implan

Systeemontwikkeling

Van waterval naar agile

Erik van der Heijden

implan
versie 1
maart 2013

INHOUDSOPGAVE

1. INLEIDING

Tot in de jaren negentig van de vorige eeuw werden IT-systemen vooral ontwikkeld met behulp van de zogenaamde *lineaire* of *watervalmethoden*. Doordat projecten die op basis van die methoden werden uitgevoerd tot problemen leidden, zowel financieel, kwalitatief als qua doorlooptijd, ontstond er een behoefte aan alternatieve systeemontwikkelingsmethoden. Vanuit die behoefte ontstonden de zogenaamde *incrementele en iteratieve methoden* voor systeemontwikkeling. Vanaf de publicatie door een groep software-ontwikkelaars van het *Manifesto for Agile Software Development* in 2001 wordt voor deze vorm van systeemontwikkeling in het algemeen de term *agile* gebruikt. Deze term zal hier ook gebruikt worden.

In de laatste twintig jaar zijn de *agile* methoden meer en meer dominant geworden. In de jaren negentig ging de meeste aandacht uit naar wat *Rapid Application Development* (RAD) genoemd werd. Op dit moment is er vrijwel geen vacature op het gebied van systeemontwikkeling te vinden waarin niet gevraagd wordt naar kennis van en ervaring met de moderne variant *Agile/Scrum*.

In de literatuur over incrementele systeemontwikkeling stelt de auteur steevast, dat die waterval-methoden niet meer voldoen aan de eisen van snelheid en flexibiliteit die in onze dynamische tijd en snel veranderende wereld gesteld worden aan de systeemontwikkeling, en dat die methode verantwoordelijk zou zijn voor allerlei problemen bij de software-ontwikkeling. Voor het gemak worden daarbij meestal alle “traditionele” benaderingen (dat wil zeggen: alle methoden van voor 1991 toen James Martin de term *rapid application development* voor het eerste gebruikte) een lineaire- of waterval-methode genoemd en op een hoop geveegd.

Toch zijn er wel wat vragen te stellen bij deze volledige afwijzing van de watervalmethoden en de even volledige omarming van de agile-systeemontwikkeling:

1. Welke problemen met systeemontwikkeling doen zich in de praktijk voor en welke oorzaken zijn daarbij te onderkennen?
2. In hoeverre worden deze problemen door de watervalmethode veroorzaakt of versterkt?
3. Wat zijn de kenmerken van de agile-methoden?
4. In hoeverre kan agile-systeemontwikkeling de gesignaleerde problemen bij systeemontwikkeling oplossen, of in ieder geval verminderen?

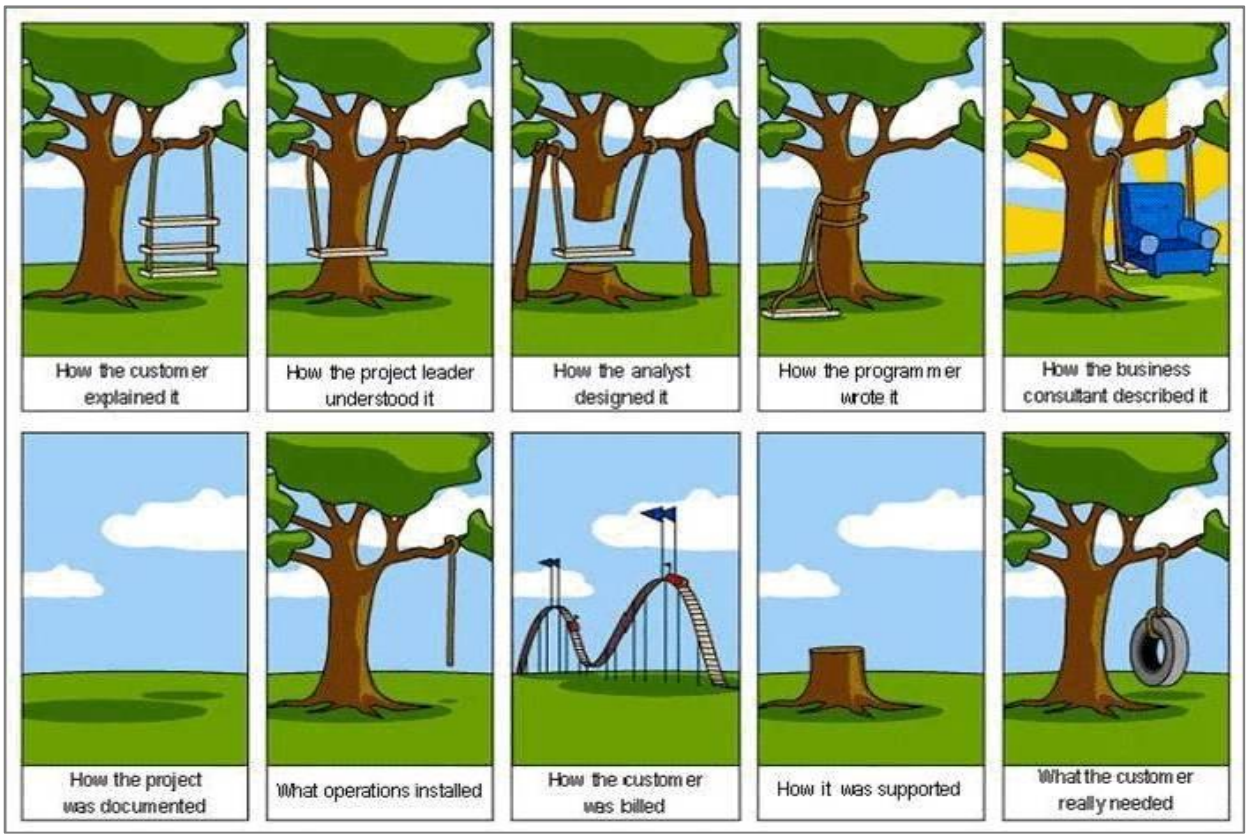
Op basis van de antwoorden op deze vragen kunnen conclusies getrokken worden met betrekking tot verbeteringen die van de agile-systeemontwikkeling verwacht mogen worden in vergelijking met de watervalmethoden en welke problemen zullen blijven bestaan of mogelijk juist ontstaan.

2. PROBLEMEN MET SOFTWAREPROJECTEN

Dat er in het verleden grote problemen zijn geweest bij de uitvoering van veel softwareprojecten, en dat die er ook nu nog zijn, zal niemand ontkennen die regelmatig met systeemontwikkeling te maken heeft, of heeft gehad. Vaak hoor je stellingen als: het ontwikkelen van software kost aan tijd en geld meestal 50% meer dan gepland en levert slechts zo'n 50% van de gewenste functionaliteit en kwaliteit. Los van de exacte juistheid van deze alarmerende getallen, dat niet iedereen helemaal tevreden is (was?), kan worden beschouwd als een *understatement*.

Allerlei deelproblemen zijn daarbij in laatste instantie te verdelen over drie hoofdproblemen:
- te lange doorlooptijd
- te duur
- onvoldoende kwaliteit en/of functionaliteit.

Figuur 1 – veel voorkomend probleem bij softwareontwikkeling (bron:www.opensw.wordpress.com)



Het moge duidelijk zijn, dat deze drie hoofdproblemen nauwe relaties met elkaar hebben. Onder het motto "een ongeluk komt nooit alleen", kun je volhouden, dat als in een project een van deze problemen ontstaat, de andere nooit ver uit de buurt zullen zijn.

Hieronder worden (in paragraaf 2.1 t/m 2.7) de mogelijke oorzaken van deze problemen over zes gebieden verdeeld. Hierbij blijft de eventuele relatie tussen de problemen en de gebruikte systeemontwikkelingsmethode zoveel mogelijk (soms is een probleem niet los te zien van de aanpak)

nog buiten beschouwing. Die komt aan bod in hoofdstuk 3 en 4. Daarin wordt besproken in hoeverre die methoden de problemen verminderen of versterken.

2.1. Onduidelijke of veranderende gebruikerseisen en -wensen

Vaak is bij het begin van een project niet helemaal duidelijk wat de gebruiker nu precies van het systeem verwacht. Daar komt nog bij dat de eisen en wensen gedurende het traject kunnen wijzigen, zeker als sprake is van een langlopend project. Hierbij is een organisatie bovendien (deels) afhankelijk van externe factoren (zie paragraaf 2.7). Onduidelijkheid met betrekking tot de eisen en wensen kan leiden tot onduidelijke of verkeerde afspraken en vervolgens tot misverstanden tussen opdrachtgever (business) en aannemer/uitvoerder (IT, intern of extern). Uiteindelijk is de kans groot dat het project een niet geheel gewenst, of zelfs ronduit ongewenst resultaat oplevert.

Als er onduidelijkheid bestaat over de eisen en wensen, bestaat er een gerede kans, dat de gebruiker en de ontwikkelaar die samen het systeem moeten maken, houvast zoeken bij het oude systeem (gesteld dat dat er is, uiteraard). Op zichzelf hoeft dat niet verkeerd te zijn, maar het oude systeem wordt niet voor niets vervangen en het gevaar bestaat, dat het nieuwe systeem in te veel opzichten een kopie van het oude wordt. Vaak wordt de bestaande technische oplossing en niet de in de toekomst gewenste effectiviteit en efficiëntie als uitgangspunt genomen.

Het bovenstaande heeft in eerste instantie betrekking op veranderingen op een relatief gedetailleerd niveau. Er kunnen echter op hoger niveau ook veranderingen in de reikwijdte van het project aangebracht worden. Het kan aantrekkelijk lijken, om ‘dat ook even mee te nemen nu we toch bezig zijn’, de zogenaamde *scope creep*. Op abstract niveau lijkt dat in een moeite mee door te gaan. Later blijkt op gedetailleerd niveau vaak dat veranderingen in de scope in veel gevallen toch complexer blijken dan in eerste instantie voorzien. Met alle consequenties van dien voor de beheersbaarheid en de doorlooptijd van het project.

2.2. Te veel aandacht voor details

Vaak verliezen projecten zich in details die voor de individuele betrokkene (gebruiker, ontwikkelaar) essentieel lijken, maar dat op het hogere project- of bedijfsniveau niet werkelijk zijn. De kosten/baten-verhouding wordt dan uit het oog verloren. Dat kan gebeuren zowel aan de kant van de klant/gebruiker (die bijvoorbeeld per se een bepaalde schermlayout gerealiseerd wil zien die technisch complex is en functioneel niet heel veel toevoegt) als aan de kant van de ontwikkelaar (die misschien veel tijd spendeert aan een technisch detail dat de klant helemaal niet zo relevant vindt).

In dit soort gevallen zou gewezen kunnen worden op de zogenaamde 80/20-regel: 80% van het systeem wordt gerealiseerd met 20% van de tijd en kosten. Consequentie: de laatste 20% van het systeem, de perfectie, kost 80% van de inspanning. Het is maar een vuistregel, maar het kan geen kwaad die altijd in het achterhoofd te hebben, zeker als éxtra tijd en/of geld gevraagd worden.

2.3. Gebrek aan kennis bij gebruiker en/of ontwikkelaar

In de meeste gevallen hebben de gebruiker en de ontwikkelaar begrijpelijkerwijs weinig kennis van elkaars discipline. Dat is dan ook de voornaamste reden dat zij samen aan een project werken. Een probleem ontstaat als zij eveneens te weinig kennis van de eigen discipline hebben. Als een gebruiker

te weinig materiekennis heeft, zal het systeem functionele lacunes en fouten vertonen. Een adequate ontwikkeling van een geautomatiseerd informatiesysteem is in een dergelijke situatie onmogelijk.

Als de ontwikkelaar te weinig kennis van systeemontwikkeling heeft, of de aanwezige kennis niet of te weinig gebruikt, zal het systeem in het gunstigste geval slecht gestructureerd zijn (en dus slecht onderhoudbaar en niet of nauwelijks herbruikbaar) en in het ongunstigste geval pas na heel veel extra tijd en geld, of zelfs helemaal niet, operationeel te krijgen.

Dit probleem lijkt een open deur, maar komt in de praktijk veel vaker voor dan je zou verwachten.

2.4. Onvoldoende (goede) communicatie

Vaak verloopt de communicatie tussen de medewerkers van een softwareproject zeer moeizaam. Zij hebben verschillende expertises en spreken in veel gevallen elkaars “taal” niet. Dit kan gemakkelijk leiden tot misverstanden, en daarmee tot fouten.

Deze fouten worden vaak pas laat ontdekt, doordat de verschillende talen ook tot gevolg hebben dat de door de ontwikkelaar opgeleverde documenten voor een gebruiker vaak onbegrijpelijk zijn. Een mondige gebruiker zal zeggen dat hij het niet snapt, maar ontelbare gebruikers gaan er van uit dat het aan hen ligt als ze het niet begrijpen. Zij durven vaak niets te zeggen uit angst voor dom te worden versleten en accepteren het document dan maar “op hoop van zegen” en vanuit de redenering: “de ontwerper zal z’n werk wel goed gedaan hebben en bovendien heb ik het toch al zo druk.”

Een andere oorzaak voor problemen, is de omstandigheid dat in veel softwareprojecten de communicatie geheel wegvalt in de periode tussen het opstellen van de functionele specificaties en het testen van het gebouwde systeem: de ontwikkelaar schrijft de technische specificaties, bouwt en test (als het goed is) het systeem en komt dan weer terug bij de gebruikers voor de acceptatietest. De gevolgen zijn zonneklaar: fouten in de functionele specificaties, in het technisch ontwerp of in de gerealiseerde programma’s komen pas tijdens het testen naar voren. Als het een beetje tegenzit kun je vervolgens bij de functionele specificaties, of zelfs bij de gebruikerseisen en -wensen opnieuw beginnen.

Overigens kunnen de communicatieproblemen al in een zeer vroeg stadium optreden, vaak al vóór het project begint: hoe vaak is het niet zo dat er geen of onduidelijke afspraken worden gemaakt tussen opdrachtgever aan de ene kant en opdrachtnemer en/of projectleider aan de andere. Bijvoorbeeld over de gebruikerseisen- en wensen, over de koppeling tussen die eisen en wensen en het beschikbare budget en de geplande doorlooptijd en over de handelwijze als er wijzigingen moeten worden aangebracht in de eisen en wensen. Formele *change control* ontbreekt vaak, of wordt niet of maar half gevolgd.

Behalve met communicatie tussen de opdrachtgever/gebruiker en de ontwikkelaar, heb je ook te maken met de communicatie tussen ontwikkelaars onderling. Spreken die wel dezelfde taal? Bestaat er een standaard ten aanzien van methoden en technieken voor systeemontwikkeling? Hiermee heb je niet alleen te maken als meerdere ontwikkelaars aan één project werken, maar ook als taken binnen een project, om welke reden dan ook, moeten worden overgedragen aan een andere ontwikkelaar.

2.5. Verkeerde verwachtingen bij (lijn)management / opdrachtgever

Ook tegenwoordig nog hebben veel (lijn)managers het idee dat ze alleen maar een opdracht hoeven te geven om een geautomatiseerd systeem te maken of te kopen en dat ze vervolgens wel horen wanneer het klaar is. Vaak gaan ze er van uit dat het in een paar weken “gepiept” is en dat de kosten wel mee zullen vallen, “we doen immers alleen maar dit of dat”. En zeker als het om een kant-en-klaar aangekocht systeem/pakket gaat, wordt vaak onderschat hoe groot de inspanning is om het systeem te implementeren, te configureren en in te richten. En dan heb ik het nog niet eens over het ontwikkelen van interfaces met andere systemen, een eventueel noodzakelijke dataconversie van het oude naar het nieuwe systeem of gebruikerstrainingen.

Als het gaat om systemen die operationele bedrijfsprocessen moeten ondersteunen, lukt het meestal nog wel om een redelijk systeem werkend te krijgen, omdat de gebruikers domweg niet zonder kunnen en het systeem er dus móét komen, maar met name veel *business intelligence*-projecten mislukken, doordat senior managers denken, dat dat mooie *dashboard* vanzelf verschijnt en zich niet bewust zijn van het feit dat ze zelf moeten aangeven wat ze precies willen zien en weten, en ook dat de basisgegevens voor die overzichten en metertjes wel eerst ergens moeten worden vastgelegd.

Vaak schrikt de opdrachtgever uit de *business* van de schatting van de doorlooptijd van het project en van de kosten die gemoeid zijn met de ontwikkeling en/of implementatie van de software. Meestal wordt dan de maker van de schatting, niet zelden de toekomstige projectleider, ter verantwoording geroepen. Zoveel geld en zoveel tijd? Kan dat niet wat minder? En uiteraard zonder concessies te doen aan de gewenste functionaliteit en kwaliteit! Als de projectleider dan niet al te sterk in zijn schoenen staat, zal hij al snel geneigd zijn om, als korte termijn oplossing, de planning dan maar wat aan te passen, om zijn opdrachtgever (niet zelden ook nog zijn baas) tevreden te stellen.

Als de planning vervolgens niet haalbaar blijkt, gaat de projectleider naar zijn opdrachtgever voor meer tijd en geld of wordt er functionaliteit of, op lange termijn meestal erger, onderhoudbaarheid ingeleverd. Net als bij onduidelijkheid met betrekking tot de gebruikerseisen en -wensen kan dit leiden tot onduidelijke of verkeerde afspraken en vervolgens tot misverstanden tussen opdrachtgever (business) en aannemer/uitvoerder (IT, intern of extern).

Ook op een ander vlak heeft de lijnmanager nogal wat invloed. Is hij bereid de bevoegdheid tot het nemen van beslissingen te delegeren aan de gebruikers die namens hem bij het project betrokken zijn? Voor de (senior) manager hebben projectactiviteiten vaak niet de hoogste prioriteit en als hij moeite heeft met het delegeren van beslissingsbevoegdheden, kan het lang duren voor een beslissing wordt genomen. Dit kan verlammend werken op de voortgang van een project.

Overigens kunnen de problemen uit deze paragraaf in het algemeen worden samengevat onder de noemer: *onvoldoende prioriteit bij het senior management*.

2.6. Te krappe planning

Uit het bovenstaande zijn al wat omstandigheden af te leiden die problemen met de planning kunnen veroorzaken. De planning zelf kan echter ook een bron voor problemen zijn.

In veel gevallen ontbreken betrouwbare ervaringscijfers over vroegere projecten. In dat geval moet gepland worden op basis van allerlei aannames. En het behoeft geen betoog dat daarbij gemakkelijk verkeerde inschattingen kunnen worden gemaakt. Vooral als de opdrachtgever (zie hierboven) ook nog druk uitoefent, omdat hij alles toch al zo duur vindt en het eigenlijk ook allemaal erg lang gaat duren (hij had zíjn baas namelijk beloofd, dat het veel eerder klaar zou zijn!).

De kans is aanzienlijk, dat zo een te krappe planning worden afgegeven die er de oorzaak van kan zijn dat er te snel gewerkt gaat worden, waardoor verkeerde beslissingen worden genomen. En in de regel worden die fouten pas heel laat ontdekt, vaak pas tijdens het testen, zodat verschillende fasen in het project moet worden teruggegaan, en van alles overhoop gehaald, om de problemen op te lossen.

Andere concessies die worden gedaan ter verkorting van de doorlooptijd en verlaging van het budget betreffen de *kwaliteit op langere termijn*: onduidelijke of zelfs onvolledige documentatie en weinig aandacht voor onderhoudbaarheid en herbruikbaarheid.

Een laatste probleem in verband met de planning is regelmatig de beschikbaarheid van de gebruiker, of liever: het gebrek aan beschikbaarheid van de gebruiker. In een planning wordt uitgegaan van een bepaalde capaciteit die door de gebruikers geleverd kan worden. Als op dat punt ruimer is gepland dan in de praktijk mogelijk blijkt, is dat een bron voor irritatie en oponthoud verderop in het project.

2.7. Problemen door externe factoren

Tenslotte kunnen problemen binnen een softwareproject veroorzaakt worden door oorzaken van buiten het project. Zo kunnen er problemen zijn met een leverancier door onduidelijke of onvolledige afspraken (zie 2.1 en 2.5) of omdat die zijn beloften niet na wil of kan komen.

Daarnaast kunnen veranderingen in de wetgeving of andere eisen van de overheid veranderingen in de vereiste functionaliteit, en dus in de doorlooptijd, veroorzaken.

Ook de beschikbaarheid van de projectmedewerkers heb je niet altijd in de hand. Ze kunnen een andere baan aannemen of ziek worden. Als op zo'n moment de taken niet goed overdraagbaar zijn, levert dat meteen een bron voor problemen en misverstanden op.

Tot slot kunnen ook veranderingen in het bedrijf (zoals reorganisaties) of in het economisch klimaat gevolgen hebben voor de uitvoering van projecten.

3. DE WATERVALMETHODEN

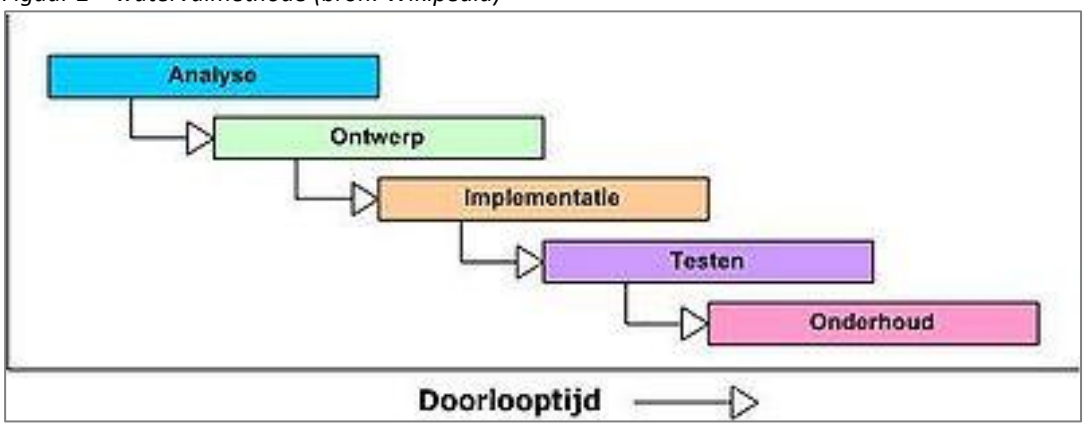
3.1. Wat is de watervalmethode?

Voor je kunt aangeven of een probleem veroorzaakt wordt door de lineaire watervalmethode, zal eerst gedefinieerd moeten worden wat eigenlijk precies onder een watervalmethode wordt verstaan. Binnen dit document wordt de volgende definitie gehanteerd:

Binnen een zuivere (lineaire) watervalaanpak wordt het proces van systeemontwikkeling verdeeld in fasen die achtereenvolgens worden uitgevoerd en afgesloten. Om de resultaten van een eenmaal afgesloten fase te kunnen wijzigen moet het gehele project terug worden gezet naar een vorige fase.

Deze definitie wordt hieronder strikt gehanteerd om duidelijk te maken wat de gevolgen zijn als de watervalmethode consequent gevolgd wordt. In de praktijk is de grens tussen een (lineaire) watervalaanpak en een (iteratieve, incrementele) agile-aanpak meestal minder scherp dan vaak wordt gesuggereerd.

Figuur 2 – watervalmethode (bron: Wikipedia)



Waarschijnlijk het in Nederland meest bekende en gebruikte voorbeeld van een watervalmethode is de *System Development Methodology* (SDM).

3.2. De watervalmethode en de problemen met softwareprojecten

In hoofdstuk 2 zijn de belangrijkste problemen met softwareprojecten behandeld en over zeven hoofdgroepen verdeeld. In dit hoofdstuk wordt de relatie gelegd tussen deze hoofdgroepen en de watervalmethode voor systeemontwikkeling.

1. *Onduidelijke of veranderende gebruikerseisen en –wensen*
   Bij de watervalmethoden worden activiteiten uit een eenmaal afgesloten fase niet opnieuw uitgevoerd. Ingeval van problemen op dit punt zou theoretisch dus het gehele project terug moeten naar de fase waarin de gebruikersspecificaties worden opgesteld.
   Zeker in grote, langlopende projecten kan dat veel tijd- en geldverlies met zich meebrengen. Dat kan organisaties in de verleiding brengen om dan maar met de bestaande (en dus niet meer

helemaal correcte) specificaties verder te gaan, zodat er uiteindelijk een systeem wordt opgeleverd, dat niet aan de eisen en wensen voldoet.
Voor veranderingen in de reikwijdte van een project geldt hetzelfde, omdat deze ook als veranderende eisen en wensen beschouwd kunnen worden.

2. *Te veel aandacht voor details*
   Als de gebruiker zich bij het opstellen van de eisen en wensen verliest in details, kan dat de specificatiefase verlengen, maar hoeft dat niet per se negatieve gevolgen te hebben voor de daaropvolgende ontwikkelfase. Integendeel , hoe meer details bekend zijn, hoe beter de ontwikkelaar weet wat er van hem verwacht wordt. Daarbij moet het wel zo zijn, dat de gebruiker zich beperkt tot zuivere eisen en wensen. Vaak worden er ook oplossingsaspecten meegenomen in de eisen en wensen en dan loop je het risico dat de gebruiker op de stoel van de ontwikkelaar gaat zitten, zodat onnodige discussie kan ontstaan, of dat de ontwikkelaar maakt wat de gebruiker voorstelt, wat niet de beste oplossing voor de gebruiker en de organisatie hoeft te zijn.
   Als de ontwikkelaar zich onnodig verliest in details, is dat nog risicovoller: dat kan de ontwerp- en realisatiefasen onnodig verlengen en omdat contact tussen gebruiker en ontwikkelaar in de ontwerp- en realisatiefasen van de watervalmethoden beperkt is, wordt deze vertraging mogelijk pas laat opgemerkt.

3. *Gebrek aan kennis bij gebruiker en/of ontwikkelaar*
   Als een gebruiker te weinig materiekennis heeft, kan er met geen enkele methode een fatsoenlijk systeem worden ontwikkeld, of pas na heel veel moeite, dus ook niet met de watervalmethode.
   Als een ontwikkelaar te weinig kennis heeft, is de kans groot dat je dat bij de watervalmethode pas heel laat merkt, omdat de ontwikkelaar de technische fasen relatief autonoom uitvoert. Dit risico is groter bij kleine projecten en/of kleine organisaties waar vaak maar één ontwikkelaar alle activiteiten uitvoert, en kleiner bij grote projecten en/of grote organisaties waar vaak ervaren en minder ervaren ontwikkelaars samenwerken.

4. *Onvoldoende (goede) communicatie*
   Hierboven is al enkele keren geschreven dat problemen regelmatig te maken hebben met een gebrek aan communicatie tussen gebruiker en ontwikkelaar, met name tijdens de ontwerp- en realisatiefasen van de watervalmethoden, maar eigenlijk in alle fasen, omdat iedere fase in principe sequentieel door specialisten wordt uitgevoerd.
   Overigens is er geen enkele watervalmethode die communicatie, bijvoorbeeld via projectteams, uitsluit, maar het is wel zo dat de structuur van de watervalmethoden communicatie niet uitdrukkelijk stimuleert.
   Een ander veelvoorkomend communicatieprobleem is, dat gebruiker en ontwikkelaar elkaars taal niet spreken. Door de relatief autonome uitvoering van taken en het feit dat communicatie vaak via de *phase deliverables* plaatsvindt, lijkt dit een serieus risico van de watervalmethoden. Dit zorgt er ook nog voor dat misverstanden als gevolg van onvoldoende goede communicatie bij de watervalmethoden vaak laat ontdekt worden.
   Een specifieke vorm van communicatie betreft het maken van afspraken, met name tussen opdrachtgever/business en opdrachtnemer/IT. Zijn die volledig, duidelijk en eenduidig? Voor iedere methode is dit uiteraard een cruciaal punt, maar bij de watervalmethoden geldt, als zo vaak, het extra risico dat de problemen mogelijk pas laat ontdekt worden.

5. *Verkeerde verwachtingen bij (lijn)management / opdrachtgever*
   Voor een deel is dit hetzelfde punt als het laatste punt hierboven (zijn de afspraken duidelijk?), en geldt bij de watervalmethoden ook hetzelfde risico van de late ontdekking.
   Een extra risico is hier, dat de opdrachtgever extra druk gaat opleggen om het project sneller

en/of goedkoper uit te voeren. Dit kan leiden tot een onacceptabele vermindering van kwaliteit. Bij de watervalmethoden kan dit ertoe leiden dat (vrijwel) het gehele project moet worden overgedaan, of dat het project wordt stopgezet, zodat er uiteindelijk geen enkel werkend resultaat is opgeleverd.
Als de manager/opdrachtgever geen bevoegdheden wil delegeren, kan dat in de watervalaanpak leiden tot ellenlange beslissings- en autorisatietrajecten.

6. *Te krappe planning*
   De punten die hierboven genoemd zijn, kunnen tot gevolg hebben, dat een planning niet gehaald wordt of al bij voorbaat te krap wordt opgesteld. Specifiek voor de watervalmethode geldt dat door de sequentiële uitvoering van de fasen, vaak de neiging bestaat om uitloop in een van de eerste fasen, in de latere fasen te willen inhalen, om uiteindelijk toch tijdig de eindstreep te bereiken. Hierdoor bestaat het risico dat de realisatie- en zeker de testfase in onverantwoord korte tijd moeten worden uitgevoerd. In veel softwareprojecten is de testfase, qua tijd, een sluitpost.
   Doordat bij watervalmethoden de planning meestal al aan het begin van het project voor het gehele project wordt opgesteld, op een moment dus dat nog veel onduidelijk is, zal deze regelmatig, in ieder geval na iedere fase, moeten worden bijgesteld en kunnen de afwijkingen ten opzichte van de oorspronkelijke planning groot zijn. En daarmee zijn ook de afwijkingen ten opzichte van de oorspronkelijke verwachtingen van de opdrachtgever groot.

7. *Problemen door externe factoren*
   Externe factoren zullen vaak leiden tot veranderingen in de gebruikerseisen en –wensen of tot problemen met de planning, en hebben voor watervalprojecten daarom dezelfde gevolgen als hierboven bij 1 en 6 beschreven. Het grote verschil is, dat je op externe factoren minder invloed hebt en de gevolgen vaak dus veel dwingender zijn.

Uit bovenstaande kunnen enkele conclusies worden getrokken ten aanzien van de relatie tussen problemen met softwareprojecten en de watervalmethoden:

- Vaak worden problemen (te) laat ontdekt, waardoor ver terug moet worden gegaan in het project en veel tijd en geld verloren gaat; en veel keuze is er vaak niet, omdat het alles of niets is: het gehele systeem wordt pas aan het eind van het project opgeleverd.
- De watervalmethode verhindert goede communicatie tussen de betrokkenen niet, maar stimuleert die zeker ook niet.
- Bij de watervalmethode bestaat het risico, dat door uitloop van de eerste fasen, de laatste fasen onverantwoord worden ingekrompen om de deadline te kunnen halen.
- De watervalmethode vergroot de kans dat een systeem uiteindelijk niet aan de gebruikerseisen en –wensen voldoet.
- Problemen met softwareprojecten slechts gedeeltelijk veroorzaakt of versterkt door de watervalmethode.

In het volgende hoofdstuk zal bekeken worden in hoeverre de aanpak van RAD, DSDM en Scrum, als voorbeelden van agile-methoden, een verbetering kan betekenen.

4. DE AGILE-METHODEN

4.1. Van waterval (lineair) naar agile (incrementeel/iteratief)

Op de zuivere (lineaire) watervalmethoden kwam rond 1980 van verschillende kanten reactie. Voorbeelden hiervan zijn de semi-gestructureerde, en later de gestructureerde analyse van Yourdon c.s. en de proces-geöriënteerde benadering van Floyd.

Floyd ventileerde de gebruikelijke kritiek op de lineaire benadering:
- De gebruiker is alleen in de eerste fase(n) bij het project betrokken.
- Er wordt geen rekening gehouden met wijzigende eisen en wensen.
- Er is geen mogelijkheid om zaken te verbeteren in een vorige fase (lineair proces).
- Er worden omvangrijke documenten opgeleverd die voor de gebruiker onbruikbaar zijn.

Als alternatief introduceerde Floyd in 1981 haar *process oriented* benadering die in een aantal opzichten als een voorloper van de incrementele, iteratieve agile aanpak kan worden beschouwd:
- Systeemontwikkeling wordt beschouwd als een voortdurende opeenvolging van ontwikkelingscycli.
- Iedere cyclus levert een tussentijds deelresultaat op (een systeemversie).
- De gebruikers en ontwikkelaars staan in voortdurend contact met elkaar.

Het behoeft geen betoog dat in de ideeën van Floyd veel ruimte is ingeruimd voor het iteratieve aspect van de systeemontwikkeling, terwijl ook de voortdurende communicatie tussen gebruikers en ontwikkelaars een belangrijke rol speelt, net als bij de agile-methoden, zoals we nog zullen zien.

De iteratieve en incrementele systeemontwikkeling kwam goed op gang toen James Martin in 1991 zijn boek *Rapid Application Development* (RAD) publiceerde en daarin zijn gelijknamige methode voor systeemontwikkeling als alternatief voor de watervalmethoden beschreef. Vlak daarna verschenen aan RAD gerelateerde methoden als IBM's *Rational Unified Process* (RUP, 1994), *Scrum* (1995), *Dynamic Systems Development Method* (DSDM, 1995) en *Extreme Programming* (XP, 1996). Hieronder ga ik op drie van deze methoden verder in: RAD, DSDM en Scrum.

Feitelijk gaat de historie van flexibele, iteratieve en incrementele systeemontwikkeling dus al meer dan dertig jaar terug, maar de term *agile* werd pas voor het eerst gebruikt in 2001 toen zeventien softwareontwikkelaars in de staat Utah (US) bijeenkwamen om over *lightweight* systeem-ontwikkeling (in tegenstelling tot de *heavyweight* watervalmethoden) te praten en het *Manifesto for Agile Software Development* publiceerden. In dat manifest geven zij aan dat ervaring hen heeft geleerd wat zij moeten prefereren:
- Individuën en interactie boven processen en tools
- Werkende software boven allesomvattende documentatie
- Samenwerking met de klant boven contractonderhandelingen
- Reageren op verandering boven het volgen van een plan.

Met als toevoeging: "while there is value in the items on the right, we value the items on the left more."

Volgens het manifest kent agile-systeemontwikkeling *twaalf grondbeginselen*:
1. Klanttevredenheid (snelle oplevering van werkende software).

2. Accepteren dat gebruikerseisen en -wensen veranderen, ook later in het project.
3. Werkende software wordt regelmatig opgeleverd (liever in weken dan in maanden).
4. Werkende software is het belangrijkste meetpunt voor het bepalen van de voortgang.
5. Duurzame systeemontwikkeling.
6. Nauwe, dagelijkse samenwerking tussen klanten/gebruikers en ontwikkelaars.
7. *Face-to-face* communicatie heeft de sterke voorkeur (hele team in dezelfde ruimte).
8. Projecten worden opgezet rond gemotiveerde individuën die je kunt vertrouwen.
9. Voortdurende aandacht voor technische *excellence* en goed ontwerp.
10. Eenvoud (de kunst om de hoeveelheid werk die níét wordt gedaan te maximaliseren) is essentieel.
11. Zelf-organiserende teams.
12. Voortdurende aanpassing aan veranderende omstandigheden.

4.2. Iteraties en incrementen

Een *iteratieve* benadering van systeemontwikkeling betekent niet meer dan dat de resultaten van een of meer activiteiten worden teruggekoppeld, besproken en eventueel aangepast. Als zodanig is deze aanpak niet het alleenrecht van de moderne agile-benaderingen, zoals hierboven blijkt uit het belang dat Floyd aan het iteratieve aspect hecht. Maar ook Edward Yourdon wijst voortdurend op de iteratieve mogelijkheden van zijn gestructureerde analyse. Hij spreekt liever over activiteiten dan over fasen, om niet te suggereren dat de systeemontwikkeling een puur sequentieel proces is. “Bijna elke activiteit (…) kan, en doet dat doorgaans ook, gegevens opleveren die bepaalde wijzigingen in één of meer voorafgaande activiteiten tot gevolg hebben”, schrijft Yourdon in zijn boek *Gestructureerde Analyse*. Toch wordt de aanpak van Yourdon vaak geassocieerd met de (lineaire) watervalmethode. Onterecht dus, in ieder geval ten dele.

Dat het systeemontwikkelingstraject nooit een puur lineair proces kan zijn, is een idee dat iedere ontwikkelaar weet uit zijn praktijk. Professor W. Hartman schreef al in 1984 in *Het ontwerpen van informatiesystemen; een inleiding*: “elke ontwerper weet dat hij met een *iteratief proces* bezig is: hij wil voorwaarts, maar moet dikwijls één of meer stappen terug.” En zo is het natuurlijk ook.

Incrementele systeemontwikkeling is het stap voor stap ontwikkelen van informatiesystemen, waarbij elke stap een werkend en bruikbaar – klein of groot – deelsysteem oplevert dat wordt toegevoegd aan hetgeen in eerdere stappen is opgeleverd. Hierbij moet eerst bepaald worden welke stappen of bouwstenen in het gewenste informatiesysteem te onderkennen zijn. Een increment is vervolgens de stap die nodig is voor analyse, ontwerp, realisatie en invoering van een bouwsteen. Alle incrementen, na de eerste, profiteren van wat in het voorafgaande geleerd is. Een increment wordt ontwikkeld in een zogenaamde *time box*. Over de wenselijke doorlooptijd van zo’n time box verschillen de meningen nogal. Sommigen vinden drie tot negen maanden acceptabel, terwijl anderen één tot vier weken ook voldoende vinden.

En voor incrementele aanpak geldt hetzelfde als voor de iteratieve: ook deze kan worden gebruikt binnen de zogenaamde traditionele methoden. Zo leent Yourdons *middle-out* benadering via het (van McMenamin en Palmer overgenomen) concept van *event-partitioning* zich bij uitstek voor een incrementele benadering. Immers: als event-partitioning wordt toegepast, wordt het systeem per event gemodelleerd en aangezien events tot op zekere hoogte onafhankelijk van elkaar zijn, is een incrementele ontwikkeling zeer goed mogelijk.

Maar ook Floyds voorstellen voor het werken met systeemversies zijn in wezen niets anders dan een pleidooi voor de incrementele benadering: ieder nieuw onderdeel wordt toegevoegd aan de reeds

bestaande systeemversie, zodat een nieuwe versie ontstaat. En zelfs de watervalmethode SDM geeft expliciet de mogelijkheid van een incrementele aanpak aan. Nadat de fasen Definitiestudie en Basisontwerp voor het gehele systeem zijn uitgevoerd, kan het traject vanaf het Detailontwerp per deelsysteem worden afgelegd. Volgens SDM kan dit in verschillende volgorde gebeuren, zowel sequentieel als parallel.

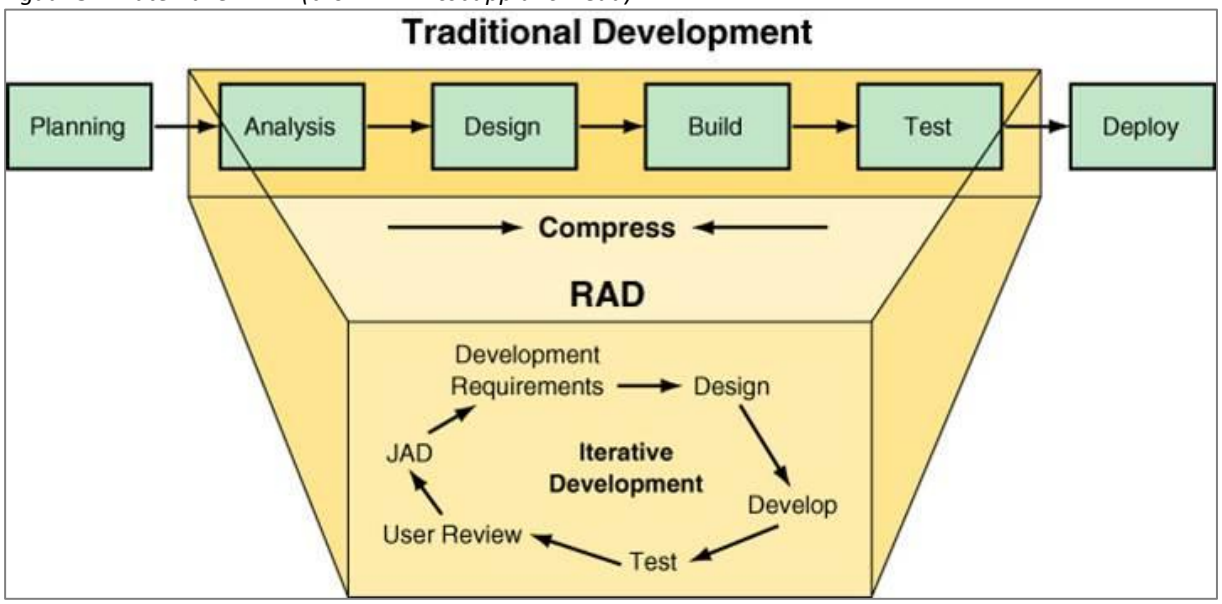
We kunnen concluderen, dat een iteratieve en/of incrementele aanpak ook bereikt kan worden met methoden die traditioneel met een lineaire benadering geassocieerd worden. Het lijkt er op dat een lineaire aanpak van softwareprojecten meer vastzit aan de organisatiecultuur, dan aan de gekozen systeemontwikkelingsmethode. Aan de andere kant is het wel zo, dat de agile-methoden (zoals we hieronder zullen zien) sterker de nadruk leggen op incrementele en iteratieve aspecten en zich op die manier profileren.

4.3. Rapid Application Development (RAD)

Doelstelling

In de zeventiger en tachtiger jaren van de vorige eeuw werden de meeste softwareprojecten lineair uitgevoerd volgens de/een watervalmethode. Grote, logge en daardoor langlopende en (te) dure projecten waren het resultaat. De ontevredenheid bij menig opdrachtgever was groot en pogingen om de bestaande systeemontwikkelmethoden te verbeteren bleven zonder praktisch resultaat. Als reactie daarop introduceerde James Martin in 1991 een alternatieve methode voor systeemontwikkeling: *Rapid Application Development* (RAD). Velen zagen RAD als niet minder dan een radicale doorbraak, waarmee voldaan kon worden aan de behoefte aan het *sneller* en *goedkoper* ontwikkelen van *kwalitatief betere* systemen, aan een meer *resultaatgerichte* aanpak, in tegenstelling tot de *activiteitgerichte* aanpak van de jaren zeventig en tachtig (zoals binnen SDM).

Figuur 3 – waterval en RAD (bron: www.csbapp.uncw.edu)



Kenmerken

- *Iteratief en incrementeel ontwikkelen binnen time boxes*

De meest genoemde en waarschijnlijk voornaamste kenmerken van een RAD-project zijn de iteratieve en incrementele aanpak (al zijn die eigenschappen niet exclusief voor RAD, zoals we eerder gezien hebben, maar kenmerkend voor alle agile-methoden, en zelfs ook mogelijk binnen veel watervalmethoden).

Een RAD-project begint met de begrenzing van de incrementen en de vaststelling van de volgorde waarin zij worden ontwikkeld. Over het algemeen zal daarbij besloten worden de belangrijkste functies het eerst te realiseren.

Daarbij wordt het RAD-project uitgevoerd met behulp van het zogenaamde *Time Box Management*. Binnen Time Box Management worden onwrikbare mijlpalen vastgesteld: rotsvaste en niet-onderhandelbare tijdstippen waarop 'iets' klaar moet zijn. En zoals eerst de belangrijkste incrementen ontwikkeld worden, zo zullen binnen een increment eerst de belangrijkste functies worden gerealiseerd, zodat die in ieder geval binnen de Time Box gerealiseerd zijn. Het restant kan dan eventueel in een volgend increment worden ontwikkeld. Het werken met Time Boxes zal de resultaatgerichtheid van de systeemontwikkeling bevorderen: er moet immers een werkend resultaat worden opgeleverd aan het eind van de Time Box.

De mogelijkheid om reeds opgeleverde incrementen aan te vullen en te verbeteren tijdens de ontwikkeling van volgende incrementen is een iteratief aspect dat de kwaliteit van het eindproduct ten goede kan komen. Bovendien biedt het werken met RAD-teams (zie hieronder) de mogelijkheid van voortdurende terugkoppeling en daarmee van iteratieve bijstelling binnen een increment.

Overigens kan een onderscheid worden gemaakt tussen incrementen en subsystemen. Het aantal incrementen kan bij grote systemen zo groot worden dat de totale doorlooptijd een probleem wordt. Dan kan besloten worden om het totale systeem te verdelen in subsystemen en de subsystemen parallel op een incrementele manier te ontwikkelen. Subsystemen kunnen als relatief autonome functionele eenheden worden gezien, terwijl een increment een te ontwikkelen eenheid is. Een subsysteem kan uit één of meer incrementen bestaan.

- *Teamleden die aan hoge eisen voldoen en een grote en actieve betrokkenheid tonen*

Onontbeerlijk binnen een RAD-project is de grote betrokkenheid van de gebruiker. Concreet betekent die, dat hij full-time deel uitmaakt van het zogenaamde *RAD-team*. Dit RAD-team is multidisciplinair van opzet, hetgeen wil zeggen dat zowel de gebruikers als de ontwikkelaars in het team vertegenwoordigd zijn. Binnen het team worden zij geacht intensief samen te werken, waarbij ieder teamlid de eigen expertise inbrengt, maar niet persoonlijk verantwoordelijk is voor bepaalde keuzes. Het RAD-team als geheel is verantwoordelijk voor het resultaat. Om dit resultaat zo snel mogelijk te realiseren, moeten de teamleden beschikken over vergaande beslissingsbevoegdheden.

Er wordt gewerkt in *workshops*, waarbinnen de teamleden in gezamenlijk overleg de (*high level*) specificaties opstellen (JRP: *Joint Requirements Planning*), het applicatieontwerp maken (JAD: *Joint Application Design* of *Joint Application Development*) en het ontwerp vervolgens direct concretiseren met behulp van *prototyping*. Een prototype is een vereenvoudigd, maar werkend model van (delen van) een informatiesysteem. Prototypes worden ook wel beschouwd als halffabrikaten die op iteratieve en incrementele wijze tot eindproducten worden vervolmaakt.

Gezien de taken die een RAD-team moet uitvoeren, en de daarbij behorende bevoegdheden die het team heeft, is het begrijpelijk dat de eisen die aan de leden van een RAD-team gesteld worden bijzonder zwaar zijn: ervaring, initiatief, samenwerking, zelfstandigheid, doorzettingsvermogen, verantwoordelijkheidsgevoel en zeer goed ontwikkelde communicatieve vaardigheden. Daarnaast moeten de ontwikkelaars zowel functioneel als technisch goed onderlegd zijn en moeten de gebruikers in het team full-time beschikbaar zijn, liefst kennis van automatiseren hebben en in ieder geval een zeer goede materiekennis bezitten, niet alleen met betrekking tot de 'eigen' functionaliteit, maar, vanwege de aansluiting met andere incrementen, ook van die van de 'buren'.

Ook van de organisatie wordt het een en ander verwacht. Teamleden moeten full-time worden vrijgemaakt van hun normale werkzaamheden en het management moet bereid zijn beslissingsbevoegdheden aan de projectmedewerkers te delegeren. Daarnaast moet er voor de benodigde specialistische ondersteuning gezorgd worden, aangezien een RAD-team weliswaar multidisciplinair is opgezet, maar ook beperkt van omvang en dus niet alle specialismen kan bevatten. Omdat echte specialisten vaak ook niet voortdurend nodig zijn, zou het ook niet efficiënt zijn om deze full-time aan het RAD-team toe te voegen.

- *Geavanceerde technische hulpmiddelen*

Een RAD-team moet al discussiërend en overleggend tot voortdurende verbeteringen in de specificaties kunnen komen en tot een werkend prototype dat eveneens continu aangepast moet kunnen worden. Bovendien moeten eenmaal ontwikkelde elementen hergebruikt kunnen worden binnen andere incrementen. Om de gewenste kwaliteit en snelheid in de ontwikkeling te kunnen bereiken, zijn daarom de juiste hulpmiddelen van groot belang, maar ook, wat vaak vergeten wordt, voldoende kennis van die hulpmiddelen. Aanschaf van en opleiding in tools voor *Computer Aided Software Engineering* (CASE) is dus essentieel voor het slagen van een RAD-project.

Minimaal moet zo'n CASE-tool:
- gebaseerd zijn op modellen
- de beschikking hebben over een systeemencyclopedie (*repository*) met versiebeheer
- de vereiste documentatie geautomatiseerd kunnen genereren
- de beschikking hebben over een programmagenerator (in ieder geval voor programma's, liefst ook voor het creëren en onderhouden van de database).

- *Adequate incrementen in plaats van perfecte*

Het gevolg van het leggen van de prioriteit bij prijs en snelheid, is dat in eerste instantie concessies mogen worden gedaan aan de kwaliteit/functionaliteit van het systeem. Gestreefd wordt naar een *adequate* en niet naar een *perfecte* oplossing.

Hierbij mogen niet zoveel concessies worden gedaan aan de kwaliteit, dat niet meer gesproken kan worden van de in de doelstelling genoemde "kwalitatief betere systemen". Doordat ieder, op zichzelf adequaat maar onvolmaakt increment bij de ontwikkeling van een volgend increment kan worden aangepast en verbeterd, heeft de gebruiker aan het eind van het totale project de beschikking over een kwalitatief hoogwaardig(er) product. Het begrip *kwalitatief beter* moet hier geïnterpreteerd worden als: 'meer tegemoet komend aan de eisen en wensen van de gebruiker'. Over de kwaliteit van andere aspecten, bijvoorbeeld onderhoudbaarheid, zegt het nog niets.

4.4. Dynamic Systems Development Method (DSDM)

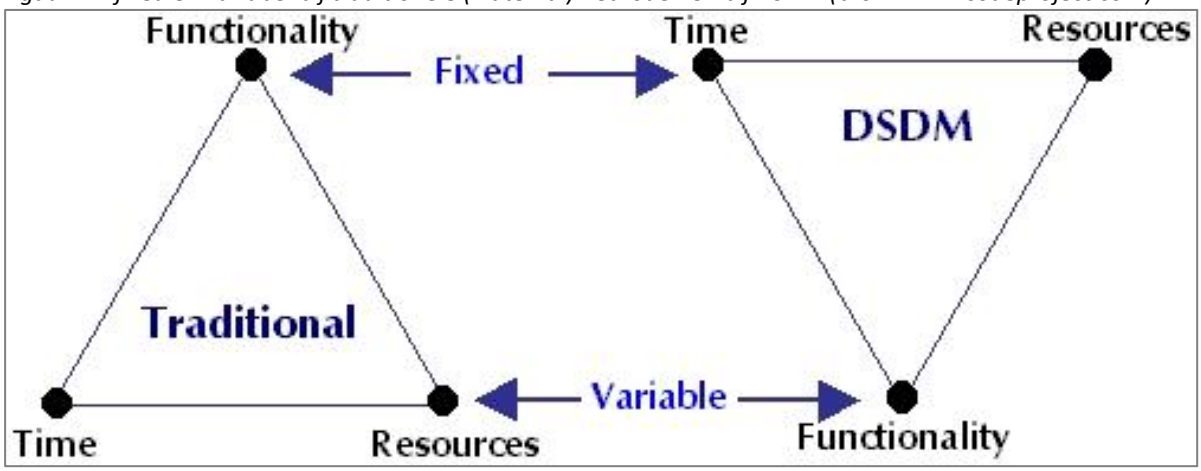
Negen principes

In de loop van de jaren negentig ontstond er behoefte aan een meer concreet raamwerk voor het uitvoeren van RAD-projecten. Zo'n raamwerk was de *Dynamic Systems Development Method* (DSDM).

DSDM is gebaseerd op een 9-tal principes:
1. Actieve gebruikersparticipatie tijdens het gehele proces is noodzakelijk.
2. Teams moeten over voldoende bevoegdheden beschikken om beslissingen te nemen.
3. De focus is gericht op een frequente oplevering van producten (werkende prototypes en/of systeemdelen).
4. De bruikbaarheid voor zakelijke en organisatorische doeleinden is het essentiële criterium voor de acceptatie van *deliverables*.
5. Een iteratieve en incrementele ontwikkelaanpak is absoluut noodzakelijk om te komen tot een accurate business-oplossing.
6. Alle veranderingen, aangebracht tijdens de ontwikkeling, zijn onomkeerbaar.
7. Systeemeisen worden gedefinieerd op algemeen niveau.
8. Testen vindt plaats gedurende de gehele ontwikkelfase.
9. Een coöperatieve houding van alle betrokken disciplines is essentieel.

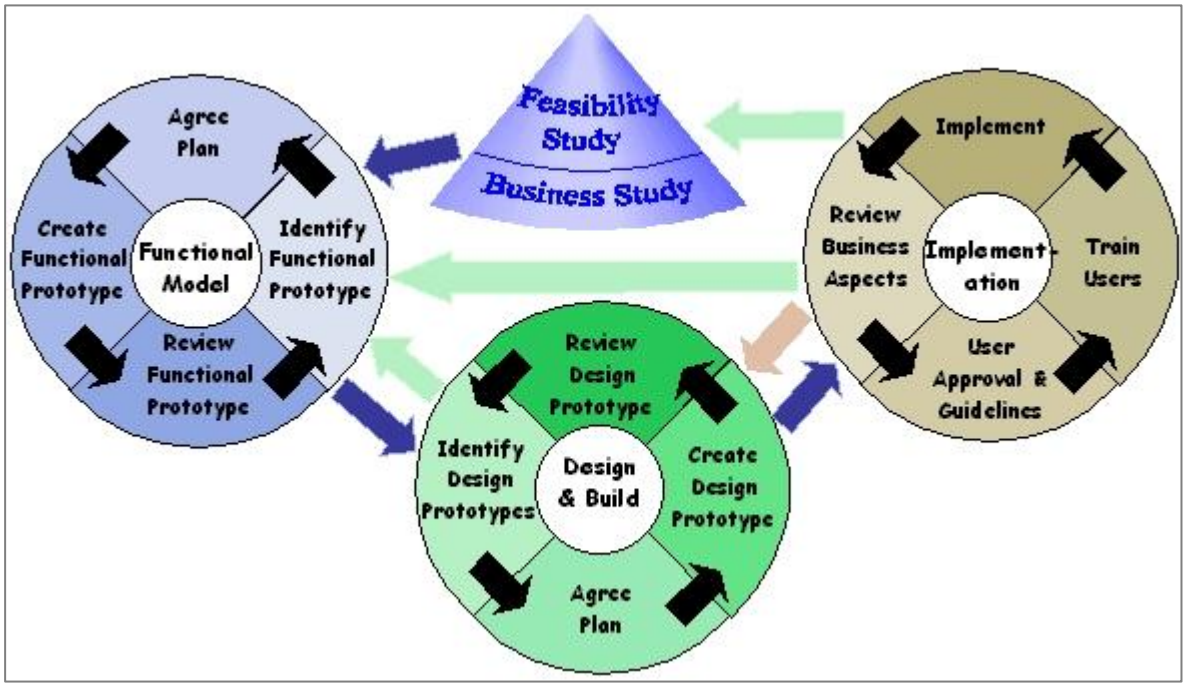
Vraagtekens zijn te plaatsen bij het zesde principe: "Alle veranderingen, aangebracht tijdens de ontwikkeling, zijn onomkeerbaar". Hoe moeten we dit rijmen met de flexibiliteit die RAD/DSDM zo hoog in haar vaandel voert? Incrementen moeten toch voortdurend bijgesteld en verbeterd kunnen worden?

Figuur 4 – fixed en variabel bij traditionele (waterval)methoden en bij DSDM (bron: www.codeproject.com)



Het zal duidelijk zijn dat deze principes voor het merendeel gelijk zijn aan de principes van de RAD-aanpak in het algemeen. Op basis van deze principes worden binnen DSDM de volgende vijf stappen onderkend:
- Twee basisstappen: (1) toepasbaarheidsonderzoek en (2) bedrijfsonderzoek.
- Drie iteratieve stappen: (3) ontwerp, (4) realisatie en (5) implementatie van een prototype.

Figuur 5 – DSDM: twee basisstappen en drie iteratieve stappen (bron: www.codeproject.com)



Vijf stappen

(1) Toepasbaarheidsonderzoek

Kort onderzoek naar de mogelijkheden binnen de bedrijfsomgeving van de toepassing van RAD/DSDM. Niet elk systeem leent zich even goed voor een incrementele ontwikkeling en implementatie. Zo lenen transactieverwerkende real-time systemen of systemen waarvan de functionaliteit moeilijk te visualiseren is, zich minder goed voor RAD/DSDM, omdat prototyping niet goed mogelijk is. En ook pakketimplementaties, rekenintensieve technisch-wetenschappelijke toepassingen en onderhoud van bestaande systemen zijn minder geschikt voor de RAD/DSDM-aanpak.

Wel uitermate geschikt voor een RAD/DSDM-aanpak zijn projecten met de volgende eigenschappen:
- een hoge graad van gebruikersinterface (schermen en rapporten)
- een duidelijk definieerbare gebruikersgroep
- een hoge mate van onduidelijkheid met betrekking tot de gebruikerseisen en -wensen.

Het toepasbaarheidsonderzoek levert de volgende producten op:
- De doelstellingen van de ontwikkeling.
- De afbakening van de belangrijkste producten.
- De technische en zakelijke vereisten.
- Een globaal projectplan.

(2) Bedrijfsonderzoek

Het bedrijfsonderzoek heeft de volgende doelstellingen:
- Het afbakenen van de belangrijkste zakelijke functies die moeten worden ondersteund.
- Het vaststellen van de verschillende gebruikersgroepen.
- Het vaststellen van de noodzakelijke interfaces.

- Het prioriteren van de systeemfuncties.
- Het opstellen van een prototypeplan.
- Het geven van een aanzet voor de te realiseren systeemarchitectuur.

Het bedrijsonderzoek levert de volgende producten op:
- Een organisatorisch functiemodel.
- Een beschrijving van gebruikersgroepen, processen en informatiebehoeften.
- De prioriteitstoekenning aan functies.
- Een prototypeplan.
- Een definitie van de systeemarchitectuur.

(3) Functionele model-iteratie

Deze stap behelst het uitwerken van de onderkende functies in volgorde van prioriteit, waarbij zowel functionele als niet-functionele (performance, beveiliging etc.) eisen aan de orde komen. De implementatie-strategie wordt bepaald en een risico-analyse uitgevoerd. Het resultaat is een modelmatig prototype.

Op te leveren producten:
- functionele modellen (processen, gegevens).
- implementatiestrategie.
- risico-analyse.
- kosten/baten-analyse.

(4) Systeemontwerp- en systeembouw-iteratie

In deze stap vindt de realisatie van werkende prototypes plaats. De nadruk ligt op het technische vlak, zoals databases en gebruikersinterfaces. Deze prototypes worden voortdurend getest en beoordeeld. Testen is binnen DSDM geen aparte activiteit.

(5) Implementatie-iteratie

In deze stap wordt het ontwikkelde systeem(deel) gereed gemaakt voor beschikbaarstelling aan de gebruikersorganisatie. De nadruk ligt op de samenwerking tussen de verschillende gedeelten en het systeem als geheel. Daarnaast vindt een *review* van het project plaats om ervaring op te bouwen voor vervolgtrajecten, het opleiden van gebruikers en het opstellen van gebruikershandleidingen.

Conclusie

De eerste twee stappen van DSDM (Toepasbaarheidsonderzoek en Bedrijfsonderzoek) zijn niet incrementeel en ook niet iteratief van aard. Als zodanig zijn zij niet wezenlijk anders dan vergelijkbare stappen in een project met een lineaire watervalaanpak. De laatste drie stappen van DSDM zijn wel iteratief. In de praktijk blijkt dat in elke fase gemiddeld drie iteraties plaatsvinden voordat wordt gestart met een volgende fase. Echter in elke fase kan worden teruggekeerd naar een voorgaande fase, indien hiertoe aanleiding bestaat, en zij worden per increment uitgevoerd.

Alles bij elkaar lijkt het er op dat DSDM vooral een combinatie is van RAD en SDM, waarbij nieuw is dat wordt vermeld dat de fasen ontwerp, realisatie en implementatie iteratief, per increment en deels parallel kunnen worden uitgevoerd (hoewel we hierboven in paragraaf 4.2 gezien hebben dat

dat tot op zekere hoogte voor SDM ook al gold), en dat er binnen incrementen werkende en zich ontwikkelende prototypes worden opgeleverd.

4.5. Scrum

Algemeen

Als de functie-eisen in de vacatures een juist beeld geven van de verspreiding van de verschillende systeemontwikkelingsmethoden in Nederland, dan kunnen we concluderen dat Scrum verreweg de meest gebruikte methode is. Sterker nog, er is vrijwel geen vacature voor een softwareontwikkelaar waarin niet wordt gevraagd naar kennis van of ervaring met *Agile/Scrum*.

Scrum is een raamwerk voor *agile management* van softwareontwikkeling en, net als bijvoorbeeld DSDM en het traditionele SDM, meer een projectmanagementmethode dan een methode waarin concrete tools en technieken worden aangeboden om software mee te ontwikkelen. De voorlopers van Scrum kwamen op in het midden van de jaren tachtig van de vorige eeuw. In 1995 werd de methode geformaliseerd toen Ken Schwaber en Jeff Sutherland hun *Scrum Methodology* presenteerden.

Rollen binnen een Scrumteam

Net als bij de andere agile-methoden, wordt de basis van Scrum gevormd door een multidisciplinair team. Scrum stelt wel expliciete regels met betrekking tot de organisatie van zo'n s*crum team*. Om te beginnen kent Scrum drie *core* rollen en een aantal *ancillary* rollen. De core rollen vormen de vaste kern van het scrumteam. De ancillary rollen zijn slechts indien nodig bij het scrumproces betrokken. De core rollen zijn:

- *Producteigenaar*
  Vertegenwoordigt de belanghebbenden, de opdrachtgever(s). Verantwoordelijk voor het leveren van *business value*. De producteigenaar schrijft en prioriteert specificaties, meestal in de vorm van *user stories* en voegt deze toe aan de zogenaamde *product backlog* (een lijst met geprioriteerde gebruikersspecificaties). Een scrumteam heeft één producteigenaar.
- *Development team*
  Bestaat uit 3-9 medewerkers met *cross-functional* vaardigheden. Organiseert zichzelf en werkt bij voorkeur in één ruimte. Voert het technische werk uit. Verantwoordelijk voor de oplevering van een werkend increment aan het eind van een s*print* (zie hieronder). Dit team kan ook de testen uitvoeren, maar daarvoor kan ook een apart *Test and evaluation team* worden opgezet.
- *ScrumMaster*
  Is niet de teamleider, maar faciliteert. Zorgt ervoor dat het scrumproces wordt gevolgd en dat alles wat het opleveren van de de gewenste resultaten binnen een sprint belemmert, uit de weg wordt geruimd. Deze rol verschilt van een projectmanager, doordat de ScrumMaster geen *people management*-taken heeft.

Het scrumproces

De basiseenheid binnen Scrum-systeemontwikkeling is de zogenaamde *sprint*. Binnen zo'n sprint wordt door het scrumteam binnen een vaststaande Time Box (van één tot vier weken), net als bij de andere agile-methoden, een increment opgeleverd.

Elke sprint wordt voorafgegaan door een *planning meeting*, waarin, op basis van de product-backlog, de prioriteiten en de capaciteit van het team, wordt vastgesteld wat de taken zijn en wat er gaat

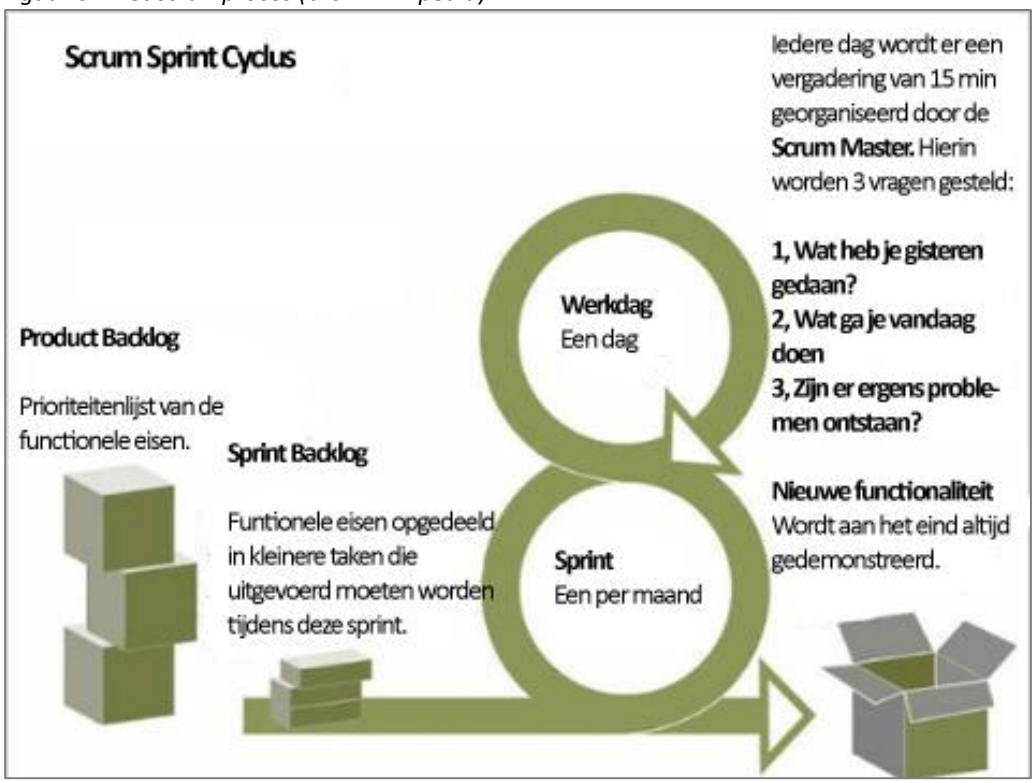
worden opgeleverd. Dit wordt vastgelegd in de *sprint backlog*. Door het Time Box-principe eindigt een sprint altijd op tijd. Wat aan het eind van een sprint nog niet af is, wordt teruggeplaatst in de product-backlog. De planning-meeting duurt maximaal acht uur. Tijdens de eerste vier uur bespreekt het hele team de product-backlog en de prioriteiten daarbinnen en tijdens de tweede vier uur stelt het development-team de sprint-backlog op.

Tijdens een sprint wordt er iedere dag een maximaal vijftien minuten durende (Time Box) *daily scrum* gehouden (ook wel *daily standup* genoemd, omdat staand vergaderen bevordert dat mensen *to the point* blijven en de vijftien minuten zo niet zo snel worden overschreden). In deze vergadering beantwoordt ieder teamlid drie vragen:

1. Wat heb je gedaan sinds de vergadering van gisteren?
2. Wat ben je van plan om vandaag te gaan doen?
3. Zijn er problemen die het opleveren van de geplande resultaten kunnen belemmeren?

De ScrumMaster faciliteert de daily scrum, noteert de geïdentificeerde problemen en zorgt er (buiten de vergadering) voor dat ze opgelost worden.

Figuur 6 – het scrumproces (bron: Wikipedia)



Een sprint eindigt met twee vergaderingen:

- *Review meeting* van maximaal vier uur, waarin wordt beproken welke geplande resultaten wel en welke niet zijn opgeleverd; het behaalde resultaat wordt gepresenteerd aan de stakeholders.
- *Retrospective meeting* van maximaal drie uur, gefaciliteerd door de ScrumMaster, waarin het sprint-team bespreekt wat er goed ging en wat beter kan in de volgende sprint.

Als er door meerdere scrumteams parallel wordt gewerkt, kan er behoefte zijn aan coördinatie van die teams. Daarvoor is de *scrum of scrums*, die dagelijks na de daily scrum plaatsvindt. In deze meeting ligt de focus op de raakvlakken tussen en de integratie van de teams en hun werk.

Om onzekerheid weg te nemen, kan het soms nodig zijn om een concept, een *proof of concept* of een prototype te ontwikkelen, voordat een item uit de product backlog tot een werkend geheel wordt ontwikkeld. Dit gebeurt dan in een speciale Time Box, de zogenaamde *spike*. Een speciaal soort spike is de *tracer bullet*, een werkend geheel, maar met beperkte functionaliteit, dat dient als voorbeeld of richtsnoer voor verdere ontwikkeling.

4.6. Voorwaarden voor succesvolle agile-systeemontwikkeling

Na het voorgaande moge het duidelijk zijn, dat, als een organisatie een agile-methode voor systeemontwikkeling wil invoeren, die organisatie eerst aan de nodige eisen zal moeten voldoen om die invoering succesvol te laten zijn. Binnen veel organisaties zal dit een heel bewustwordingsproces vragen. In ieder geval zal aan de volgende essentiële voorwaarden moeten worden voldaan:

1. Ontwikkelaars, vaak gewend om te werken op de traditionele manier met een sequentiële fasering en zonder directe betrokkenheid van de klantorganisatie tijdens technische fasen als ontwerp en realisatie, moeten de betrokkenheid van de gebruikers niet beschouwen als bemoeizucht of op-de-vingers-kijken, maar als een mogelijkheid om via samenwerking tot een optimaal (wat, zoals we eerder gezien hebben, niet per se hetzelfde is als “perfect”) resultaat te komen.
2. Niet alleen in het denken van de ontwikkelaars, maar ook in dat van de klanten/gebruikers is zo’n radicale omslag noodzakelijk, omdat zij veel nauwer betrokken zijn bij het hele proces van systeemontwikkeling, ook bij de technische fasen.
3. Gebruikers (en ontwikkelaars) moeten gedurende de ontwikkeling van een increment fulltime werkzaam kunnen zijn binnen een ontwikkelteam. Daarnaast moeten specialisten op allerlei gebied voortdurend ter beschikking staan van het team.
4. Alle teamleden moeten aan zeer zware eisen voldoen op gebieden als kennis, ervaring en (communicatieve) vaardigheden.
5. Agile-methode vergen vaak aanzienlijke investeringen, met name in geavanceerde ontwikkeltools. Deze technische hulpmiddelen moeten niet alleen beschikbaar zijn, de teamleden moeten ook voldoende kennis en ervaring hebben om er effectief gebruik van te kunnen maken. Als de kennis ontbreekt, moeten de teamleden worden geschoold.
6. Opdrachtgevers moeten (a) bereid zijn om beslissingsbevoegdheden te delegeren en (b) kunnen leven met nogal wat onzekerheden vooral aan het begin van, maar ook nog tijdens het project:
   - een dynamische en soms schijnbaar chaotische aanpak;
   - oplevering van incomplete deelsystemen;
   - het vaak ontbreken van nauwkeurige schattingen bij aanvang.

En ook al wordt een agile-methode ingevoerd in een organisatie die aan alle voorwaarden voldoet, dan nog moet het besef aanwezig zijn dat, zoals eerder ook al werd opgemerkt, niet alle systemen even geschikt zijn om met een agile-methode te worden ontwikkeld.

4.7. De agile-methoden en de problemen met softwareprojecten

In hoofdstuk 2 zijn de belangrijkste problemen met softwareprojecten behandeld en over zeven hoofdgroepen verdeeld. In hoofdstuk 3 is besproken welke relaties er zijn tussen deze problemen en de watervalmethoden voor systeemontwikkeling. Hieronder wordt ingegaan op de relatie tussen de zeven hoofdgroepen en de agile-methoden. Om duidelijk het verschil met de watervalmethoden in beeld te brengen, wordt er van uitgegaan dat de agile-methode zuiver wordt gehanteerd.

1. *Onduidelijke of veranderende gebruikerseisen en –wensen*
   Door de incrementele en iteratieve aanpak en door het fulltime samenwerken in teamverband van gebruikers en ontwikkelaars is een agile-methode bij uitstek geschikt om dit probleem te minimaliseren.
   Hierbij moet wel de kanttekening worden gemaakt, dat onduidelijke en/of veranderende eisen en wensen er bij agile-methode voor kunnen zorgen, dat eenmaal werkend opgeleverde (en mogelijk zelfs al in productie zijnde) onderdelen ook aangepast moeten worden.

2. *Te veel aandacht voor details*
   Ook bij de agile-methoden is het risico van te veel aandacht voor en discussie over (meer of minder relevante) details zeker aanwezig. De voortdurende samenwerking tussen gebruikers en ontwikkelaars kan hiertoe leiden, maar aan de andere kant kan deze samenwerking in combinatie met de aanwezigheid van andere teamleden, er juist voor zorgen dat deze aandacht en discussies niet ontsporen. Ook het Time Box-principe en de eis dat er (vaak binnenkort) iets werkends moet worden opgeleverd, kunnen hier heilzaam werken.

3. *Gebrek aan kennis bij gebruiker en/of ontwikkelaar*
   Door de aanpak met opeenvolgende fasen en de omstandigheid dat in de vorige fase(n) alles gedetailleerd (en goedgekeurd) is gedocumenteerd, biedt de watervalmethode in principe meer houvast aan degene die in een volgende fase met die documentatie aan de slag moet. Die medewerker kan zich bovendien volledig richten op zijn eigen specialisatie.
   Bij de agile-methoden is dit volkomen anders: documentatie is slechts beperkt voorhanden, eisen en wensen kunnen voortdurend veranderen of worden ter plekke bedacht, iedere medewerker moet met de andere specialismen kunnen praten en dus over de grenzen van zijn specialisatie heen kunnen kijken en tegelijkertijd moet er aan het eind van ieder increment iets werkends worden opgeleverd dat een basisstructuur (zoals een datamodel) heeft die zoveel mogelijk ook voor de nog te ontwikkelen incrementen geschikt is.
   Alles bij elkaar betekent dit, dat er van de leden van een agile-team heel veel kennis en ervaring gevraagd wordt, aanzienlijk meer dan bij een watervalproject. Dit probleem zal door de agile-aanpak dus niet verkleind worden. Sterker nog, gebrek aan kennis en ervaring, zowel bij gebruiker als bij ontwikkelaar, zal bij de agile-aanpak tot nog grotere problemen leiden, dan bij de watervalmethode. Het voordeel van de agile-aanpak is wel, dat het gebrek aan kennis door de nauwe samenwerking in veel gevallen eerder zichtbaar zal worden, zodat sneller maatregelen kunnen worden genomen.

4. *Onvoldoende (goede) communicatie*
   Door de opzet en de werkvormen (teams, workshops, prototyping, review meetings, etc.) wordt communicatie bij de agile-aanpak min of meer afgedwongen, meer dan bij de watervalmethode. Bovendien zal een communicatieprobleem, net als een gebrek aan kennis, bij een agile-project eerder worden opgemerkt.

5. *Verkeerde verwachtingen bij (lijn)management / opdrachtgever*
   Doordat aan het eind van ieder increment een werkend geheel wordt opgeleverd, komen verkeerde verwachtingen bij de opdrachtgever eerder aan de oppervlakte. Datzelfde geldt als de opdrachtgever moeite heeft met het delegeren van bevoegdheden. Al snel zal blijken, dat dat het werken in het agile-team sterk belemmert. Ten aanzien van dit laatste punt kan dus worden gesteld, dat de agile-aanpak bij dit probleem positief werkt, als de opdrachtgever bereidt is te leren en te veranderen, maar de voortgang aanzienlijk bemoeilijkt als dat niet het geval is.

6. *Te krappe planning*
   Door de incrementele aanpak, komen planningsproblemen bij de agile-aanpak in de meeste gevallen eerder naar boven dan bij de watervalmethode en dat betekent dat ook eerder maatregelen kunnen worden genomen om de planningsproblemen op te lossen. Bovendien is het door de incrementele oplevering van werkende producten mogelijk om op ieder gewenst moment te stoppen (bijvoorbeeld bij het bereiken van een projectdeadline), zonder dat alles dan voor niets is geweest.
   Bij de agile-aanpak is het risico wel groter, dat onder de (tijds)druk van het Time Boxen aan het eind van de incrementen losse eindjes overblijven die bij elkaar achteraf nog veel werk opleveren of desastreus zijn voor het toekomstige onderhoud van het systeem.

7. *Problemen door externe factoren*
   Net als bij de watervalmethoden zullen externe factoren bij agile-projecten vaak leiden tot veranderingen in de gebruikerseisen en –wensen of tot problemen met de planning, en hebben zij daarom dezelfde gevolgen als hierboven bij 1 en 6 beschreven. Het grote verschil is ook hier, dat je op externe factoren minder invloed hebt en de gevolgen vaak dus veel dwingender zijn. Maar het grote voordeel van de agile-methoden is, dat deze expliciet zijn ingesteld op veranderende omstandigheden. En dat maakt dat de gevolgen aanzienlijk kleiner zullen zijn dan bij de watervalmethoden.

Uit bovenstaande kunnen enkele conclusies worden getrokken ten aanzien van de relatie tussen problemen met softwareprojecten en de agile-aanpak:

- Veel problemen kunnen (veel) eerder ontdekt worden, dan bij de watervalaanpak, en dus ook eerder aangepakt.
- En als een project om wat voor reden voortijdig gestopt moet worden, bijvoorbeeld omdat de deadline bereikt is, is in ieder geval het werkende resultaat van de afgeronde incrementen beschikbaar.
- De watervalmethode verhindert goede communicatie tussen de betrokkenen niet, maar de opzet van de agile-projecten stimuleert communicatie, dwingt die zelfs af.
- De agile-aanpak is veel dynamischer dan de watervalmethode en speelt voortdurend in op veranderende omstandigheden.
- De voortdurende samenwerking tussen gebruiker en ontwikkelaar levert in veel gevallen een beter eindresultaat op.
- De agile-aanpak kan een aantal veelvoorkomende projectproblemen verminderen, maar:
  - de agile-aanpak stelt zwaardere eisen aan kennis, ervaring en (communicatieve) vaardigheden van alle teamleden, zowel gebruikers als ontwikkelaars;
  - al deze teamleden moeten gedurende lange(re) tijd full time beschikbaar kunnen zijn voor het projectteam
  - niet alle projecten/systemen zijn even geschikt voor agile-ontwikkeling.

5. CONCLUSIES

Op basis van het voorgaande kan worden geconcludeerd, dat een strikt lineaire watervalaanpak ongewenst is. Die zal dan ook veel minder vaak voorkomen, dan vaak wordt aangenomen. Altijd zal het iteratieve aspect een rol spelen en als er een goede *change control* procedure is, hoeft dat, ook in een voor het overige lineair project, geen probleem te zijn.

Incrementele systeemontwikkeling lijkt belangrijke voordelen te kunnen bieden. Er lijkt eigenlijk maar één echt bezwaar te zijn: al vroeg in het traject van systeemontwikkeling moeten keuzes gemaakt worden, bijvoorbeeld met betrekking tot datamodel/database, die gevolgen (kunnen) hebben voor nog te ontwikkelen incrementen. Maar zoals gezegd: daar staan grote voordelen tegenover. Overigens wordt incrementele systeemontwikkeling in feite vaak al toegepast, aangezien veelvuldig systemen worden opgesplitst in subsystemen die min of meer onafhankelijk van elkaar worden gerealiseerd en geïmplementeerd.

De incrementele vorm van systeemontwikkeling kent trouwens nog een voordeel dat niet eerder genoemd is, omdat het niet direct gekoppeld is aan de problemen met software-projecten: doordat steeds kleine delen in productie gaan, heeft de organisatie de mogelijkheid om eerder de investeringen terug te gaan verdienen, een snellere *return-on-investment*.

De iteratieve en incrementele aanpak kan zeer goed concreter worden ingevuld met, bijvoorbeeld, de gestructureerde analyse van Yourdon. Door eerst voor het gehele systeem een *environmental model* te maken, eventueel met een geheel of gedeeltelijk *behavioral model* en vervolgens op basis daarvan incrementen te definiëren en die één voor één verder te ontwikkelen, ontstaat een gestructureerde wijze van incrementeel ontwikkelen.

De agile-methoden bieden nog extra voordelen op het gebied van de communicatie, maar stellen zulke zware eisen aan de organisatie, dat invoering van de volledige aanpak voor veel organisaties niet realistisch lijkt. Bovendien zijn er duidelijke gevaren verbonden aan het Time Box-principe en aan prototyping. Onder tijdsdruk kunnen verkeerde keuzes worden gemaakt, met name voor de lange termijn, en de basisstructuur van het systeem zal in veel projecten risico's lopen. Hierdoor wordt de kans op problemen in de toekomst, bijvoorbeeld een slechte onderhoudbaarheid, vergroot.

Johan Cruijff was een groot speler en een groot coach, met vaak onnavolgbare redeneringen. Als hij ter verantwoording werd geroepen voor een nederlaag van eerst Ajax en later Barcelona, reageerde hij altijd hetzelfde: het lag niet aan de tactiek, die was goed, nee, het lag aan de spelers, die hadden individuele fouten gemaakt en als ze die niet hadden gemaakt, dan… Wat Cruijff hier over het hoofd ziet, is, dat mensen nu eenmaal niet perfect zijn en dus fouten maken, ook voetballers. En in de tactiek moet je daarmee rekening houden. Als je dat niet doet, kan de tactiek op papier, in de ideale situatie, nog wel kloppen, in de praktijk zal het nogal eens mis gaan. En dan heb je een grote kans om de wedstrijd te verliezen.

Gezien de hoge eisen die in de zuivere vormen van agile-systeemontwikkeling aan organisatie en teamleden gesteld worden, zou wat voor de redenering van Cruijff geldt, ook wel eens van toepassing kunnen zijn op de agile-aanpak. Mogelijk dat de (zuivere) agile-methoden prima zouden kunnen werken in een ideale situatie, maar gevreesd moet worden, dat die ideale situatie eigenlijk vrijwel nooit bestaat. Er zullen altijd essentiële voorwaarden zijn waar niet aan voldaan kan worden. In de literatuur kom je (te) vaak passages tegen als: "wordt vervolgens de pilot snel en effectief

ontwikkeld" of "effectieve samenwerking in beslissingsbevoegde, multidisciplinaire en vooral *enthousiaste, doelgerichte* teams". Tja, allemaal goed en wel, maar hoe bereik je dat dan, die effectieve samenwerking? En wat als de medewerkers in het team net iets minder enthousiast of doelgericht zijn?

Vooral in organisaties, of organisatieonderdelen, met een streng gereguleerde IT, zoals in de farmaceutische- of voedingsmiddelenindustrie, of in transactiegerichte omgevingen, lijkt een zuivere agile-benadering niet goed mogelijk. In dat soort omgevingen staan stabiliteit, zekerheid en betrouwbaarheid voorop. Dat soort eisen zorgt er vaak ook voor dat de cultuur in zo'n organisatie niet geschikt is voor de zuivere agile-benadering. Waar van agile meer succes verwacht kan worden, is in dynamische omgevingen
- waar de gebruikerseisen en –wensen onduidelijk of veranderlijk zijn,
- die in eerste instantie gericht zijn op flexibiliteit en veranderende omstandigheden,
- waarin de *user interfaces* een dominante rol spelen.

Bij het ontwikkelen van bijvoorbeeld marketing-gerichte web-applicaties lijkt agile echt toegevoegde waarde te kunnen bieden.

Tot slot: de zuivere agile-benadering lijkt dus voor veel organisaties en/of organisatieonderdelen nog altijd te ver te gaan, maar elementen uit die benadering, zoals het incrementele en iteratieve, de nauwe samenwerking tussen business en ontwikkelaars en het gebruik van Time Boxing, kunnen heel goed gecombineerd worden met een meer traditionele methode, bijvoorbeeld met de gestructureerde analyse van Yourdon. Optimale systeemontwikkeling zou in veel gevallen in zo'n combinatie gevonden kunnen worden.

SAMENVATTING IN TIEN STELLINGEN

1. Er zijn drie hoofdproblemen met softwareprojecten:
   - te lange doorlooptijd
   - te duur
   - onvoldoende kwaliteit en/of functionaliteit.

2. De oorzaken voor de problemen met softwareprojecten zijn te verdelen over zeven gebieden:
   - onduidelijke of veranderende eisen en wensen van de gebruikers
   - te veel aandacht voor details
   - onvoldoende kennis en ervaring bij gebruiker en ontwikkelaar
   - onvoldoende (goede) communicatie
   - verkeerde verwachtingen bij (lijn)management en opdrachtgever
   - te krappe planning
   - externe factoren.

3. Een strikt lineaire systeemontwikkeling is ongewenst, maar komt in de praktijk ook niet zo vaak voor.

4. Incrementele en iteratieve systeemontwikkeling is zeer goed mogelijk in combinatie met de meer traditionele watervalmethoden.

5. De incrementele/iteratieve aanpak van de systeemontwikkeling heeft grote voordelen in vergelijking met de lineaire methode, mits voldoende aandacht wordt geschonken aan enkele risico's die aan de aanpak kleven.

6. Ook het Time Boxen en de nauwe samenwerking tussen de gebruikersorganisatie en de IT-ontwikkelaars vormen een belangrijke toegevoegde waarde van agile-methoden.

7. Zuivere agile-systeemontwikkeling vraagt zoveel van de organisatie en de teamleden, en brengt zoveel risico's met zich mee, dat het binnen veel organisaties en/of organisatieonderdelen niet realistisch is de aanpak in zijn zuivere vorm in te voeren.

8. Omdat de belangrijkste risico's van agile-systeemontwikkeling de basisstructuur van het systeem en de daarmee samenhangende matige onderhoudbaarheid betreffen, zullen de grootste problemen met een systeem dat met een agile-aanpak is ontwikkeld, pas tot uiting komen nádat de invoering is afgerond.

9. Net als Johan Cruijff speculeert de agile aanpak te veel op de perfecte prestaties van de mensen die de uitvoering voor hun rekening moeten nemen.

10. Als de risicovollere elementen uit de agile-benadering worden gecompenseerd door enkele krachtige onderdelen van de watervalmethode, kan de optimale methode voor systeem-ontwikkeling worden geformeerd.