeboxes. Dit impliceert een vaste einddatum. Timebox ook de iteraties. Dit maakt uw project planbaar. Prioriteer bij de start van een nieuwe iteratie steeds de resterende use cases. Realiseer in een iteratie steeds één use case tegelijk en niet allemaal tegelijk. Beter vijf use cases helemaal af, dan tien half. Als er al use cases over zijn aan het eind van de laatste iteratie, dan zijn dit de minst belangrijke. *Nice-to-haves*.

**VOORTGANG ZICHTBAAR MAKEN** Weet u hoe ver uw projecten zijn? "Ja, we zijn al op tachtig procent van het ontwerp." Maar wat is tachtig procent van het ontwerp? Hoe weet je nu wat tachtig procent is? Of is mis-

schien tachtig procent van de tijd op? Fred Brooks zei het al: "*Don't confuse effort for progress*."

Voortgang motiveert medewerkers. Maak de voortgang in uw project daarom direct zichtbaar. "Maar ik schrijf toch ieder week een voortgangsrapportage?", verzucht een projectleider. Maar voor wie?

Een duidelijke eenheid van werk is essentieel voor het meten van voortgang. Neem use cases. Op ieder willekeurig moment in een project is voortgang eenvoudig aan te geven. Tel het aantal gerealiseerde use cases. Het visualiseren van de voortgang kan ook anders. Noteer alle use cases op post-its. Hang ze in categoriën aan de muur. Bijvoorbeeld *New, In current iteration, Working on, Testing and Accepted*. Zo ziet iedereen hoe de vlag erbij hangt. Open communicatie.

**Tester en ontwikkelaar schrijven samen use cases en test cases**

**PAIR PROGRAMMING** Stimuleer het werken in paren. Een goed voorbeeld is pair programming. Twee steeds wisselende ontwikkelaars realiseren samen code. Maar ook het opstellen van plannen en testscenario's of het uitvoeren van de tests kan in paren. Het werken in paren lijkt contra-productief. Ik werkelijkheid levert het betere oplossingen, hogere kwaliteit, minder fouten en zelfs minder code. Werken in paren kost nauwelijks meer tijd. Het is bovendien leuker. Belangrijke bijkomstigheid? De kennis over het project, het ontwerp en de code raakt snel verspreid over het team. Het inwerken van nieuwe medewerkers is makkelijker. Dit beschermt het project tegen onverhoopte gebeurtenissen, zoals het wegvallen van een medewerker.

**MODELLEER PRAGMATISCH** Modelleren is effectiever dan beschrijven. Een model is beter onderhoudbaar dan een document met vergelijkbare inhoud. Minimaliseer de hoeveelheid documentatie in uw projecten. Maximaliseer de hoeveelheid niet gedaan werk in projecten. Zorg ervoor dat er altijd een whiteboard is. Whiteboards nodigen uit tot modelleren. Waarom schrijven we eigenlijk zulke lijvige ontwerpen? Niemand leest ze. Veel documentatie in projecten lijkt alleen te worden geschreven om ter zijner tijd een zondebok aan te kunnen wijzen. Een tijd geleden leende ik één van vier delen van een functioneel ontwerp uit een lopend linear project, als voorbeeld bij een lezing. Vier maanden later vond ik de map terug in mijn bureau. Het was niemand opgevallen. Het vastleggen van kennis is niet het belangrijkste. Het overdragen van kennis wel.

**ONTWERP REALISTISCH** Een applicatiearchitectuur is onontbeerlijk. Houdt het definiëren van uw architectuur uit de achterkamertjes en ivoren torentjes. Laat de applicatiearchitectuur ontwikkelen door het team. Met het domein en de ontwikkelomgeving in het achterhoofd. Sluit de architectuur aan op gebruikte modelleertechnieken. Houdt de applicatiearchitectuur zo eenvoudig mogelijk. Pragmatisch en realiseerbaar. Start het project liever met te weinig architectuur dan met te veel.

**EERST TESTEN** Testers zien de wereld anders dan ontwikkelaars. Ontwikkelaars gaan er vanuit dat alles goed gaat, testers zoeken uitzonderingen. Gebruik dit. Ontwikkelaars en testers ontwerpen samen de use cases. In paren. Stel direct de bijbehorende functionele en technische tests op. Realiseer pas daarna de use cases. Test de use case direct na realisatie. Het herstellen van een fout kost expontieel meer effort, naarmate de fout later een project wordt ontdekt. Een groot probleem in lineare systeemontwikkeling. Hier is iedere use case ontwikkeld op basis van de tests. Dit voorkomt heel veel fouten. Bovendien stimuleert dit uw testers enorm.

**NET ALS IN DE FILM?** Ik hoop dat dit epos u heeft geboeid. Het bood een blik achter de scherm van traditionele en moderne systeemontwikkelprojecten. Net als films zijn projecten alleen succesvol als er chemie is tussen producer, regisseur, acteurs en publiek. Het is nooit alleen de regisseur die de film succesvol maakt. Nooit alleen de acteurs. En zeker niet alleen het script! Het is uiteindelijk altijd het publiek dat bepaalt hoe lang uw film draait in hun theater. Cut!

*Dit artikel is de director's cut uit het boekje "Smart. Succesvol samenwerken in systeemontwikkeling", dat recent door Ordina is gepubliceerd.*

*Tekst en fotografie: Sander Hoogendoorn, partner Ordina (www.sanderhoogendoorn.com)*