Agile Scrum Handboek

3de druk

Nader K. Rad

Van Haren PUBLISHING

Agile Scrum Handboek

Andere uitgaven bij Van Haren Publishing

Van Haren Publishing (VHP) is gespecialiseerd in uitgaven over Best Practices, methodes en standaarden op het gebied van de volgende domeinen:

- IT en IT-management;
- Enterprise-architectuur;
- Projectmanagement;
- Businessmanagement.

Deze uitgaven zijn beschikbaar in meerdere talen en maken deel uit van toonaangevende series, zoals *Best Practice, The Open Group series, Project management* en *PM series.*

Van Haren Publishing is tevens de uitgever voor toonaangevende instellingen en bedrijven, onder andere: Agile Consortium, ASL BiSL Foundation, CA, Centre Henri Tudor, CM Partners, Gaming Works, IACCM, IAOP, IPMA-NL, ITSqc, NAF, KNVI, PMI-NL, PON, The Open Group, The SOX Institute.

Onderwerpen per domein zijn:

IT en IT-management
ABC of ICT
ASL®
CMMI®
COBIT®
e-CF
ISM
ISO/IEC 20000
ISO/IEC 27001/27002
ISPL
IT4IT®
IT-CMF™
IT Service CMM
ITIL®
MOF
MSF
SABSA
SAF
SIAM™
TRIM
VeriSM

Enterprise-architectuur
ArchiMate®
BIAN
GEA®
Novius Architectuur Methode
TOGAF®

Projectmanagement
A4-Projectmanagement
DSDM/Atern
ICB / NCB
ISO 21500
MINCE®
M_o_R®
MSP®
P3O®
PMBOK® Guide
Praxis®
PRINCE2®

Businessmanagement
BABOK® Guide
BiSL® en BiSL® Next
BRMBOK™
BTF
CATS CM®
DID®
EFQM
eSCM
FSM
IACCM
ISA-95
ISO 9000/9001
OBM
OPBOK
SixSigma
SOX
SqEME®

Voor een compleet overzicht van alle uitgaven, ga naar onze website: www.vanharen.net

Agile Scrum Handboek

Derde druk

Nader K. Rad

Van Haren PUBLISHING

Colofon

Titel:	Agile Scrum handboek – derde druk
Auteur:	Nader K. Rad
Vertaling:	Theo Wanders, Maarssen
Uitgever:	Van Haren Publishing, ‘s-Hertogenbosch-NL www.vanharen.net
ISBN Hard copy:	978 94 018 0793 7
ISBN ebook:	978 94 018 0794 4
ISBN ePUB:	978 94 018 0795 1
Druk:	Derde druk, eerste oplage, augustus 2021 Engelse versie: Third edition, first impression, April 2021

Voor verdere informatie over Van Haren Publishing, e-mail naar: info@vanharen.net.

Inhoudsopgave

1. HET AGILITY CONCEPT 1

1.1 Projectleveringsmethode en levenscyclus 2
1.1.1 De voorspellende benadering 2
1.1.2 De adaptieve aanpak 4
1.1.2.1 Iteraties met een vast bereik versus een vaste duur 5
1.1.2.2 De duur van een iteratie 5
1.1.2.3 Dezelfde tijdsduur of verschillende doorlooptijden 5
1.1.2.4 Wat gebeurt er binnen iteraties? 6
1.1.2.5 Increment versus leverbaar resultaat 7
1.1.2.6 Iteraties versus cycli 7
1.1.2.7 Testen en kwaliteit in Agile 8
1.2 Een ontwikkelingsaanpak selecteren 8
1.3 Is Agile alleen geschikt voor IT-ontwikkeling? 9
1.3.1 Projecten 9
1.3.2 Programma's 10
1.3.3 Operations 10
1.4 Is Agile sneller? 10
1.5 Is Agile nieuw? 11

2. SCRUM 13

2.2 Scrum als verpakking 14
2.3 De Scrum-structuur 14
2.3.1 Mensen 16
2.3.1.1 Kenmerken 16
2.3.1.2 Rollen 19
2.3.1.3 Extra rollen 25

2.3.2 Gebeurtenissen ... 26
2.3.2.1 Sprint ... 26
2.3.2.2 Sprint Planning ... 29
2.3.2.3 Daily Scrum ... 32
2.3.2.4 Sprint Review ... 34
2.3.2.5 Sprint Retrospective ... 36
2.3.3 Artefacten ... 37
2.3.3.1 Product Backlog ... 38
2.3.3.2 Sprint Backlog ... 44
2.3.3.3 Increment ... 47
2.4 Scaled Scrum ... 49
2.4.1 Rollen ... 49
2.4.1.1 Developers ... 50
2.4.1.2 Scrum Master ... 51
2.4.1.3 Product Owner ... 51
2.4.1.4 Aanvullende rollen ... 51
2.4.2 Gebeurtenissen ... 52
2.4.2.1 Sprint ... 52
2.4.2.2 Sprint Planning ... 52
2.4.2.3 Daily Scrum ... 53
2.4.2.4 Sprint Review ... 54
2.4.2.5 Sprint Retrospective ... 54
2.4.3 Artefacten ... 54
2.4.3.1 Product Backlog ... 54
2.4.3.2 Sprint Backlog ... 54
2.4.3.3 Incrementen ... 55

3. CRYSTAL ... 57

3.1 De Cockburn-schaal ... 57
3.2 Frequent releasen ... 58
3.3 Osmotische communicatie ... 58
3.4 Walking skeleton ... 59
3.5 Information Radiators ... 59
3.5.1 Escaped defects ... 61
3.5.2 Voortgangsinformatie ... 61
3.5.2.1 Burn-up chart ... 62
3.5.2.2 Burn-down chart ... 63
3.5.2.3 Burn-down bar ... 65
3.5.2.4 Cumulatieve flow diagrammen ... 66
3.5.3 Niko-Niko kalender ... 67

4. EXTREME PROGRAMMING 69

4.1 Dagelijkse routine 69
4.1.1 Pair programming 69
4.1.2 Toewijzing 70
4.1.3 Ontwerpen 71
4.1.4 Schrijf de test 71
4.1.5 Coderen 72
4.1.6 Refactoring 72
4.1.7 Integreren 73
4.1.8 Naar huis 74
4.1.9 Stand-up meetings 74
4.1.10 Tracking 74
4.1.11 Risicomanagement 74
4.2 Spiking 75
4.3 De aard van items 75
4.3.1 De twee regels 76
4.3.2 INVEST 77
4.3.3 User stories 78
4.4 Schatten 79
4.4.1 Ideale tijd 79
4.4.1.1 Relatie met tijd 80
4.4.1.2 Zelfcorrectie 80
4.4.2 Story points 81
4.4.2.1 Schatten in story points 82
4.4.2.2 Relatie met tijd 82
4.4.2.3 Zelfcorrectie 83
4.4.3 T-shirt maten 83
4.4.4 Velocity 84
4.4.4.1 Plannen van iteraties 84
4.4.4.2 Schatting van de opleverdatum 85
4.4.4.3 Prestatiemeting 85
4.4.4.4 Velocity vergelijken 85
4.4.4.5 Onvoltooid werk versus velocity 85
4.4.4.6 Afwijkingen in velocity 87
4.4.4.7 Startsnelheid 87
4.4.5 Planning poker 88
4.4.5.1 Partijdige schatting 88
4.4.5.2 Onpartijdige schatting 89
4.4.5.3 Opnieuw stemmen 89
4.4.5.4 Maximaal of gemiddeld 90
4.4.5.5 Planning poker kaarten 90

4.4.6 Triangulatie ... 91
4.4.7 Affiniteitsschatting ... 92
4.4.8 Herschatten ... 93
4.5 Feedbackloops ... 93
4.6 Planning onion ... 94

5. DSDM® ... 97

5.1 Projectbeperkingen ... 97
5.2 Vooraf plannen ... 99
5.3 MoSCoW-prioritering ... 100
5.4 Uitzonderingen ... 101
5.5 Zelforganisatie ... 101
5.6 Contractsoorten ... 102

6. KANBAN ... 103

6.1 Visualiseren ... 103
6.2 Beperken van Work In Progress (WIP) ... 104
6.3 Pull versus Push ... 105

7. FILOSOFEREN! ... 111

7.1 eXtreme Programming ideeën ... 111
7.1.1 Rechten van de klant ... 111
7.1.1.1 Algemeen plan ... 112
7.1.1.2 De hoogste waarde ... 112
7.1.1.3 Voortgang ... 112
7.1.1.4 Verandering ... 113
7.1.1.5 Schema ... 113
7.1.2 Rechten van de programmeur ... 113
7.1.2.1 Prioriteiten ... 114
7.1.2.2 Kwaliteit ... 114
7.1.2.3 Hulp ... 114
7.1.2.4 Schattingen ... 115
7.1.2.5 Empowerment ... 115
7.1.3 Waarden ... 115
7.1.3.1 Communicatie ... 115
7.1.3.2 Eenvoud ... 115
7.1.3.3 Feedback ... 116
7.1.3.4 Moed ... 116
7.1.3.5 Respect ... 116

7.2 DSDM®-ideeën ... 116
7.2.1 Filosofie ... 116
7.2.2 Principes ... 117
7.2.2.1 Focus op de zakelijke behoefte ... 117
7.2.2.2 Op tijd leveren ... 118
7.2.2.3 Samenwerken ... 118
7.2.2.4 Doe nooit concessies aan kwaliteit ... 118
7.2.2.5 Stapsgewijs bouwen vanuit een stevige basis ... 118
7.2.2.6 Iteratief ontwikkelen ... 119
7.2.2.7 Communiceer continu en duidelijk ... 119
7.2.2.8 Demonstreer beheersbaarheid ... 119
7.3 Scrum-ideeën ... 119
7.3.1 Pijlers ... 120
7.3.2 De vijf Scrumwaarden ... 120
7.3.2.1 Commitment ... 120
7.3.2.2 Moed ... 120
7.3.2.3 Focus ... 120
7.3.2.4 Openheid ... 121
7.3.2.5 Respect ... 121
7.4 Het Agile Manifesto ... 121
7.4.1 Waardestatement #1 ... 122
7.4.2 Waardestatement #2 ... 123
7.4.3 Waardestatement #3 ... 123
7.4.4 Waardestatement #4 ... 123
7.4.5 De principes ... 124
7.4.5.1 Principe #1 ... 124
7.4.5.2 Principe #2 ... 124
7.4.5.3 Principe #3 ... 124
7.4.5.4 Principe #4 ... 125
7.4.5.5 Principe #5 ... 125
7.4.5.6 Principe #6 ... 125
7.4.5.7 Principe #7 ... 125
7.4.5.8 Principe #8 ... 126
7.4.5.9 Principe #9 ... 126
7.4.5.10 Principe #10 ... 126
7.4.5.11 Principe #11 ... 127
7.4.5.12 Principe #12 ... 127

OVER DE AUTEUR ... 129

INDEX ... 131

1. Het Agility concept

Er zijn veel mythes en misleidende concepten over Agile, te beginnen met het antwoord op de meest fundamentele vraag in deze context: Wat is Agile? Agile staat voor “wendbaarheid”.

Wat vaak gezegd wordt is “Agile is een mindset”. Het punt is dat Agile een mindset nodig heeft, net zoals al het andere. Maar het is onjuist om te zeggen dat het een mindset is. Zeggen dat “Agile een mindset is” heeft slechts één praktische consequentie: het laat bepaalde mensen doen wat ze willen en het gewoon Agile noemen omdat het tegenwoordig in de mode is.

Een ander veel voorkomend probleem in onze maatschappij is de illusie van de externe vijand. Degenen onder jullie die bekend zijn met de manier waarop autoritaire systemen werken, weten dat ze altijd een vijand nodig hebben. Het helpt de hiaten die ze in hun systeem hebben te dichten door afleiding te creëren, en dat creëert een gemeenschappelijk doel om het gebrek aan echte, haalbare interne doelen te dekken. Het is triest om te zien dat veel Agile-beoefenaars dezelfde aanpak hebben, meestal voor het persoonlijk gewin van een paar leiders.

Het is voor uw professionele leven het beste om open te staan voor verschillende ideeën en van al deze ideeën te leren zonder dat u een sektelid wordt. Deze benadering is het eerste principe in de Nearly Universal Principles of Projects: https://nupp.guide.

Laten we beginnen met onze uitleg over de echte aard van Agile.

1.1 Projectleveringsmethode en levenscyclus

Wanneer u een stukje software ontwikkelt, worden de volgende stappen op een of andere manier uitgevoerd voor afzonderlijke functies, of voor de oplossing als geheel:

- Analyseren;
- Ontwerpen;
- Construeren/bouwen;
- Integreren;
- Testen.

U kunt natuurlijk andere namen voor die stappen gebruiken, ze samenvoegen, in minder stappen opsplitsen, of in meer; dat is prima. Deze stappen noemen we **leveringsprocessen** .

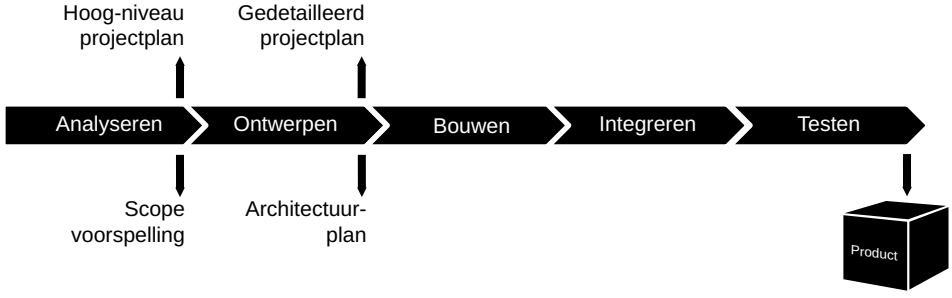
Nu is de vraag: hoe gaat u deze processen organiseren en uitvoeren? Denk aan een paar opties voordat u de rest van dit hoofdstuk leest.

En? Hoeveel opties hebt u bedacht?

1.1.1 De voorspellende benadering

U hebt misschien veel opties in gedachten, maar ze behoren allemaal tot een van de twee generieke vormen. Overigens noemen we deze opties de **ontwikkelingslevenscyclus** (development lifecycle) of **ontwikkelingsaanpak** (development approach).

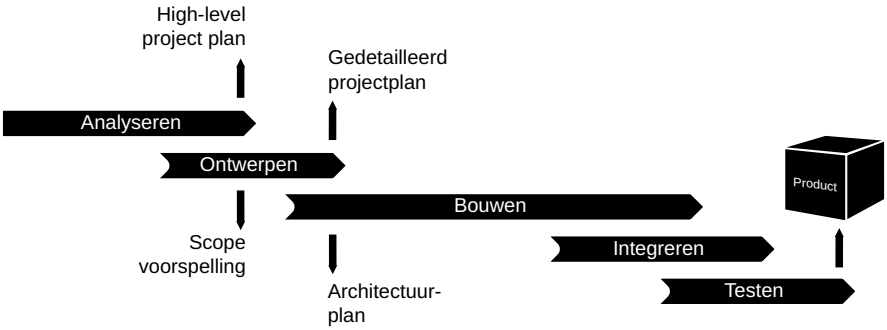
De volgende figuur toont ons het algemene levenscyclusmodel.



In dit levenscyclusmodel is elke processtap voltooid voordat we naar de volgende gaan:

1. Dat wil zeggen dat wij eerst alle eisen compleet analyseren en dan pas beslissen wat we als oplossing willen hebben.
2. Daarna ontwerpen we de architectuur van de oplossing en zoeken we uit wat de beste manier is om alle functies te realiseren en vorm te geven.
3. Daarna gaan programmeurs de verschillende onderdelen bouwen.
4. Vervolgens worden de verschillende onderdelen geïntegreerd in één oplossing.
5. Als laatste wordt de oplossing volledig getest en fouten worden hersteld.

Het is duidelijk dat de stappen elkaar kunnen overlappen; u hoeft bijvoorbeeld niet te wachten tot alle oplossingsonderdelen compleet zijn voordat ze worden geïntegreerd en getest. De levenscyclus kan er dan als volgt uitzien:



Dit is geen fundamenteel verschil met het voorgaande model, we hebben nog steeds een volgorde van ontwikkelprocessen als de belangrijkste factor voor de levenscyclus.

Zoals u ziet, is dit type levenscyclus gebaseerd op een initieel onderzoek om een idee te vormen wat we moeten gaan produceren. We hebben een **upfront-specificatie**, een **upfront-ontwerp** en bijgevolg een passend plan. Daarom noemen sommigen het een **planning-gestuurde ontwikkeling**. We proberen te voorspellen wat we willen en hoe het geproduceerd kan worden, en daarom bevat het de term “voorspellend”. Voorspellende levenscycli zijn een normale en geschikte manier om veel projecten te ontwikkelen, zoals een bouwproject. U plant en ontwerpt eerst en u volgt vervolgens het plan. Dit is echter voor sommige soorten projecten geen comfortabele manier om te werken.

Denk aan een typisch IT-ontwikkelingsproject. U kunt veel tijd besteden aan het opgeven en het analyseren van de eisen, en vervolgens al het andere hierop baseren. Wat gebeurt er nu? De klant zal niet altijd blij zijn als hij het resultaat ziet! Ze zullen vragen om wijzigingen. Het verwerken van wijzigingen is duur in deze levenscyclus omdat u misschien al het voorgaande werk moet herzien.

Zoals gebruikelijk in deze branche, weten de klanten pas wat ze willen als ze het product zien. Wanneer zien ze het product? Tegen het einde van het project. Op dat moment zijn de kosten om het product te veranderen maximaal.

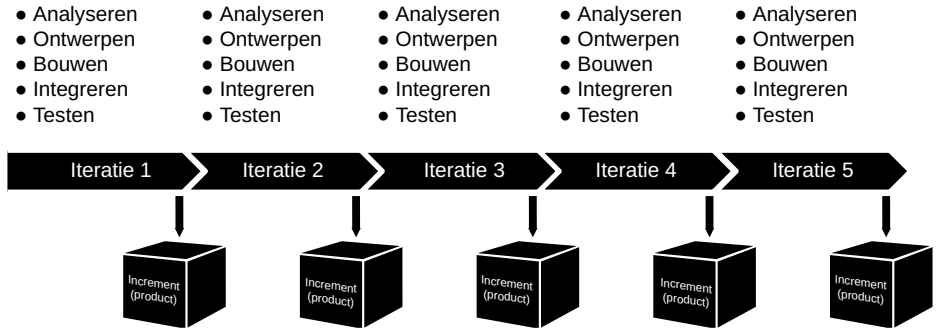
De Agile-gemeenschap verwijst meestal naar voorspellende systemen als **watervalsystemen**. Het is echter geen goed idee om deze term te gebruiken, omdat deze een negatieve klank heeft gekregen, en het gebruik ervan zou een verstorende werking hebben op een tot dan toe rationeel verlopend gesprek over de ontwikkelingsbenaderingen.

1.1.2 De adaptieve aanpak

Om de problemen te overwinnen die IT-ontwikkelingsprojecten hebben met voorspellende levenscycli, kunnen we het comfort en de structuur van een voorspellend systeem opofferen en een andere levenscyclus gebruiken die het product stapsgewijs creëert, om het gaandeweg in overleg met de klanten en eindgebruikers te controleren. Dit is een luxe die we hebben in IT-ontwikkelingsprojecten die niet iedereen heeft. Denk aan een bouwproject: daar zijn geen zinvolle deelopleveringen denkbaar en is het product pas op het einde bruikbaar.

Om eerlijk te zijn, dit nadeel van een bouwproject is in evenwicht met het feit dat als u een project start om een ziekenhuis te bouwen, het niet veel uitmaakt hoeveel veranderingen er zijn. Het eindresultaat zal een ziekenhuis zijn, en niet bijvoorbeeld een pretpark! Echter, in IT-ontwikkelingsprojecten zou u een project kunnen starten om zoiets als een ziekenhuis te maken en eindigen met zoiets als een pretpark.

We kunnen dus een incrementele levering krijgen in IT-ontwikkelingsprojecten; laten we deze kans aangrijpen met een levenscyclus als deze:



Er is geen echte voorspelling voor het eindresultaat in deze levenscyclus. In plaats van het eindproduct te voorspellen en daarop te vertrouwen hebben we korte perioden (**iteraties**) waarin we een **increment** (deel) van het product maken. Wij laten de incrementen (de laatste versie van het product) aan de klant en eindgebruikers zien, ontvangen hun feedback en beslissen wat te doen in de komende iteratie. Dus in plaats van het doen van voorspellingen, gaan we door met het project en gebruiken telkens de feedback. Deze aanpak wordt de **adaptieve levenscyclus** genoemd. Agile is de populaire naam voor adaptieve systemen.

Om elk increment te maken, doorlopen we alle ontwikkelingsprocessen in die tijdsperiode. In de volgende periode zullen we deze processen herhalen: we itereren. Dat is de reden waarom deze methode van ontwikkeling **iteratieve ontwikkeling** genoemd wordt. Bij iteratieve ontwikkeling wordt elk proces (zoals het ontwerp) meerdere keren herhaald voor verschillende elementen van het product, in plaats van één keer voor het hele product te worden uitgevoerd.

Normaal gesproken vinden iteratieve ontwikkeling en incrementele levering samen plaats.

1.1.2.1 Iteraties met een vast bereik versus een vaste duur

Is het naar uw mening beter om iteraties met een vast bereik (scope) of met een vaste tijdsduur te hebben? Theoretisch kunnen beide werken, maar in de praktijk zijn iteraties met een vaste duur superieur, omdat het vast houden van de reikwijdte van de iteratie de volgende resultaten kan hebben:

- Het kan zijn dat u te veel tijd aan elke functie besteedt en te veel toeters en bellen toevoegt. Het hebben van een vaste duur dwingt u om uzelf eerst te concentreren op de meest waardevolle dingen.
- De tijd die u nodig hebt om de scope te voltooien, is meestal langer dan u verwacht, waardoor de iteraties langer worden en het aantal feedbackloops afneemt. Als er minder feedback is, is er minder aanpassing.

Daarom hebben bijna alle Agile-methoden iteraties met een vaste duur, en ze staan er meestal op om deze timeboxen te respecteren. Een **timebox** is een venster met een maximale (of vaste) hoeveelheid tijd, die onder geen enkele omstandigheid wordt verlengd (want als u deze eenmaal verlengt, doet u dat altijd).

1.1.2.2 De duur van een iteratie

Nu de iteraties moeten worden getimeboxed, hoe lang moet dat duren?

We kunnen op elk moment feedback ontvangen, maar de gestructureerde feedback die we aan het einde van elke iteratie ontvangen, is de sleutel. Kortere iteraties geven ons daarom meer gestructureerde feedback en daardoor meer aanpassingsmogelijkheden. Aan de andere kant moet elke iteratie voldoende tijd hebben om een aantal functies te produceren die een serieuze beoordeling met de klant waard zijn, wat betekent dat ze niet te kort mogen zijn.

In de begintijd van de Agile-systemen leek 4 tot 8 weken een goed idee. Tegenwoordig zijn kortere doorlooptijden meer acceptabel. De maximaal aanvaardbare tijdsduur is bij de meeste systemen 4 weken, en een duur van slechts 1 week lijkt praktisch voor de huidige technologieën.

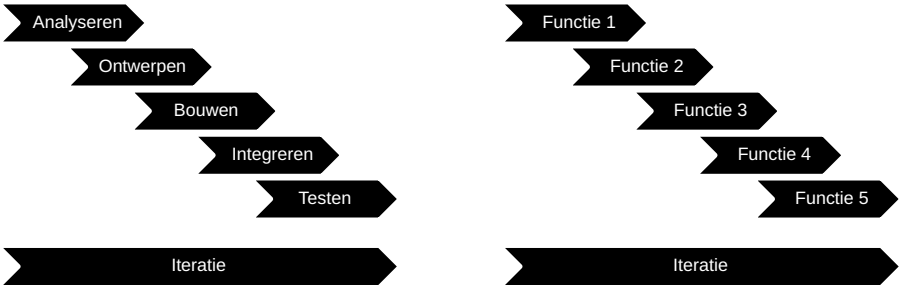
1.1.2.3 Dezelfde tijdsduur of verschillende doorlooptijden

Is het volgens u beter dat alle iteraties dezelfde tijdsduur hebben, of moet dit flexibel zijn? Het hebben van dezelfde tijdsduur is meer gedisciplineerd en wekt regelmaat op. In de meeste gevallen is het niet echt nodig om te beslissen over de duur van elke timebox afzonderlijk, daarom stellen de meeste systemen dezelfde tijdsduur in voor alle iteraties. U kunt de tijdsduur van een timebox aanpassen, maar u beslist niet over de duur van elke iteratie afzonderlijk.

1.1.2.4 Wat gebeurt er binnen iteraties?

Een iteratie is een periode waarin we de ontwikkelprocessen herhalen. Maar hoe doe je dat?

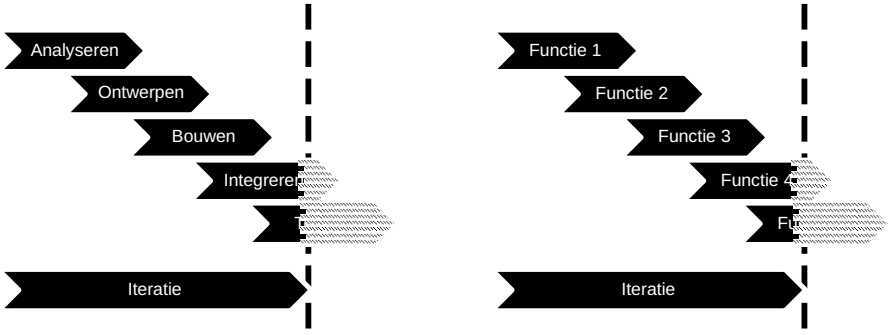
Hier zijn twee mogelijkheden:



In de figuur links worden voor alle gevraagde functies die tot de iteratie horen, tegelijkertijd alle ontwikkelingsprocessen doorlopen. We kunnen het opvatten als een “mini-voorspellend systeem”.

In de figuur rechts gaan alle stappen per functie, één of enkele tegelijk, en worden voor elk van hen alle ontwikkelingsprocessen doorlopen. Dus analyseren, ontwerpen, bouwen, etc. voor functie 1, daarna voor functie 2, etc. We kunnen het beschouwen als een mini-mini-voorspellend systeem (d.w.z. bijna niet voorspellend). We geven de voorkeur aan de tweede, op functies gebaseerde optie, vooral omdat deze compatibel is met timeboxed iteraties.

Een reden is dat we besloten hebben om iteraties met een timebox te gebruiken, dus er is altijd een kans dat we niet alles kunnen voltooien. Met de mini-voorspellende aanpak aan de linkerkant bent u niet in staat om een nieuwe functie volledig en compleet af te maken als de timebox voorbij is, en krijgt u geen feedback; terwijl met de rechter aanpak er altijd een paar functies gereed zijn om aan de klant te demonstreren en aan te passen bij de volgende iteratie. Onderstaande figuur demonstreert deze situatie.



Als er een maximale duur is, zijn we aan het einde van de iteratie misschien niet met alles klaar. Dat betekent dat er met de functie-gebaseerde benadering een aantal functies zijn die nog niet af zijn, terwijl we bij de benadering aan de linkerkant niet klaar zijn met een of meer van de ontwikkelingsprocessen van elk van de functies. Dit heeft tot gevolg dat we aan het einde van de iteratie geen bruikbare output hebben en geen demonstratie kunnen geven en dus geen feedback kunnen ontvangen.

1.1.2.5 Increment versus leverbaar resultaat

Elke increment is een deliverable, maar niet elke deliverable is een increment. We gebruiken de term “increment” om te verwijzen naar de incrementen van het product, wat in het geval van IT-ontwikkeling verschillende versies van werkende software zijn. Elke nieuwe increment is een bruikbare versie van hetzelfde product, maar met meer functies, en het moet bruikbaar zijn om betrouwbare feedback mogelijk te maken. Een deliverable kan daarentegen bijna alles zijn dat u in uw project produceert. In een voorspellend project zijn het ontwerp vooraf en het plan vooraf bijvoorbeeld resultaten die niet kunnen worden beschouwd als incrementen, omdat dit niet bruikbaar is voor de klant en hij er ook geen feedback op kan leveren. Omdat Agile in de mode is, noemen sommige mensen hun deliverables gewoon incrementen en claimen ze dat hun werkwijze op basis daarvan Agile is, wat dus niet correct is.

1.1.2.6 Iteraties versus cycli

Elke iteratie is een cyclus, maar niet elke cyclus is een iteratie.

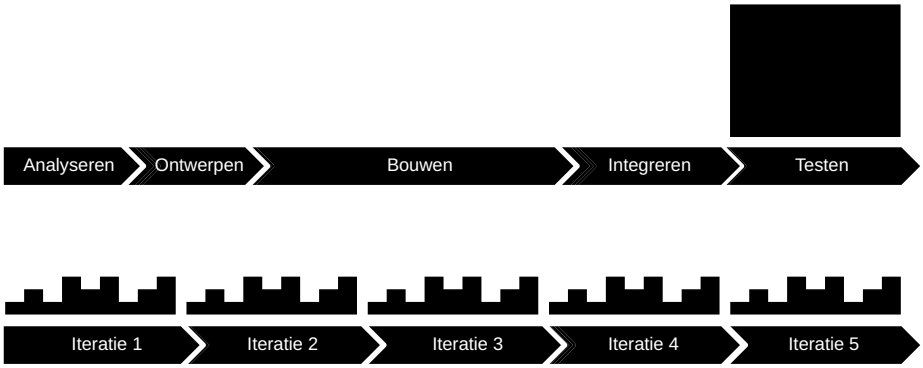
Een iteratie is een speciaal type cyclus waarin we zowel onze ontwikkelingsprocessen als onze managementprocessen herhalen. Veel systemen hebben cycli, maar die cycli herhalen alleen de managementprocessen en niet noodzakelijk de ontwikkelingsprocessen. De grote maandelijkse cyclus en de kleine wekelijkse cyclus in P3.express, de fasen in PRINCE2® en de fasen in de *PMBOK® Guide* zijn daar allemaal voorbeelden van.

Om dit verschil duidelijker te maken, stelt u zich een cyclus voor die zijn eigen planning, monitoring, beheer en afsluiting heeft. Het feit dat deze managementprocessen worden herhaald, is de reden dat hun containers cycli worden genoemd. Stel dat het een voorspellend project is, en de ene cyclus gaat over het specificeren van de vereisten, de volgende cyclus gaat over het ontwerpen van het product, enzovoort. Dit is een cyclisch systeem zonder iteraties.

Helaas denken sommige mensen dat zolang ze cycli in hun projecten kunnen identificeren, ze deze iteratief en dus Agile kunnen noemen, wat niet correct is. Dit is nog erger dan managementprocessen verwarren met ontwikkelingsprocessen. Sommige mensen noemen willekeurige tijdsperioden in hun projecten iteraties, en overwegen bijvoorbeeld wekelijkse “iteraties” als er geen echte iteratie van processen is.

1.1.2.7 Testen en kwaliteit in Agile

Het volgende diagram toont een vereenvoudigd schema van de manier waarop testen in elke benadering worden uitgevoerd:



De meeste testactiviteiten vinden plaats aan het einde van een voorspellend project, en dan zijn we waarschijnlijk te laat en staan we onder grote druk om het project zo snel mogelijk af te ronden. Deze druk kan ertoe leiden dat sommige tests worden geschrapt en dat de kwaliteit in gevaar komt.

Hoe zit het dan met adaptieve systemen?

Welnu, dit probleem bestaat niet in adaptieve levenscycli omdat er continu wordt getest, en het maakt dus niet uit wanneer we het project stoppen, omdat we altijd de juiste testverhouding hebben.

Er zijn ook andere verschillen. De aard van adaptieve systemen maakt het bijvoorbeeld bijna essentieel om geautomatiseerde tests te hebben. Geautomatiseerde tests dekken mogelijk niet elke afzonderlijke regel code, en er is een optimale testcodedekking die we nodig hebben in ons project. **Testcodedekking** (test code coverage) is de verhouding tussen de coderegels die door geautomatiseerde tests zijn getest en het totale aantal regels.

1.2 Een ontwikkelingsaanpak selecteren

Elk van de voorspellende en adaptieve levenscycli heeft voor- en nadelen. De juiste keuze is afhankelijk van vele factoren, maar de belangrijkste is de aard van het product. U kunt twee essentiële vragen stellen voordat u beslist welk type levenscyclus nodig is voor het project:

1. Moet het adaptief zijn? Want als u dat niet doet, is het een voorspellende levenscyclus! Die is eenvoudiger en meer gestructureerd. Een adaptief systeem is nodig

als er een risico is om te beginnen met het idee om zoiets als een ziekenhuis te bouwen en te eindigen met zoiets als een pretpark.
2. Kan ik adaptief zijn? Deze vraag is nog belangrijker dan de vorige vraag. Om adaptief te zijn, moet u de mogelijkheid hebben om iteratief te ontwikkelen en incrementeel te leveren om feedback te ontvangen en aan te passen. Laten we nogmaals een bouwproject beschouwen: kunt u het gebouw iteratief ontwerpen? Kunt u de fundering van een gebouw ontwerpen zonder de rest van het gebouw te ontwerpen dat bepalend is voorde belasting op de fundering? Het antwoord is simpelweg nee. Het is niet mogelijk om bouwprojecten iteratief te ontwikkelen. Bovendien is incrementele levering in dit soort situaties niet mogelijk omdat enerzijds de subsets van een gebouw niet bruikbaar zijn en anderzijds de feedback die door een subset wordt gegenereerd, mogelijk niet van toepassing is op de rest. We kunnen dus geen adaptieve levenscyclus gebruiken om een gebouw te bouwen (verwar dit niet met interieurontwerp en decoratie, of zelfs renovatie, waarvoor we mogelijk wel een adaptief systeem kunnen gebruiken).

De belangrijkste boodschap is dat een voorspellende of een adaptieve aanpak geen kwestie is van goed of slecht, maar eerder afhangt van diverse factoren. Het zijn beide geldige benaderingen, en elk van hen is geschikter voor sommige soorten producten. Als kleine oefening: denk aan een IT-project voor het upgraden van de besturingssystemen van 300 computers in een organisatie, of een IT-project om een netwerkinfrastructuur voor een zeer grote organisatie met kantoren op zes locaties. Welke ontwikkelingscyclus is naar uw mening het meest geschikt voor deze twee projecten?

1.3 Is Agile alleen geschikt voor IT-ontwikkeling?

De meeste voorbeelden in dit boek, evenals andere bronnen over Agile, gaan over IT-ontwikkelingsprojecten. Betekent dit dat Agile beperkt is tot IT-ontwikkelingsprojecten?

1.3.1 Projecten

Sommige mensen beweren dat Agile voor elk type project kan worden gebruikt, en dezelfde mensen beweren meestal dat dit de enige juiste manier is om projecten uit te voeren. Het zijn meestal mensen die geen ander serieus project hebben meegemaakt dan niet-kritieke IT-ontwikkelingsprojecten. In werkelijkheid zijn er veel soorten projecten waarbij een adaptieve methode niet nodig of niet mogelijk is, omdat we ze niet iteratief kunnen ontwikkelen en stapsgewijs kunnen opleveren.

Afgezien van het simpele feit dat Agile niet de enige absolute waarheid is en niet in elk project kan worden gebruikt, kunnen we nog steeds kijken naar de reeks projecten die kunnen profiteren van een adaptief systeem. Is het beperkt tot IT-ontwikkeling, of zijn er andere geschikte typen? Het is misschien mogelijk om Agile in andere soorten