Temporele doorloop in systeemontwikkeling: recht-toe-recht-aan versus herhaling

Bachelorscriptie geschreven door Juul Coolen

juulcoolen@student.ru.nl
Informatiekunde, NIII, Radboud Universiteit Nijmegen.
16 juni 2008

Begeleid door dr. S.J.B.A. (Stijn) Hoppenbrouwers

"Iteration, like friction, is likely to generate heat instead of progress."

- George Eliot (Engelse Victoriaanse schrijver, 1819-1880)

Inhoudsopgave

HOOFDSTUK 1

Probleemschets

1.1 Inleiding

Systeemontwikkeling, *System Development* of *System Engineering* in het Engels, is een typisch vakgebied waarin de informatiekundige te vinden is. In het bijzonder de *requirements engineering* fase binnen de systeemontwikkeling is informatiekundig van aard. [**Kulak**] besteden hier uitgebreid aandacht aan. Requirements engineering wordt daar omschreven als een proces bedoeld voor het helder voor ogen krijgen van de eisen en wensen van opdrachtgever en gebruiker voor een nieuw te ontwikkelen systeem. Een onderscheid wordt gemaakt tussen de requirements enerzijds en de rest van het systeemontwikkelingsproces anderzijds. De informatiekundige wordt geacht in deze fase door middel van de 'gathering' (het vergaren) en analyse van de requirements een requirements document op te stellen. Dit document vormt vervolgens de fundering voor het vervolg van de systeemontwikkeling.

Er zal nu een definitie gegeven worden van systeemontwikkeling die in de rest van de tekst consequent toegepast zal worden;

def. **Systeemontwikkeling:** Het proces dat doorlopen wordt ten behoeve van de ontwikkeling van een softwaresysteem, bestaande uit de fasen requirements gathering, analyse, ontwerp, bouw, testing, implementatie en onderhoud. **(def. 1.1)**

Bovenstaande definitie is gebaseerd op andere in omloop zijnde definities en is voornamelijk overeenstemmig met die gebruikt in het boek van Kulak en Guiney. Andere definities zijn ook mogelijk, maar de gekozen definitie is gangbaar in de literatuur en gekozen met het oog op het hier besproken onderwerp.

Het onderwerp van deze scriptie is de temporele doorloop in systeemontwikkeling. Daarmee worden specifiek de fasen bedoeld zoals hierboven omschreven, of meer precies: de doorloop van deze fasen. Bij de ontwikkeling van een systeem is het aan de ontwikkelaar de keuze op welke manier de verschillende fasen doorlopen worden. Hij of zij kan bijvoorbeeld kiezen voor een lineaire 'stap-voor-stap' fasering, of een die iteratief (herhalend) van aard is. De implicaties van deze beslissing moeten niet genegeerd worden. Met name die consequenties die van invloed zijn op het resultaat, of liever gezegd het succes ervan, zijn zeer interessant om naar te kijken.

Systeemontwikkeling heeft immers tot doel de creatie van een systeem dat aan de eisen en wensen van de klant en de (eind-)gebruikers voldoet. Voor het slagen van het systeem is het daarom van het grootste belang dat deze personen (de stakeholders) er tevreden over zijn - zowel over het proces om er te komen als het eindproduct zelf. Dat dit nog niet zo makkelijk is en dat de keuze voor een bepaalde temporele doorloop, zo wordt hier gesteld, z'n uitwerking heeft op het succes, valt te lezen in het vervolg van dit hoofdstuk.

1.2 Het probleem

De ontwikkeling van softwaresystemen is een complex proces. Een proces waarin vanalles fout kan gaan. Het is er alles aan gelegen om de risico's zoveel mogelijk te verkleinen. Het ontwikkeltraject moet daarom met zorg ingericht worden.

Zoals te lezen viel in het inleidende deel wordt systeemontwikkeling (hier) gedefinieerd als een soort van stappenplan; verschillende fasen die elkaar min of meer opvolgen. Dit laatste staat centraal in deze scriptie. Men kan namelijk grofweg onderscheid maken tussen

lineaire en *iteratieve*[1] procesdoorlopen. In het eerste geval vindt deze doorloop stap-voor-stap, van-voor-naar-achter plaats. Dit betekent bijvoorbeeld concreet dat eerst een ontwerp van het systeem gemaakt wordt, dit vervolgens gevalideerd wordt, waarna men start met de bouw van het systeem. Kenmerkend aan lineaire systeemontwikkeling is dat men niet meer achterom kijkt; '*what's done is done*'. Het idee is eenvoudig en maakt het werken relatief eenvoudig. Nadeel is alleen dat fouten in eerdere fasen erin blijven zitten tijdens latere fasen, tot en met het eindproduct. Stel dat een programmeur tijdens de bouw erachter komt dat bepaalde implementaties zoals voorgesteld in het ontwerp helemaal niet mogelijk zijn, of alleen wanneer meer tijd - en daarmee meer geld - beschikbaar is, dan is dat in deze traditionele vorm van werken eigenlijk niet meer te veranderen. Wat hier onder lineaire doorloop verstaan wordt, staat in de literatuur ook wel bekend als het 'watervalmodel'.

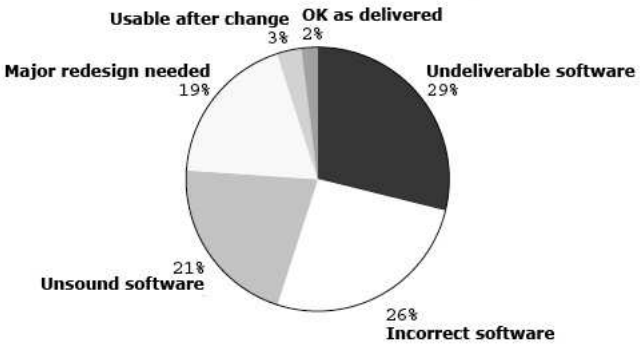
Werken aan de hand van het watervalmodel wordt tegenwoordig vooral als negatief bestempeld door het statische karakter ervan. Missers begaan in de beginstadia zoals tijdens de requirements gathering of het ontwerp worden meegenomen in de erop volgende fasen en zullen uiteindelijk ook terechtkomen in het het systeem dat geacht wordt gebruikt te gaan worden. Een volgende fase betekent meteen ook een afgesloten fase binnen het watervalmodel. In figuur 3.1 zien we het zogenaamde succes van in het heden ontwikkelde softwaresystemen. Al met al blijft slechts een schamele hoeveelheid over van systemen die nog te redden zijn.

Een heleboel geld wordt dus nog steeds verkwist. Fouten kunnen in elk van de fasen voorkomen, maar het blijkt met name dat hoe verder naar het einde toe, hoe duurder het wordt om een fout te herstellen[2]. In dat kader lijkt het belangrijk om zo snel mogelijk eventuele oneffenheden op te sporen en glad te strijken. Of, wellicht, een andere benadering is te kiezen voor een procesdoorloop die robuuster is en daarmee minder foutgevoelig.

Juist dit laatste vinden we terug als basis voor de iteratieve pro-

[1] Vaak worden de de termen 'iteratief' en 'incrementeel' samen genoemd. Iteratief duidt dan op het herhaaldelijk doorlopen van fasen, waarbij incrementeel betrekking heeft op de geleidelijke deel-voor-deel ontwikkeling. Het belangrijkste voor de context van deze scriptie is echter het iteratieve aspect - als tegenhanger van het lineaire 'watervallen'. Het incrementele zal daarom alleen daar waar nodig explicitiet aangehaald worden.

[2] [**Schach**] laat cijfers zien van kosten van wijziging na oplevering die al gauw zo'n honderd keer zo hoog liggen als die tijdens de initiële definitie fasen. In andere literatuur zijn vergelijkbare cijfers zichtbaar.



Figuur 1.1: Het 'succes' van softwaresystemen (bron: A. Loconsole, cs.ume.se)

cesdoorloop. Door het herhaaldelijk uitvoeren van stappen - en daar waar nodig terug te springen - wordt gemeend dat de kwaliteit in het algemeen hoger zal liggen. Wanneer diezelfde programmeur nu iets ontdekt in het ontwerp wat echt niet kan, dan kan bij de iteratieve aanpak teruggegaan worden naar de tekentafel, dus één of meerdere stappen terug. Net zo lang tot alles opgelost is. Hoogstwaarschijnlijk resulteert dit in een systeem met minder fouten.

Zeker zo belangrijk als het aantal fouten is uiteindelijk de tevredenheid van opdrachtgever en gebruiker over het systeem. Zij zullen gebaat zijn bij een systeem dat o.a. gebruiksvriendelijk is en doet wat het moet doen (functioneel valide). Het verschijnsel wat hierbij optreedt - en waar het watervalmodel met name niet mee lijkt overweg te kunnen - is dat de eisen en wensen van de stakeholders over het algemeen continue veranderen gedurende het ontwikkelproces. Dit is het gevolg van de dynamische IT-wereld die voortdurend in beweging is. De concurrentie zit niet stil. Wanneer een bedrijf een nieuw systeem laat ontwikkelen moet dit daarom zo snel mogelijk gebeuren en het liefst heeft men dat halverwege of zelfs tegen het einde toe nog veranderingen erin opgenomen kunnen worden. Wederom wordt iteratieve procesdoorloop hier gezien als de oplossing voor dit euvel. Net zoals bij de ontdekking van een fout, kan bij een veranderde requirement teruggegaan worden en een nieuwe iteratie gestart worden. Het systeem dat er op het eind uit komt rollen wordt geacht nagenoeg foutloos te zijn en perfect aan te sluiten op wat de opdrachtgever en de gebruikers voor ogen hadden.

Om terug te komen op die kosten van wijziging; is het wel zo dat

een iteratieve aanpak over het algemeen goedkoper is dan een lineaire aanpak? Je kunt daar je vraagtekens bij plaatsen, in ieder geval dat is wat wordt gedaan in deze scriptie. Het klinkt allemaal aannemelijk - iteratief impliceert minder fouten impliceert lagere kosten - maar zijn er bijvoorbeeld harde cijfers te vinden die aantonen dat dit ook het geval is? De cijfers die [**Schach**] presenteert geven dan wel aan dat de kosten van wijziging zo'n beetje exponentieel toenemen naarmate het proces vordert, maar wat zegt dit? Als men gaat watervallen, en men doet dit zo secuur dat alleen naar de volgende fase gegaan wordt wanneer alle fouten uit de huidige fase eruitgehaald zijn, dan kost het niet meer dan welke andere methode dan ook - of dit ook haalbaar is, is weer een andere vraag (en is maar de vraag). Iteratief werken daarentegen bevordert misschien zelfs het maken van fouten, omdat men gewoon weer een nieuwe iteratie kan inzetten zodra een fout gevonden wordt. Echter, iedere iteratie kost ook weer geld. Het is dus interessant om te kijken wat hierover geschreven en hopelijk ook aangetoond is. Aannemelijk is wel dat iteratief werken zo z'n voordelen heeft, met name voor de grotere en complexere projecten lijkt het in ieder geval veel beter geschikt dan watervallen. Extra aardig kan het in dat kader zijn om te kijken wanneer het watervalmodel wèl goed uit de verf komt en of het er inderdaad zo beroerd mee is gesteld.

In deze scriptie komt niet alleen het kostenplaatje aan de orde, maar wordt gekeken naar een breder scala aan 'succesfactoren' bij iteratief versus lineair. Welke dit zijn valt te lezen in de hierna volgende opzet en operationalisatie (hst. 2), waarbij beschreven zal worden hoe het geheel is opgezet, hoe de onderzoeksvragen samenhangen, wat de variabelen zijn en welke wetenschappelijke methoden toegepast worden. Voor nu resten nog de hoofd- en deelvragen die behandeld zullen worden in het vervolg en met name onder de resultaten (hst. 3):

Hoofdvraag:

Hebben iteratieve procesdoorlopen meer kans op slagen dan lineaire?

Deelvragen:

1) Welke systeemontwikkelingsmodellen komen we tegen, en wat zijn hun voors en tegens?

2) En hoe zijn ze te categoriseren (lineair/iteratief/...)?
3) Welke succesfactoren kunnen we onderscheiden?
4) Welke modellen zijn succesvol te noemen, in welke situatie en wat zijn hun implicaties?

HOOFDSTUK 2

Opzet

Is iteratief beter dan lineair? Dat is in essentie de vraag die centraal staat in deze scriptie. Maar hoe beantwoord je zo'n vraag? Zoals geschetst in het eerste hoofdstuk hebben we ('de literatuur', ikzelf) zo onze vermoedens, wanneer de een beter werkt dan de ander en welke over het algemeen genomen de betere is. Maar wat is dat "beter" dan? Hoe druk je zoiets uit? Een keuze moet hierbij gemaakt worden alvorens men echt verder kan en dat zal dan ook gebeuren in paragraaf 2.2 onder de noemer "operationalisatie". De eerder genoemde succesfactoren komen daarbij aan de orde. Want 'beter' of 'succesvol(ler)' zijn abstracte en brede begrippen. Vandaar dat een uitwerking in de vorm van definiëring noodzakelijk is alvorens op de vragen van het onderzoek zelf ingegaan kan worden.

Dat laatste gebeurt in hoofdstuk 3, Resultaten. Het is hierbij eerst nog op z'n plaats om de onderzoeks hoofd- en deelvragen toe te lichten. Waarom ze zo gekozen zijn als dat ze hier definitief gesteld staan en hoe ze samenhangen. Paragraaf 2.1 biedt hiervoor de ruimte. Ook zal per vraag toegelicht worden hoe hier gedacht wordt antwoord op gegeven te kunnen worden.

Ten slotte zal in het laatste deel van dit hoofdstuk onder "methoden" een wetenschappelijke verantwoording afgelegd worden in de hoedanigheid van de gebruikte methoden binnen dit onderzoek.

2.1 De aanpak

Om antwoord te kunnen geven op de hoofdvraag, zijn er de deelvragen die gezamenlijk moeten leiden tot een antwoord op die allesoverkoepelende vraag. Gekozen is voor de volgende vier deelvragen:

1) Welke systeemontwikkelingsmodellen komen we tegen, en wat zijn hun voors en tegens?
2) En hoe zijn ze te categoriseren (lineair/iteratief/...)?
3) Welke succesfactoren kunnen we onderscheiden?
4) Welke modellen zijn succesvol te noemen, in welke situatie en wat zijn hun implicaties?

De eerste vraag is voornamelijk oriënterend van aard. Welke 'smaken' zijn er zoal te vinden? Deze vraag staat deswege niet in directe relatie tot de onderzoeksvraag, maar de bevindingen zouden wel een beter inzicht in het het kader van het onderwerp en het probleem moeten geven.

Vraag twee speelt in op de eerste door een categorisatie van deze modellen voor te stellen. Een die al reeds voorgesteld werd in de titel van deze scriptie. Namelijk die van lineair ('recht-toe-recht-aan') versus iteratief ('herhaling'). Deze twee vragen met hun bijbehorende bevindingen zullen ons laten zien (dat kan alvast verklapt worden) dat er inderdaad zoiets bestaat als een tweedeling binnen de systeemontwikkeling tussen een lineaire temporele doorloop en een iteratieve temporele doorloop.

Een verscheidenheid aan bronnen zal geraadpleegd worden bij de beantwoording van deze eerste twee deelvragen. Aangezien de beantwoording nogal opsommend van aard zal zijn, hoeft de geraadpleegde literatuur niet een dermate kritische inhoud te hebben. Enkele tekstboeken en een encyclopedie als die van Wikipedia zullen voldoen.

De derde deelvraag hangt nauw samen met de operationalisatie van het onderzoek, welke hierna in paragraaf 2.2 uitgewerkt zal worden. Als je een vergelijking wilt maken tussen twee verschijnselen, in ons geval de verschijnselen lineaire en iteratieve doorloop, waarbij aan te geven moet zijn wanneer de een *beter* of *succesvoller* is dan de ander, dan moet dat uitdrukbaar zijn. De "succesfactoren" zijn dan eigenlijk de variabelen waarnaar gekeken wordt. Welke dat zijn

valt al voor een klein deel te lezen in de operationalisatie van 2.2 en uitgebreider in het derde hoofdstuk onder het kopje van deze derde deelvraag.

Deelvraag nummer vier is toch wel de vraag waar het hier allemaal om draait. Van wat zaken ontdaan kunnen we misschien wel stellen dat het niet veel meer is dan een verkapte versie van de hoofdvraag. De deelvraag echter kijkt meer naar de verschillende modellen, waar de hoofdvraag puur de categorisatie ervan in lineair versus iteratief belicht. Een beantwoording van de hoofdvraag, of in ieder geval een poging daartoe, is onderwerp van discussie in de afsluitende conclusie (3.2). Antwoorden op deze deelvraag en ook meteen de hoofdvraag zijn hopelijk te vinden in literatuur die specifiek ingaat op respectievelijk het succes van deze modellen en het succes van de twee temporele doorlopen. Voor een deel is het dan zelf conclusies trekken en hopelijk ook voor een deel bevindingen van anderen in de vorm van harde cijfers tegenkomen. Wetenschappelijke artikelen zullen hierbij de belangrijkste en interessantste bron van informatie zijn.

2.2 Operationalisatie

In essentie draait het in deze scriptie om de geldigheid van de stelling *iteratief* > *lineair* (of het omgekeerde), in zekere situaties bovendien. Zo stellend kunnen we drie termen onderscheiden: *lineair*, *iteratief* en het > teken (of <). Een (voldoende) exacte omschrijving van deze drie termen hebben we nodig voor het vervolg, om antwoord te kunnen geven op de onderzoeksvraag. Een poging tot het geven van een dergelijke omschrijving zal dan ook hier plaatsvinden en 'stiekem' is dat ook wat gebeurt bij de deelvragen een, twee en drie, vanuit een verschillend perspectief weliswaar. De eerste twee vragen samen namelijk laten inzichtelijker zien wat we verstaan onder 'lineair' en wat onder 'iteratief'. Deelvraag drie gaat dan over het 'groter dan'-teken.

Kulak & Guiney ([**Kulak**]) beschrijven *iteratieve systeemontwikkeling* metaforisch als het verven van een huis. Eerst brengt men een grondverf aan - welke een totaal andere kleur kan hebben dan de uiteindelijke kleur. Daarna komt de goede kleur, welke wellicht nog één of meer keren moet worden aangebracht voordat het geheel egaal is. Het herhaaldelijk toevoegen van niveaus van detaillering en

definiëring is wat hier verstaan wordt onder *iteratief*. [**Press**] echter gebruikt in zijn boek een andere indeling. Namelijk die in waterval, incrementele en "evolutionaire" procesmodellen. Hij noemt zowel de incrementele als de evolutionaire procesmodellen iteratief. Het verschil zit 'm erin dat incrementeel gaat over iteraties die functionaliteit toevoegen, waar evolutionair slaat op een product dat 'evolueert' (resultaat in het begin vaak nog niet tot in detail bekend) over de tijd maar niet per sé met extra functionaliteit per iteratie. Hier zit een bepaalde nuance tussen, maar wordt hier in de definiëring samengeslagen en staat iteratief meer voor alles dat niet 'watervallend' van aard is.

Zoals, ietwat suggestief, voorgesteld in de inleidende probleemschets (hoofdstuk 1) verdacht ondergetekende ontwikkelaars ervan dat zij binnen een iteratieve systeemontwikkelingsdoorloop wellicht gemakszuchtiger te werk te gaan dan wanneer men een watervalmodel zou hanteren. Dit omdat iteratie foutherstel over de fasen heen als een noodzakelijk iets accepteert, in tegenstelling tot bij watervallen. Een fout wordt dan al snel geclassificeerd als 'minder erg', wat misschien niet terecht is. Kulak & Guiney verwijzen in dit kader naar een van de werken van Jacobson ([**Jacob**]), waarin onder iteratief **niet** het excuus voor ontwikkelaars om 'zomaar wat te doen' verstaan wordt. Maar de praktijk is vaak anders. Watervallen werkt in theorie ook prima, in de praktijk daarentegen schiet het regelmatig te kort, zeker tegenwoordig met issues als 'time-to-market' en 'adaptability'.

def. **Iteratieve temporele doorloop:** Het doorlopen van het systeemontwikkelingsproces met de verschillende fasen ten behoeve van de ontwikkeling van een softwaresysteem, waarbij elke iteratie een niveau van detaillering en definiëring toevoegt en waar nodig als het ware 'teruggesprongen' kan worden. **(def. 2.1)**

Lineair is de tegenhanger van iteratief en wordt hier gelijkgesteld aan watervallen, vertegenwoordigd in het alombekende watervalmodel. Simpelweg houdt dit in dat pas naar een volgende fase binnen het systeemontwikkelingsproces wordt doorgegaan als de huidige fase volledig is afgerond en alleen voorwaarts, niet terug. Terugspringen wordt, in tegenstelling tot bij iteratief, nu gezien als 'fout' en ongewild. Iets wat vertraging en kosten met zich meebrengt en niet

is ingecalculeerd.

def. **Lineaire temporele doorloop:** Het doorlopen van het systeemontwikkelingsproces met de verschillende fasen ten behoeve van de ontwikkeling van een softwaresysteem, waarbij sequentieel door de fasen heen gelopen wordt en alleen naar de volgende fase gegaan wordt zodra de huidige volledig is afgerond. Terugspringen wordt zoveel mogelijk vermeden. **(def. 2.2)**

Om het verschil tussen iteratief en lineair in één zin samen te vatten: de iteratieve aanpak onderkent de uitdagingen van hedendaags softwareontwerp en calculeert terugspringen door middel van iteraties in, waar de lineaire aanpak terugspringen ziet als iets wat ten alle tijden voorkomen dient te worden omdat het niet bevordelijk zou zijn voor het ontwikkelingsproces. Verschillende visies waarbij er in de loop van de tijd een sterke verschuiving richting de iteratieve aanpak heeft plaatsgevonden.

Ons rest nu alleen nog een definiëring van het 'groter dan'-teken. Veel hoeft er op dit moment niet over gezegd te worden, aangezien het voor het overgrote deel het onderwerp betreft van deelvraag nummer drie. De $>$ zouden we kunnen invullen als 'beter dan'. En dat 'beter dan' kan op verschillende dingen, of variabelen, slaan. Variabelen zijn dan bijvoorbeeld *time-to-market* (of simpelweg *tijd*), *(geld)kosten* of *klanttevredenheid*. Deze variabelen vormen de succesfactoren van de verschillende doorloopmodellen. Uiteindelijk wanneer we willen beantwoorden welk van deze modellen de betere is, kunnen we antwoorden in termen van 'a kost minder geld dan b', of 'de klanttevredenheid bij a is groter dan bij b', bijvoorbeeld. Dit is nu nog behoorlijk breed en simpel uitgedrukt, maar hier zullen vanzelf gaandeweg de nodige nuances en verfijningen in aangebracht worden.

2.3 Methoden

De toegepaste wetenschappelijke methode in dit onderzoek is die van een (vergelijkende) literatuurstudie. Er zal getracht worden om zoveel mogelijk informatie over de tweestrijd tussen lineair en iteratief vanuit verschillende invalshoeken en uit diverse literatuur verkregen te worden. De gebruikte boeken en artikelen zijn allemaal

voor zover mogelijk ‘wetenschappelijk verantwoord’ en bronnen van Internet zullen veelal ter inspiratie en ondersteuning dienen en niet in de eerste plaats als enige bron gelden.

Literatuur is voor een groot deel afkomstig uit in colleges gebruikt werk en via hierin vermelde referenties. Helaas is een dienst als Gartner niet gratis en is Springer niet altijd even bruikbaar. Gelukkig zijn ook op plaatsen als de ACM Portal, de digitale bibliotheek van de IEEE Computer Society en Google Scholar interessante items te vinden.

HOOFDSTUK 3

Resultaten

We zijn aanbeland bij de daadwerkelijke uitwerking van de onderzoeksvraag. In paragraaf 3.1 wordt gepoogd antwoorden te geven op de verschillende deelvragen (zie voor een overzicht 1.2 en 2.1). Samen moeten deze antwoorden, incrementeel, leiden tot een antwoord op de hoofdvraag. Deze eerste paragraaf zal bestaan uit een viertal subsecties, één voor ieder van de deelvragen. Echter, het is niet altijd mogelijk om puur op de vraag in kwestie in te gaan zonder soms een uitstap naar of vermelding van een van de andere onderzoeksonderdelen te moeten maken. De structuur zoals aangegeven door de onderzoeks deel- en hoofdvragen zal wel altijd zo goed als het kan gehandhaafd blijven. Enige overlap is alleen moeilijk te voorkomen. Paragraaf 3.2 ten slotte trekt samenvattend conclusies met betrekking tot de hoofdvraag. Het streven is om tegen die tijd een goed beeld te hebben van het hele 'lineair versus iteratief' en wellicht wel een 'winnaar' uit te kunnen roepen.

3.1 Bevindingen

3.1.1 De modellen

- *Welke systeemontwikkelingsmodellen komen we tegen, en wat zijn hun voors en tegens?*

In het kort draait het hier om de modellen die er bestaan voor

systeemontwikkeling in het licht van de temporele doorloop. Ieder van de modellen heeft zo z'n eigen manier van het doorlopen van de stappen requirements tot en met onderhoud[1]. Het watervalmodel bijvoorbeeld doet dit 'recht-toe-recht-aan'. De fasen worden van voor naar achter doorlopen, als in de metafoor van het stromen van water. Voor zoverre hebben we naast het watervalmodel alleen nog iteratie gezien, maar iteratie is een categorisatie en geen model op zich. Zo hadden we het watervalmodel in zekere zin als definitie voor de categorie lineaire temporele doorlopen gebruikt. Het watervalmodel staat gelijk aan lineair, buiten dit model zijn geen van de modellen lineair te noemen - uitgezonderd van het sterk van het watervalmodel afgeleide V-model, waarover later meer. Iteratief van aard zijn meerdere modellen. Deze categorisatie in lineair, iteratief en misschien nog andere categorieën is echter het onderwerp van deelvraag 2 (3.2). Hier worden nu de bekende systeemontwikkelingsmodellen voor temporele doorloop beschreven met hun voor- en nadelen. Achtereenvolgens zijn dit:

- Watervalmodel
- V-model
- Spiral Model
- Agile[2]
- RAD
- RUP
- XP
- Scrum

Je kunt kritische noten plaatsen bij bovenstaande lijst. Waarom gekozen voor juist deze modellen? Want er zijn er nog wel meer in de omloop. Vanzelfsprekend moet ergens een streep getrokken worden en is een volledige lijst haast onmogelijk realiseerbaar. Nou is deze lijst al vrij omvangrijk en bevat, de literatuur erop naslaand (gebruik

[1] Dit is corresponderend met definitie 1.1. In paragraaf 2.2 echter hadden we bij iteratief de fasen implementatie en onderhoud erbuiten gelaten. Niet ieder model focust dus op exact dezelfde fasen.

[2] Agile wordt ook weleens in één adem genoemd met iteratief. De keuze om hier agile op te nemen en niet iteratief zal in de loop van 3.1.1 en in 3.1.2 bij de beantwoording van de tweede deelvraag duidelijk worden.

makende van het overzicht op [**Wiki**] en de boeken van [**Kulak**] en [**Press**]), toch wel de bekendste modellen. Ook moet met het oog op deelvraag twee de categorisatie in de gaten gehouden worden - een voorbeeld van een overweging horende hierbij is wat staat beschreven onder de voetnoot bij agile in de lijst. Nogmaals benadrukkend, bovenstaande indeling is gebaseerd op de categorisering op het vlak van de temporele doorloop van de lifecycle procesmodellen, volgend in 3.1.2. In [**Press**] wordt dit beschreven als het perspectief vanuit de workflow; de sequentie van activiteiten.

Nu de keuze voor de modellen vaststaat, volgt in de aansluitende onderdelen korte beschrijvingen en de voornaamste voor- en nadelen van deze modellen.

Watervalmodel

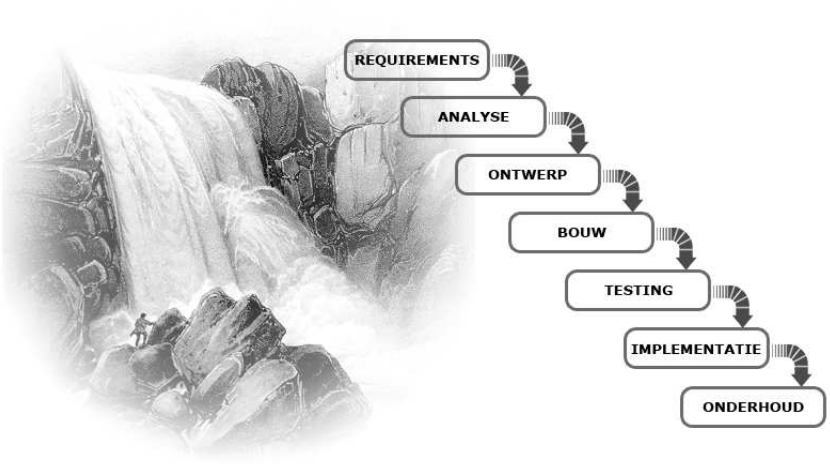
Watervallen wordt in één zin door Kulak & Guiney samengevat als: "*waterfall refers to a way of developing software that proceeds through one activity at a time, completes it fully, then moves on to the next*". De activiteiten of fasen waarnaar hier verwezen wordt zijn die van definitie 1.1 (tenminste dat is de definitie zoals die hier gehanteerd wordt). De voor- en nadelen zoals aangegeven door Kulak & Guiney:

+ overzichtelijk doordat alles strikt gefaseerd plaatsvindt	- fouten ontdekt in een latere fase zijn lastig tot niet te herstellen
+ deadlines, personeel en kosten makkelijker in te plannen	- wijzigingen vanuit de klant zijn ongewenst
+ geen dure herstelwerkzaamheden, omdat terugspringen zoveel mogelijk vermeden wordt	- het verzamelen van alle requirements ineens is in de praktijk onmogelijk

Tabel 3.1: Voor- en nadelen van watervallen

Het meeste van het tabel 3.1 vermelde zijn we al eens eerder tegengekomen. En met name op de nadelen zal nog uitgebreid op teruggekomen worden bij deelvraag vier en de conclusie van 3.2. Echter, puntje drie van de nadelen verdient nu wel een kleine toelichting. Dat een volledige requirements gathering vooraf niet mogelijk is komt 1) door de *human nature* van de stakeholders van het systeem: mensen zijn nu eenmaal niet in staat om altijd volledig en voldoende precies te zijn ('we zijn ook maar mensen') en 2) door on-

vermijdelijke veranderingen binnen bedrijven van de eenentwintigste eeuw.



Figuur 3.1: Watervallen

V-model

Het V-model is niet veel meer dan een uitbreiding van het watervalmodel zoals we dat hiervoor zagen. Wat het V-model in de kern toevoegt aan het watervalmodel zijn parallelle fasen van testing. Dus gelijk met elke fase uit het watervalmodel loopt een test fase. Het model dankt z'n naam - hoe onverwacht - aan de V-vorm die het uitbeeldt; het loopt eerst neerwaarts tot en met de coding/bouw fase en daarna opwaarts met de parallelle testing fasen. Ter illustratie, in de uiteinden bevinden zich dus de requirements analyse (links) en acceptance testing (rechts). De voor- en nadelen, hoewel niet altijd even duidelijk vast te stellen (bronnen niet eenduidig), zijn o.a.:

Spiral Model

Ruw gezegd steunt het Spiral Lifecycle Model op drie pilaren, te weten: het watervalmodel, prototyping[3] en iteraties. Je zou daarmee kunnen zeggen dat het Spiral Model precies tussen lineair en iteratief

[3]Er wordt ook weleens gesproken van het Prototyping Model, waarmee meestal verwezen wordt naar een op protypes gebaseerd watervalmodel.

+ fouten worden snel gevonden en hersteld (goedkoper)	- veel parallel werk, veel mensen op ieder tijdstip nodig
+ voorkomt de neerwaartse stroom van fouten zoals bij het watervalmodel	- niet geschikt voor korte-termijn projecten door de vele review-momenten
+ overzichtelijke fasering	- houdt nog steeds geen rekening met veranderingen

Tabel 3.2: Voor- en nadelen van het V-model

inzit. Het doorloopt namelijk oppervlakkig de stappen van het watervalmodel door gebruik te maken van prototypes, welke weer door middel van iteraties steeds verder verfijnd worden. Dit gebeurt net zolang totdat de klant tevreden is; het prototype komt voldoende overeen met het product dat men voor ogen heeft.

+ belangrijke issues komen snel aan het licht	- iteraties i.c.m. prototyping kunnen leiden tot eenvoudig te voorkomen redundant werk
+ veranderingen kunnen in de volgende iteratie worden opgenomen	- niet meer probleemgestuurd, aandacht komt nu eerder op de (in eerste instantie) minder belangrijke details te liggen
+ veel feedback met de klant, daardoor een aannemelijk hogere kwaliteit realiseerbaar	

Tabel 3.3: Voor- en nadelen van het Spiral Model

Agile

Agile is hét begrip van vandaag. Je ontkomt er niet meer aan binnen de systeemontwikkeling. Agile staat voor alles wat goed is en voor alles wat watervallen niet is. Tenminste, zo wordt verkondigd (agilemanifesto.org). In principe is agile system development een vorm van iteratief werken - we lopen hier noodgedwongen al enigszins op de zaken vooruit (zie 3.1.2) - gekenmerkt door de volgende specifieke eigenschappen:

- iedere iteratie levert een stukje software op (hoge productiegraad);
- een iteratie omvat een geheel softwareproject van requirements tot en met testing;
- projectteam verdeeld in kleinere, zelfstandige en heterogene groepen (informatiekundigen, programmeurs, testers, klanten etc.);
- voorkeur voor face-to-face communicatie;
- de software vormt de maat voor de voortgang, documentatie ondergeschoven kind.

+ springt makkelijk om met continu veranderende requirements	- zelfstandigheid en iteraties kunnen leiden tot zichzelf voorbij werkende groepen (als 'kippen zonder kop')
+ intensief contact binnen de groepen en met de klant, bevordert kwaliteit en tevredenheid	- projectgroepen kunnen niet te groot worden omdat dat face-to-face communicatie lastig maakt
+ zichtbare en constante voortgang	- beperkte structuur en noodzakelijke documentatie
	- hoge vereisten op het gebied van cultuur en discipline om het geheel te laten functioneren
	- inefficiëntie wanneer requirements veranderen en programmatuur opnieuw geschreven moet worden

Tabel 3.4: Voor- en nadelen van Agile

Het aantal nadelen ten opzichte van de voordelen lijkt iets te suggereren. Niet dat agile per definitie zo slecht is - bovendien kunnen die enkele voordelen zwaarder wegen - er is alleen her en der wel wat kritiek leesbaar. Dat terwijl agile en iteratief zo'n beetje overal gepredikt wordt als tegenhanger én vervanger van watervallen. We raken hier weer de kern van de onderzoeksvraag die in deze scriptie gesteld staat. Het is daarom nog niet het moment om hier verder op in te gaan. Dat zullen we bewaren voor verderop in het verhaal.

RAD (Rapid Application Development)

RAD oftewel Rapid Application Development heeft veel weg van agile werken. Kulak & Guiney beschrijven RAD als een methode waarbij de participatie van de business users groot is dankzij zo-

genaamde JAD (joint application development) sessies, waarbij in feedback loops prototypes ontwikkeld worden. Het heeft wat weg van agile werken, met het verschil dat RAD intensief gebruik maakt van wat men aanduidt met *time boxing*. Hierbij wordt een project opgedeeld in relatief korte perioden van normaal zo'n twee tot zes weken.

+ versnelde ontwikkeling dankzij (rapid) prototyping, daardoor minder tijdsgerelateerde risico's	- opgeleverde tussentijdse producten zijn nog vaak gebrekkig in functionaliteit
+ intensieve user participatie, bevordelijk voor functionele kwaliteit	- prototypes kunnen afleiden van de functionele requirements en zorgen voor een grotere focus op de (onbelangrijke) details (algemeen prototype-probleem)

Tabel 3.5: Voor- en nadelen van RAD

RUP (Rational Unified Process)

Het Rational Unified Process framework is een product van Rational Software en tegenwoordig IBM ([**IBM**]). De nadruk ligt hier meer op dat het een (aanpasbaar) framework betreft dan een enkel, concreet voorschrijvend proces. Software teams hebben de mogelijkheid het aan hun eigen wensen en behoeften aan te passen. Het RUP framework is beschikbaar in de vorm van een softwarepakket dat een knowledge base omvat met voorbeeld artifacts en beschrijvingen voor de verschillende activiteiten. RUP steunt heel erg op de *best practices* voor systeemontwikkeling en het management hiervan. Op deze best practices komen we later nog terug, bij de beantwoording van de laatste deelvraag en de conclusies van dit onderzoek.

De RUP lifecycle verdeelt de verschillende activiteiten onder in fasen en iteraties. De fasen heten achtereenvolgens de 'Inception phase', 'Elaboration phase', 'Construction phase' en de 'Transition phase'. Een exacte beschrijving van deze fasen valt buiten de scope van deze scriptie (zie hiervoor [**Kruch**]), maar grofweg kunnen deze fasen gemapt worden naar de verschillende stappen zoals onder andere beschreven in het watervalmodel. De fasen van RUP kunnen geïtereerd worden wanneer nodig. Daarmee heeft het onder de motorkap ook iets weg van het Spiral Model zoals eerder beschreven.

+ sterk gestructureerd en gemanaged framework, bevorderlijk voor kwaliteit	- wellicht teveel van het goede voor kleinere organisaties
+ fundering gevormd door best practices en key principles, wederom bevordelijk voor kwaliteit	- RUP experts geen overbodige luxe bij het installeren van het framework
+ sterke fasering en iteraties maken het geschikt voor moderne bedrijven	- sterk multi-disciplinair, wat betekent dat een grote variëteit aan mensen aanwezig moet zijn

Tabel 3.6: Voor- en nadelen van RUP

We zien dan ook dat met name de grotere organisaties implementaties van RUP gebruiken; Ernst & Young om er maar een te noemen. Hiermee wordt het grootste nadeel van RUP samengevat: voor kleinere organisaties zal het framework zich niet lonen.

XP (eXtreme Programming)

Extreme Programming is in de kern agile, in de zin dat het ontwikkelingsproces flexibel reageert op user needs (veranderende requirements). De gebruiker staat wederom centraal. Kosten (van wijziging) zouden dankzij het agile karakter laag moeten blijven, in ieder geval lager dan bij traditionele methoden zoals het watervalmodel. De vier hoofdactiviteiten zoals beschreven binnen XP, zijn: 'Coding', 'Testing', 'Listening' en 'Designing', waarbij XP-adepten aan de eerste de meeste waarde hechten. [**Wiki**]

+ nadruk op communicatie met klant, bevordelijk voor kwaliteit en tevredenheid	- code-georiënteerd i.p.v. design-georiënteerd
+ nadruk op teamwork en communicatie, performance verhogend	- er wordt weinig gedocumenteerd wat ten koste gaat van het overzicht
+ incrementele ontwikkeling, conform iteratief/agile werken	- testing en reviewing verloopt vaak weinig gestructureerd
+ continu testing en reviewing, fouten worden daarmee snel gevonden	- beperkte kwaliteitsplanning

Tabel 3.7: Voor- en nadelen van XP

Met name blijkt dat bij software vanaf enkele duizenden regels code

of bij projectteams bestaande uit meer dan een handvol mensen, de XP methodiek al snel het overzicht doet laten verliezen.

Scrum

De termen 'scrum' en 'sprint' hoor je tegenwoordig binnen de systeemontwikkeling nog weleens vallen. Het is een vorm, respectievelijk onderdeel van agile werken. Het lijkt heel sterk op RAD wat we eerder besproken hebben. Scrum procesdoorlopen zijn uitdrukkelijk vastgelegd (met meetings, logs, user roles, etc.) en bestaan uit zogeheten sprints van meestal een maand. Het bedrijf Planon gevestigd in Nijmegen hanteert onder andere 'de scrum', en is een van de 'inspiratiebronnen' geweest bij de keuze van het onderwerp voor deze scriptie.

Tot slot

Zoals aangegeven was deze deelvraag voornamelijk oriënterend van aard. Deelvraag twee (3.1.2 hierna) zal hierop verder inspelen door deze verschillende modellen te gaan categoriseren. Geleidelijk aan zullen we gaan toewerken naar een beantwoording van de hoofdvraag.

3.1.2 Categorisatie

- *Hoe zijn de systeemontwikkelingsmodellen te categoriseren (lineair/iteratief/...)?*

De beantwoording van deze deelvraag is tweeledig. Aan de ene kant betreft het een beantwoording van de vraag als zodanig, aan de andere kant omvat het een verdediging van de titel van de scriptie. Er volgt nu eerst een indeling van de bij deelvraag één opgesomde modellen aan de hand van deze categorisatie (lineair tegenover iteratief) in tabel 3.6.
Het watervalmodel en het V-model zijn de twee lineaire modellen, waarvan watervallen de meest 'pure' is. Het Spiral Model zou je het overgangsmodel van lineair naar iteratief kunnen noemen aangezien het elementen van beide bevat. Onder de noemer iteratief valt een bonte verzameling van modellen, echter veel van hen zijn op de hoop 'agile' te vegen.

Lineair:	Iteratief:
Watervalmodel	Spiral Model
V-model	Agile
	RAD
	RUP
	XP, Scrum ...

Tabel 3.8: De categorisatie van de modellen uit 3.1.1 in lineair en iteratief

Kijkend naar tabel 3.6 lijkt de categorisatie zoals vanaf het begin voorgesteld niet onredelijk. De modellen zijn netjes op te delen. Natuurlijk hadden we ook een indeling kunnen maken op een andere basis. Bijvoorbeeld tussen prototype-based en *non*-prototype-based modellen. Maar de focus ligt hier op de temporele doorloop. Het dichtst in de buurt komende alternatief zou dan nog tussen lineaire (non-agile) en agile modellen zijn, maar aangezien agile in de kern iteratief is ($agile \subset iteratief$), vormt iteratief een beter label.

In de literatuur is bovenstaande tweedeling algemeen aanvaard. Iteratief wordt gezien als de opvolger van lineair ontwikkelen, waarbij iteratief vaak wordt aangeduid met agile. Iteratief/Agile springt in op de behoefte aan een dynamische ontwikkeling van softwaresystemen waar veranderende requirements aan de orde van de dag zijn en zonder problemen kunnen worden meegenomen in de software. Op dit laatste komen we nog uitgebreid terug in het vervolg.

Samenvattend kijken we naar de doorloop van lineaire systeemontwikkeling versus die van iteratieve systeemontwikkeling. Voor een formelere beschrijving van de twee zie definities 2.1 en 2.2. Eerder was in het licht van iteratief de term 'incrementeel' al een keer gevallen. Beloofd was dat hier nog op teruggekomen zou worden. Daarvoor nu het moment. Incrementeel hangt nauw samen met iteratief. Het verschil is subtiel. Waar iteratief (volgens def. 2.1) in iteraties verbeteringen en detaillering toevoegt aan eenzelfde object, daar voegt incrementeel telkens een nieuw object toe. Kulak & Guiney omschrijven het als het bouwen van een huis kamer voor kamer. Iteratief en incrementeel zijn onlosmakelijk met elkaar verbonden, met tot gevolg dat iteratief werken incrementeel werken noodgedwongen impliceert.

3.1.3 Succesfactoren

- *Welke succesfactoren kunnen we onderscheiden?*

We waren er bij de operationalisatie van 2.2 al aan begonnen, maar een complete uitwerking van deze succesfactoren hadden we uitgesteld tot dit moment. Om te kunnen zeggen wanneer iteratief of lineair 'beter is dan' de ander hadden we in 2.2 het 'groter dan'-teken als een soort van placeholder gebruikt. Dit teken wordt nu vervangen door enkele succesfactoren. Met deze factoren duiden we dus bepaalde eigenschappen van systeemontwikkelingsdoorlopen aan. Denk om te beginnen aan de kosten; kost iteratief ontwikkelen over het algemeen genomen meer of minder dan lineair ontwikkelen? Of bevat een van hen voorzorgsmaatregelen of capaciteiten om bepaalde kosten veroorzaakt door bepaalde verschijnselen te minimaliseren (b.v. costs of change)? We zullen nu een aantal verschillende succesfactoren bespreken welke volgen uit de literatuur. Ook zal aangegeven worden welke hier het belangrijkst geacht worden (prioriteiten).

De meest voor de hand liggende succesfactor die we kunnen onderscheiden zijn de kosten (*costs*). Deze zijn op te delen in kosten voor ontwikkeling (oftewel *development costs*) en kosten door wijziging (*costs of change*). Alvast enigszins vooruitlopend op de zaken, het is niet moeilijk voor te stellen dat de eerste variant lager uitvalt bij lineair ontwikkelen en de tweede bij iteratief ontwikkelen. Maar daarover ... later meer. Development costs zijn dan alle kosten die komen kijken bij de ontwikkeling van het softwaresysteem, welke voor een groot deel afhankelijk zijn van de omvang en complexiteit van het systeem. Deze vertalen zich weer naar de grootte en samenstelling van het projectteam, de looptijd, etc. De costs of change horen hier dus niet bij, dit zijn de kosten als gevolg van veranderende requirements of herstel van fouten. Kosten zijn belangrijk, maar dat hangt ook van de situatie af. Wellicht dat een opdrachtgever die zich begeeft in een extreem concurrerende markt het belangrijker vindt dat zijn of haar nieuwe softwaresysteem als eerste op de markt komt dan wat het prijskaartje is.

Zeker in de IT sector is *time-to-market* van essentieel belang. De markt verandert continu. In dit licht wordt hier aan deze tijdsfactor een hogere prioriteit toegekend dan aan de ontwikkelingskosten. De costs of change echter zijn ook zeer belangrijk, omdat deze al snel

exponentieel kunnen oplopen. De development costs zijn wat vlakker en daarmee minder interessant. Time-to-market gaat over de tijdspanne die er is tussen het tot stand komen van het verzoek van de klant voor een nieuw systeem en het moment van oplevering. Dit kan ook een tussenproduct betreffen. Met de marktwerking van nu is het niet ongewoon om een nog niet volledig product op de markt te zetten. Het is dan vaak belangrijker dat het product er is en dat het tevens op tijd genoeg is om de concurrentie voor te blijven, dan wel bij te benen. Met dit gegeven zou je kunnen denken dat iteratieve, incrementele en/of bijvoorbeeld prototyping modellen op dit vlak de voorsprong hebben.

Een derde succesfactor die we hier onderscheiden is de *foutgevoeligheid* van modellen. Er werd al eerder gesuggereerd dat modellen die doorlopen worden middels iteraties wellicht een soort van vrijbrief aan ontwikkelaars geven om fouten te maken, 'omdat deze toch wel in een volgende iteratie hersteld worden'. Of er sprake is van een dergelijke instelling en of dat een negatieve invloed heeft op project en product zullen we hopelijk later zien.

Nauw hiermee samenhangend zijn de factoren *agility* en *end-user satisfaction*. Met de eerste verwijzen we naar de capaciteit van modellen om met veranderende requirements om te gaan. Samen met de foutgevoeligheid zijn dit drie kwaliteitsaspecten van de modellen. End-user satisfaction betreft de tevredenheid van de eindgebruiker en is meer het gevolg van de mate van agility van het toegepaste model. End-user satisfaction heeft in zekere zin al van de hiervoor genoemde factoren in zich. Maar uiteindelijk is dit de belangrijkste factor. Is de eindgebruiker ontevreden over het opgeleverde product, dan kan het nog zo snel opgeleverd zijn, nog zo goedkoop ontwikkeld zijn, het product zal niet of met tegenzin gebruikt worden en is gedoemd te falen.

Opsommend en rangschikkend

> End-user satisfaction
>> Agility
>>> Costs of change
>>> Time-to-market
>>> Foutgevoeligheid
>>>> Development costs

Bovenstaande rangschikking geeft de prioriteit aan van hoog naar laag. End-user satisfaction heeft de hoogste prioriteit toegewezen gekregen. Costs of change, time-to-market en foutgevoeligheid zijn van gelijke prioriteit.

3.1.4 Succesvolle modellen

3.1.3 en 3.1.1 worden nu gecombineerd door per succesfactor de modellen door te nemen. Is de time-to-market van het watervalmodel hoog? Zijn de costs of change bij XP laag? Al deze aspecten komen hier aan de orde. Per succesfactor zal bovendien in een overzicht aangegeven worden hoe ieder van de modellen hierop 'scoort'.

Development costs

Over de ontwikkelkosten bij de diverse modellen valt moeilijk gegevens te vinden. Bovendien zijn, zoals al eerder aangegeven, de costs of change van veel groter belang omdat deze soms wel exponentieel kunnen toenemen en daardoor een veel aanzienlijker deel van de totale kosten voor hun rekening nemen.

In een door Marko van Eekelen ([**MvE**]) gebruikte collegepresentatie valt een cijfer te zien van 15%, het aandeel van de ontwikkelkosten in de totale kosten. Afgezien van de significantie van de hoogte van dit cijfer, valt in een voetnoot te lezen dat het gekozen lifecyle model alleen van invloed is op de verdeling van deze 15%, niet van het totaal. Dit impliceert dat de ontwikkelkosten in hun totaliteit niet (bijzonder) variëren naar gelang het gekozen model.

Redenerend zou wel gesteld kunnen worden dat iteratieve modellen over het algemeen hogere development costs met zich meebrengen. Iteratieve modellen vereisen namelijk meer redundant werk en grotere projectteams met een sterkere verdeling van de functies. Requirements engineers, coders, testers, maar ook klanten (i.v.m. feedback loops) moeten bij iedere iteratie aanwezig zijn. In lineaire modellen zoals het watervalmodel kan vaak met een minder omvangrijk projectteam volstaan worden. Is de requirements fase afgerond, dan zijn in principe de requirements engineers klaar met hun werk. Bij iteratief ontwikkelen vergen bovendien al de iteraties en functies meer management - een 5-tot-1 ratio van ontwikkelaars tot managers volgens [**Kruch2**].

Laag: (lineair)	Hoog: (iteratief)
Watervalmodel	Spiral Model
V-model	Agile
	RAD
	RUP
	XP
	Scrum

Tabel 3.9: Developments costs

Costs of change

Typerend genoeg vertonen deze kosten een omgekeerd patroon; ze zijn hoger bij de lineaire modellen en lager bij de iteratieve. Dit komt simpelweg door de natuur van deze twee categorieën van modellen. Iteratieve modellen kunnen veranderende requirements en fouten in de eerstvolgende iteratie meenemen. Bij watervallen wordt er niet snel teruggesprongen en komen fouten bovendien pas vaak laat aan het licht, wanneer het eigenlijk al te laat is ([**Kruch2**]). Volgens de gegevens gepresenteerd door [**MvE**] zijn de kosten van wijziging in deze laatste fasen zestig tot honderd keer zo hoog dan wanneer ontdekt tijdens de definitiefase (index).

Laag: (iteratief)	Hoog: (lineair)
Spiral Model	Watervalmodel
Agile	V-model
RAD	
RUP	
XP	
Scrum	

Tabel 3.10: Costs of change & Time-to-market

Time-to-market

Time-to-market gaat over hoe snel een product op de markt gezet kan worden. Dit hoeft niet per sé een product te zijn dat al aan alle requirements voldoet. Soms is het belangrijker om een product op de markt te hebben dan een volledig product. Gaandeweg kan dan zo'n product uitgebreid en verbeterd worden.

Bovenstaande past exact bij de iteratieve aanpak, of preciezer gezegd de agile en prototyping aanpak. [**Sumrell**] heeft het over "*lengthy six month waterfall practices*" en "*we would uncover an issue that could take significant amount of time to fix and retest*", als redenen voor de overstap van watervallen naar een agile aanpak.

Tabel 3.10 is daarom ook van toepassing op de time-to-market.

Foutgevoeligheid

Onder de foutgevoeligheid wordt verstaan de mate waarin fouten binnen een project onstaan en deze gerepareerd worden. In [**Gold**] valt te lezen dat de oorzaak van fouten in een systeem voor 45% afkomstig zijn van "*incorrect or misunderstood requirements and specifications*", met andere woorden: in de beginstadia van het project. Dit hoeft niet per definitie een slecht iets te zijn, zolang deze fouten maar gaandeweg gerepareerd worden. En dit is juist niet het geval bij lineaire procesdoorlopen. [**Kruch2**] heeft hierover te vertellen:

"*In a waterfall approach, too much rework comes at the very end, as an annoying and often unplanned consequence of finding nasty bugs during final testing and integration. Even worse, you discover that most of the cause of the "breakage" comes from errors in the design, which you attempt to palliate in implementation by building workarounds that lead to more breakage.*"

Naast de lineaire procesdoorlopen blijkt ook XP te maken te hebben met een hogere foutgevoeligheid. De nadelen van XP (zie 3.1.1 onder kopje "XP") opsommend beloven zaken als weinig documentatie, weinig gestructureerde testing en reviewing en een beperkte kwaliteitsplanning niet veel goeds voor de foutgevoeligheid van de methodiek.

Dat iteraties ontwikkelaars (onbewust) zouden aanzetten tot het maken van meer fouten, was niets over terug te vinden in de literatuur. Wel kwam het verschijnsel *gold plating* (zie pag. 33) in deze context aan het licht. Dit issue zal nog even ter sprake komen in de conclusies van 3.2.

Laag:	Hoog:
Spiral Model	Watervalmodel
Agile	V-model
RAD	XP
RUP	
Scrum	

Tabel 3.11: Foutgevoeligheid

Agility

Het lijkt een gegeven te zijn binnen het vakgebied en de bijbehorende literatuur: voor hedentendaagse systeemontwikkeling zijn dynamische procesdoorlopen nodig. Met andere woorden: modellen en methodieken die iteraties ondersteunen om de vrijwel oneindige stroom van wijzigende requirements het hoofd te kunnen bieden. Of dit gegeven inderdaad een gegeven genoemd mag worden, laten we voor nu even in het midden en bewaren we tot de conclusies van 3.2. We nemen gemakshalve aan dat 'agility' ofwel de mate van dynamiek van een model als een succesfactor aangeduid mag worden. De volgende indeling zien we dan (niet verrassend gelijk aan de indeling van tabel 3.8):

Laag: (lineair)	Hoog: (iteratief)
Watervalmodel	Spiral Model
V-model	Agile
	RAD
	RUP
	XP
	Scrum

Tabel 3.12: Agility

Tot slot

De laatste succesfactor, end-user satisfaction, vormt eigenlijk de samenvatting van alle bovenstaande succesfactoren. Daarom dat een uitwerking hiervan bewaard zal worden voor de hiernavolgende conclusies (3.2). Het Spiral Model hadden we eerder (3.1.1) omschreven als een overgangsmodel tussen de lineaire doorlopen van het

watervalmodel en het V-model, en de iteratieve doorlopen. Op basis hiervan zou in sommige van de bovenstaande tabellen de keuze gemaakt kunnen worden om het Spiral Model in de andere kolom op te nemen. Echter, omdat het Spiral Model voornamelijk iteratief van aard is, is om de grens duidelijk aan te geven deze keuze uitdrukkelijk niet gemaakt.

3.2 Conclusies

- *"Waterfalls, beautiful in nature, lousy in software development"*

Na een tocht langs verscheidene modellen en methodieken voor de temporele doorloop van systeemontwikkelingsprojecten zijn we aanbeland bij het slot. Deze afsluitende paragraaf staat in het teken van de beantwoording van de hoofdvraag: zijn iteratieve procesdoorlopen succesvoller dan lineaire? We hebben in 3.1.1 gekeken naar een aantal gangbare modellen in de omloop, met hun voor- en nadelen. 3.1.2 voorzag deze modellen van een categorisatie in lineair tegenover iteratief. In 3.1.3 vervolgens werd "succesvoller" beschreven in termen van succesfactoren, welke ten slotte uitgewerkt werden in 3.1.4. Eén van de succesfactoren was nog onbesproken gebleven en het geheel moet nog in het licht gezet worden van de categorisering 'iteratief versus lineair'.

Een antwoord op de hoofdvraag is ook meteen een uitwerking van deze laatste succesfactor, end-user satisfaction. Immers, we hadden end-user satisfaction bovenaan het lijstje (3.1.4) gezet, en natuurlijk voor een reden. [**Larman**] hebben het over een slagingspercentage van 2% voor op waterval-gebaseerde projecten. Maar het zit 'm vooral in het twistpunt wanneer een project "slaagt" of niet. Hoe je het ook wendt of keert, de cruciale factor is meestal niet het kostenplaatje, of de time-to-market, maar juist hoe tevreden de eindgebruiker over het systeem is. Alles hangt natuurlijk wel met elkaar samen, maar is de gebruiker niet in staat om met het systeem te werken, dan zal het snel in de prullenbak verdwijnen. Het succesvol zijn van de modellen heeft daarom alles te maken met de end-user satisfaction.

Iteratief versus Lineair

Op de website van ACM Queue is een artikel terug te vinden ([**ACMQ**]) waarin onder andere de resultaten van een zekere enquête besproken worden. Deze enquête (2003) gaat in op de toepassing van ontwikkelingsmodellen in de praktijk en is gehouden onder een omvangrijke groep van system development professionals. De resultaten zijn uitgewerkt in [**Neill**]. Een aantal bevindingen hieruit zijn interessant voor onze context. Zo blijkt maar liefst een derde (35%) het watervalmodel te gebruiken bij de ontwikkeling van een systeem. Agile heeft een te verwaarlozen aandeel. Daarentegen worden incrementele methoden aanzienlijk meer ingezet bij projecten met een doorlooptijd van meer dan twee jaar. De voorkeur gaat dan niet meer uit naar watervallen, maar het aandeel van het watervalmodel is zelfs dan nog aanzienlijk.

Vanwaar deze sterke drang naar watervallen? [**Larman**] beschrijven dat velen toch voor het dit model kiezen om twee redenen: "*the waterfall model is simple to explain and recall*" en "*it gives the illusion of orderly, accountable and measurable processes and milestones*". Zij laten echter tevens zien dat iteratieve en incrementele ontwikkeling door de jaren heen succesvol(ler) is geweest.

[**Davis**] schetsen de problematiek van systeemontwikkeling krachtig: "*the system being constructed is always aiming at a moving target*", wat volgens hen leidt tot twee dingen: "*this is the primary reason for delayed schedules*" en "*failing to meet customer expectations*". Dit zou je kunnen lezen als een bevestiging van het nut van iteratief en incrementeel ontwikkelen.

Davis en consorten dragen in dit artikel bovendien vijf metrieken aan (*shortfall*, *lateness*, *adaptability*, *longevity* en *inappropriateness*) waarmee het watervalmodel vergeleken wordt met een aantal andere modellen, zoals prototyping en incrementele ontwikkeling. Wat blijkt is dat het watervalmodel op al deze punten het slechtst scoort.

Dus, watervallen of iteratief en incrementeel ontwikkelen?

[**Kruchten2**] schrijft op een gegeven moment: "*The product that results from an iterative process will be of better overall quality than are products that result from a conventional sequential process. The system will have been tested several times, improving the quality of testing. The requirements will have been refined and will therefore*

be more closely related to the users' real needs. And at the time of delivery, the system will have been running longer."

Je zou zeggen dat daar geen speld tussen te krijgen is. En eigenlijk is dat ook zo. In geen enkel artikel of boek dat ik tegengekomen ben geeft iemand uitdrukkelijk de voorkeur aan watervallen. Het blijkt dat met de huidige aard van systeemontwikkeling het watervalmodel niet meer op z'n plaats is.

Dit hoeft echter niet te betekenen dat het watervalmodel overbodig is geworden. Op [**CIO**] valt bijvoorbeeld te lezen: "*don't use linear processes for non-algorithmic tasks, but also don't use iterative processes for a well defined task*". Het IBM artikel [**IBM2**] beschrijft zelfs een manier om RUP en het watervalmodel met elkaar te combineren. Bovendien zou de waterval benadering nog zinvol zijn bij projecten met 1) weinig risico's, 2) een korte duur (2 tot 3 maanden) en 3) wanneer het een evolutie van een bestaand systeem betreft. Een toepasselijke quote hier nog: "*the right question is: "what is the right development lifecycle for my project?", instead of labeling the waterfall model as "bad"* ". Iteratie zou dan in sommige gevallen "overkill" zijn.

De watervalaanpak mag dan wel nog zo z'n toepassingen hebben, maar een iteratieve en incrementele ontwikkelingsdoorloop is toch wel 'the way to go'. Wat niet wil zeggen dat er hier geen valkuilen zijn. Zo hoeft een iteratieve ontwikkeling niet minder werk en minder tijd te betekenen. Er komt namelijk veel meer planning en management bij kijken. [**Kruchten2**] stelt dat weliswaar een iteratieve en incrementele aanpak op de lange termijn succesvoller is, maar juist management is hierbij wel de randvoorwaarde. Ook zaken als *gold plating* en *requirements creep* doen zich hierbij voor. De eerste wil zeggen dat ontwikkelaars willen profiteren van de iteraties door steeds weer een betere techniek te introduceren, wat redundant werk met zich meebrengt. Requirements creep is het verschijnsel dat klanten requirements toevoegen omdat ze dat simpelweg kunnen.

Zolang een iteratieve en incrementele aanpak bij een systeemontwikkelingsproject op z'n plaats is en zolang de planning en het management in orde zijn, is er geen reden om te kiezen voor een watervalbenadering. Daarmee is dus niet gezegd dat er voor watervallen binnen de systeemontwikkeling geen plek meer is, het heeft nog steeds z'n waarde, maar weliswaar in beperkte mate. Dat nog steeds eenderde (zie eerder) deze benadering toepast, zegt eerder iets

over de starheid binnen deze groep van professionals. Objectieve data en bronnen tonen telkens weer aan dat iteratief en incrementeel ontwikkelen de toekomst heeft. De succesfactor end-user satisfaction (zie 3.1.3) valt hier over het algemeen een stuk positiever uit. Dus om antwoord te geven op de vraag welke beter is - lineair of iteratief ontwikkelen - kunnen we toch wel stellen dat tegenwoordig de laatste de voorkeur geniet. Helaas, als we nog eens terugkijken naar figuur 1.1, dan zijn we er nog lang niet. Misschien wanneer engineers steeds meer iteratief gaan ontwikkelen i.p.v. lineair, kunnen we wat doen aan deze schrikbarende cijfers.

HOOFDSTUK 4

Referenties

[**Kulak**] *Use Cases: Requirements in Context.* D. Kulak & E. Guiney. Addison-Wesley, 2004.

[**Schach**] *Software Engineering with Java.* S.R. Schach. Irwin, 1997.

[**Jacob**] *The Unified Software Development Process.* I. Jacobson, G. Booch, J. Rumbaugh. Addison-Wesley, 1999.

[**Press**] *Software Engineering: A Practitioner's Approach.* R. Pressman, D. Ince. McGraw-Hill Publishing, 2004.

[**Kruch**] *The Rational Unified Process: An Introduction.* P. Kruchten. Addison-Wesley, 2003.

[**Kruch2**] *From Waterfall to Iterative Development - A Challenging Transition for Project Managers.* P. Kruchten. The Rational Edge, Rational Software, 2001.

[**Sumrell**] *From Waterfall to Agile - How does a QA Team Transition?* M. Sumrell. Misys Healthcare Systems, IEEE, 2007.

[**Gold**] *Succeeding with Objects* A. Goldberg, K.S. Rubin. Addison-Wesley, 1995.

[**Larman**] *Iterative and Incremental Development: A Brief History.* C. Larman, V.R. Basili. IEEE, 2003.

[**Neill**] *Requirements Engineering: The State of the Practice.* C.J. Neill, P.A. Laplante. Penn State University, IEEE, 2003.

[**Davis**] *A Strategy for Comparing Alternative Software Development Life Cycle Models.* A.M. Davis, E.H. Bersoff, E.R. Comer. IEEE, 1988.

WEBSITES:

[**Wiki**] http://en.wikipedia.org/wiki/Software_development_process

[**IBM**] http://www-306.ibm.com/software/rational/

[**MvE**] http://www.cs.ru.nl/ marko/onderwijs/se/

[**ACMQ**] http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=110

[**CIO**] http://www.cio.com.au/index.php/id;1489602947

[**IBM2**] http://www.ibm.com/developerworks/rational/library/4626.html