THESIS

CONTINUOUS DELIVERY… HYPE OF MUST?

EEN KWALITATIEVE ANALYSE NAAR HET SUCCES VAN CONTINUOUS DELIVERY

Rotterdam School of Management

Erasmus University

CONTINUOUS DELIVERY… HYPE OF MUST?

EEN KWALITATIEVE ANALYSE NAAR HET SUCCES VAN CONTINUOUS DELIVERY

Student:	Ing. A.R. Baronner rsm@baronner.nl
Studentnummer:	140171
Opleiding:	Parttime master Bedrijfskunde Major New Business: Innovation & Entrepreneurship
Onderwijsinstelling:	RSM Erasmus University
Plaats en datum:	Gorinchem, 16 juli 2017
Begeleider:	prof.dr.ir. J.C.M. van den Ende
Meelezer:	Drs. A. Simons

'If you want to be incrementally better, be competitive.
If you want to be exponentially better, be cooperative.'

(William Shakespeare, English writer, poem and actor, 1564-1616)

SAMENVATTING

Software staat aan de basis van het merendeel van de radicale of disruptieve innovaties van het afgelopen decennium. Onder invloed van software worden gehele waardeketens opnieuw gedefinieerd door bedrijven als Uber, Airbnb en Facebook. De afhankelijkheid van software is niet alleen beperkt tot deze *disrupters*; software is in toenemende mate missiekritiek voor organisaties. Daarmee wordt ook het belang om goede software te ontwikkelen vergroot. Ook softwareontwikkeling is aan innovatie onderhevig. De introductie van op Agile gebaseerde kort cyclische ontwikkelmethoden kenden een hoop voordelen en deze innovatie werd dan ook binnen de grotere software-gedreven organisaties snel omarmd. Veel IT-beheerafdelingen hebben echter moeite om de dramatisch hogere frequentie van opleveringen te kunnen installeren, integreren en testen, waardoor het merendeel van de Agile software-opleveringen 'op de plank blijven liggen' en niet of aanzienlijk minder-frequent wordt ontsloten voor de eindgebruikers. Hierdoor worden de voordelen van Agile softwareontwikkeling slechts ten dele gebruikt. Continuous Delivery is een innovatieve en op Agile-gebaseerde software-ontwikkelmethode die er op gericht is om goed werkende software kort cyclisch wel in een productie-omgeving te ontsluiten voor eindgebruikers.

Het doel van dit onderzoek is om te achterhalen of Continuous Delivery in vergelijking met reguliere Agile software-ontwikkeltrajecten een toegevoegde waarde is. Hiervoor is de volgende onderzoeksvraag opgesteld: *Wat is het effect van Continuous Delivery op het succes van software-ontwikkeltrajecten?* Om een antwoord te geven op de onderzoeksvraag is een vergelijkende meervoudige case-study uitgevoerd bij vier grote software-gedreven organisaties. Per organisatie zijn twee cases geanalyseerd: een Agile software-ontwikkeltraject met Continuous Delivery en een zonder (*Agile-only*). Uit het onderzoek blijkt dat Continuous Delivery – afhankelijk van een drietal aspecten – een positieve invloed heeft op *alle* succesindicatoren van een Agile software-ontwikkeltraject: scope, kwaliteit, tijd, kosten en wendbaarheid. Er is dus geen sprake van een *trade-off.*

De gevonden aspecten zijn 'mate van hergebruik van Continuous Delivery', 'IT-wendbaarheid' en '*time-to-market*'. De voor Continuous Delivery benodigde procesautomatisering is kostbaar in termen van geld en tijd en zal alleen rendabel zijn wanneer de voordelen hiervan frequent worden genuttigd. Hergebruik is mogelijk door de inzet van Continuous Delivery niet te beperken tot de projectfase van een traject, maar ook te gebruiken tijdens de verdere levenscyclus (beheerfase) of zelfs voor meerdere software-ontwikkeltrajecten. Een ander gevonden aspect is de IT-wendbaarheid: indien de organisatie of de architectuur niet geschikt is voor Continuous Delivery, dan zal het aanpassen daarvan behoorlijk impactvol zijn en dus zal de investering voor een Continuous Delivery implementatie minder snel rendabel zijn. Het laatste gevonden aspect is de noodzakelijke *time-to-market*. Wanneer deze laag is kan dat de organisatie een concurrentievoordeel bieden en zal in dat geval sneller een positieve business case voor Continuous Delivery bieden.

VOORWOORD

"Uber, the world's largest taxi company, owns no vehicles. Facebook, the world's most popular media owner, creates no content. Alibaba, the most valuable retailer, has no inventory. And Airbnb, the world's largest accommodation provider, owns no real estate. Something interesting is happening." (Goodwin, 2015).

Software is niet meer weg te denken als *enabler* van radicale en disruptieve innovaties. Andreessen (2011) stelde zes jaar geleden al in zijn essay "Why Software Is Eating the World" dat elk bedrijf – om te voorkomen dat het wordt 'disrupted' – een softwarebedrijf zou moeten worden of in ieder geval zo zou moeten denken. Software is dus een belangrijk om te kunnen concurreren, waarbij *time-to-market*, flexibiliteit, prijs en kwaliteit belangrijke criteria zijn. In de afgelopen decennia zijn er veel software-ontwikkelmethoden ontstaan, die optimaliseerden op een of meerdere criteria, maar altijd was er een sprake van een *trade-off* of andere nadelen.

Doordat ik binnen mijn organisatie betrokken was bij de implementatie van Continuous Delivery zag ik de potentie van deze nieuwe ontwikkelmethode en besloot ik bij mijn afstuderen van mijn parttime master bedrijfskunde aan de RSM Erasmus University het effect van Continuous Delivery op het succes van Agile software-ontwikkeltrajecten te onderzoeken. Het onderzoek is uitgevoerd van januari tot en met juli 2017 en gebaseerd op zeven cases bij vier organisaties.

Zoals uit het onderzoek blijkt is een belangrijke succesfactor van Continuous Delivery de korte en kwalitatief goede feedbackloop. Dat geldt voor het schrijven van code, maar ook van een scriptie. Ik wil dan ook mijn coach Jan van den Ende hartelijk bedanken voor de concrete en snelle feedback en de praktische suggesties ten aanzien van het onderzoek, maar vooral voor zijn positieve grondhouding. Mijn meelezer Aart Simons wil ik bedanken dat hij ondanks de bouw van een varend woonschip toch tijd kon vinden om mij buiten de drukke reguliere werkzaamheden om van waardevolle suggesties te voorzien.

Daarnaast wil ik mijn leidinggevende Gerrit Jan van den Toorn en mijn toenmalig directeur Claudia de Andrade bedanken voor het geloof in mij en het faciliteren van deze opleiding. Mijn collega Johannes Sim dank ik voor zijn optimisme en niet-aflatende steun in zowel raad als daad bij het uitvoeren van dit onderzoek. Voor wat betreft de gehele studietijd ben ik een woord van dank verschuldigd aan mijn studiegenoten Remco, Karina, Marika, Willemijn, Linda en Coby voor hun humor, steun en kameraadschap bij de vele waardevolle projecten die we in verschillende samenstellingen hebben mogen uitvoeren.

Het laatste woord van dank gaat uit naar mijn gezin. Jelle en Douwe: dank voor jullie begrip als papa de studieboeken moest verkiezen boven een gezamenlijke gezinsactiviteit. En Caroliene: deze studie had ik niet kunnen doen zonder jouw steun en hulp. Ook jij hebt heel veel moeten laten om deze studie mogelijk te maken. Deze studie hebben we daarom dan ook samengedaan!

16 juli 2018,
Allard Baronner

Inhoudsopgave

Begrippenlijst

Voor het lezen van dit onderzoek is geen technische kennis vereist. Helaas wordt er niet aan ontkomen om een aantal IT-termen te gebruiken. Bij de introductie worden deze begrippen beschreven, maar als naslag hieronder eveneens toegelicht.

Accepteren	Volgens het OTAP-principe opgeleverde software wordt voordat deze op de productie-omgeving wordt ontsloten geaccepteerd door IT-beheer en/of de business om te zorgen dat de software goed aansluit bij de gebruiker en eventuele issues bekend zijn. Onderdeel van de acceptatie kunnen gebruikersacceptatietest, productie-acceptatietest en (functioneel) beheeracceptatietest zijn.
Bug(fix)	Een bug is een softwarefout, waardoor de functie afwijkt van de specificaties. Een bugfix is aanvullende software die een softwarefout verhelpt.
Deployen	Een technische handeling waarbij de software van de ene omgeving naar de andere omgeving wordt gezet. Een deploy naar de productieomgeving wordt een release genoemd.
Development	De organisatieonderdelen die verantwoordelijk zijn voor het ontwikkelen van de software.
Last mile	De stappen die na een software-oplevering noodzakelijk zijn om deze beschikbaar te stellen voor de eindgebruikers in de productie-omgeving.
Lead-user	Een gebruiker die meestal voorloopt op de gemiddelde gebruikers en helpt met de verdere ontwikkeling van een product (ook wel *key-user* genoemd).
Operations	De organisatieonderdelen die verantwoordelijk zijn voor het technisch beheer van de software, onderliggende infrastructuur en koppelingen met andere systemen.
OTAP:	Een fasering in softwareontwikkeling die ervoor zorgt dat nieuwe releases beheersbaar en efficiënt worden gereleased. Software wordt in de O(ntwikkel)-omgeving gebouwd, in de T(est)-omgeving getoetst, in de A-(cceptatie) omgeving geaccepteerd en tot slot in de P(roductie)-omgeving gereleased.
Productie-omgeving	De omgeving waarop de software is ontsloten voor de gebruiker (soms ook live-omgeving genoemd).
Releasen	Opgeleverde software voor de eindgebruiker beschikbaar stellen (dus in de productie-omgeving).
Rework	Rework is het werk wat opnieuw gedaan moet worden omdat deze niet voldoet.

Lijst van Figuren en Tabellen

1 INLEIDING

In dit hoofdstuk wordt de aanleiding van dit onderzoek geschetst, gevolgd door de probleemstelling, de onderzoeksvragen en de onderzoeksdoelstellingen. Daarna wordt gekeken naar de wetenschappelijke en praktische relevantie en vervolgens wordt het onderzoek verder verscherpt met het conceptueel ontwerp en de afbakening. Het hoofdstuk sluit af met een beschrijving van de opbouw van deze scriptie.

1.1 AANLEIDING

In 2001 zag het Agile Manifesto het levenslicht en sindsdien is de adoptie van Agile software ontwikkelmethoden spectaculair te noemen (Fuggetta & Di Nitto, 2014; Ignatius, 2016). Agile heeft de klassieke 'muur' tussen softwareontwikkelaars (*developers*) en de business sterk verlaagd waardoor de software beter aansluit bij de wensen van de klant en gebruiksvriendelijker is. Bovendien zijn Agile softwareontwikkeltrajecten wendbaarder, is de kwaliteit en ontwikkelsnelheid van de software hoger en zijn de kosten lager dan bij traditionele ontwikkelmethoden (Begel & Nagappan, 2007; Dybå & Dingsøyr, 2008; Laanti, Salo, & Abrahamsson, 2011; Petersen & Wohlin, 2009, 2010; Ramesh & Devadasan, 2007; Vijayasarathy & Turk, 2008; Solinski & Petersen, 2016). De toenemende complexiteit van software en een verder stijgende druk om de *time-to-market* te verkorten tonen echter de noodzaak aan om te kijken naar de stap na Agile.

Continuous Delivery is zo'n stap en verschuift de focus van het opleveren van software naar het daadwerkelijk beschikbaar krijgen van de software voor de eindgebruiker, binnen de IT 'releasen' genoemd. Software voegt pas waarde toe als deze voor de gebruiker ontsloten is, niet als deze verstoft op een CD of USB-stick. De activiteiten die hiervoor nodig zijn worden de *'last mile'* genoemd. Dat is een eufemisme, want deze laatste loodjes wegen – zeker bij business software in grote organisaties – zwaar. De *'last mile'* is vaak een pijnlijk, risicovol en tijdrovend proces (Krusche & Alperowitz, 2014; Neely & Stolt, 2013), waardoor software niet of te laat voor de gebruiker beschikbaar is en de voordelen van Agile zoals kortere *time-to-market* en kortere feedbackloop met gebruikers slechts ten dele benut worden. Hoofdoorzaak is de *Wall of Confusion* (Remi Jullian & Sangeetha, 2016), een denkbeeldige muur tussen de ontwikkelaars van de software (*Development*) en *IT-Operations* (hierna te noemen *Operations*), het organisatieonderdeel wat verantwoordelijk is voor het releasen, en beschikbaar houden van IT.

Continuous Delivery verlaagt onder andere de *Wall of Confusion* door verregaande integratie van onder andere processen, systemen, organisatieonderdelen en culturen. Het is dus meer dan een nieuwe innovatieve manier van softwareontwikkeling en een heel nieuw paradigma om een organisatie te managen die afhankelijk is van software (Akerele, Ramachandran, & Dixon, 2014; Chen, 2015; Humble & Farley, 2010; Maalej, Happel, & Rashid, 2009). Continuous Delivery vervangt Agile niet, maar biedt een set van samenhangende principes en best-practises die de *time-to-market* verder verkorten zonder dat dit ten

koste gaat van de kwaliteit en efficiëntie. Sterker nog: in de literatuur worden juist op deze vlakken – bovenop de voordelen van Agile – diverse voordelen van Continuous Delivery geschetst: verdere verbetering van productiviteit en efficiency, hogere productkwaliteit en een verhoogde klanttevredenheid, mede door het verbeteren en versnellen van de communicatie tussen developer en gebruikers (Chen, 2015; Fitzgerald & Stol, 2014; Humble & Farley, 2010; Neely & Stolt, 2013). De onderbouwing van deze claims zijn echter vaak indirect en niet gedegen empirisch onderbouwd. Aangezien het realiseren van Continuous Delivery aanzienlijke investeringen vraagt is een empirische onderbouwing van Continuous Delivery op het succes van software-ontwikkeltrajecten in organisaties zeer relevant.

1.2 Probleemstelling en onderzoeksvragen

De probleemstelling van het onderzoek luidt:

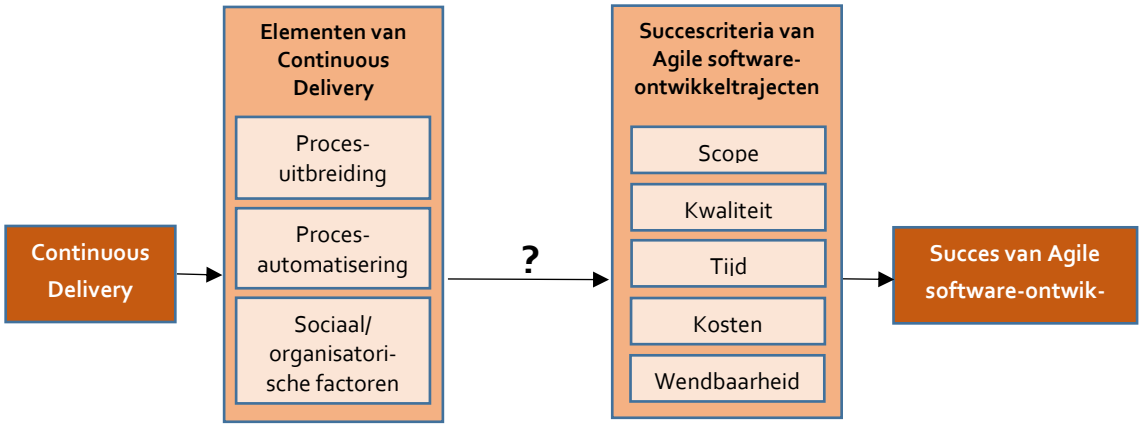
Wat is het effect van het gebruik van Continuous Delivery op het succes van Agile software-ontwikkeltrajecten?

Onder succes van en software-ontwikkeltraject wordt zowel het softwareproduct als de implementatie daarvan in de organisatie verstaan. Om deze probleemstelling verder te kunnen operationaliseren dienen de onderstaande deelvragen te worden beantwoord:

1. Wat zijn Agile software-ontwikkeltrajecten?
2. Wat is Continuous Delivery?
3. Uit welke elementen bestaat Continuous Delivery?
4. Wat zijn de indicatoren van een succesvol Agile software-ontwikkeltraject?

1.3 Conceptueel Ontwerp

Het conceptueel ontwerp visualiseert de onderzoeksvraag en wordt hieronder weergegeven.



Figuur 1: Conceptueel ontwerp

In het model is 'Continuous Delivery' de onafhankelijke variabele en het 'succes van Agile software-ontwikkeltrajecten' de afhankelijke variabele. Hoewel het onderzoek andere uitkomsten kan genereren, wordt een positief effect van Continuous Delivery op het succes van Agile-softwareontwikkeling verwacht. Het onderzoek beoogt niet alleen het effect tussen beide variabelen te analyseren, maar ook de effecten tussen de verschillende aspecten van Continuous Delivery en de diverse succesindicatoren van Agile ontwikkelingstrajecten. In het theoretische kader (hoofdstuk 2) worden deze elementen en indicatoren verder uiteengezet.

1.4 ONDERZOEKSDOELSTELLINGEN EN RELEVANTIE

De doelstelling van dit onderzoek is als volgt geformuleerd:

> *Het doel van dit onderzoek is inzicht te krijgen in de effecten van het gebruik van Continuous Delivery op het succes van Agile software-ontwikkeltrajecten.*

Dit is wetenschappelijk relevant omdat – in tegenstelling tot het concept Agile – de theorievorming rondom Continuous Delivery beperkt is. Zowel in de theorie als in de markt is er geen eenduidige definitie van deze innovatie, desondanks wint Continuous Delivery aan populariteit. Over de relatie tussen het gebruik van Continuous Delivery en het succes van een Agile software-ontwikkeltraject is nog geen gedegen empirisch bewijs in de wetenschappelijk literatuur voorhanden. Het vullen van deze 'gap' is de academische relevantie van dit onderzoek. Daarnaast is dit onderzoek ook praktisch relevant: het biedt besluitnemers (waaronder CIO's, IT-managers, projectmanagers) objectieve handvatten voor investeringsbeslissingen rondom eventuele inzet van Continuous Delivery in software-ontwikkeltrajecten. Dat is relevant, aangezien er aanzienlijke investeringskosten en -inspanningen benodigd zijn voordat Continuous Delivery binnen een organisatie kan worden geïmplementeerd.

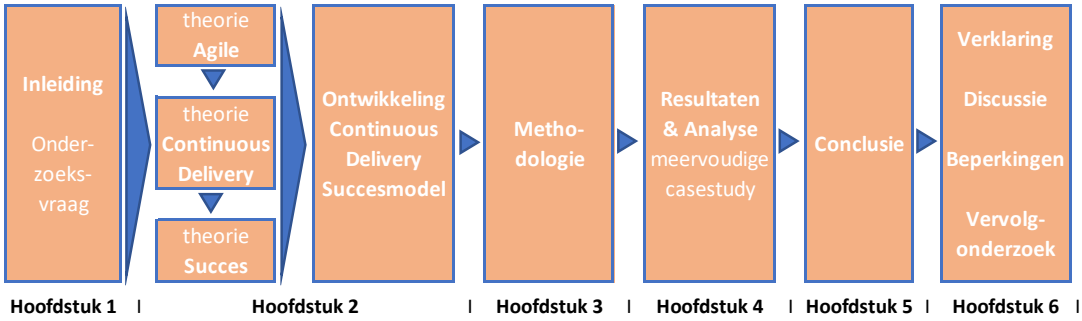
1.5 AFBAKENING

Gezien de beperkt beschikbare tijd voor dit onderzoek zijn veel aspecten van softwareontwikkeling niet behandeld. Centraal staat het effect van Continuous Delivery op Agile software-ontwikkeltrajecten. Hierbij is alleen gekeken naar softwareontwikkeling binnen grotere, software-gedreven organisaties die in Nederland opereren. Bij de implementatie van Continuous Delivery binnen een organisatie wordt niet specifiek stilgestaan. Een analyse naar de barrières en succesfactoren van een Continuous Delivery-implementatie is dan ook niet uitgevoerd.

1.6 LEESWIJZER

In het voorliggend hoofdstuk is de aanleiding en doelstelling van het onderzoek beschreven. In hoofdstuk 2 wordt een theoretische verkenning gedaan naar de thema's Agile, Continuous Delivery en succes binnen de scope van software-ontwikkeltrajecten. Het hoofdstuk wordt afgesloten met het Continuous Delivery succesmodel, een model wat op basis van de literatuur is opgesteld en verder in dit onderzoek

wordt geëxploreerd om de onderzoeksvraag te kunnen beantwoorden. De manier waarop dat is gebeurd, alsmede een onderbouwing van de gekozen aanpak is beschreven in hoofdstuk 3: methodologie. In hoofdstuk 4 worden de onderzoeksresultaten gepresenteerd. Per organisatie zijn twee cases opgesteld: mét en zonder Continuous Delivery. Paarsgewijs worden deze in dit hoofdstuk beschreven en geanalyseerd, waarbij het hoofdstuk wordt afgesloten met een overkoepelende cross-case analyse. De conclusies die uit de onderzoeksresultaten getrokken kunnen worden zijn toegelicht in hoofdstuk 5. Tot slot worden in hoofdstuk 6 de resultaten en in een bredere context bediscussieerd. In paragraaf 6.3 worden de beperkingen van het onderzoek beschreven. De thesis eindigt met voorstellen ten behoeve van vervolgonderzoek.



Figuur 2: Visuele weergave onderzoeksopzet en scriptieopbouw

2 THEORETISCH KADER

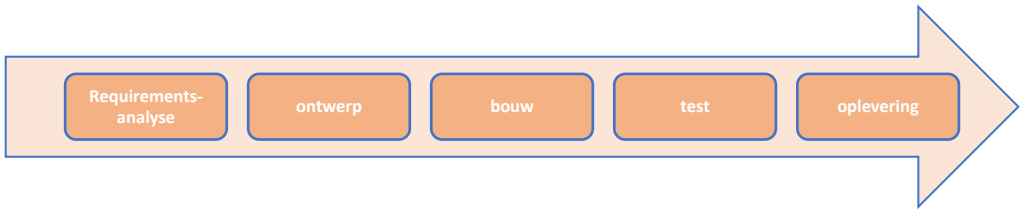
In hoofdstuk 1 zijn de probleemstelling en onderzoeksvragen geïntroduceerd. In dit hoofdstuk worden bestaande theorieën en ideeën hierover in kaart gebracht. Paragraaf 2.1 geeft inzicht in de theoretische principes van Agile ontwikkelmethoden. In paragraaf 2.2 wordt de relevante literatuur rondom Continuous Delivery behandeld en de verschillende elementen van Continuous Delivery uiteengezet. Vervolgens wordt in paragraaf 2.3 uiteengezet wanneer een Agile software-ontwikkeltraject succesvol is. Om richting te geven aan het verdere empirisch onderzoek wordt in paragraaf 2.4 het Continuous Delivery succesmodel geïntroduceerd. Hierin wordt op basis van het literatuuronderzoek het effect van de verschillende elementen van Continuous Delivery op de verschillende succesindicatoren van Agile software-ontwikkeltrajecten voorspeld die verder in het empirisch onderzoek worden geëxploreerd.

2.1 AGILE ONTWIKKELMETHODEN

Om de complexiteit rondom het bouwen van softwaresystemen te beheersen zijn vanaf de jaren zestig methoden ontworpen voor softwareontwikkeling. Deze zijn grofweg te verdelen in traditionele en Agile ontwikkelmethoden.

2.1.1 *TRADITIONELE ONTWIKKELMETHODEN*

De traditionele software ontwikkelmethoden, waaronder de veelgebruikte watervalmethode van Royce (1970) en het Verification & Validation-model (V-model) van Rook (1986) zijn sequentieel van aard. Deze methoden kenmerken zich door een planmatige gefaseerde aanpak met duidelijke omschreven activiteiten en deliverables, waarbij de volgende fase pas start indien de voorafgaande is gecontroleerd en afgetekend. Bij deze planmatige ontwikkelmethoden worden de eisen aan het systeem en de software (requirements) geanalyseerd, vervolgens wordt het systeem ontworpen en daarna pas gebouwd. De implementatie wordt getest en indien succesvol wordt het systeem door de developer opgeleverd aan *Operations*, de IT-beheerders (Bassil, 2012; Rook, 1986; Royce, 1970).



Figuur 3: Fasen in traditionele planmatige softwareontwikkelingsmethodes

Het sterke punt van deze methoden is de 'eerst denken dan doen'-aanpak, waardoor al bij het ontwerp rekening kan worden gehouden met de complexiteit. De planmatige methoden maakt het mogelijk het werk te plannen en daarmee voorspelbaar te maken (Salah, Paige, & Cairns, 2014). Nadeel van de traditionele methoden is dat deze weinig flexibel is door de strikte scheiding tussen fasen. Voor het doorvoeren van nieuwe of aangepaste requirements dienen alle fasen weer te worden doorlopen, iets

wat kostbaar is en daardoor slechts nauwelijks wordt ingezet. Zeker in combinatie met lange doorlooptijden van software-ontwikkeltrajecten leidt dit ertoe dat de software bij oplevering vaak al verouderd is. Omdat het systeem als één geheel wordt gebouwd is er geen ruimte voor tussentijdse feedback door bijvoorbeeld gebruikers, waardoor fouten in de requirements of de softwarecode pas aan het eind ontdekt worden en daarmee de kans nog groter wordt dat deze niet aansluit bij de business. Een risico dat de grondlegger van de watervalmethode al direct onderkende (Royce, 1970). In een steeds sneller veranderende wereld werd vanaf midden jaren negentig gekeken naar alternatieve ontwikkelmethoden.

2.1.2 *AGILE ONTWIKKELMETHODEN*

Met het Software Development Manifesto (later hernoemd naar Agile Manifesto) werd de uit de industriële sector afkomstige term Agile geïntroduceerd. Veel van de zeventien specialisten die het manifest hebben opgesteld en getekend hadden in de jaren daarvoor al alternatieve ontwikkelmethoden geïntroduceerd. Deze methoden kenmerkten zich door hun *mensgerichte* en *iteratieve* aanpak. De menselijke factor werd door Nygaard (1990) geïntroduceerd, die aantoonde aan dat software ontwikkelen een sociale activiteit is die specifieke aandacht nodig heeft om de effectiviteit en efficiency te verhogen. Door iteratief te werken kunnen risico's in het software-ontwikkeltraject worden verkleind en daarmee de kans op succes worden vergroot. Het Agile Manifesto verenigt deze nieuwe zienswijzen op systeemontwikkeling en fungeert daarmee als een paraplu voor deze nieuwe iteratieve en mensgerichte ontwikkelmethoden. Recent wetenschappelijk onderzoek over de populariteit van de verschillende Agile-ontwikkelmethoden ontbreekt. In een survey-onderzoek van het bedrijf VersionOne (2016) blijkt SCRUM echter verreweg de meest gebruikte Agile-methode.

Agile Methode	**Beschrijving**	**Grondlegger / Introductiejaar**
Rapid Application Development (RAD)	Proces is bij RAD belangrijker dan planning. De methode is Agile omdat detailspecificaties niet bij het begin van de softwareontwikkeling wordt opgesteld, maar pas als deze noodzakelijk zijn. Hierdoor kan voortschrijdend inzicht en gewijzigde wensen worden verwerkt. In plaats van specificaties op papier wordt vaak met prototypes gewerkt.	Barry Boehm James Martin en Scott Shultz 1991
Dynamic Software Development Method (DSDM)	De DSDM-methode kent drie fasen: pre-project, project life cycle, en post-project. Het steunt op negen principes, waaronder betrokkenheid van de gebruiker, empowerment van het projectteam, frequente levering en het centraal stellen van de behoefte van de business. Voor het prioriteren van te bouwen onderdelen wordt de MoSCoW-methode toegepast (Must-have, Should-have, Could-have en Would-like-to-have). Prioritering DSDM-softwareontwikkeling start pas als een high-level scope is vastgelegd.	Agile Business Consortium (voorheen DSDM-consortium) 1994
SCRUM	SCRUM is ontwikkeld om software te kunnen ontwikkelen in situaties waarin het lastig vooruitplannen is. Software wordt in stappen (*sprints*) ontwikkeld door zelfsturende teams. Werkzaamheden worden geplaatst op een *backlog*. De *product owner* bepaalt de prioriteit en planning. In de *daily SCRUM* worden de activiteiten besproken. De *SCRUM-master* bewaakt en begeleid het proces. Feedback loops zijn een belangrijk onderdeel van de SCRUM-methode.	Jeff Sutherland* Ken Schwaber* 1995

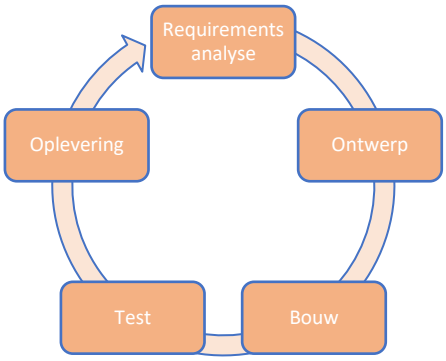
eXtreme Programming (XP/XP2)	eXtreme Programming bestaat uit twaalf (best) practices van softwareontwikkeling, waaronder kleine releases, collectief eigenaarschap, simpel ontwerp en continuous integration. In een herziene versie (XP2) worden ook best practices uit andere methoden geïncorporeerd: het *test-first programming* is ontleed uit de TDD-methode, en de dagelijkse de *sit-in* heeft sterke verwantschap met de *daily SCRUM*. In tegenstelling tot SCRUM is er meer focus op technieken.	Kent Beck*, Ken Auer, Ward Cunningham*, Martin Fowler* en Ron Jeffries* 1996
Crystal	In tegenstelling tot de navolgende methoden is de Crystal family een verzameling van methoden en processen. Familieleden verschillen in zwaarte en worden onderscheiden met kleuren, waarbij de meest Alle methode Crystal Clear is genoemd. Afhankelijk van de omvang en de risico's wordt gekozen voor een methode. Wat de methoden met elkaar gemeen hebben is het streven naar veiligheid, bruikbaarheid en efficiëntie.	Alistair Cockburn* 1996
Feature-Driven Development (FDD)	De eerste drie fasen van FDD hebben veel overeenkomsten met het klassieke waterval ontwikkelen: hierin wordt een globaal ontwerp vastgelegd, verdeeld in stukjes (features) en vervolgens wordt een ontwikkelingsplan opgesteld. De laatste twee fasen zijn meer Agile: tweewekelijks worden features verder ontworpen en gerealiseerd.	Jeff de Luca* 1997
Lean Software development (LSD)	LSD past traditionele Lean-principes toe op softwareontwikkeling. Centraal staan de zeven principes: elimineer verspilling, versterk het leren, beslis zo laat zo mogelijk, lever zo snel als mogelijk, empower het team, ga voor kwaliteit en zie het geheel. De principes worden aangevuld met 22 best-practices van Agile (gereedschappen).	Mary Poppendieck en Tom Poppendieck 2003
Test Driven Development (TDD)	Bij Test Driven Development staat het vooraf automatiseren van testen centraal. Eerst worden de testscripts gebouwd, pas daarna de functionaliteit.	Kent Beck* 2003
Agile Unified Process (AUP)	AUP is gebaseerd op IBM's Rational Unified Process, maar vereenvoudigd en met gebruikmaking van Agile technieken. Bij AUP worden zeven disciplines waaronder modelleren, implementeren, testen en uitrollen iteratief uitgevoerd.	Scott Ambler (2005)

* tevens een van de opstellers/ondertekenaars van het Agile Manifesto.

Figuur 4: Chronologisch overzicht voornaamste Agile ontwikkelmethoden

2.1.3 VISIE VAN AGILE

In het manifest stellen de auteurs dat zij "laten zien dat er betere manieren zijn om software te ontwikkelen door in de praktijk aan te tonen dat dit werkt en door anderen ermee te helpen" (Beck et al., 2001, p. 1). Het Agile Manifesto is gebaseerd op een twaalftal principes waaruit blijkt dat de focus ligt op het toevoegen van waarde aan de klant, óók als zijn behoefte tijdens de bouw verandert. Dit wordt gerealiseerd door een nauwe samenwerking tussen business en ontwikkelaars, continue verbetering en iteratieve, kort cyclische opleveringen van werkende software (Beck et al., 2001). Op basis van deze principes is een gedeelde visie opgesteld, bestaande uit een viertal stellingen die duidelijk een paradigmaverschuiving ten aanzien van de klassieke



Figuur 5: Iteratie in een Agile software ontwikkelingstraject

waterval denken weergeeft. Deze worden hieronder verder toegelicht. Het is bij deze stellingen niet zo dat de begrippen aan de rechterkant er niet toe doen; wanneer er echter een afweging moet worden gemaakt dan weegt het begrip aan de linkerkant zwaarder.

Allereerst stelt het Agile Manifesto dat "Individuen en interacties boven processen en hulpmiddelen" gaan (Beck et al., 2001). Belangrijke elementen in de watervalmethode zijn de diverse documenten waarin de requirements en de ontwerpen nauwkeurig worden beschreven. Binnen Agile softwareontwikkeling wordt gesteld dat een goede samenwerking binnen teams cruciaal is en dat de beste resultaten worden behaald door zelfsturende multidisciplinaire teams waarin ook de business vertegenwoordigd is. Eveneens wordt gesteld dat communicatie de belangrijkste driver voor productiviteit en kwaliteit is en dus boven het volgen van processen gaat. De meest efficiënte interacties zijn die tussen individuen (Fowler & Highsmith, 2001). Een tweede standpunt is dat "samenwerking met de klant boven contracten en onderhandeling" gaat (Beck et al., 2001). Binnen de op Agile gebaseerde ontwikkelmethoden neemt de opdrachtgever een belangrijke plaats in. Immers: hij is degene voor wie de software gebouwd wordt en moet dan ook de mogelijkheid hebben om te kunnen sturen. Het is dan ook nadelig voor de op te leveren software als de requirements onwrikbaar zijn vastgelegd in contracten (Fowler & Highsmith, 2001). Bij softwareontwikkeling dragen documenten niet of nauwelijks bij aan het verhogen van waarde voor de business; werkende software wél. Daarom wordt bij Agile softwareontwikkeling in korte incrementele iteraties gestreefd naar het snel opleveren van belangrijke functionaliteit in werkende software. Uitgebreide documentatie is daarmee ondergeschikt aan een werkend product (Beck et al., 2001; Fowler & Highsmith, 2001, p. 3; Highsmith & Cockburn, 2001, p. 121). Tot slot stelt het Agile Manifesto dat "Aanpassen op veranderingen boven het uitvoeren van een plan" gaat (Beck et al., 2001). Agile is het vermogen om te kunnen veranderen; door te werken in korte iteraties kan meer voortschrijdend inzicht en feedback vanuit de business worden verwerkt. Door niet te ver vooruit te kijken blijft het softwarevoortbrengingsproces flexibel (Fowler & Highsmith, 2001).

2.2 Continuous Delivery

In 2001 zag het Agile Manifesto het levenslicht en sindsdien is de adoptie van Agile ontwikkelmethoden spectaculair te noemen (Fuggetta & Di Nitto, 2014; Ignatius, 2016). Maar de toenemende complexiteit van software en een toenemende druk om de *time-to-market* te verkorten tonen de noodzaak aan om te kijken naar de stap na Agile. Agile methoden hebben ervoor gezorgd dat de

Continuous *Deployment*

In sommige literatuur wordt naast Continuous Delivery ook gesproken over Continuous *Deployment*. Regelmatig gaat het hierbij om synoniemen (zoals bij Claps et al., 2015; Neely & Stolt, 2013; Olsson et al., 2012).

In een enkel geval wordt Continuous Deployment uitgelegd als het vermogen om automatisch te deployen (zie: Fitzgerald & Stol, 2014). In deze opvatting is Continuous Deployment een randvoorwaarde voor Continuous Delivery.

In de meeste artikelen echter worden beide begrippen naast elkaar gebruikt om een verschil aan te duiden dat bij Continuous Delivery de software na elke aanpassing in productie *kan* worden gereleased, terwijl bij Continuous *Deployment* elke aanpassing direct en automatisch *wordt* gereleased (Claps et al., 2015; Fitz, 2009; Fitzgerald & Stol, 2014; Gfader, 2013; Humble, 2010; Humble & Farley, 2010, 2010, Sharma, 2014, 2014). In deze opvatting is Continuous Delivery dus randvoorwaardelijk voor Continuous Deployment.

gebruiker bij de softwareontwikkeling wordt betrokken, maar er is een urgente noodzaak om van het daadwerkelijke gebruik ervan in productie (en dus niet op een test- of demo-omgeving) te leren (Olsson, Alahyari, & Bosch, 2012). Continuous Delivery verschuift de focus van het schrijven van code naar het bouwen, testen en releasen van software. Continuous Delivery vervangt Agile niet, maar biedt een set van samenhangende principes en best-practises die de *time-to-market* verder verkorten zonder dat dit ten koste gaat van de kwaliteit en efficiëntie. Sterker nog: in de literatuur worden op dit vlak diverse voordelen van Continuous Delivery geschetst: verbetering van productiviteit en efficiency, hogere productkwaliteit en een verhoogde klanttevredenheid, mede door het verbeteren en versnellen van de communicatie tussen developer en gebruikers (Chen, 2015; Fitzgerald & Stol, 2014; Humble & Farley, 2010; Neely & Stolt, 2013). Deze claims zijn echter niet empirisch onderbouwd.

2.2.1 *Definitie van Continuous Delivery*

Continuous Delivery is het vermogen om op elk moment veilig softwareaanpassingen op de productie-omgeving te kunnen releasen en daarmee te ontsluiten voor de eindgebruiker (Fitzgerald & Stol, 2014; Humble & Farley, 2010; Humble, Molesky, & O'Reilly, 2015; Krusche & Alperowitz, 2014; Neely & Stolt, 2013). Dat de software altijd release-klaar is wil nog niet zeggen dat dat ook daadwerkelijk bij elke wijziging in de code gebeurt. Continuous Delivery is dus niet het verkorten van de duur van een sprint (Gfader, 2013). Sterker nog: de cadans van een sprint hoeft niet gelijk te lopen met die van de release. Zo hoeft het implementeren van een oplossing voor een softwarefout niet te wachten tot het einde van een sprint, maar kan deze tussentijds op de productie-omgeving worden doorgevoerd. Daarentegen kan ook besloten worden om een groot stuk functionaliteit waaraan in meerdere sprints wordt gewerkt niet tussentijds na iedere sprint te releasen, maar in één keer. De verantwoordelijkheid voor het bepalen wanneer software wordt gereleased is bij Continuous Delivery dan ook geen technische keuze meer (en dus bepaalt door IT), maar een keuze die door de business kan worden gemaakt (Gfader, 2013). Uiteraard geldt hierbij dat frequent releasen de betrouwbaarheid van het proces vergroot. Het verdient dan ook aanbeveling om een groot stuk functionaliteit toch tussentijds in productie te releasen. Releasestrategieën als *feature toggle* en *dark launch* kunnen er dan voor zorgen dat de onderhanden zijnde features niet voor de gebruikers beschikbaar zijn, maar voor slechts een beperkte groep developer en testers.

2.2.2 *Beïnvloeding*

Continuous Delivery heeft sterke wortels in de Agile softwareontwikkeling. Het eerste Agile-principe "het tevredenstellen van de klant door het vroegtijdig en voortdurend opleveren van waardevolle software" (Beck et al., 2001, p. 1) heeft sterke overeenkomst met de doelstelling van Continuous Delivery om op elk moment veilig softwareaanpassingen op de productie-omgeving te kunnen releasen en daarmee te ontsluiten voor de eindgebruiker. Een andere beïnvloeding komt vanuit *Lean Thinking,* en dan met name het begrip *flow* (Fitzgerald & Stol, 2014). Hierbij wordt werk in een continue stroom met een zo kort mogelijke doorlooptijd gerealiseerd en contrasteert daarmee sterk met op *'batch & queue'* gebaseerde processen (Womack & Jones, 2010). Het woord 'continuous' in Continuous Delivery is een duidelijke verwijzing naar dit *flow*-gedachtegoed.

2.2.3 *PRINCIPES*

Continuous Delivery is gestoeld op een aantal principes (Humble & Farley, 2010), die tezamen het fundament vormen. Deze worden hieronder – enigszins gecategoriseerd – samengevat.

HERHAALBAAR EN BETROUWBAAR PROCES / AUTOMATISEER ALLES / ALLES ONDER VERSIEBEHEER

Om voortdurend en veilig softwareaanpassingen te kunnen doen is een herhaalbaar en betrouwbaar proces noodzakelijk. Om dit te bereiken geldt binnen Continuous Delivery het adagium "*If it hurts, do it more often"*. Door het sterk verhogen van de releasefrequentie wordt de betrouwbaarheid van het proces getoetst en – in combinatie met het laatste 'continue verbetering' – dus voortdurend verbeterd indien nodig. Betrouwbaarheid wordt ook verhoogd door in alle omgevingen hetzelfde deploymentproces te hanteren. Wanneer voor elke omgeving een ander deploymentproces nodig is "dan vinden issues een manier om op te duiken" (Humble & Farley, 2010, p. 25). Handmatige activiteiten zijn in de regel minder goed herhaalbaar en betrouwbaar en bovendien veel minder efficiënt. Het tweede principe propageert dan ook om alle stappen om van een wijziging in de code tot een wijziging in de productie-omgeving te komen te automatiseren: *Computers perform repetitive tasks, people solve problems*. Voor een stevige basis onder het automatiseren en het proces is het noodzakelijk dat er slechts één *single point of truth* is. Versiebeheer dient dan ook op alles wat relevant is ingericht te worden: code, configuratie, scripts, databases, documentatie, etc.

DOE WAT MOEILIJK IS EERST / BOUW KWALITEIT IN

'Doe wat moeilijk is eerst' is wat heuristisch te noemen en kan breder worden toegepast dan alleen op softwareontwikkeling. Humble & Farley (2010, p. 26) stellen dat "*Bring the pain forward – deal with the hard stuff first*" efficiënter is dan hiermee wachten tot aan het einde van het ontwikkelingstraject. In het begin staan meer oplossingsrichtingen open en zal minder verspilling van code door herbouw (*refactoring*) nodig zijn dan aan het eind van een software-ontwikkeltraject. Hoe eerder een issue wordt opgelost in het proces hoe efficiënter deze is. Het principe 'bouw kwaliteit in' gaat over het verkorten van de feedbackloop door bijvoorbeeld testen in te bouwen die de developer al op een softwarefout wijzen zodra deze wordt gemaakt. Op deze manier wordt sneller kwalitatieve code geschreven en worden er aan het einde van het software-ontwikkeltraject (wanneer het oplossen van fouten veel inefficiënter is) aanzienlijk minder issues naar boven.

'AF' BETEKENT 'IN PRODUCTIE' / IEDEREEN IS VERANTWOORDELIJK

Software is pas waardevol als deze voor de gebruiker ontsloten is, en dit principe geeft aan dat iedereen in de keten, dus ook de developer hierin een verantwoordelijkheid hebben. De cultuurverandering die dit met zich meebrengt kan lastig zijn om te implementeren (Humble & Farley, 2010). Continuous Delivery breidt de scope van het ontwikkelingsproces uit dat software niet alleen wordt opgeleverd, maar ook wordt getest en gereleased zodat deze beschikbaar is voor de gebruiker. Pas dan is de software, ook voor *Development*, 'af'.

CONTINUE VERBETERING

Continue verbetering (*Continuous Improvement*) is een belangrijk onderdeel van Continuous Delivery, met name waar het aankomt op de automatisering. "Als 'oefening kunst baart', dan is automatisering de overtreffende trap: perfect, herhaalbaar, betrouwbaar en efficiënt (Humble & Farley, 2010, p. 28). Om zover te komen is een cultuur en aanpak van continue verbetering randvoorwaardelijk.

2.2.4 *ELEMENTEN VAN CONTINUOUS DELIVERY*

In de literatuur wordt niet exact gespecificeerd uit welke *best-practises* Continuous Delivery bestaat. Wel worden testautomatisering, sterke teamsamenwerking, deploymentautomatisering, een goede teamcultuur, versiebeheer en Continuous Integration als de belangrijkste *practices* gezien om de effectiviteit van Continuous Delivery te stimuleren. (Akerele et al., 2014; Humble & Farley, 2010). Continuous *Integration is* het automatiseren van het *merge*- en *build*-proces: het samenvoegen van de wijzigingen in de software en deze compileren zodat deze geschikt is om te kunnen installeren. In dit onderzoek wordt om een aantal redenen van de bovenstaande opsomming afgeweken. De benodigde procesextensie om opgeleverde software ook daadwerkelijk voor de eindgebruikers aan te kunnen bieden wordt mist in deze opsomming, maar wordt wel impliciet genoemd in de literatuur. 'Procesuitbreiding' is echter gezien een essentieel onderdeel van Continuous Delivery en wordt daarom in dit onderzoek dan ook expliciet als element benoemd. Daarnaast is gekozen om een aantal vergelijkbare onderdelen samen te voegen: 'sterke teamsamenwerking' en 'goede teamcultuur' worden in dit onderzoek samengevoegd in het element 'sociaal/organisatorische factoren'. Vervolgens zijn 'versiebeheer', 'Continuous Integration', 'testautomatisering' en 'deploymentautomatisering ondergebracht in het element 'procesautomatisering'. Dat resulteert in onderstaand overzicht, waarbij de elementen onder de gelijknamige kopjes verder worden toegelicht.

CD-element	Korte omschrijving	CD-principes
Procesuitbreiding	Een procesuitbreiding waarbij het softwareontwikkelproces niet ophoudt bij de oplevering van de software aan *Operations*, maar wordt verlengd totdat de software ook daadwerkelijk herhaalbaar en betrouwbaar in productie ontsloten is voor de gebruikers.	*- Herhaalbaar en betrouwbaar proces* *- 'Af' betekent 'in productie'*
Procesautomatisering	Het verregaand automatiseren van de processen rondom het samenstellen, deployen en testen van software.	*- Automatiseer alles* *- Alles onder versiebeheer* *- Herhaalbaar en betrouwbaar proces* *- Doe wat moeilijk is eerst*
Sociaal/ organisatorisch factoren	De voor een Continuous Delivery implementatie benodigde aanpassingen in de organisatie, de organisatiecultuur en de wijze van samenwerking.	*- Doe wat moeilijk is eerst* *- 'Af' betekent 'in productie'* *- Iedereen is verantwoordelijk* *- Continue verbetering*

Figuur 6: elementen van Continuous Delivery

PROCESUITBREIDING

Het eerste principe van het Agile Manifesto is: "onze hoogste prioriteit is het tevredenstellen van de klant door het vroegtijdig en voortdurend opleveren van waardevolle software." (Beck et al., 2001, p. 1). Bij de hiervoor beschreven Agile ontwikkelmethoden zoals SCRUM, eXtreme Programming en Rapid Application Development is de scope beperkt tot het opleveren van de software. Om na een oplevering van software deze ook voor de gebruiker beschikbaar te krijgen in de *live* of productieomgeving dienen nog additionele stappen te worden gevolgd, ook wel bekend als de *'last mile'*. Alleen dan kan de software de volledig toegevoegde waarde bieden. De eerste stap is dat de oplevering voor Quality Assurance (QA)-doeleinden wordt getoetst in een acceptatie-omgeving die vergelijkbaar is met de uiteindelijke productieomgeving. Zo kan QA - zonder dat de productie-omgeving hiervan last ondervindt - toetsen of de software goed integreert met de rest van het systeemlandschap en voldoet aan de acceptatiecriteria. Zo lang geconstateerde bevindingen niet zijn opgelost of gemitigeerd kan de software niet door naar productie. In de laatste stap wordt de software aangeboden aan de eindgebruikers door de opgeleverde software uit te rollen op de productieomgeving. Continuous Delivery integreert deze twee stappen in het software-ontwikkelproces.

Figuur 7: Scope Continuous Delivery-proces

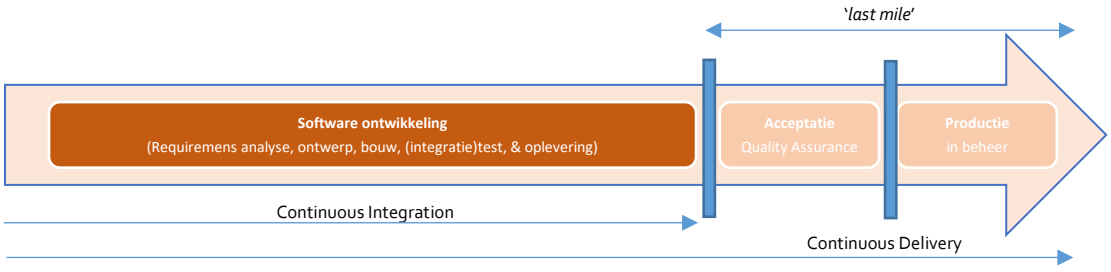
De *'last mile'* is vaak een pijnlijk, risicovol en tijdrovend proces (Humble & Farley, 2010; Krusche & Alperowitz, 2014; Neely & Stolt, 2013). Ten eerste bevat een deployment op een andere omgeving veel handmatig werk en is dus naast tijdrovend ook foutgevoelig. Ten tweede wordt pas in de acceptatiefase de software getoetst in het reeds bestaande systeemlandschap. Op dat moment komen pas de integratie-issues tussen software en infrastructuur naar boven. Daarnaast wordt analyse bemoeilijkt doordat de issues zich op het snijvlak van meerdere specialisaties en meerdere afdelingen bevindt: de software (*Development*) en infrastructuur (*Operations*). Dit snijvlak wordt in de literatuur ook wel aangeduid als de *Wall of Confusion* (Remi Jullian & Sangeetha, 2016). Het vijfde punt is dat ontwikkeling, acceptatie en beheer vaak aparte silo's binnen een organisatie met andere werkwijzen en culturen waar alleen verantwoordelijkheden worden genomen binnen de eigen silo (over de muur gooien). Hierdoor wordt het oplossen van issues bemoeilijkt. Een ener laatste oorzaak ligt in het feit dat releases in de regel niet-frequent worden uitgevoerd, waardoor expertise en documentatie nauwelijks wordt opgebouwd. Tot slot botst het belang van *development*, geënt op vernieuwing met het belang van *Operations* om de omgeving zo stabiel mogelijk te houden, want elke verandering is een kans op instabiliteit (Chen, 2015).

Het voordeel van het integreren en frequent uitvoeren van de *'last mile'* in het software-ontwikkelproces is drieledig. Ten eerste zorgt deze ervoor dat integratie-issues tussen software en de productieomgeving eerder in het proces worden opgemerkt, waardoor afstemming en aanpassing nog relatief eenvoudig is. Ten tweede verkort het de feedbackloop met de gebruikers. Zeker bij de snel wijzigende requirements in Agile-trajecten helpt het de ontwikkelaars om hun begrip van de requirements te toetsen (Krusche & Alperowitz, 2014). Dit verhoogt de kwaliteit van de software en wordt 'verspilling' (een begrip uit *Lean Thinking*) eerder gedetecteerd en dus verminderd. Voor succesvolle integratie van de *'last mile'* zijn additionele aanpassingen nodig. Dat klinkt eenvoudig, maar er liggen – zeker in bestaande ontwikkel-en-release omgevingen van grote organisaties – aanzienlijke uitdagingen en zelfs barrières op het technische en organisatorische vlak (Chen, 2015; Olsson et al., 2012). Tot slot wordt de *'last mile'* door de hoge releasefrequentie onderdeel van de reguliere werkzaamheden en daarmee een *non-event*.

PROCESAUTOMATISERING

Centraal bij Continuous Delivery staat de Delivery Pipeline, een (bijna) volledig geautomatiseerde workflow. De pipeline bestaat uit een drietal onderdelen die tezamen zorgen voor de procesautomatisering: Continuous *Integration*, deploymentautomatisering en testautomatisering.

De eerste helft van de pipeline, Continuous *Integration* genoemd, heeft zijn oorsprong in de eXtreme Programming softwaremethoden (Fowler & Foemmel, 2006; Stolberg, 2009), maar is ook toepasbaar op de andere Agile ontwikkelmethoden (Beck, 2000). In de wat oudere literatuur wordt een aantal principes van Continuous Delivery ook onder Continuous Integration geschaard (Fowler & Foemmel, 2006; Stolberg, 2009), maar in de recente literatuur wordt onder Continuous Integration het vermogen verstaan om na elke wijziging in de softwarecode de software opnieuw samen te stellen, te integreren (*builden*) en tot slot deels te testen. Bij Continuous Integration werken softwareontwikkelaars samen aan hetzelfde softwareproject, waarbij de door de developer individueel opgeleverde code automatisch integreert met de bestaande code (de zogenaamde *master branch*) en in samenhang getest wordt. Wanneer een test niet slaagt krijgt de developer die de wijziging heeft doorgevoerd de opdracht om direct de gewijzigde code ofwel te verbeteren ofwel terug te trekken. Hierdoor is er altijd een werkende softwareversie beschikbaar voor release en reduceert het software integratieproblemen, waardoor sneller betrouwbare software kan worden opgeleverd (Fowler & Foemmel, 2006; Ståhl & Bosch, 2014; Stolberg, 2009; Waller, Ehmke, & Hasselbring, 2015).



Figuur 8: Scope Continuous Integration

Continuous Delivery brengt "het concept van Continuous *Integration* naar het volgend plateau" (Remi Jullian & Sangeetha, 2016, p. 12571) door in het tweede deel van de pipeline de *'last mile'* te standaardiseren en te automatiseren. Zo verkleint Continuous Delivery de 'gap' tussen developer, *Operation*-medewerkers en de eindgebruiker. Hierdoor kunnen automatisch (acceptatie)testen worden gestart en wanneer deze slagen wordt automatisch gedeployd op een andere omgeving. Hoewel ook de laatste stap, de release van de software naar de productie-omgeving, geautomatiseerd is, wordt deze bij Continuous Delivery niet automatisch geïnitieerd, maar door een mens, script of business-regel afgeroepen (Chen, 2015; Humble, 2010; Neely & Stolt, 2013; Remi Jullian & Sangeetha, 2016; Sharma, 2014).

Een deployment van software op een willekeurige omgeving is niet slechts het kopiëren van wat bestanden (*files*) van de ene naar de andere plek. Met name bij bedrijfssoftware van grote organisaties is een deployment een complexe en risicovolle activiteit (Chen, 2015; Humble, 2010; Neely & Stolt, 2013). Het is een orkestratie van activiteiten met onderlinge afhankelijkheden om de diverse onderdelen van de code op de juiste plekken en in de juiste volgorde geoperationaliseerd te krijgen op de verschillende IT onderdelen, waaronder applicaties, applicatieservers, databases en database schema's, systeemprocessen, proxy-instellingen en uitwijkomgevingen (Sharma, 2014). Deploymentautomatisering maakt het deploymentproces herhaalbaar en betrouwbaarder. Bovendien verkort het de doorlooptijd aanzienlijk (Akerele et al., 2014; Chen, 2015; Humble & Farley, 2010; Maalej et al., 2009). Dit omdat automatische activiteiten sneller worden uitgevoerd dan handmatige. Bovendien worden de wachttijden die ontstaan doordat meerdere teams of personen een activiteit moeten uitvoeren weggenomen. Een laatste reden is dat automatische deployments minder foutgevoelig zijn waardoor er minder tijd gaat zitten in foutopsporings- en -herstelactiviteiten. De hoge frequentie van deployments, gecombineerd met continue verbetering voor een verdere afname van falende deployments.

Het automatiseren van testen is bij Continuous *Integration* reeds geïntroduceerd, maar dit beperkt zich tot unit en hooguit unitintegratietesten (Stolberg, 2009). Hierbij wordt gekeken of de opgeleverde software-onderdelen, individueel respectievelijk in samenhang voldoen aan de opgestelde specificaties. Voorheen was dit het domein van de developer. Om te controleren of de software ook productie-klaar is werd de software overhandigd aan Quality Assurance (QA) die vervolgens systeemtesten uitvoert op een op productie-lijkende omgeving, waaronder interface-, performance-, volume-, stress-, stabiliteits- en beveiligingstesten. Daarna controleerde de klant en/of gebruiker in een (gebruikers) acceptatietest of deze voldoet aan de business case die ten grondslag lag van de software en of deze tot slot gereleased kon worden in productie (W. E. Perry, 2006). Het doel van Continuous Delivery is om al deze testen te automatiseren. En aangezien de code continu verandert zal bij elke wijziging in de code moeten worden bepaald of de testscripts moeten worden aangepast. Om te voorkomen dat niet-geteste onderdelen zonder te testen in productie kunnen komen moeten eerst de testscripts worden aangepast, alvorens de code op te leveren. Goed versiebeheer is hierbij randvoorwaardelijk. Omdat bij testautomatisering alleen vaste kosten zijn voor het maken en onderhouden van de testscripts kunnen de testen zelf kosteloos worden uitgevoerd. Hierdoor kunnen testen frequenter en eerder in het proces

worden uitgevoerd, wat de feedbackloop verkleind en de kwaliteit en efficiency van de software verhoogt (Akerele et al., 2014; Chen, 2015; Humble & Farley, 2010; Maalej et al., 2009)

SOCIAAL/ORGANISATORISCHE FACTOREN

Hoewel Continuous Delivery zwaar leunt op proces- en technische aanpassingen is de grootste uitdaging van niet-technische aard (Chen, 2015). In een onderzoek van Claps (2015) naar de uitdagingen van Continuous Delivery implementaties vond hij negen technische, tegenover elf sociale uitdagingen. Fitz (2009) stelt dat cultuur gezien kan worden als een kritieke succesfactor. De organisatie IMVU ziet een cultuur van frequent releasen als pilaar onder het succes van CD (Fitz, 2009). Claps et al. (2015) zien als succesfactor van Continuous Delivery een cultuur waarin 'met elkaar gedeeld' wordt. Met name bij de *'last mile'* zijn veel afdelingen betrokken, allen met verschillende belangen, werkwijzen en beelden over verantwoordelijkheden en hoe samen te werken. Door aanpassing van organisatie en cultuur kan de voor succesvolle Continuous Delivery benodigde nauwere samenwerking worden bereikt (Akerele et al., 2014; Chen, 2015). Het gaat hierbij om de implementatie van de eerder beschreven principes van Continuous Delivery, en met name: 'doe wat moeilijk is eerst', '*af* betekent *in productie*', 'iedereen is verantwoordelijk' en 'continue verbetering'. Als een extensie van Agile zijn bij Continuous Delivery uiteraard ook de in paragraaf 2.1.2 beschreven Agile principes van toepassing: 'mensen en hun onderlinge interactie boven processen en hulpmiddelen', 'werkende software boven allesomvattende documentatie', 'samenwerking met de klant boven contractonderhandelingen' en tot slot 'inspelen op verandering boven het volgen van een plan (Beck et al., 2001).

Continuous delivery steunt op zowel *Development* als *Operations* en heeft daarom sterke raakvlakken met het DevOps-fenomeen (Fitzgerald & Stol, 2014). DevOps is een cultuur, beweging en practise die probeert de communicatie, samenwerking en integratie tussen *Development* en *Operations* te verbeteren (Remi Jullian & Sangeetha, 2016; Waller et al., 2015). Maar voor succesvolle implementatie van Continuous Delivery is betrokkenheid van meerdere afdelingen noodzakelijk (Olsson et al., 2012). Dus niet alleen *Development* en *Operations*, maar ook Q&A, IT-security, de business, etc. De veranderende verantwoordelijkheden, waarbij iedereen verantwoordelijk is voor een succesvolle release is makkelijker gezegd dan geïmplementeerd (Claps et al., 2015).

2.3 SUCCESVOLLE AGILE SOFTWARE-ONTWIKKELTRAJECTEN

Software-ontwikkeltrajecten bij grotere organisaties worden in de regel gemanaged als projecten. Pinto en Slevin (1988) stelden decennia geleden al dat er weinig onderwerpen in de projectmanagementliteratuur te vinden zijn waarover zoveel geschreven is en toch zo weinig consensus over heerst als de definitie van projectsucces. Om de onderzoeksvraag te kunnen beantwoorden zal het begrip succes toch gedefinieerd moeten worden. In deze paragraaf wordt een voorzet gedaan voor succesindicatoren voor Agile software-ontwikkeltrajecten. Vervolgens worden op basis van empirisch onderbouwde onderzoeken de effecten van Agile op deze succesfactoren in kaart gebracht. Tot slot wordt op basis van een analyse van de literatuur de verwachte toegevoegde effecten van Continuous Delivery op het succes van Agile ontwikkelingstrajecten geanalyseerd, als basis voor verdere empirische validatie.

2.3.1 TRADITIONELE SUCCESINDICATOREN VOOR SOFTWARE-ONTWIKKELTRAJECTEN

Traditioneel in projectmanagement is de duivelsdriehoek (in het Engels bekend onder verschillende namen, variërend van '*projectmanagement/iron/golden triangle*' tot *'holy trinity')*, een model waarin succes wordt gemeten aan het vermogen om het project zodanig te managen, dat de verwachte resultaten (kwaliteit) worden gerealiseerd binnen tijd en binnen budget. 'Kwaliteit', 'tijd' en 'kosten' worden in deze modellen gezien als de drie succesindicatoren en vormen de hoeken van de duivelsdriehoek (Atkinson, 1999; Ika, 2009). In sommige modellen wordt 'kwaliteit' vervangen door 'scope' en wordt de oppervlakte van de driehoek gezien als 'kwaliteit'. Het succes van een software traject kan eveneens worden gemeten op basis van deze vier indicatoren: 'kwaliteit' (wordt goed werkende software opgeleverd), 'scope' (sluit de software aan bij de behoeften van de business), 'tijd' (wordt er tijdig geleverd) en 'kosten' (is er binnen budget en uren gebleven) (Cohn & Ford, 2003; Lindvall et al., 2004).

2.3.2 AGILE SUCCESINDICATOREN VOOR SOFTWARE-ONTWIKKELTRAJECTEN

Deze hiervoor genoemde indicatoren worden ook geacht bruikbaar te zijn voor het beoordelen van het succes van Agile software projecten (Chow & Cao, 2008). In sommige onderzoeken wordt 'gebruikerstevredenheid' en 'productiviteit' als aparte indicatoren opgevoerd (Rico, 2008); anderen zien deze als onderdelen van 'kwaliteit', respectievelijk 'tijd'. Een succesvol project leidt niet per definitie tot succes voor een organisatie (Ika, 2009). Serrador en Pinto (2015) beargumenteren daarom om de succesindicatoren niet te beperken tot alleen de traditionele succesindicatoren. Traditionele en Agile methoden laten zich onderling slecht vergelijken, aangezien traditionele methoden zijn geoptimaliseerd voor productiviteit, terwijl Agile-methoden geoptimaliseerd zijn voor wendbaarheid (Rico, 2008). Dat suggereert dat wendbaarheid de vijfde indicator is om het succes van Agile softwareontwikkeling te kunnen beoordelen. In dit onderzoek is wendbaarheid gedefinieerd als de mate van impact die het aanpassen van functionele specificaties gedurende de looptijd van het software-ontwikkeltraject teweegbrengt, waarbij geldt dat met een hoge wendbaarheid de impact laag zal zijn en vice versa.

Succesindicator	Omschrijving
Scope	De mate waarin de functionaliteit van de software aansluit bij de (expliciete en impliciete) eisen en wensen van klant, business en gebruikers?
Kwaliteit	De mate waarin goed werkende en geïntegreerde software wordt opgeleverd, waaronder: - Doet de software wat het volgens de functionele eisen moet doen? - Verstoren softwarefouten niet het gebruik van de software? - Is de software in de toekomst eenvoudig aanpasbaar? - Is de security van de software in lijn met de eisen die gesteld worden?
Tijd	De mate waarin de gevraagde functionaliteiten op tijd wordt opgeleverd.
Kosten	De mate waarin resources (geld, tijd, etc.) worden aangewend voor het software-ontwikkeltraject.
Wendbaarheid	De mate van impact die het aanpassen van functionele specificaties gedurende de looptijd van het software-ontwikkeltraject teweegbrengt, waarbij geldt dat bij een hoge wendbaarheid de impact laag zal zijn en vice versa.

Figuur 9: Succesindicatoren voor Agile software-ontwikkeltrajecten

2.3.3 *EFFECTEN VAN AGILE OP HET SUCCES VAN SOFTWARE-ONTWIKKELTRAJECTEN*

In de afgelopen jaren is er veel empirisch onderzoek gedaan naar de toegevoegde waarde van Agile ontwikkelmethoden (Begel & Nagappan, 2007; Dybå & Dingsøyr, 2008; Laanti, Salo, & Abrahamsson, 2011; Petersen & Wohlin, 2009, 2010; Ramesh & Devadasan, 2007; Vijayasarathy & Turk, 2008; Solinski & Petersen, 2016). Hoewel deze onderzoeken sterk verschillen qua opzet en omvang, zijn de uitkomsten redelijk consistent en in grote lijnen vergelijkbaar. Effecten van Continuous Delivery die in de literatuur vaak genoemd zijn, waaronder medewerkerstevredenheid, samenwerking, goede communicatie en kennisopbouw worden in dit onderzoek niet gezien als aparte indicatoren, maar dragen bij aan de hiervoor omschreven succesindicatoren. Klanttevredenheid wordt eveneens frequent genoemd in de literatuur, maar in dit onderzoek gezien als een gevolg van de performance op de succesindicatoren. Tot slot worden bij Agile worden *(lead-)users* meer betrokken bij de softwareontwikkeling. Voor dit soort '*utilitarian'* innovaties geldt dat *co-creatie* tot betere producten leidt en dus (Candi, van den Ende, & Gemser, 2016) een positief effect heeft 'scope' en 'kwaliteit'. De relevante effecten ten opzichte van traditionele ontwikkelmethoden worden hieronder als succesindicatoren voor Agile software-ontwikkeltrajecten samengevat.

Succesindicatoren	Effect van Agile op softwareontwikkeling
Scope	Opgeleverde software voldoet meer aan de wensen van de klant en sluit beter aan bij de business/gebruikers.
Kwaliteit	Software wordt opgeleverd met minder softwarefouten, een hogere betrouwbaarheid en meer gebruiksgemak.
Tijd	Er worden meer regels code in minder tijd geschreven. Dit resulteert echter niet altijd in een snellere *time-to-market* omdat de *last-mile* niet expliciet een onderdeel is bij Agile.
Kosten	Kosten voor ontwikkeling en onderhoud van de software.
Wendbaarheid	Minder impact bij aanpassen van de functionele eisen gedurende het software-ontwikkeltraject.

Figuur 10: Toegevoegde waarde van Agile op een software-ontwikkeltraject

Met uitzondering van de indicator 'tijd' is de toegevoegde waarde op al deze indicatoren significant (Solinski & Petersen, 2016). Laanti (2011) en Vijayasarathy & Turk (2008) zien 'tijd' als één van de grootste succesfactoren, terwijl Solinski & Petersen (2016) 'tijd' juist de minst significante bijdrage ziet leveren. Deze verschillen worden veroorzaakt doordat het onderzoek van Laanti en Vijayasarathy & Turk zich beperkt tot de ontwikkelsnelheid (dus de doorlooptijd tot aan de oplevering) en de *time-to-market* (dus beschikbaar voor de gebruiker) niet in het onderzoek meeneemt. Het verschil hiertussen is de in paragraaf 2.2.4 eerder beschreven '*last mile'*, die vaak tijdrovend is (Humble & Farley, 2010; Krusche & Alperowitz, 2014; Neely & Stolt, 2013).

2.3.4 TOEGEVOEGDE EFFECTEN VAN CONTINUOUS DELIVERY OP AGILE SOFTWARE-ONTWIKKELTRAJECTEN

Omdat Continuous Delivery een extensie is op Agile, worden in de literatuur de effecten van Continuous Delivery vaak ten opzichte van Agile beschreven. Deze worden in de tabel hieronder samengevat.

Succesindicatoren	Effect van Continuous Delivery op Agile softwareontwikkeling
Scope	Opgeleverde software sluit nog beter aan bij de wensen van de klant doordat daadwerkelijk gebruik van de in productie kan worden meegenomen. Het gaat hierbij niet alleen om feedback van productiegebruikers, maar ook om objectieve *real usage data*.
Kwaliteit	Minder *refactoring* noodzakelijk doordat integratie van de software met de klant (productie)omgeving continu plaatsvindt. Releases zijn betrouwbaarder. Softwarefouten in productie kunnen sneller worden hersteld.
Tijd	Niet alleen hogere ontwikkelsnelheid, maar ook een snellere *time-to-market*.
Kosten	Ontwikkelingskosten worden verder verlaagd, lagere kosten in de beheerfase.
Wendbaarheid	Nog wendbaarder doordat aanpassingen nu ook continu kunnen worden doorgevoerd in productie en aan de gebruiker worden ontsloten.

Figuur 11: Toegevoegde waarde van Continuous Delivery op Agile software-ontwikkeltrajecten

2.4 ONTWIKKELING VAN EEN CONTINUOUS DELIVERY SUCCES-MODEL

Dit onderzoek bestudeert het effect van de verschillende elementen van Continuous Delivery op de diverse succesfactoren van Agile softwareontwikkeling. Hoewel nadrukkelijk alle uitkomsten worden opengehouden, lijken zich vanuit de literatuur al wat contouren af te tekenen voor wat betreft de matrix die hierdoor ontstaat: het Continuous Delivery succesmodel. In dit model lijkt er sprake te zijn van een tweetal effecten: no-regret effecten (een positief effect en in het succesmodel weergegeven als '+') en effecten die - afhankelijk van bepaalde aspecten van het software-ontwikkeltraject - zowel negatief als positief kunnen zijn (weergegeven als -/+). Beide verwachte effecten worden in sub paragraaf 2.4.1 respectievelijk 2.4.2 toegelicht, een nadere detaillering van de betreffende aspecten van een agile software-ontwikkeltraject volgt in 2.4.3.

2.4.1 POSITIEVE EFFECTEN (NO REGRET)

De elementen 'procesuitbreiding' en 'organisatorische factoren' lijken in vergelijking met het element procesautomatisering om een beperkte investering te vragen, terwijl een meer geïntegreerde aanpak van het softwareontwikkelings- en -releaseproces en de bijbehorende organisatorische veranderingen al snel ten goede zullen komen aan alle succesindicatoren. Verwacht wordt dan ook dat activiteiten op het gebied van deze twee elementen een positief effect hebben op het succes van het Agile project. Iets vergelijkbaars geldt voor het element 'procesautomatisering'. De verwachting is dat de inspanningen op deze drie elementen al snel positief zullen bijdragen aan de succesindicatoren 'kwaliteit', 'scope', en 'wendbaarheid'. Dat resulteert in een eerste schets van het Continuous Delivery Succesmodel.

Continuous Delivery element	Scope	Kwaliteit	Tijd	Kosten	Wendbaarheid
Procesuitbreiding	+	+	+	+	+
Procesautomatisering	+	+	**?**	**?**	+
Sociaal/organisatorische factoren	+	+	+	+	+

+ positief effect (*no regret*) ? Afhankelijk van het software-ontwikkeltraject

Figuur 12: eerste aanzet Continuous Delivery succesmodel

2.4.2 SOFTWARE-ONTWIKKELTRAJECT-AFHANKELIJKE EFFECTEN

Bij het element 'procesautomatisering' is de verwachting dat deze zowel negatief als positief kunnen bijdragen aan de succesindicatoren 'kosten' en 'tijd'. Oorzaak is dat dit element initieel hoge (automatiserings)inspanningen vragen, en dus initieel een negatief effect zullen hebben op kosten en doorlooptijd, dat mogelijk positief wordt naarmate deze elementen frequenter worden ingezet.

Continuous Delivery element	Scope	Kwaliteit	Tijd	Kosten	Wendbaarheid
Procesuitbreiding	+	+	+	+	+
Procesautomatisering	+	+	**- /+**	**- /+**	+
Sociaal/organisatorische factoren	+	+	+	+	+

+ positief effect (*no regret*) -/+ effect afhankelijk van het software-ontwikkeltraject

Figuur 13: Concept Continuous Delivery succesmodel

2.4.3 BEÏNVLOEDENDE ASPECTEN VAN SOFTWARE-ONTWIKKELTRAJECTEN

Zoals hierboven beschreven lijken sommige elementen van Continuous Delivery zowel een negatief als positief effect te kunnen hebben op de succesindicatoren 'tijd' en 'kosten' (in het model weergegeven als -/+), afhankelijk van een aantal aspecten van het software-ontwikkeltraject. Deze aspecten zijn 'mate van hergebruik van Continuous Delivery', "wendbaarheid IT-organisatie' en 'noodzakelijke *time-to-market*'. Deze worden hieronder verder toegelicht.

MATE VAN HERGEBRUIK VAN CONTINUOUS DELIVERY

Hoe meer releases er in de toekomst verwacht worden, hoe kleiner de gemiddelde investering in termen van tijd en kosten per release en dus hoe eerder een investeringsbeslissing rondom de inzet van Continuous Delivery loont.

WENDBAARHEID IT-ORGANISATIE

Het lijkt aannemelijk dat investeringen voor een Continuous Delivery-implementatie lager zullen zijn bij een wendbare IT-organisatie dan bij een starre. Indien er al (delen van) Continuous Delivery binnen de organisatie zijn geoperationaliseerd dan zullen de investeringskosten naar verwachting eveneens lager uitvallen. Hoe lager de investeringskosten hoe sneller Continuous Delivery een positief effect zal hebben op de succesindicator 'prijs' en 'tijd'.

NOODZAKELIJKE *TIME-TO-MARKET*

Continuous Delivery mikt op het verkorten van de *time-to-market*. In sommige markten is een korte *time-to-market* een belangrijk concurrentiecriterium en ontstaat zo dus sneller een positieve business case voor een Continuous Delivery implementatie. Hoewel voor missiekritieke software de klassieke waterval-ontwikkelmethoden preferabel zijn (Salah et al., 2014) is dat niet altijd mogelijk, bijvoorbeeld indien de veranderingsbehoefte continu hoog is in combinatie met een vereiste korte *time-to-market*. De risico's die dit met zich meebrengt kunnen gedeeltelijk door Continuous Delivery worden gemitigeerd, aangezien oplossingen voor softwarefouten hiermee sneller kunnen worden gerealiseerd en gereleased.

Het kwalitatief onderzoek beoogt het in deze paragraaf geïntroduceerde Continuous Delivery succesmodel empirisch te exploreren. De methodologische opzet van dit onderzoek wordt beschreven in hoofdstuk 3. In hoofdstuk 4 worden de resultaten ervan gepresenteerd.

3 ONDERZOEKSMETHODE

Doel van het onderzoek is om het effect is van de inzet van Continuous Delivery op het succes van Agile software-ontwikkeltrajecten te achterhalen. Het conceptueel ontwerp is reeds in paragraaf 1.3 beschreven. Om deze te onderzoeken is een inductief kwalitatieve vergelijkende (multiple) case-study uitgevoerd. De keuze voor dit type onderzoek wordt bij bespreking van het onderzoeks-ontwerp in paragraaf 3.1 toegelicht. In paragraaf 3.2 wordt de methode van dataverzameling beschreven. In paragraaf 3.3 wordt het hoofdstuk afgesloten met een beschrijving van de data-analyse.

3.1 ONDERZOEKSONTWERP

Bij het opzetten van een onderzoek zijn verschillende ontwerpkeuzes te maken, waarvan de belangrijkste in deze paragraaf worden toegelicht. Een veelvuldig toegepast onderscheid betreft het kennis leidend belang en is die tussen fundamenteel (theoretisch) en toegepast (praktisch) onderzoek (Verschuren & Doorewaard, 2007). Aangezien dit onderzoek vooral invulling geeft aan hiaten in de wetenschappelijke literatuur kan worden gesteld dat deze voornamelijk theoretisch en theorie-ontwikkelend van aard is. Om dezelfde reden kan het onderzoek ook worden gekwalificeerd als exploratief en inductief. Een andere keuze die toelichting behoeft betreft de onderzoekbenadering. Primair worden kwalitatieve methoden gebruikt om de probleemstelling te duiden en mogelijke oplossingsrichtingen te toetsen.

Zoals gebruikelijk in de wetenschap is dit onderzoek voorafgegaan door een literatuuronderzoek, waarin de huidige stand van zaken in de relevante wetenschap inventariseert, als ook de hiaten. Daarnaast zijn andere onderzoeksstrategieën overwogen, maar vanwege de slechte toepasbaarheid niet geselecteerd. Zo is bijvoorbeeld vanwege het ontbreken van een gemeenschappelijk referentiekader rondom de elementen van Continuous Delivery een onderzoeksstrategie gebaseerd op surveys afgevallen. En zo ook de onderzoekstrategie 'experiment' omdat deze praktisch slecht uitvoerbaar lijkt aangezien de context van Agile-projecten altijd aan verandering onderhevig is en deze dus altijd ongewild het experiment beïnvloedt.

Software engineering is een multidisciplinair onderzoeksveld, dus om te begrijpen hoe developer software bouwen moet niet alleen gekeken worden naar de tools en de processen die worden gebruikt, maar ook naar cognitieve en sociale zaken die hen omringen. Vanwege het belang van de menselijke factor in de software-ontwikkeltrajecten (Nygaard, 1990) worden veel van de onderzoeksmethoden die gebruikt worden in disciplines die het menselijk gedrag onderzoeken, zoals psychologie en sociologie, ook gebruikt in onderzoek naar softwareontwikkeling (Easterbrook, Singer, Storey, & Damian, 2008). Casestudies vormen "een krachtige en flexibele empirische methode. Ze worden voornamelijk gebruikt voor exploratief onderzoek om een verschijnsel te begrijpen en te verklaren of een theorie te verschijnsel een theorie te ontwikkelen." (D. E. Perry, Sim, & Easterbrook,

2004, p. 1045). Voor onderzoek naar software engineering zijn casestudies als methode zeer geschikt, in het bijzonder bij onderzoek naar nieuwe technieken, werkwijzen, functies en processen in het software ontwikkelingsdomein (D. E. Perry et al., 2004).

De basis van dit onderzoek vormt een vergelijkende meervoudige casestudy bestaande uit een zevental cases binnen vier organisaties. Vergelijkende casestudies kunnen goed worden ingezet bij onderzoek rondom innovaties (Yin, 2013). Doordat cases met elkaar kunnen worden vergeleken zijn vergelijkende casestudies robuuster dan enkelvoudige casestudies. Daarnaast kunnen de resultaten beter worden gegeneraliseerd. (Yin, 2013). Exploratieve casestudies zijn geschikt "om te analyseren wat gemeenschappelijk is of verschilt tussen cases die bepaalde criteria delen" (Easterbrook et al., 2008, p. 288). In dit onderzoek is het gedeelde criteria het gebruik van Agile in software-ontwikkeltrajecten. Een onderbouwde selectie van te onderzoeken cases is in casestudies van essentieel belang. Bij de selectie van cases in dit onderzoek zijn de volgende inclusiecriteria gehanteerd:

- De organisatie is gevestigd in Nederland en qua omvang groot (> 500 FTE)
- De organisatie heeft een aanzienlijke afhankelijkheid van software (is 'software gedreven')
- Software wordt ontwikkeld op basis van Agile ontwikkelmethoden
- Per organisatie zijn minimaal twee cases beschikbaar: mét en zonder Continuous Delivery.

De onderzoekseenheid betreft een software-ontwikkeltraject, waarbij bij de selectie gezocht is naar paren van cases mét en zonder Continuous Delivery (hierna vaak *Agile-only* genoemd) binnen dezelfde organisatie zodat beide perspectieven vergeleken kunnen worden in een vergelijkbare context. Om de generaliseerbaarheid van het onderzoek te vergroten is gekozen naar een sterke variatie van de organisaties. Een hogere variëteit vergroot de kans dat alle factoren die een rol spelen ook worde aangetoond. Binnen een organisatie zijn de cases echter met een minimum aan variatie geselecteerd waardoor externe factoren worden verminderd (Eisenhardt, 1989; Verschuren & Doorewaard, 2007). Met het aantal cases is geprobeerd een goede balans te vinden tussen breedte van het onderzoek en het detailniveau per case (Baker, Edwards, & Doidge, 2012; Voss, Tsikriktsis, & Frohlich, 2002). De casestudies zijn uitgevoerd in de periode van april tot en met juni 2017.

Organisatie	**Rijkswaterstaat**		**Centric**		**ING-bank**		**Coolblue**
Sector	overheid		IT-dienstverlening		financiële dienstverlening		retail
# medewerker	ca 8.700		ca 4.300		12.000+		1.500+
Afzetgebied	Nederland		Europe		Wereldwijd		Benelux
Case	**VCM**	**IVS Next**	**S4O**	**Storeworld**	**mijnING**	**CDaaS**	**CD-backoffice**
Continuous Delivery	*Agile-only*	ja	*Agile-only*	ja	*Agile-only*	ja	ja
# betrokken teams in case	1	2	2	2	1	2	2

Figuur 14: Overzicht uitgevoerde casestudies. Een uitgebreider overzicht is opgenomen in bijlage A.

3.2 METHODE VAN DATA VERZAMELEN

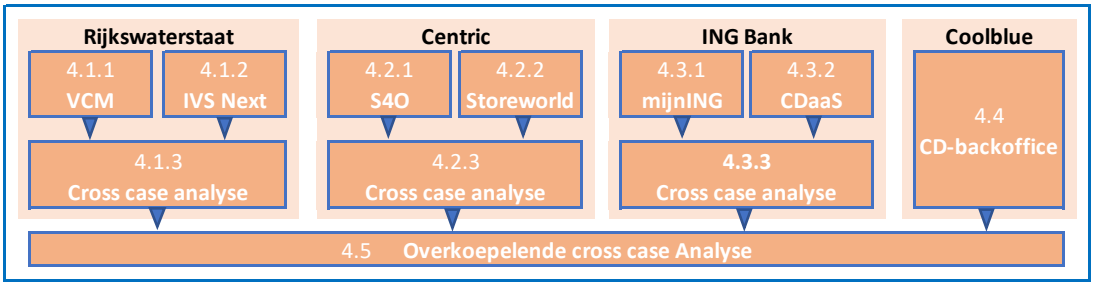
De zestien interviews vormen naast de literatuurstudie de basis van dit onderzoek. Met interviews kan dieper worden ingegaan op de achterliggende motivatie, meningen en behoeften (Easterby-Smith, Thorpe, & Jackson, 2012). Exploratief onderzoek vereist een open manier van het verzamelen van gegevens (Verschuren & Doorewaard, 2007). Toch is er in dit onderzoek gekozen voor semi-open interviews, omdat het doel van de interviews was om het vooropgestelde Continuous Delivery succesmodel te exploreren. Per case is gesproken met degene die verantwoordelijke is voor de realisatie van software. Deze bevindt zich meestal aan de IT-zijde van de organisatie: IT-manager, projectleider, applicatiemanager, etc. Deze interviews duurden ongeveer één uur. Onderwerpen waren onder andere de inzet van de verschillende elementen van Agile en (indien relevant) Continuous Delivery en de bijdrage daarvan aan het succes van het software-ontwikkeltraject. Daarnaast is om redenen van triangulatie de persoon die de voordelen van een succesvol software-ontwikkeltraject nuttigt geïnterviewd: het gaat hierbij om personen die de businesszijde vertegenwoordigen, zoals de product owner of unitmanager. In uitzonderlijke gevallen waren beide rollen bij één persoon belegd of was deze persoon niet beschikbaar voor interviews. In deze gevallen is op andere wijze getriangu-leerd. Alle interviews zijn – na goedkeuring door de geïnterviewde – digitaal opgenomen om een betere samenvatting op te kunnen stellen en nauwkeuriger te kunnen citeren. Vanwege privacy-redenen zijn geïnterviewden niet bij naam genoemd in het onderzoeksrapport. Een overzicht van de afgenomen interviews is opgenomen in bijlage B. In deze studie zijn van alle cases ook secondaire bronnen gebruikt. Denk hierbij aan jaarverslagen, presentaties, voortgangsrapportages blogs, etc. Tot slot is een bureaustudie uitgevoerd om secondaire data zoals rapportages, rapporten, interne enquêtes, presentaties en websites in de context van de organisatie en de case te interpreteren.

3.3 METHODE VAN DATA-ANALYSE

Met behulp van triangulatie-technieken is de informatie binnen de eigen context geïnterpreteerd. Dit is bereikt door het vergelijken van informatie uit verschillende bronnen. Bijvoorbeeld binnen bepaalde onderzoekstechnieken (bijvoorbeeld interviews met verschillende mensen), maar ook tussen onderzoekstechnieken (bijvoorbeeld interview en rapportage). De eerste analyse vindt plaats door van ieder interview een samenvatting te maken. Deze is vervolgens ter verificatie neergelegd bij de respondent. Daarna is een within-case analyse uitgevoerd om een diepgaand begrip te vormen over de volgordelijkheid en causaliteit van de case-data: hoe hebben in deze case de elementen van Continuous Delivery het succes van Agile-software-ontwikkeltrajecten beïnvloedt? Bij enkele complexe cases is een tijdlijn opgesteld die het onderzoeken van causale verbanden vergemakkelijkte. Per case is gekeken naar welke Continuous Delivery elementen zijn gebruikt en wat hierop het effect was op de succesindicatoren van het software-ontwikkeltraject. Zo zijn de patronen en verbanden in kaart gebracht. Bij de *Agile-only* cases is eveneens een analyse uitgevoerd over wat de effecten *zouden* zijn als er meer Continuous Delivery zouden zijn ingezet. Daarna is voor de cases binnen dezelfde organisatie een cross case analyse uitgevoerd en afsluitend een overkoepelende cross case analyse. Hierbij zijn de cases vergeleken om verschillen en overeenkomsten zichtbaar te maken en te verklaren.

4 ONDERZOEKSRESULTATEN & ANALYSE

In dit hoofdstuk worden de onderzoeksresultaten van he empirisch onderzoek gepresenteerd en geanalyseerd. In de navolgende drie paragrafen worden paarsgewijs per organisatie twee cases toegelicht, de onderzoeksresultaten gepresenteerd en in een cross case analyse paarsgewijs per organisatie vergeleken. Bij een vierde organisatie, Coolblue, was de validiteit van een van beide cases wegens gebrek aan triangulatiemogelijkheden onvoldoende om in het onderzoek op te kunnen nemen. In paragraaf 4.4 worden dan ook alleen de resultaten van de Coolblue-case mét Continuous Delivery beschreven en is een cross-case analyse dan ook niet aan de orde. In paragraaf 4.5 is een overkoepelende cross case analyse uitgevoerd waarin de resultaten van alle onderzochte organisaties tezamen worden geanalyseerd.



Figuur 15: Visuele weergave beschrijving en analyse onderzoeksresultaten

De onderzoeksresultaten van de zeven cases worden in scoretabellen samengevat. Deze tabellen geven enerzijds de mate aan waarin de drie elementen van Continuous Delivery zijn ingezet, en anderzijds het effect daarvan op het succes van het Agile software-ontwikkeltraject in termen van scope, kwaliteit, tijd, kosten en wendbaarheid. In de tabel wordt de mate waarin de elementen worden ingezet weergegeven met het aantal stippen op een schaal van nul tot en met drie, waarbij geldt hoe meer stippen hoe groter 'de mate'. Hiervoor is een coderingstabel gebruikt, welke in bijlage C is opgenomen. Het effect van het betreffende element op de succesindicatoren is in de tabel met een minteken (▪), vierkantje (□) of plusteken (✚) weergegeven voor respectievelijk een negatief, een niet-significant of een positief effect. Bij de cases zónder Continuous Delivery is tevens een scoretabel opgenomen, waarin geanalyseerd wordt wat de verwachte effecten zouden zijn indien Continuous Delivery wél zou zijn geïmplementeerd. Bij de cross case analyses is per organisatie een vergelijkbare scoretabel gebruikt als bij de cases. Naast de eerdergenoemde tekens is voor trajectafhankelijke effecten het teken '▪ / ✚' gebruikt. In de *overkoepelende* cross case analyse wordt gekeken naar de overeenkomsten en verschillen tussen de analyses van de vier organisaties.

4.1 ORGANISATIE A: RIJKSWATERSTAAT

Rijkswaterstaat is een agentschap van het Ministerie van Infrastructuur en Milieu en ontwikkelt en onderhoudt enerzijds het hoofd(vaar)wegennet en de hoofd(vaar)wegen voor vlot en veilig verkeer en anderzijds de hoofdwatersystemen voor een adequate bescherming tegen overstromingen. Binnen

Rijkswaterstaat zijn twee cases onderzocht: de Verkeercentrale van Morgen (VCM), een project waarvan de softwareontwikkeling *Agile-only* is uitgevoerd, en IVS Next, een project op basis van Continuous Delivery. Deze cases worden in de navolgende twee sub-paragrafen toegelicht. In de laatste sub-paragraaf worden de uitkomsten van de cross case analyse beschreven.

4.1.1 CASE A1: VERKEERCENTRALE VAN MORGEN (VCM)

De komst van de Tweede Maasvlakte leidt tot meer scheepvaartverkeer op de Nederlandse hoofdvaarwegen, die nu al tot de drukst bevaren binnenwateren van Europa behoren. Om deze groei in de toekomst veilig en efficiënt op te kunnen vangen wordt een nieuwe manier van scheepvaart verkeersmanagement beproefd. Het personeel op de diverse objecten zoals bruggen, sluizen en verkeerposten hebben een beter zicht op het scheepvaartverkeer dan de schipper en kunnen het verkeer dus vlotter en veiliger organiseren. Elk object heeft echter een beperkt werkgebied, en onderlinge afstemming tussen de objecten ontbreekt of is slechts zeer beperkt. Voor een vaarweg met diverse objecten is de verkeersafhandeling daarmee suboptimaal.

In het programma *Verkeerscentrale van Morgen* (VCM) is onderzocht of deze beperking is op te lossen door alle systemen, communicatie en bediening onder te brengen in één verkeerscentrale en van hieruit de scheepvaart over de twee hoofdtransportassen (Amsterdam-Antwerpen en Rotterdam-Duisburg) actief te begeleiden. De schipper verzendt elektronisch zijn vaarplan waarop Rijkswaterstaat een tijdindicatie afgeeft voor de brugopeningen en sluisschuttingen op de route. Op basis hiervan kan een schipper zijn snelheid aanpassen. Rijkswaterstaat kan op basis van de vaarplannen en de actuele posities beter het verkeer voorspellen en daarmee ook weer betere indicaties afgeven. Hierdoor worden de vaarwegen efficiënter gebruikt, kan de schipper brandstof besparen en krijgen de verladers een betere tijdindicatie wanneer een schip voor laden of lossen beschikbaar is.

Vanwege het hoog experimentele karakter van VCM is binnen Rijkswaterstaat gekozen voor een SCRUM/Agile aanpak voor het ontwikkelen van de benodigde software voor VCM. Agile was binnen Rijkswaterstaat nog relatief nieuw. Door Agile te werken kon sneller dan bij klassieke ontwikkelmethoden worden bijgestuurd, was de scope flexibel waardoor kansen benut konden worden en missers voortijdig werden beëindigd. Dit droeg bij aan de succesvolle softwareontwikkeling: efficiënter, betere kwaliteit, scope beter aansluitend op de business behoefte door de hoge mate van flexibiliteit, wat resulteerde in een hoge wendbaarheid en een hogere klanttevredenheid. De succesindicator *kosten* laat zich lastig beoordelen: Rijkswaterstaat heeft aanzienlijk meer tijd moeten steken in het project dan bij een klassiek project, waarbij veel meer is uitbesteed.

Er zijn bij dit project geen CD-elementen ingezet, met één uitzondering. Bij een tussentijdse oplevering van de software bleek dat de integratie tussen de opgeleverde software en de systemen bij Rijkswaterstaat heel veel issues opleverde. Dit leidde tot een uitloop van tien weken en om verdere vertraging te voorkomen werd een DevOps-achtige oplossing geforceerd door beheerders en developer wekelijks een dag bij elkaar te zetten om aan deze integratie issues te werken. Hoewel er nog sterk in termen van

'issue ligt niet in mijn domein' gedacht werd, zijn uiteindelijk deze issues wel vijf voor twaalf opgelost. Zonder deze DevOps-achtige aanpak was volgens de geïnterviewden de deadline zeker niet gehaald en hadden er *"tonnen aan Europese subsidies moeten worden terugbetaald"*.

VCM-score	**Inzet van element in de case**	**Effect van element op succesindicator**				
Continuous Delivery element		Scope	Kwaliteit	Tijd	Kosten	Wend-baarheid
Procesuitbreiding	○○○	□	□	□	□	□
Procesautomatisering	○○○	□	□	□	□	□
Sociaal/organisatorische factoren	●○○	□	□	✚[1]	✚[1]	□

[1] Implementatie van een integratie dag met *Development* en *Operations* had een positief effect op 'tijd' en 'kosten'.

Figuur 16: Samenvatting resultaten casestudy A1: VCM (Rijkswaterstaat)

Een aantal respondenten is ook betrokken geweest bij de tweede case waarbij wél Continuous Delivery is ingezet en kan dus een goede inschatting maken van de effecten van Continuous Delivery elementen als deze zouden zijn ingezet bij VCM. Wanneer de '*last mile*' integraal onderdeel zou zijn geweest van het softwareontwikkelproces dan is het aannemelijk dat dit een positief effect zou hebben op alle succesindicatoren: met het element procesuitbreiding wordt een kortere en meer integrale feedbackloop gerealiseerd, waardoor frequenter en makkelijker kan worden bijgestuurd (scope, wendbaarheid) en integratie wordt verbeterd (kwaliteit, tijd, kosten, wendbaarheid). Voor de sociaal/organisatorische factoren geldt iets vergelijkbaars: een cultuur van frequent releasen en samenwerken heeft ook een positief effect op alle succesindicatoren vanwege de betere integratie en het verkorten van de feedbackloop. De effecten van procesautomatisering ondersteunen eveneens de kortere feedbackloop en een beter integratie, maar de investeringen in tijd en geld voor implementatie van een Continuous Delivery Pipeline zouden aanzienlijk zijn, waardoor het project niet meer op tijd zou zijn gerealiseerd en duurder zou zijn uitgevallen.

VCM: *wat-als-CD* score?	**Hypothetische inzet element**	**Effect van element op succesindicator**				
Continuous Delivery element		Scope	Kwaliteit	Tijd	Kosten	Wend-baarheid
Procesuitbreiding	✚	✚	✚	✚	✚	✚
Procesautomatisering	✚	✚	✚	-[1]	-[1]	✚
Sociaal/organisatorische factoren	✚	✚	✚	✚	✚	✚

[1] Gezien de relatief kleine omvang van het project en de korte doorlooptijd zou procesautomatisering onrendabel en vertragend zijn geweest.

Figuur 17: Verwachte score wanneer Continuous Delivery zou zijn ingezet bij VCM (Rijkswaterstaat)

4.1.2 CASE A2: INFORMATIE EN VOLGSYSTEEM VOOR DE SCHEEPVAART (IVS NEXT)

Rijkswaterstaat heeft als missie onder andere om vlot en veilig verkeer op Rijksvaarwegen te realiseren. Hiervoor wordt het missiekritieke systeem Informatie en Volgsysteem voor de Scheepvaart (IVS90) ingezet, wat inmiddels sterk verouderd en aan vervanging toe is (Baronner & Burgt, 2017). In 2012 startte de vervanging van het systeem conform het klassieke watervalmodel maar werd in 2014 halverwege afgebroken omdat deze niet tot de gewenste resultaten leidde: ontwikkeling was kostbaarder dan verwacht, het project liep inmiddels flink achter op schema en de opgeleverde software bleek na testen niet naar te functioneren. Om de missie van Rijkswaterstaat niet in gevaar te brengen diende het verouderde IVS90 echter op korte termijn te worden vervangen. Het management heeft dan ook besloten om de klassieke watervalaanpak te vervangen door een Agile-aanpak. *Bottom-up* is door IT-*Operations* gekozen om dit met behulp van Continuous Delivery in te richten, daar een reguliere release van IVS90 van oplevering van de software tot het opleveren in productie *best case* zes weken duurt, maar meestal aanzienlijk langer in verband met heroplevingen voor benodigde bugfixes.

In lijn met het Rijkswaterstaatbeleid om softwareontwikkeling uit te besteden is ook voor dit project Agile-ontwikkelcapaciteit van derden ingekocht. Dit bemoeilijkte de implementatie van de procesuitbreiding aanzienlijk, daar in het contract en bijbehorende sturingsmechanismen nog uit werd gegaan van de klassieke verdeling waarbij de ontwikkelpartij de software aanlevert en Rijkswaterstaat deze test, accepteert en deployd. Na meerdere integratie-issues werd echter duidelijk dat procesuitbreiding voordelen had en werd de *defintion of done* aangepast van oplevering in de demo-omgeving van de leverancier naar oplevering in de RWS-omgeving. Deze wijziging kwam ook het Continuous Delivery element sociaal/organisatorische factoren ten goede: hoewel de samenwerkingsbereidheid op de werkvloer al vanaf het begin af aan hoog was, werd deze verder geïntensiveerd na dit besluit. De gebruikelijke ergernis tussen Dev (leverancier) en Ops (Rijkswaterstaat) is door de open en intensieve samenwerking, snelle feedback en snelle resultaten vervangen door enthousiasme en verhoogd wederzijds vertrouwen. Het afstemmen en implementeren van de integrale procesautomatisering (Continuous Delivery pipeline) ging voorspoedig: de softwareleverancier had al Continuous Integration ingericht, en voor wat betreft de integratie hiervan met Rijkswaterstaat en over de automatisering van deployments en tests ontstond al snel een gezamenlijk beeld.

De vier respondenten, waaronder de product owner en de projectmanager hadden al ervaring met Agile ontwikkelmethoden. Er bestaat een collectieve eensgezindheid over de toegevoegde positieve effecten van Continuous Delivery op het Agile projectsucces in het algemeen en de succesindicatoren kwaliteit, scope, tijd en wendbaarheid in het bijzonder. Kortere feedbackloop en een aanzienlijk betere integratie tussen DEV en OPS waren de voornaamste pluspunten. Gezien de hoge investeringen voor procesautomatisering in tijd en geld heeft toepassing van Continuous Delivery in de projectfase slechts een beperkte kostenbesparing teweeggebracht. Omdat deze investeringen al gedaan zijn wordt verwacht dat dit effect groter zal zijn bij het verder door ontwikkelen van IVS Next in de verdere levenscyclus.

IVS Next: score	Inzet van element in de case	Effect van element op succesindicator				
Continuous Delivery element		Scope	Kwaliteit	Tijd	Kosten	Wend-baarheid
Procesuitbreiding	●●●	✚	✚	✚	✚	✚
Procesautomatisering	●●○	✚	✚	✚	✚	✚
Sociaal/organisatorische factoren	●●○	✚	✚	✚	✚	✚

Figuur 18: Samenvatting resultaten casestudy A2: IVS Next (Rijkswaterstaat)

4.1.3 CROSS CASE ANALYSE CASES RIJKSWATERSTAAT

In deze cross case analyse worden de cases van VCM en IVS Next onderling vergeleken, om zodoende de effecten van Continuous Delivery software-ontwikkeltrajecten bovenop Agile te kunnen analyseren. Beide cases laten zich onderling goed vergelijkbaar aangezien er veel overeenkomsten zijn: zelfde organisatie, zelfde domein (scheepvaart), gebruik van dezelfde Agile-elementen, vergelijkbare druk om te leveren en veel dezelfde spelers/ afdelingen. Daarnaast werd bij beide cases sterk geëxperimenteerd: bij de start van VCM was er nog nauwelijks ervaring met Agile en Continuous Delivery was niet alleen voor IVS Next nieuw, maar voor Rijkswaterstaat als geheel.

Een factor die afwijkt is dat beide cases qua periode op elkaar aansluiten en dus niet gelijktijdig zijn. Omdat veel medewerkers van IVS Next ook bij VCM betrokken waren zou de opgedane Agile-ervaring ook ten grondslag kunnen liggen aan de positievere resultaten van de softwareontwikkeling bij IVS Next. In de interviews is hiernaar nadrukkelijk gevraagd, en juist omdat een aantal respondenten in beide trajecten hebben geparticipeerd kon goed worden doorgevraagd naar de onderliggende redenen. Hieruit blijkt dat de invloed van het tijdsverschil tussen beide cases geen significante invloed heeft op de uitkomsten. De vertraging bij VCM is niet door een tekort aan Agile-ervaring veroorzaakt, maar door integratie-issues tijdens de 'de *last mile*', terwijl het succes van IVS Next juist vergroot is door inzet van Continuous Delivery, waardoor de impact van integratie-issues aanzienlijk is verkleind. Bij IVS Next zijn er nog steeds uitdagingen m.b.t. de integratie, maar deze worden nu 1.) vroegtijdig gedetecteerd, 2) vanuit het reguliere proces opgelost en dat 3) in goede samenwerking tussen Dev en Ops.

Wanneer de *wat-als-Continuous Delivery*-analyse van VCM worden vergeleken met de analyse van IVS Next dan blijkt dat - op twee uitzonderingen na - alle Continuous Delivery elementen een positieve impact hebben op alle succesindicatoren. Vanwege de hoge investeringen van de procesautomatisering zijn deze pas rendabel wanneer deze intensief worden gebruikt, bijvoorbeeld bij doorontwikkeling in de verdere levenscyclus of andere software-ontwikkeltrajecten. Daarnaast kost implementatie van een Continuous Delivery pipeline doorlooptijd en kan deze bij een kortlopend project als VCM een negatief effect hebben op een tijdige oplevering. Interessant om te melden is dat als VCM in tijd zou zijn gerealiseerd na IVS Next de *wat-als-Continuous Delivery* op deze twee indicatoren positiever had uitgepakt, aangezien dan geen eigen pipeline gebouwd zou hoeven te worden, maar die van IVS Next uitgebreid had kunnen worden.

Cross case analyse Rijkswaterstaat	Effect van element op succesindicator				
Continuous Delivery element	Scope	Kwaliteit	Tijd	Kosten	Wendbaarheid
Procesuitbreiding	+	+	+	+	+
Procesautomatisering	+	+	- / +[1]	- / +[1]	+
Sociaal/organisatorische factoren	+	+	+	+	+

[1] Effect is afhankelijk van de omvang, respectievelijk de doorlooptijd van het ontwikkeltraject

Figuur 19: Samenvatting cross-case analyse Rijkswaterstaat

4.2 ORGANISATIE B: CENTRIC

Centric levert met zo'n 5000 medewerkers IT- en financiële dienstverlening aan de voornamelijk grotere organisaties. Het is een Nederlands bedrijf, maar heeft vestigingen in België, Duitsland, Frankrijk, Luxemburg, Nederland, Noorwegen, Roemenië, Zweden en Zwitserland. Binnen Centric zijn twee cases onderzocht: Suite4Omgevingsdienten waarvan de softwareontwikkeling voornamelijk zonder Continuous Delivery wordt uitgevoerd en Storeworld, waarbij wél Continuous Delivery wordt ingezet. Deze cases worden in de navolgende twee sub-paragrafen toegelicht. In de laatste sub-paragraaf worden de uitkomsten van de cross case analyse beschreven.

4.2.1 CASE B1: SUITE4OMGEVINGSDIENSTEN (S4O)

Centric biedt lokale overheden verschillende softwarepakketten aan om informatie te beheren en uit te wisselen. Enerzijds met burgers, anderzijds met andere instanties, waaronder overheden en bedrijven. Suite4Omgevingsdiensten is de software-oplossing die Centric gemeenten biedt het ondersteunen van alle werkzaamheden rondom ruimtelijke ordening en naastgelegen zake zoals monumentenbeheer en beperkingenregistratie. De softwareontwikkeling voor de plusminus 25 applicaties die onder deze suite vallen wordt sinds 2011 door twee gedistribueerde teams uitgevoerd volgens SCRUM/Agile-werkwijze. Het pakket was technologisch sterk verouderd en de eerste periode is dan ook veel tijd besteed aan het uitfaseren van legacy Delphi-code. Sindsdien is de klanttevredenheid, die in opdracht van de onafhankelijke gebruikersvereniging GV Centric is gemeten, gestegen (Infact Marktonderzoek, 2015). Hoewel de ontwikkelteams in sprints van twee weken software bouwen komt deze in de regel niet verder dan de testomgeving. Slechts twee keer per jaar (en bij spoedreparaties vaker) wordt deze software aangeboden als een (patch)release aan de gemeenten.

"Zonder Continuous Delivery kan er eigenlijk niet worden geSCRUMd, omdat er nu niet continu, maar slechts twee keer per jaar waarde wordt toegevoegd. Bovendien ontbreekt nu waardevolle feedback van gebruikers: als gebruikers merken dat de lijntjes kort worden en er wat met feedback gebeurd zijn ze sneller bereid om feedback te leveren en schiet de kwaliteit omhoog".

Binnen de Suite4Omgevingsdiensten is dan ook een korte periode geëxperimenteerd met een tweewekelijkse delivery van software aan de klant. Dit was echter behoorlijk arbeidsintensief en door het

gebrek aan (geautomatiseerde) testcapaciteit was de *trade-off* dat de kwaliteit achteruitging. Het verder automatiseren van de testen werd niet haalbaar geacht omdat door het monolithische karakter en de omvang van de software verdere testautomatisering niet rendabel zijn, aangezien er plannen waren om de applicatie om te bouwen naar een moderne architectuur gebaseerd op microservices om wendbaarder te kunnen zijn. Het experiment is dan ook gestopt. Sindsdien is 95-98% van de testen nog steeds handmatig. Een integratietest voorafgaand aan het vrijgeven van de software kost één à twee weken, waar met meerdere personen aan wordt gewerkt. Een groot deel van het samenstellen en deployen van een softwareversie is al wel geautomatiseerd en kan in een dagdeel worden gerealiseerd. Overigens zaten de gemeenten niet te wachten op een frequente oplevering. Zonder deploymentautomatisering is het releasen van nieuw opgeleverde Centric-software in de omgeving van een gemeente een kostbare en bij tijden risicovol proces, waarvan de verantwoordelijkheid en kosten bij de gemeenten liggen.

S4O: score **Continuous Delivery element**	**Inzet van element in de case**	**Effect van element op succesindicator**				
		Scope	Kwaliteit	Tijd	Kosten	Wend-baarheid
Procesuitbreiding	○○○	□	□	□	□	□
Procesautomatisering	●○○	□	✚[2]	✚[2]	✚[2]	□
Sociaal/organisatorische factoren[1]	●○○	□	□	✚[3]	✚[3]	□

[1] Cultuur van frequent releasen was slechts zeer tijdelijk en daardoor niet meegenomen in deze score.
[2] Gedeeltelijke deploymentautomatisering heeft de kwaliteit iets verhoog doordat tussentijds (handmatig) testen mogelijk is geworden. Bovendien heeft het de kosten verlaagd en de kans op tijdige levering verhoogd.
[3] Aandacht voor een betere samenwerking tussen *Development* en Quality Assurance heeft een positief effect.

Figuur 20: Samenvatting resultaten casestudy B1: Suite4Omgevingsdiensten (Centric)

"Continuous Integration is met onze huidige architectuur nog geen optie. Voor de software die we al onder nieuwe architectuur hebben gebracht zou het wel kunnen maar omdat die services moeten integreren met opleveringen van onze 'oude' productlijn halen we ons daar heel veel gedoe mee op de hals. Voorlopig hebben we gekozen om hiermee nog te wachten."

Sinds 2015 wordt gewerkt om deze suite als Software-as-a-Service (SaaS) aan te kunnen bieden. Hierbij wordt de software gehost en beheerd in de rekencentra van Centric in plaats van bij, respectievelijk door de lokale gemeenten. Gemeenten worden hiermee ontlast en Centric kan hiermee de beheercomplexiteit van bij elke gemeente verschillende implementaties verlagen. Doel is om de software dan ook middels Continuous Delivery te ontwikkelen. Verwachting is dat met Continuous Delivery de *time-to-market* wordt verlaagd en door onder andere de frequente releases en verregaande automatisering de kwaliteit verder wordt verhoogd. Daarnaast verwacht Centric een besparing te kunnen realiseren door het verminderen van de administratieve overhead en het minimaliseren van integratie-issues. Hierdoor denkt Centric wendbaarder te zijn en door de hogere kwaliteit tegen lagere kosten meer klantwaarde te kunnen bieden. Respondenten zien een architectuur op basis van micro-services echter als een noodzakelijke randvoorwaarde voor Continuous Delivery. De huidige software is echter aan het einde van de levenscyclus; en om de bestaande software om te schrijven naar een service-georiënteerde architectuur

zou een desinvestering zijn en jaren duren. Het is de verwachting dat bij een volledige vervanging van de software het effect van de procesautomatisering dan wél op de indicatoren kosten en tijd positief zullen bijdragen.

S4O: *Wat-als-CD* score?	Hypothetische inzet element	Effect van element op succesindicator				
Continuous Delivery element		Scope	Kwaliteit	Tijd	Kosten	Wendbaarheid
Procesuitbreiding	+	+	+	+	+	+
Procesautomatisering	+	+	+	- / +[1]	- / +[1]	+
Sociaal/organisatorische factoren	+	+	+	+	+	+

[1] Effect hangt af of de benodigde architecturale aanpassingen nu of bij een geplande vervanging van het systeem worden doorgevoerd. In het eerste geval zal gezien de korte terugverdientijd en de lange doorlooptijd de effecten negatief zijn.

Figuur 21: Verwachte score wanneer Continuous Delivery zou zijn ingezet bij Suite4Omgevingsdiensten (Centric)

4.2.2 CASE B2: STOREWORLD

Centric's Storeworld is een suite van retail-applicaties en bestaat onder andere uit kassa- en back officesystemen en is geschikt voor gegevensuitwisseling voorraad- en bestelsystemen. De Storeworld suite kan zowel bij klanten als bij Centric operationeel zijn. Rond 2010 is gestart met een SCRUM/Agile aanpak voor een van de ontwikkelteams, later volgde een tweede. Vanaf 2015 werken de teams met Continuous Delivery en mede daardoor kon het aantal SCRUM-teams worden vergroot naar de huidige zeven. *Operations* is geen onderdeel van deze teams, maar er wordt zwaar ingezet op een nauwe samengewerkt tussen Dev en Ops, waardoor de klassieke muren tussen beide onderdelen sterk zijn verlaagd. In totaal werken zo'n vijftig mensen aan de Storeworld suite, verdeeld over drie landen en zeven teams.

Alle softwareaanpassingen aan Storeworld gaan via één Continuous Delivery-pipeline. Continuous Integration is operationeel en deployments zijn volledig geautomatiseerd, waarbij de 'eigenaar' van een QA-, acceptatie- of productie-omgeving kan beslissen wanneer een deployment daadwerkelijk plaatsvindt. Developer krijgen binnen minuten de uitslag van de unittest bij het inchecken van de software. In de nacht wordt de suite als geheel – wederom geautomatiseerd – gebuild, gedeployd in samenhang getest. De laatste testen voordat de zes-wekelijkse release wordt vrijgegeven voor productie zijn echter nog deels handmatig. Het realiseren voor een CD-organisatie met dito cultuur heeft behoorlijk wat aandacht gekost, mede doordat teams fysiek ver van elkaar verwijderd zijn en vaak niet dezelfde moedertaal spreken. Om toch een gezamenlijke team- en verantwoordelijkheidsgevoel te creëren wordt er veel gereisd. Daarnaast wordt de werkwijze van de teams zoveel mogelijk uniform gehouden.

Kwaliteit is door invoering van Continuous Delivery sterk verhoogd: "sinds de invoering van Continuous Delivery en met name de procesautomatisering hebben wij geen enkele release meer hoeven uit te stellen vanwege integratie-issues, terwijl dat in de jaren daarvoor wel regelmatig gebeurde". Deze automatisering is kostbaar, maar verdient zich met zo'n zestig deployments en testen per nacht al snel terug. Door de snellere feedback vanuit productie kunnen ook betere keuzes worden gemaakt voor wat

betreft scope en inhoud van de volgende sprint: "we zijn wendbaarder en dat heeft uiteraard ook een positief effect op de klanttevredenheid". Als belangrijkste element van Continuous Delivery wordt procesautomatisering genomineerd, aangezien deze ook als de *enabler* van de sociaal/culturele aspecten van Continuous Delivery wordt gezien.

"Continuous Delivery is eigenlijk een randvoorwaarde om met meerdere teams efficiënt en effectief te kunnen SCRUMmen. Ik zie eigenlijk alleen maar voordelen. Maar als ik toch iets negatiefs moet noemen dan is dat misschien dat CD ook stress lijkt te veroorzaken bij een aantal mensen. Enerzijds doordat we "fail fast" toepassen is het gemakkelijker te achterhalen wat er fout is (en vaak ook door wie specifiek). Ik denk niet dat we dan verwijtend optreden maar ik merk toch dat sommige mensen het als stresserend ervaring. Overwegend toch wat oudere collega (zonder echt te willen veralgemenen) voor wij het allemaal (ook technologisch) nu wat snel gaat."

Storeworld: score	**Inzet van element in de case**	**Effect van element op succesindicator**				
Continuous Delivery element		Scope	Kwaliteit	Tijd	Kosten	Wend-baarheid
Procesuitbreiding	●●●	✚	✚	✚	✚	✚
Procesautomatisering	●●●	✚	✚	✚	✚	✚
Sociaal/organisatorische factoren	●●●	✚	✚	✚	✚	✚

Figuur 22: Samenvatting resultaten case-study B.2: Storeworld (Centric)

4.2.3 CROSS CASE ANALYSE CASES CENTRIC

In deze cross case analyse worden de twee Centric-cases onderling vergeleken, om zodoende de effecten van *Agile-only* (bij S4O) en Continuous Delivery (bij Storeworld) te kunnen analyseren. Veel externe factoren zijn gelijk: de cases spelen in dezelfde organisatie, in beide cases wordt gebruik gemaakt van multiple-SCRUM-teams en in beide gevallen is geëxperimenteerd met Continuous Delivery, het zij met wisselend succes. Een belangrijke randvoorwaarde voor het succesvol toepassen van Continuous blijkt de architectuur te zijn. Een sterk monolithische architectuur is in sterke mate belemmerend om de voordelen van Continuous Integration en testautomatisering te kunnen benutten. De belemmering is bij echter van die mate dat Continuous Delivery daar alleen na implementatie van een nieuwe architectuur kan worden gerealiseerd.

Wanneer de *wat-als*-analyse van S4O wordt vergelijken met de analyse van Storeworld dan blijkt dat alle Continuous Delivery elementen een positieve impact hebben op alle succesindicatoren, met uitzondering van twee. De kosten en doorlooptijd van het aanpassen van de architectuur in een bestaande applicatie kunnen zo hoog zijn dat deze vele malen hoger zijn dan de besparingen die met Continuous Delivery kunnen worden gerealiseerd. Hoe korter de resterende levensduur, hoe onrendabeler de transitie naar Continuous Delivery. Dat resulteert in onderstaande samenvatting.

Cross case analyse Centric	Effect van element op succesindicator				
Continuous Delivery element	Scope	Kwaliteit	Tijd	Kosten	Wendbaarheid
Procesuitbreiding	+	+	+	+	+
Procesautomatisering	+	+	- / + [1]	- / + [1]	+
Sociaal/organisatorische factoren	+	+	+	+	+

[1] Effect van procesuitbreiding op kosten en tijd is afhankelijk van of de architectuur geschikt is voor Continuous Delivery.

Figuur 23: Samenvatting overkoepelende cross case analyse Centric

4.3 ORGANISATIE C: ING BANK

De ING Bank is een Nederlandse bank en onderdeel van de ING Groep. De bank levert in 40 landen financiële diensten zoals bankieren, verzekeren en beleggen. In Nederland zijn er meer dan 12000 mensen voor de bank werkzaam. Binnen de ING-bank zijn twee cases onderzocht: Vernieuwen homepage MijnING (aanvankelijk zonder Continuous Delivery) en Continuous Delivery as a Service (mét Continuous Delivery). Deze cases worden in de navolgende twee sub-paragrafen toegelicht. In de laatste subparagraaf worden de uitkomsten van de cross case analyse beschreven.

4.3.1 CASE C1: VERNIEUWEN HOMEPAGE MIJNING (MIJNING)

De internetapplicatie MijnING is een van de belangrijkste applicaties van ING. Klanten kunnen hier hun bankzaken over internet doen en voor marketing een belangrijk communicatiekanaal. Door de ontwikkelmethode en architectuur was de doorlooptijd van een release van MijnING met gemiddeld negen maanden niet in overeenstemming met de gewenste *time-to-market* van de interne opdrachtgevers, die nieuwe functionaliteiten veel sneller live wilde hebben. Vanuit de hoofdafdeling Internet is gestart met een SCRUM/Agile experiment: er werd een team gevormd van opdrachtgever, architect, analist, ontwikkelaars, testers, en acceptanten uit verschillende subafdelingen waaronder *Operations* (binnen ING System Management genoemd) die gezamenlijk de nieuwe homepage van mijn MijnING moest realiseren. Na afloop van de vierde sprint levert het team al een werkende en geaccepteerde homepage op het werkstation van een van de ontwikkelaars: het bewijs dat er in een klein aantal sprints behoorlijk veel van de gewenste functionaliteit kan worden ontwikkeld. Maar de beheerprocessen rondom de '*last mile*' waren niet ontworpen om binnen een aantal dagen te kunnen releasen en met het verkorten van de release cyclus naar drie weken in plaats van negen maanden steeg het aantal releases schrikbarend, waarvoor geen release capaciteit beschikbaar was.

"Aan de voorkant hadden we dus met Agile wat opgelost, maar daarmee kregen we de problemen meer achter in de keten."

Na bestudering van het Continuous Delivery-concept besloot het management een aantal medewerkers vrij te spelen om te onderzoeken of deze methode mogelijk de oplossing was voor de problemen die zich bij de introductie van Agile nu achterin de keten manifesteerden. Formeel gezien eindigt de case hier, omdat besloten werd Continuous Delivery te gaan inzetten voor MijnING. Hiervoor werd een

apart SCRUM-team Continuous Delivery opgericht, wat later is geëvolueerd tot de CDaaS squads (zie 4.3.2).

MijnING: score Continuous Delivery element	Inzet van element in de case	Effect van element op succesindicator				
		Scope	Kwaliteit	Tijd	Kosten	Wend-baarheid
Procesuitbreiding	○○○	□	□	□	□	□
Procesautomatisering	○○○	□	□	□	□	□
Sociaal/organisatorische factoren	●○○	□	✚ [1]	✚ [1]	□	□

[1] Multidisciplinaire teams met acceptanten uit verschillende subafdelingen waaronder *Operations* had een positief effect.

Figuur 24: Samenvatting resultaten casestudy C1: MijnING (ING-bank)

Omdat bij deze case al vrij snel Continuous Delivery elementen werden toegevoegd is het *wat-als-Continuous Delivery-scenario* redelijk nauwkeurig vast te stellen. Stap voor stap werden deployment en testen samen met het SCRUM-team Continuous Delivery geautomatiseerd. Ook werden met hulp van het management de activiteiten van de '*last mile*' geïntegreerd en opgenomen in de processen. Tot slot werd er een reorganisatie doorgevoerd waarmee de IT-organisatie gekanteld werd en de hand-overs tussen teams werden geminimaliseerd. Deze maatregelen bij elkaar hadden een positief effect op scope, doorlooptijd, kwaliteit en wendbaarheid. Op basis hiervan besloot het management om alle software op basis van Agile en Continuous Delivery te gaan ontwikkelen. Hierdoor kwamen de kosten voor de verdere inrichting van Continuous Delivery niet alleen ten laste van één project van MijnING, maar werden deze verdeeld over alle scrumteams. Het is - gezien de totale levensduur van MijnING - echter aannemelijk dat Continuous Delivery ook voor MijnING alleen rendabel zou zijn.

MijnING: *Wat-als-CD* score? Continuous Delivery element	Hypothetische inzet element	Effect van element op succesindicator				
		Scope	Kwaliteit	Tijd	Kosten	Wend-baarheid
Procesuitbreiding	✚	✚	✚	✚	✚	✚
Procesautomatisering	✚	✚	✚	✚	✚	✚
Sociaal/organisatorische factoren	✚	✚	✚	✚	✚	✚

Figuur 25: Score wanneer Continuous Delivery zou zijn ingezet bij MijnING (ING Bank)

4.3.2 CASE C.2: CONTINUOUS DELIVERY AS A SERVICE (CDAAS)

De case MijnING was het eerste Agile team. Medio 2011 waren er al 18 SCRUM/Agile teams operationeel, waarvan enkele al CD-elementen hadden geïmplementeerd. Begin 2012 is gestart met het *top down* implementeren van Continuous Delivery binnen de ING Bank om zo een deel van de strategie van ING te kunnen operationaliseren:

"N*iet alleen voor de Uber's en AirBnB's geldt dat 'software is eating the world', maar juist ook binnen de bancaire sector.* En ING gelooft sterk in '*speed is market share'. Als je op tijd bent pak je een groot marktaandeel, en wie te laat komt vist achter het net. Het doet er heel erg toe wie de eerste is".*

De dienst die ING-breed Continuous Delivery mogelijk maakt heet (*nomen est omen*) Continuous Delivery as a Service (CDaaS). ING kent geen projecten meer, nieuwe producten en diensten komen tot stand in bestaande of nieuw te vormen Agile teams (binnen ING squads genoemd). Inmiddels maken 800+ squads binnen ING gebruik van CDaaS en wekelijks worden nieuwe squads getraind en aangesloten op de CDaaS. De belangrijkste drivers om Continuous Delivery te implementeren zijn de kortere *time-to-market*, altijd de juiste functionaliteiten kunnen bouwen, kwaliteit en efficiëntie. Deze voordelen worden gemiddeld over alle squads ruimschoots behaald. Elk land waarin ING actief was organiseerde voorheen de een eigen softwareontwikkeling, maar sinds 2015 wordt Continuous Delivery ook *global* verder uitgerold, wat een enorme kostenbesparing met zich meebrengt. Het gebruik van gemeenschappelijke tooling als CDaaS maakt het vergelijken van performance over landen erg eenvoudig en dat kan als bedreiging worden gezien. Aan de andere kant zorgen gemeenschappelijke tooling en processen zoals CDaaS ook voor samenwerking: over teams en over landen heen.

De CDaaS-squads zijn verantwoordelijk voor beheer en verdere ontwikkeling van CDaaS binnen ING en passen hierbij het motto *'Eat your own dog food'* toe. CDaaS en daarmee Continuous Delivery wordt ook gebruikt voor softwareontwikkeling binnen de eigen teams en in deze case is dat gebruik onderzocht. Het beeld wat hierbij anno 2017 ontstaat is vergelijkbaar met de overige squads die met Continuous Delivery werken: alle elementen van Continuous Delivery hebben een positief effect op alle succesindicatoren. Investeringen zijn hoog, maar verdienen zich gedurende de levenscyclus van software makkelijk terug, zeker omdat investeringen aan de CDaaS over 800+ squads worden verdeeld. Gebruik van Continuous Delivery kent voor ING geen nadelen, maar kan voor medewerkers en afdelingen wel een bedreiging vormen. Een voorbeeld: de toegevoegde waarde van een marketingafdeling wordt verkleind als de keuze tussen twee prototypes van een bepaalde marketingcampagne beter kan worden gedaan met A/B-testing (het meten van de effectiviteit van beide versies in productie en dus met echte gebruikersdata) dan met marketeers.

Niet elk software-ontwikkeltraject bij ING is geschikt voor Continuous Delivery: *"Inzet van Continuous Delivery moet altijd een afweging zijn: heb ik de hogere time-to-market nodig? Wat is de impact van fouten als je niet automatiseert? Kosten, kwaliteit en tijdbesparing zijn hierin factoren die de keuze beïnvloeden."*

CDaaS: score	**Inzet van element in de case**	**Effect van element op succesindicator**				
Continuous Delivery element		Scope	Kwaliteit	Tijd	Kosten	Wend-baarheid
Procesuitbreiding	●●●	✚	✚	✚	✚	✚
Procesautomatisering	●●●	✚	✚	✚	✚	✚
Sociaal/organisatorische factoren	●●●	✚	✚	✚	✚	✚

Figuur 26: Samenvatting resultaten casestudy C2: CDaaS (ING-bank)

4.3.3 CROSS CASE ANALYSE CASES ING BANK

In deze cross case analyse worden de twee ING-cases onderling vergeleken: MijnING en CDaaS. Hoewel veel factoren voor beide cases gelijk zijn zit er wel zo'n zes jaar tussen beide cases. Door deze tijdspanne zijn er meer externe factoren die invloed kunnen hebben op het onderzoeksresultaat, zoals kennis, ervaring en organisatie-inrichting. Bij de ene case stonden Agile en vooral Continuous Delivery nog in de kinderschoenen, bij de andere case is de organisatie volledig ingericht om de Agile werkwijze te ondersteunen en is Continuous Delivery de *defacto* standaard binnen de ING-bank. De case MijnING laat zien dat Agile problemen aan de voorkant oplost, maar dat daardoor problemen verderop in de keten ontstaan. De oplossingen die daarvoor initieel werden bedacht bij MijnING hebben veel van de CD-elementen in zich: processen op elkaar afstemmen, automatiseren en samenwerking intensiveren. Deze elementen zijn in de tijd zijn geëvolueerd tot CDaaS, het onderwerp van onderzoek van de tweede case.

Wanneer de *wat-als*-analyse van MijnING wordt vergeleken met de analyse van CDaaS dan blijkt dat alle elementen van Continuous Delivery een positieve impact hebben op alle succesindicatoren.

Cross case analyse ING Bank	**Effect van element op succesindicator**				
Continuous Delivery element	Scope	Kwaliteit	Tijd	Kosten	Wendbaarheid
Procesuitbreiding	+	+	+	+	+
Procesautomatisering	+	+	+	+	+
Sociaal/organisatorische factoren	+	+	+	+	+

Figuur 27: Samenvatting overkoepelende cross-case analyse ING Bank

4.4 ORGANISATIE D: COOLBLUE

Coolblue is een winkelformule met mee dan driehonderd webwinkels en 8 fysieke winkels en werkt sinds 2011 met Agile teams waarbij de IT-teams van meet af aan Continuous Delivery elementen inzetten om software snel naar productie te kunnen zetten. De validiteit van een van beide cases was wegens gebrek aan triangulatiemogelijkheden onvoldoende om in het onderzoek op te nemen. In deze paragraaf worden dan ook alleen de resultaten van de case mét Continuous Delivery beschreven en is een cross-case analyse dan ook niet aan de orde.

4.4.1 CASE D: CONTINUOUS DELIVERY BACKOFFICE (CD-BACKOFFICE)

De onderzochte case betreft de Continuous Delivery Backoffice (CD-backoffice), dat (omen est nomen) verantwoordelijk zijn voor de ontwikkeling en het beheer van de centrale Continuous Delivery pipeline. Behalve dat het team de eigen systemen verder ontwikkelt op basis van Continuous Delivery, heeft het ook een goed zicht op het gebruik van Continuous Delivery binnen Coolblue.

"Coolblue beschouwt ontwikkelcapaciteit als een schaars middel, dus wanneer software gebouwd wordt dient het ook daadwerkelijk waarde te leveren. En de enige manier om daarachter te komen is om het de

klanten te laten gebruiken. Dat kan alleen wanneer deze wordt gereleased in productie. Dan kunnen we met data-analyses vaststellen of een wijziging ook daadwerkelijk waarde toevoegt."

Een SCRUM/Agile development team bestaat uit zo'n vijf developer en een product owner. Elk team is verantwoordelijk voor plusminus 5 van de 65 microservices die ervoor zorgen dat de websites en overige systemen functioneren. Teams bepalen zelf wanneer hun software naar productie gaat. Sinds een jaar is ook de laatste stap naar Productie wanneer alle testen geslaagd zijn automatisch: de stap van Continuous Delivery naar Continuous Deployment. Bij de start in 2011 bepaalde elk team zelf nog hoe dat werd gedaan; er waren dus meerdere Continuous Delivery pipelines. Consequentie daarvan was dat in 2015 is gestart deze te migreren naar een centrale CD-pipeline, waarvoor geldt: 'comply or explain'. Per dag worden met behulp van deze pipeline zo'n 20 releases naar productie gezet. Dat gaat volledig automatisch: het team hoeft alleen de release te initiëren en indien alle automatische testen geslaagd zijn wordt de software vanzelf uitgerold in de productie-omgeving. De CD-backoffice bestaat uit twee Agile teams die verantwoordelijk zijn voor de softwareontwikkeling en ondersteuning van de centrale Continuous Integration en Continuous Delivery (CI/CD)-pipeline.

"*Met 65 onderling verbonden microservices en 13 ontwikkelteams het nagenoeg ondoenlijk is om Agile te werken zónder CD. De omvang en (integratie-)complexiteit die nu wordt afgevangen door de CI/CD-pipeline zou zoveel handmatig werk met zich meebrengen dat er nauwelijks capaciteit beschikbaar is voor nieuwe features, terwijl de kwaliteit door de toenemende foutgevoeligheid hard achteruit holt.".*

De voordelen van Continuous Delivery die binnen Coolblue worden ervaren zijn lagere *time-to-market*, snellere bugfixes in productie, en door de kortere feedbackloops vanuit de productie omgeving sluit de software beter aan bij de klant. Daarnaast levert het gebruik te maken van een centrale Continuous Delivery pipeline efficiëntie-voordelen op. Dat zijn precies ook de effecten die ook voor de CD-backoffice worden ervaren bij het gebruik van Continuous Delivery. Voor Continuous Delivery worden geen downsides gezien, wel randvoorwaarden zoals een service-georiënteerde architectuur, een engineering cultuur van samenwerken, opleiding en skills.

CD-backoffice: score	**Inzet van element in de case**	**Effect van element op succesindicator**				
Continuous Delivery element		Scope	Kwaliteit	Tijd	Kosten	Wend-baarheid
Procesuitbreiding	●●●	+	+	+	+	+
Procesautomatisering	●●●	+	+	+	+	+
Sociaal/organisatorische factoren	●●●	+	+	+	+	+

Figuur 28: Samenvatting casestudy D: CD-backoffice (Coolblue)

4.5 OVERKOEPELENDE CROSS CASE ANALYSE

De vier onderzochte organisaties Rijkswaterstaat, Centric, ING Bank en Coolblue verschillen onderling sterk van elkaar, maar de resultaten uit de (cross case) analyses tonen veel gelijkenis. In de onderstaande tabel worden per organisatie de uitkomsten van de analyses weergegeven. Deze worden verder onder de tabel per Continuous Delivery-element toegelicht.

Overkoepelende cross case analyse	Effect van element op succesindicator																			
Succesindicator	Scope				Kwaliteit				Tijd				Kosten				Wend-baarheid			
Continuous Delivery element	RWS	Centric	ING	Coolblue	RWS	Centric	ING	Coolblue	RWS	Centric	ING	Coolblue	RWS	Centric	ING	Coolblue	RWS	Centric	ING	Coolblue
Procesuitbreiding	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Procesautomatisering	+	+	+	+	+	+	+	+	±	±	+	+	±	±	+	+	+	+	+	+
Sociaal/organisatorische factoren	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+

Figuur 29: Samenvatting (cross case) analyses

4.5.1 PROCESUITBREIDING

Het element 'procesuitbreiding' heeft een positief effect– of in het geval van de *Agile-only* cases: had kunnen hebben – op alle succesindicatoren, te weten scope, kwaliteit, tijd, kosten en wendbaarheid. Het integreren van de ontwikkelprocessen met de deployment- en acceptatieprocessen uit de '*last mile*' dragen al snel positief bij aan het succes, terwijl de investeringen hiervoor beperkt zijn.

4.5.2 PROCESAUTOMATISERING

Uit de cases blijkt dat de effecten van het element 'procesautomatisering' door het mogelijk maken van een kortere feedbackloop, het verkorten van de doorlooptijd en het verlagen van de foutgevoeligheid al snel de succesindicatoren 'scope', 'kwaliteit' en 'wendbaarheid' positief beïnvloedt (of wat de *Agile-only* cases betreft: had kunnen beïnvloeden). Het effect van het element 'procesautomatisering' op de indicatoren 'tijd' en 'kosten' varieert van negatief tot positief. Oorzaak hiervan is dat dit element initieel hoge (automatiserings)kosten veroorzaakt die niet altijd opwegen tegen de voordelen van Continuous Delivery en waarvan implementatie zoveel impact heeft dat deze voor vertraging kan zorgen in de oplevering van de software. Uit de case-analyses komt een drietal software-ontwikkeltraject-afhankelijke aspecten naar voren die van invloed zijn of het Continuous Delivery element negatief dan wel positief bijdraagt aan 'tijd' en 'kosten'.

MATE VAN HERGEBRUIK VAN CONTINUOUS DELIVERY

Uit de *wat-als-Continuous Delivery*-analyse bij het VCM-project van Rijkswaterstaat blijkt dat het ontwikkeltraject te klein zou zijn geweest om de automatiseringskosten ten behoeve van Continuous Delivery rendabel te maken. Hoe meer software-aanpassingen in de toekomst, hoe kleiner de investering per release in termen van tijd en kosten. De business case voor procesautomatisering wordt dus

positiever naarmate deze meer wordt ingezet. Zoals blijkt uit de IVS Next case bij Rijkswaterstaat kan implementatie van Continuous Delivery al lonen voor één softwareproject. Bij ING en Coolblue werd procesautomatisering aanvankelijk voor één traject opgezet, maar al snel voor meerdere trajecten ingezet, waardoor 'procesautomatisering' bij grotere software-gedreven organisaties al snel rendabel kan zijn.

IT-WENDBAARHEID

De S4O-case bij Centric toont dat architectuur een barrière kan zijn om Continuous Delivery te implementeren. Voor het automatiseren van test- en deploymentautomatisering dienen grotere monolithische systemen opgesplitst te worden in kleinere, autonome delen (bijvoorbeeld microservices). Het aanpassen van architectuur heeft bij grotere systemen echter behoorlijk veel impact en is daardoor kostbaar en langdurig, wat dus de succesindicatoren 'kosten', respectievelijk 'tijd' negatief zal beïnvloeden. Voor Centric was de businesscase hierdoor uiteindelijk niet positief, voor ING wel.

NOODZAKELIJKE TIME-TO-MARKET

"Speed is market share" is een citaat uit de CDaaS-case bij ING en wordt veel gebruikt om aan te geven dat *time-to-market* een belangrijke concurrentiefactor voor de ING is. Zelfs al zou door Continuous Delivery het ontwikkelen van software duurder worden, dan zou het nog steeds rendabel kunnen zijn vanwege het concurrentievoordeel dat wordt behaald door een lagere *time-to-market*. *Time-to-market* is van belang zijn bij nieuwe functionaliteiten, maar kan ook voor bestaande functionaliteiten gelden bijvoorbeeld de snelheid waarmee beveiligingslekken of softwarefouten kunnen worden opgelost.

4.5.3 SOCIAAL/ORGANISATORISCHE FACTOREN

Bij alle cases blijkt dat het laatste element, 'sociaal/organisatorische aanpassingen', een positief effect heeft – of in het geval van de *Agile-only* cases: had kunnen hebben – op alle succesindicatoren: scope, kwaliteit, tijd, kosten en wendbaarheid. Het versterken van dit elementen draagt al snel positief bij, omdat de investeringen hiervoor relatief laag zijn. Bij de *Agile-only* cases van Rijkswaterstaat en ING waren sociaal/organisatorische aanpassingen de eerste maatregelen om Agile-software sneller naar productie te krijgen. Een kanttekening hierbij is dat het versterken van deze elementen proportioneel dient te gebeuren: een bedrijfsbrede reorganisatie van een grote organisatie zal minder snel lonen wanneer er slechts één team is wat software ontwikkeld.

5 CONCLUSIE

In dit hoofdstuk wordt op basis van literatuur en empirisch onderzoek de centrale onderzoeksvraag beantwoord: *"Wat is het effect van het gebruik van Continuous Delivery op het succes van Agile software-ontwikkeltrajecten?"*. De beantwoording hiervan (paragraaf 5.2) volgt uit het beantwoorden van de deelvragen (paragraaf 5.1).

5.1 BEANTWOORDING VAN DE DEELVRAGEN

Om de centrale onderzoeksvraag te kunnen beantwoorden zijn in paragraaf 1.2 deelvragen opgesteld. Hieronder volgt de beantwoording hiervan als basis voor het kunnen beantwoorden van de centrale onderzoeksvraag.

WAT ZIJN AGILE SOFTWARE-ONTWIKKELTRAJECTEN?

Agile software-ontwikkeltrajecten zijn trajecten die middels een Agile software-ontwikkelmethode software voortbrengen. Hierbij geldt dat een traject zowel als een project, als een reguliere (lijn)activiteit kan worden gezien. Kenmerken van Agile ontwikkelmethoden zijn het in kort cyclische iteraties software opleveren met kleine autonome teams en door directe communicatie met de stakeholders.

WAT IS CONTINUOUS DELIVERY?

Continuous Delivery is een Agile ontwikkelmethode, waarbij het woord 'Continuous' in Continuous Delivery verwijst naar het *flow*-gedachtegoed uit *Lean Thinking* (Fitzgerald & Stol, 2014). Met *flow* wordt bedoeld dat werk in een continue stroom met een zo kort mogelijke doorlooptijd wordt gerealiseerd en contrasteert daarmee sterk met op *'batch & queue'* gebaseerde processen (Womack & Jones, 2010).

Continuous Delivery is "het vermogen om op elk moment veilig softwareaanpassingen op de productie-omgeving te kunnen realiseren en daarmee te ontsluiten voor de eindgebruiker" (Fitzgerald & Stol, 2014; Humble & Farley, 2010; Humble et al., 2015; Krusche & Alperowitz, 2014; Neely & Stolt, 2013). Dat de software altijd productie-klaar is wil nog niet zeggen dat dat ook daadwerkelijk bij elke wijziging in de code gebeurt. Continuous Delivery is dus niet het verkorten van de duur van een sprint (Gfader, 2013). Sterker nog: de cadans van een sprint hoeft niet gelijk te lopen met die van de release. Zo hoeft het implementeren van een oplossing voor een softwarefout niet te wachten tot het einde van een sprint, maar kan deze tussentijds op de productie-omgeving worden doorgevoerd. Daarentegen kan ook besloten worden om een groot stuk functionaliteit waaraan in meerdere sprints wordt gewerkt niet tussentijds na iedere sprint te releasen, maar in één keer.

UIT WELKE ELEMENTEN BESTAAT CONTINUOUS DELIVERY?

Continuous Delivery bestaat uit een drietal elementen: procesuitbreiding, procesautomatisering en sociale/organisatorische factoren. Deze worden hieronder toegelicht.

PROCESUITBREIDING

Het eerste element van Continuous Delivery betreft een procesuitbreiding waarbij het software-ontwikkelproces niet ophoudt bij de oplevering van de software aan *Operations*, maar wordt verlengd totdat de software ook daadwerkelijk herhaalbaar en betrouwbaar in productie ontsloten is voor de eindgebruikers en dus niet alleen *lead-users*.

PROCESAUTOMATISERING

Doel van het tweede element van Continuous Delivery, procesautomatisering, is om het bouwen, testen en deployen/releasen van software verregaand te automatiseren met als doel het verminderen van fouten, het verkorten van doorlooptijden en daardoor de feedbackloop en het verhogen van de betrouwbaarheid en de kwaliteit van het softwareproces. Het element Procesautomatisering bestaat uit drie onderdelen: *Continuous Integration* (een aanpak waarbij al het werk in uitvoering van de verschillende ontwikkelaars meerdere keren per dag worden samengevoegd in een zogenoemde '*master branch*'), *testautomatisering* (het automatiseren van zo veel mogelijk testactiviteiten) en *deploymentautomatisering* (het automatiseren van deployments van de ene omgeving naar de andere).

SOCIALE/ORGANISATORISCHE FACTOREN

Fitz (2009) stelt dat cultuur gezien kan worden als een kritieke succesfactor. Claps et al. (2015) zien als succesfactor van Continuous Delivery een cultuur waarin 'met elkaar gedeeld' wordt. Met name bij de *'last mile'* zijn veel afdelingen betrokken, allen met verschillende belangen, werkwijzen en beelden over verantwoordelijkheden en hoe samen te werken. Door aanpassing van organisatie en cultuur kan de voor succesvolle CD benodigde nauwere samenwerking worden bereikt (Akerele et al., 2014; Chen, 2015). Dit element van Continuous Delivery steunt op zowel *development* als *Operations* en heeft daarom sterke raakvlakken met het DevOps-fenomeen (Fitzgerald & Stol, 2014). DevOps is een cultuur, beweging en practise die de communicatie, samenwerking en integratie tussen *Development* en *Operations* verbetert (Remi Jullian & Sangeetha, 2016; Waller et al., 2015). Maar voor succesvolle implementatie van Continuous Delivery is betrokkenheid van meerdere afdelingen noodzakelijk (Olsson et al., 2012). Dus niet alleen *Development* en *Operations*, maar ook Q&A, IT-security, de business, etc. De veranderende verantwoordelijkheden, waarbij iedereen verantwoordelijk is voor een succesvolle release wordt makkelijker gezegd dan geïmplementeerd (Claps et al., 2015).

WAT ZIJN DE INDICATOREN VAN EEN SUCCESVOL AGILE SOFTWARE-ONTWIKKELTRAJECT?

Traditioneel wordt het succes van software trajecten gemeten op basis van vier indicatoren: 'kwaliteit' (wordt goed werkende, veilige en beheerbare software geleverd), 'scope' (sluit de software goed aan bij de impliciete en expliciete eisen en wensen van klant, business en gebruikers), 'tijd' (is er snel geleverd) en 'kosten' (is er spaarzaam met resources als tijd en geld omgegaan) (Cohn & Ford, 2003; Lindvall et al., 2004). Deze indicatoren zijn ook bruikbaar te zijn voor het beoordelen van het succes van Agile software projecten (Chow & Cao, 2008), maar meten vooral productiviteit, terwijl Agile-methoden geoptimaliseerd zijn voor wendbaarheid (Rico, 2008). Daarom is de indicator 'wendbaarheid' in dit

onderzoek toegevoegd als vijfde succesindicator. Effecten van Continuous Delivery die in de literatuur vaak genoemd worden, waaronder medewerkerstevredenheid, samenwerking, goede communicatie en kennisopbouw dragen bij aan de hiervoor omschreven succesindicatoren.

5.2 BEANTWOORDING VAN DE CENTRALE ONDERZOEKSVRAAG

Het beantwoorden van de deelvragen vormt de basis voor het beantwoorden van de centrale onderzoeksvraag: *"Wat is het effect van het gebruik van Continuous Delivery op het succes van Agile software-ontwikkeltrajecten?"*. Afhankelijk van een aantal software-ontwikkeltraject specifieke aspecten die aan het einde van deze paragraaf verder worden toegelicht kan op basis van deskresearch en de meervoudige casestudy bestaande uit een zevental cases in een viertal organisaties worden geconcludeerd dat inzet van Continuous Delivery een positief effect heeft op het succes van Agile software-ontwikkeltrajecten. De uitwerking van dit effect op de vijf geïdentificeerde succescriteria wordt in onderstaande tabel weergegeven.

Succesindicatoren	Effect van Continuous Delivery op Agile softwareontwikkeling
Scope	Kort cyclische feedback en objectieve *real user data* van echte gebruikers in een representatieve (productie)omgeving laten de software beter aansluiten bij de eisen en wensen van de business, klant en gebruikers.
Kwaliteit	Minder *refactoring* noodzakelijk, minder bugs en meer betrouwbare releases. Softwarefouten in productie kunnen sneller worden hersteld.
Tijd	Niet alleen een hogere ontwikkelsnelheid, maar ook een snellere *time-to-market**.
Kosten	Ontwikkelingskosten worden verder verlaagd, lagere kosten in de beheerfase*.
Wendbaarheid	Aanpassingen kunnen sneller worden doorgevoerd in productie en dus aan de gebruiker worden ontsloten.

* Afhankelijk van een aantal software-traject afhankelijke aspecten

Figuur 30: Effect van Continuous Delivery op de succescriteria van een Agile software-ontwikkeltraject

Uit het onderzoek blijkt dat de Continuous Delivery elementen 'procesuitbreiding' en 'sociaal/organisatorische factoren' een positief effect hebben op alle hierboven genoemde indicatoren van succesvolle Agile softwareontwikkeling. Het element 'procesautomatisering' heeft eveneens een positief effect op de indicatoren 'scope', 'kwaliteit' en 'wendbaarheid', maar het effect op 'tijd' en 'kosten' kan variëren van negatief tot positief, afhankelijk van een aantal aspecten van het software-ontwikkeltraject.

Onderzoeksresultaat	Effect van element op succesindicator				
Continuous Delivery element	Scope	Kwaliteit	Tijd	Kosten	Wendbaarheid
Procesuitbreiding	+	+	+	+	+
Procesautomatisering	+	+	- / +[1]	- / +[1]	+
Sociaal/organisatorische factoren	+	+	+	+	+

[1] Effect van procesuitbreiding op kosten en tijd is afhankelijk van aspecten van het software-ontwikkeltraject: mate van hergebruik van Continuous Delivery, IT-wendbaarheid en *time-to-market*.

Figuur 31: Onderzoeksresultaat: het Continuous Delivery succesmodel

De gevonden traject afhankelijke aspecten die het effect van 'procesuitbreiding' op de indicatoren 'kosten' en 'tijd' beïnvloeden zijn 'mate van hergebruik van Continuous Delivery', 'IT-wendbaarheid' en 'noodzakelijke *time-to-market'*. De business case voor procesautomatisering wordt positiever naarmate de voordelen frequenter worden genuttigd. De *mate van hergebruik* is dus een belangrijk criterium bij de keuze om voor een software-ontwikkeltraject voor Continuous Delivery te kiezen. Hergebruik is mogelijk door de inzet van Continuous Delivery niet alleen te beperken tot de projectfase, maar ook tijdens de verdere levenscyclus. Of door meerdere software-ontwikkeltrajecten gebruik te laten maken van dezelfde Continuous Delivery-voorzieningen. Het tweede aspect is de *IT-wendbaarheid*: indien de organisatie of de architectuur niet geschikt is voor Continuous Delivery, dan kan het aanpassen daarvan behoorlijk impactvol zijn en worden met name de succesindicatoren 'kosten', en 'tijd' negatief beïnvloedt. Een business case voor Continuous Delivery zal dan minder snel positief zijn. Tot slot is een belangrijk criterium de *noodzakelijke time-to-market*. Wanneer deze laag is kan het de organisatie een concurrentievoordeel bieden en zal in dat geval de business case voor Continuous Delivery eerder positief zijn.

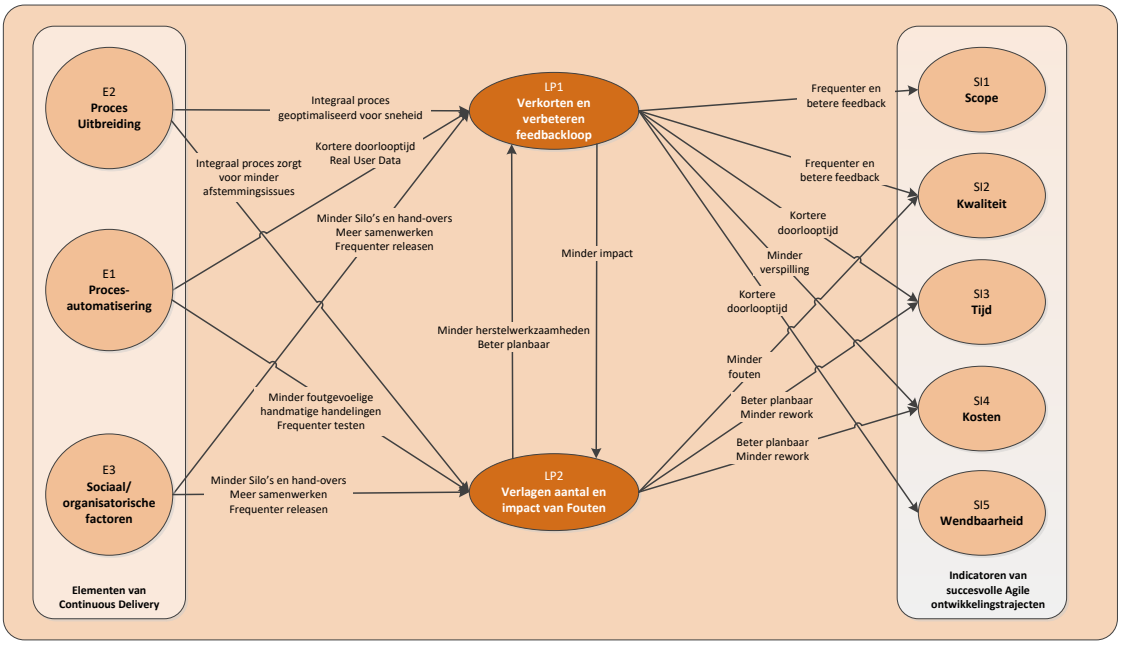
Een verklaring voor de gevonden effecten is opgenomen in het volgende hoofdstuk.

6 Verklaring, discussie, beperkingen en aanbevelingen

In dit laatste hoofdstuk worden in paragraaf 6.1 de resultaten verklaard en in een bredere context geplaatst (paragraaf 6.2). Vervolgens worden de beperkingen van het onderzoek uiteengezet (hoofdstuk 6.3). Het hoofdstuk eindigt met aanbevelingen voor vervolgonderzoek.

6.1 VERKLARING VAN DE GEVONDEN EFFECTEN

Op basis van de literatuurstudie en het empirisch onderzoek worden in deze paragraaf de effecten van de drie Continuous Delivery elementen (E1-E3) op de vijf succesindicatoren (SI1-SI5) verklaard. Omdat in de verklaring stelselmatig dezelfde redeneerlijnen gevolgd wordt, is in de verklaring een tussenstap (hierna *linking pin* genoemd) gebruikt. Het betreft hier de twee *linking pins* 'verkorten en verbeteren van de feedbackloop' en 'het verlagen van aantal en impact van fouten' (LP1-LP2). Beide *linking pins* hebben dus zowel een kwantitatief als kwalitatief element in zich.



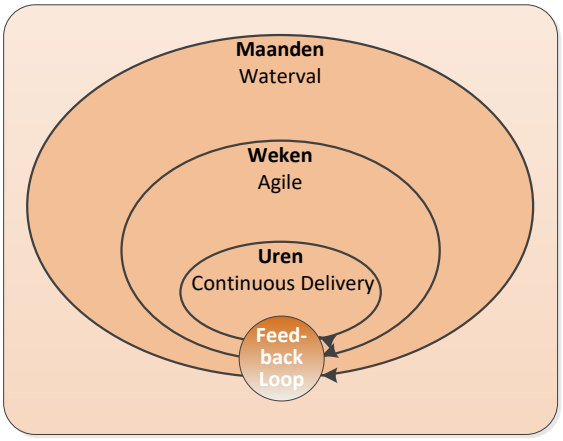
Figuur 32: Verklaring van de effecten van Continuous Delivery op het succes van Agile ontwikkeltrajecten

6.1.1 VERKLARING EFFECTEN OP LINKING PINS

Allereerst worden hieronder de twee *linking pins* beschreven en hoe deze door de Continuous Delivery elementen worden beïnvloed. Vervolgens wordt per indicator beschreven hoe deze van invloed zijn op het succes van Agile software-ontwikkeltrajecten.

LP1: VERKORTEN EN VERBETEREN VAN DE FEEDBACKLOOP

Continuous Delivery bewerkstelligt een kortere feedbackloop.



Figuur 33: Feedbackloop veelgebruikte ontwikkelmethoden

E1. Het element 'procesautomatisering', waaronder Continuous Integration, deployment- en testautomatisering maakt frequent releasen mogelijk doordat arbeidsintensief handwerk in grote mate wordt geautomatiseerd. Dit effect wordt nog eens versterkt doordat automatiseren de foutgevoeligheid verlaagd en er dus bij het bouwen, testen en deployen van software minder herstelwerkzaamheden nodig zijn, wat de feedbackloop verder verkort.

E2. Ook het element 'procesuitbreiding' zorgt voor een kortere feedbackloop door de software ontwikkelprocessen te integreren met de acceptatie- en release processen van de '*last mile*'. Hierdoor kan het proces beter geoptimaliseerd worden op doorlooptijd.

E3. Het element 'sociaal/organisatorische factoren' ondersteunt het verkorten van de feedbackloop door enerzijds een cultuur van frequent releasen en samenwerken te bewerkstelligen, en anderzijds een organisatie waarin het aantal hand-overs en silo-vorming is geminimaliseerd. De feedback wordt verbeterd doordat *real usage data*, dus objectieve data van echte productiegebruikers beschikbaar kan worden gemaakt.

LP2: VERLAGEN AANTAL EN IMPACT VAN FOUTEN

Continuous Delivery beoogt eveneens het verlagen van het aantal fouten en de impact daarvan op het ontwikkelproces.

E1. Het element 'Procesautomatisering' verkleint de hoeveelheid handmatige en daardoor foutgevoelige werkzaamheden.

E2. Door de 'procesuitbreiding' ontstaat een integrale benadering van het ontwikkelproces en de '*last mile*'. Hierdoor zijn er minder afstemmingsissues en kunnen deze door de integrale processturing sneller worden opgelost.

E3. Tot slot zorgen de 'sociaal/organisatorische factoren' voor een cultuur van samenwerken en een organisatie met minder hand-overs wat het aantal en de impact van fouten verder verlaagd. Deze effecten worden versterkt door de kortere feedbackloop en de hogere kwaliteit van feedback.

6.1.2 *VERKLARING EFFECT OP SUCCESINDICATOREN*

Nu de effecten van de verschillende Continuous Delivery elementen op de twee linking pins zijn bepaald, kan als tweede stap de beïnvloeding van de linking pins op de succesindicatoren worden beschreven.

SI1: SCOPE

LP1 Doordat de software sneller op een productie-omgeving beschikbaar is, dus met echte gebruikers en *real usage data* is niet alleen de frequentie van de feedback hoger, maar ook de representativiteit en objectiviteit ervan. Door de hogere frequentie en kwaliteit van de feedback sluit de functionaliteit van de software beter aan bij de (impliciete en expliciete) wensen en eisen van de klant en de gebruikers van de software dan bij Agile softwareontwikkeling zonder Continuous Delivery.

SI2: KWALITEIT

Continuous Delivery verhoogt de kwaliteit van software delivery. Hieraan ligt een aantal oorzaken ten grondslag.

LP1 Allereerst de hiervoor geschetste verhoogde kwaliteit van feedback. Daarnaast vergt het automatiseren van testen een investering, maar het uitvoeren van testen niet. Er kan dus frequenter en met een hogere testdekking worden getest, wat de kwaliteit ten goede komt.

LP2 Foutgevoeligheid wordt verlaagd door automatisering van handmatige werkzaamheden en proces-integratie, waardoor het releasen van software betrouwbaarder wordt. Tot slot worden implementatierisico's door het frequent releasen verkleind: enerzijds wordt hierdoor het implementeren van software en met name de '*last mile*' een non-event en anderzijds wordt door de hoge frequentie de omvang van de wijzigingen in de software per release kleiner.

SI3: TIJD

LP1 Bij het verkorten van de feedbackloop is geoptimaliseerd op snelheid van het gehele proces tot en met release in productie.

LP2 Doordat eveneens het aantal en de impact van scope-, bouw-, deployment- en integratie fouten afneemt is er minder tijd gemoeid met herstelwerkzaamheden. Bovendien kost het herstel van gevonden fouten minder tijd, omdat de developer door de snelle feedback niet zich weer moet inwerken in de code. Bovendien zijn de werkzaamheden door de vermindering en impact van (integratie)fouten beter planbaar.

SI4: KOSTEN

LP1 Verspilling resulteert vrijwel direct in het verlagen van kosten. Door het verkorten en verbeteren van de feedbackloop zal minder snel rework nodig zijn omdat duidelijker is wat er moet worden gebouwd. Het automatiseren van werkzaamheden en de procesoptimalisaties die ontstaan na procesuitbreiding resulteren eveneens in minder kosten.

LP2 Door het verminderen van aantal en impact van fouten zal er minder verspilling optreden, wat de kosten zal verlagen.

SI5: WENDBAARHEID

LP1 Door de kortere feedbackloop kan in kortere iteraties en met minder vertraging worden bijgestuurd, wat de wendbaarheid vergroot.

6.2 DISCUSSIE

In de vergelijkende meervoudige casestudy bestaande uit zeven cases bij een viertal organisaties is het effect onderzocht van Continuous Delivery op het succes van Agile software-ontwikkeltrajecten. Hieruit blijkt dat het voordelig is voor grote software-gedreven organisaties om hun Agile softwareontwikkeling op basis van Continuous Delivery in te richten. In vergelijking met *Agile-only* softwareontwikkeltrajecten wordt software geleverd die beter aansluit bij de behoefte van klant, business en gebruikers. Daarnaast levert Continuous Delivery kwalitatief betere software, sneller en tegen minder kosten. Hierbij geldt echter wel dat er voldoende software-opleveringen moeten zijn om de investeringen van Continuous Delivery in tijd en geld rendabel te maken. Deze investeringen worden hoger wanneer voor een implementatie veel aanpassingen in de IT-organisatie of architectuur noodzakelijk zijn. Aan de andere kant kan de organisatie concurrentievoordelen realiseren door de snellere *time-to-market* van Continuous Delivery, waardoor een implementatie weer sneller lonend is.

De constateringen uit het empirisch onderzoek liggen daarmee in lijn met de verwachte effecten uit het literatuuronderzoek, welke zijn samengevat in het Continuous Delivery succesmodel (zie paragraaf 2.4). Het empirisch onderzoek heeft echter twee nieuwe inzichten verschaft. Het eerste is dat niet alleen de wendbaarheid van de IT-organisatie, maar ook die van de IT-architectuur een variabele is bij de afweging om Continuous Delivery rendabel te kunnen implementeren. De tweede variabele, de herbruikbaarheid van de Continuous Delivery pipeline, was wel al vanuit de literatuurstudie voorzien, maar daar alleen binnen hetzelfde software-ontwikkeltraject. Verschillende cases toonden echter aan dat juist door het gebruik van één pipeline voor meerdere ontwikkeltrajecten, enorme besparingen kunnen worden gerealiseerd, met als extra bonus dat hierdoor binnen een organisatie standaardisatie ontstaat van softwareontwikkelprocessen en -tooling. Hierdoor zijn medewerkers en teams makkelijker uitwisselbaar, prestaties onderling beter vergelijkbaar en kan er efficiënter gewerkt worden.

Agile softwareontwikkeling verlaagt de muur tussen gebruikers(afvaardiging) en ontwikkelaars, maar introduceert bij grote organisaties door de hoge frequentie van opleveringen problemen verderop in de keten. Reguliere processen in de '*last mile*', waaronder test-, acceptatie- en deploymentprocessen, hebben lange doorlooptijden en zijn arbeidsintensief. Hierdoor is bij veel organisaties onvoldoende capaciteit om de Agile-frequentie van softwareopleveringen te ondersteunen, en komt ('*last mile*') feedback door de lange doorlooptijden veel later aan bij de ontwikkelteams. Gevolg is dat veel met Agile ontwikkelde software niet of in een veel lagere frequentie naar productie gaat. Continuous Delivery heeft de '*last mile*' wel in scope en is dan voor veel grotere software-gedreven organisaties de logische stap na het besluit om binnen een organisatie Agile ontwikkeling te doen. Vaak zijn *bottom-up*

al enkele aspecten van Continuous Delivery ingezet als mogelijke oplossing die (al dan niet *top-down*) de *spin-off* geven voor een integrale Continuous Delivery implementatie.

Continuous Delivery verandert de manier van werken aan een Agile softwaretraject echter ingrijpend: het beïnvloedt onder andere de werkzaamheden van developers, testers, *Operations*-medewerkers en het management. Deelverantwoordelijkheid van medewerkers, managers en teams wordt verruild voor een gemeenschappelijke integrale verantwoordelijkheid, processen worden geïntegreerd, geautomatiseerd en kwaliteit in het proces ingebouwd, feedbackloops worden verkort en silo's afgebroken. Continuous delivery is dus niet iets wat even snel kan worden geïmplementeerd. Het vergt aanzienlijke investeringen in techniek, processen, organisatie, houding en gedrag (Bremer & Eriksson, 2015; Chen, 2015). In de onderzochte cases worden de laatste twee vaak als het meest uitdagend gezien, alhoewel de voor Continuous Delivery benodigde aanpassingen in proces en techniek vaak wel als *enabler* voor de benodigde veranderingen in houding en gedrag worden genoemd. Veel innovaties stuiten bij invoering vaak op weerstand van medewerkers, maar doordat Continuous Delivery medewerkers 'bevrijdt' van veel repeterende handmatige handelingen zal er meer draagvlak zijn voor implementatie: het geeft medewerkers de mogelijkheid om tijd te besteden aan dingen die echt waarde toevoegen in plaats van foutgevoelige hand- en routinematige handelingen. Desondanks zullen er medewerkers zijn die Continuous Delivery zien als een bedreiging en zullen er medewerkers zijn die zich deze nieuwe manier van werken niet eigen kunnen maken. Ook hierbij zit de grootste uitdaging niet in de technische vaardigheden, maar in houding en gedrag.

Agile software-ontwikkelmethoden kennen naast voordelen ook nadelen. In een omvangrijke studie onderscheiden Turk, France & Rumpe (2014) er zes in het artikel "*Limitations of Agile Software Processes*". Aangezien Continuous Delivery gebaseerd is op Agile worden deze nadelen in meer of mindere mate overgenomen. Een tweetal nadelen wordt afgezwakt. Door het gebruik van dezelfde Continuous Delivery pipeline wordt een standaard werkwijze afgedwongen en verhoogt daarmee deels het nadeel *"beperkte ondersteuning voor gedistribueerde teams"*. Doordat bugfixes in zeer korte tijd kunnen worden gerealiseerd en gereleased, wordt de *"beperkte ondersteuning voor het ontwikkelen van missiekritieke software"* verhoogt. De nadelen *"beperkte ondersteuning voor het ontwikkelen van grote, complexe software"*, *"beperkte ondersteuning voor hergebruik van software-onderdelen"* en *"beperkte ondersteuning voor grote ontwikkelteams"* zijn onveranderd eveneens voor Continuous Delivery van toepassing. Het Agile-nadeel *"beperkte ondersteuning voor outsourcing"* wordt door de inzet van Continuous Delivery versterkt. Bij Continuous Delivery is de *defintion of done* van een sprint het opleveren van werkende software in de productieomgeving (en dus niet in de demo-omgeving bij de leverancier). Hiermee wordt de demarcatielijn tussen de verantwoordelijkheid van de opdrachtgever en die van de opdrachtnemer echter aanzienlijk lastiger te trekken waardoor het opstellen van outsourcingscontracten wordt bemoeilijkt.

Een ander nadeel van Continuous Delivery is de grote afhankelijkheid van IT, of meer precies: van de techniek. De afhankelijkheid met IT was er altijd al, maar dan in de vorm van IT-medewerkers. Mogelijk

dus dat de afhankelijkheid van IT juist met Continuous Delivery wordt verkleind, doordat kennis en ervaring van testen en deployments zich nu in code bevinden in plaats van in hoofden van mensen. Feit blijft dat de Continuous Delivery-pipeline een essentieel onderdeel is geworden van niet alleen het ontwikkelen van software, maar ook het beheer ervan. Daarmee wordt voor grote software-gedreven organisatie de beschikbarheid van de pipeline al snel even missiekritiek als de software die ermee wordt beheerd.

Een nog niet benoemd voordeel van Continuous Delivery is dat het organisaties kan helpen met Compliancy door bewijslast te automatiseren en in de vorm van rapportages te ontsluiten voor audits of indien nodig zelfs real-time. Continuous Delivery beoogt herhaalbare en betrouwbare releases, en om dat te realiseren leunt het zwaar op een (grotendeels) geautomatiseerd en dus transparant proces. Door wijzigingen in de Continuous Delivery-pipeline te versioneren kan op elk moment van de tijd teruggehaald worden hoe het wijzigingsproces is vormgegeven en kan daarmee dienen als bewijslast bij een audit. Door alle handelingen *in* de pipeline te registreren wordt het benodigde bewijs verzameld over elke stap in het software-ontwikkelproces. IT-Compliancy kan dus grotendeels geautomatiseerd en geïntegreerd worden binnen Continuous Delivery.

De resultaten in ogenschouw nemend is er *geen* sprake van een *trade-off*: Continuous Delivery is voor grotere software gedreven organisaties én flexibeler, én sneller, én kwalitatief beter én goedkoper dan *Agile-only* software-ontwikkeltrajecten. Aangezien dit resultaat reeds op basis van de literatuurstudie werd verwacht is dit in het empirisch onderzoek extra kritisch en met een gereserveerde houding benaderd. Desondanks blijkt dat grote software-gedreven organisaties die de Agile-werkwijze hebben geadopteerd Continuous Delivery willen: de voordelen zijn simpelweg te groot om te negeren. Dat een innovatie geen *trade-off* kent is bijzonder, maar niet nieuw. Een vergelijkbare revolutie voltrok zich eind jaren negentig van de vorige eeuw bij Toyota, waarbij het Toyota Production System een radicaal nieuwe manier van produceren introduceerde. Toyota was in staat gebleken om hiermee tegen minder kosten betere auto's te produceren, en dat ook nog eens in een kortere tijd en met een grotere verscheidenheid aan modellen (Ohno, 1988; Womack, Jones, & Roos, 1990). De vergelijking tussen Continuous Delivery en het Toyota Production System berust overigens niet helemaal op toeval. In zekere mate is Continuous Delivery door de sterke wortels in het Lean-gedachtengoed schatplichtig aan het Toyota Production System.

6.3 Beperkingen

Dit onderzoek is met de grootst mogelijke mate van integriteit tot stand gekomen. Toch kent de studie een aantal beperkingen die bij de interpretatie ervan bekend dienen te zijn. Deze worden hieronder toegelicht.

Ten aanzien van de selectie van cases is gebruik gemaakt van een convenience sample. Binnen deze sample is echter wel gezorgd voor een grote spreiding van organisaties waarin de cases zijn onderzocht

zodat de impact op de interne validiteit van het onderzoek is geminimaliseerd. De gebruikte Agile software-ontwikkelmethode is in alle cases het in Nederland veel gebruikte SCRUM.

De onderzoeker is inhoudelijk betrokken geweest bij de twee cases die bij Rijkswaterstaat zijn uitgevoerd en trekker de van verdere implementatie van Continuous Delivery binnen deze organisatie. Om de validiteit van het onderzoek niet te verzwakken vraagt dat om een open en neutrale houding van de onderzoeker jegens de respondenten en een professionele en kritische houding bij het interpreteren en verwerken van de onderzoeksdata. Het na afloop van het interview verifiëren van de gesprekssamenvattingen bij de respondenten beperkt de kans hierop.

Bij respondenten was de variëteit aan interpretaties van Continuous Delivery groter dan de onderzoeker op basis van twee pilotinterviews binnen de eigen organisatie had aangenomen. Om interpretatieverschillen te verkleinen is in eerste instantie ad-hoc een deel van de interviewtijd ingezet om een gemeenschappelijk referentiekader te verkrijgen. Toen bleek dat deze variëteit structureel was is gekozen om bij aanvang van het interview de in dit onderzoek gebruikte definities en elementen toe te lichten en te toetsen of deze bij aanvang van, en tijdens het gesprek ook als zodanig door de respondent wordt begrepen.

Omdat onderling vergelijkbare objectieve kwantitatieve data over het succes van software-ontwikkeltrajecten niet voorhanden was is een volgende beperking: de effecten van Continuous Delivery op de succesindicatoren zijn niet kwantitatief getoetst. Hoewel data altijd in de context is geïnterpreteerd en hiervoor gebruik is gemaakt van triangulatie kan niet worden uitgesloten dat meerdere respondenten tijdens het interview bewust of onbewust een iets te rooskleurig beeld hebben geschetst. Daar waar dit het geval leek heeft de onderzoeker dat beeld rechtgezet door wat meer gewicht toe te kennen aan andere bronnen.

Een andere beperking betreft de onderzoekperiode. In een enkel geval lag deze wat verder in het verleden, waardoor de betrouwbaarheid van de verkregen data mogelijk lager is dan de overige interviews. Daarnaast is de onderzoeksperiode van de twee samenhangende cases binnen dezelfde organisatie niet altijd dezelfde, waardoor de externe factor 'tijd' invloed kan hebben op het onderzoek. In beide gevallen heeft de onderzoeker middels triangulatie geprobeerd deze effecten te neutraliseren.

Tot slot: bij de *Agile-only* cases zijn de respondenten gevraagd wat de effecten zouden zijn geweest wanneer in het betreffende software-ontwikkeltraject wél Continuous Delivery zou zijn ingezet. Ondanks dat ook hier triangulatie is toegepast en de onderzoeker ook andere bronnen heeft betrokken bij de waardering, dragen de *what-if-Continuous Delivery*-scores een speculatief element in zich. Overigens bleken de onderzochte *Agile-only* in alle gevallen toch (zeer beperkt) enkele aspecten van Continuous Delivery te bevatten.

6.4 AANBEVELINGEN VOOR VERVOLGONDERZOEK

Ondanks de veelbelovende resultaten uit dit onderzoek zijn nog veel vragen rondom Continuous Delivery onbeantwoord. Om een volledig beeld te verkrijgen is aanvullend onderzoek waardevol.

In de onderzochte cases komen alle Continuous Delivery elementen duidelijk naar voren; maar de implementatie daarvan verschilt sterk per organisatie. In vervolgonderzoeken zou kunnen worden gefocust op de effectiviteit van deze variaties op de verschillende succesindicatoren. Van alle Continuous Delivery elementen zijn met 'procesautomatisering' de meeste investeringen gemoeid. Vanuit praktisch belang is het dan ook wenselijk om bij nader onderzoek naar de verschillende variaties dit element prioriteit te geven. Daarnaast is het interessant om de beïnvloeding van de verschillende Continuous Delivery elementen onderling te onderzoeken om meer inzicht te krijgen in hoe de verschillende elementen elkaar versterken dan wel verzwakken.

Een ander vervolgonderzoek zou zich kunnen richten op het kwantificeren van de gevonden effecten. Hiervoor kunnen verschillende *metrics* worden ingezet: *Lines of* Code per tijdeenheid per developer, aantal functiepunten per team per sprint, aantal bugs per user story, tijd gemoeid met rework i.v.m. scope-dwaling, aantal volledig behaalde sprints, et cetera.

Tot slot zou een verkennend onderzoek kunnen worden gedaan naar het effect van voorschrijvende regulering en specificaties (zoals bijvoorbeeld architectuur of functionele ontwerpen) op de succesindicatoren. Veel zaken voor de bouw van software zijn oorspronkelijk overgenomen uit de bouw in de 'fysieke wereld', waarbij de ontwerpfase vaak eveneens langer duurt dan de daadwerkelijke bouwfase. Maar met de toegenomen flexibiliteit van Continuous Delivery wordt het mogelijk om op elk moment wijzigingen door te voeren. Het herstellen van (fundamentele) fouten in de software is daarmee vele malen gemakkelijker dan in een gebouw of in software gemaakt met een klassieke waterval-ontwikkelmethode. Interessant is dan ook om te onderzoeken of het optimum voor regulering en specificatie in Continuous Delivery software-ontwikkeltrajecten verschuift ten aanzien van klassieke ontwikkelmethoden.

Deze voorgestelde vervolgonderzoeken zullen bijdragen aan de wetenschappelijke theorievorming ten aanzien van Continuous Delivery en organisaties faciliteren in een betere besluitvorming rondom de inzet van Continuous Delivery.

REFERENTIES

Akerele, O., Ramachandran, M., & Dixon, M. B. (2014). Evaluating the Impact of Critical Factors in Agile Continuous Delivery Process: A System Dynamics Approach. *International Journal of Advanced Computer Science and Applications (IJACSA)*, *5*(3), 133–143.

Andreessen, M. (2011, August 20). Why Software Is Eating The World. *Wall Street Journal*. Retrieved from https://www.wsj.com/articles/SB10001424053111903480904576512250915629460

Atkinson, R. (1999). Project management: cost, time and quality, two best guesses and a phenomenon, its time to accept other success criteria. *International Journal of Project Management*, *17*(6), 337–342.

Baker, S. E., Edwards, R., & Doidge, M. (2012). How many qualitative interviews is enough?: Expert voices and early career reflections on sampling and cases in qualitative research.

Baronner, A., & Burgt, T. van der. (2017). An Information and Tracking System for Inland Shipping. *IEEE Software*, *34*(3), 105–110.

Bassil, Y. (2012). A simulation model for the waterfall software development life cycle. *arXiv Preprint arXiv:1205.6904*.

Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., … Thomas, Dave. (2001). Manifesto for agile software development. Retrieved from http://agilemanifesto.org/

Begel, A., & Nagappan, N. (2007). Usage and perceptions of agile software development in an industrial context: An exploratory study. In *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on* (pp. 255–264). IEEE. Retrieved from http://ieeexplore.ieee.org/abstract/document/4343753/

Bremer, R., & Eriksson, J. (2015). *Understandings and Implementations of Continuous Delivery*. University of Gothenburg. Retrieved from https://gupea.ub.gu.se/handle/2077/39980

Candi, M., van den Ende, J., & Gemser, G. (2016). Benefits of Customer Codevelopment of New Products: The Moderating Effects of Utilitarian and Hedonic Radicalness. *Journal of Product Innovation Management*, (33(4),), 418–434.

Chen, L. (2015). Continuous delivery: Huge benefits, but challenges too. *IEEE Software*, *32*(2), 50–54.

Chow, T., & Cao, D.-B. (2008). A survey study of critical success factors in agile software projects. *Journal of Systems and Software*, *81*(6), 961–971.

Claps, G. G., Berntsson Svensson, R., & Aurum, A. (2015). On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software Technology*, *57*, 21–31.

Cohn, M., & Ford, D. (2003). Introducing an agile process to an organization [software development]. *Computer*, *36*(6), 74–78.

Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, *50*(9–10), 833–859. https://doi.org/10.1016/j.infsof.2008.01.006

Easterbrook, S., Singer, J., Storey, M.-A., & Damian, D. (2008). Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering* (pp. 285–311). Springer.

Easterby-Smith, M., Thorpe, R., & Jackson, P. R. (2012). *Management research*. Sage.

Eisenhardt, K. M. (1989). Building Theories from Case Study Research. *The Academy of Management Review*, *14*(4), 532–550.

Fitz, T. (2009). Continuous Deployment at IMVU: Doing the impossible fifty times a day. Retrieved February 27, 2017, from http://timothyfitz.com/2009/02/10/continuous-deployment-at-imvu-doing-the-impossible-fifty-times-a-day/

Fitzgerald, B., & Stol, K.-J. (2014). Continuous software engineering and beyond: trends and challenges. In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering* (pp. 1–9). ACM.

Fowler, M., & Foemmel, M. (2006). Continuous integration. *Thought-Works*, 122.

Fowler, M., & Highsmith, J. (2001). The agile manifesto. *Software Development*, *9*(8), 28–35.

Fuggetta, A., & Di Nitto, E. (2014). Software process (pp. 1–12). ACM Press.

Gfader, P. (2013). Use scrum and continuous delivery to build the right thing. Scrum.org. Retrieved from https://www.scrum.org/Portals/0/Documents/Community%20Work/Scrum.org%20Whitepaper_Continuous%20Delivery.pdf

Goodwin, T. (2015). *The battle is for the customer interface*. Tech Crunch.

Highsmith, J., & Cockburn, A. (2001). Agile software development: The business of innovation. *Computer*, *34*(9), 120–127.

Humble, J. (2010). Continuous Delivery vs Continuous Deployment. Retrieved February 8, 2017, from https://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment/

Humble, J., & Farley, D. (2010). *Continuous delivery: reliable software releases through build, test, and deployment automation*. Upper Saddle River, NJ: Addison-Wesley.

Humble, J., Molesky, J., & O'Reilly, B. (2015). *Lean enterprise* (First edition). Beijing: O'Reilly.

Ignatius, A. (2016). Toward a More Agile Future. *Harvard Business Review*.

Ika, L. A. (2009). Project success as a topic in project management journals. *Project Management Journal*, *40*(4), 6–19.

Infact Marktonderzoek. (2015, August). Klanttevredenheidsonderzoek Centric 2015. Infact Marktonderzoek. Retrieved from http://files.m15.mailplus.nl/user315100021/31766/GV%20Centric%20KTO%202015%20PDF.pdf

Krusche, S., & Alperowitz, L. (2014). Introduction of continuous delivery in multi-customer project courses. In *Companion Proceedings of the 36th International Conference on Software Engineering* (pp. 335–343). ACM.

Laanti, M., Salo, O., & Abrahamsson, P. (2011). Agile methods rapidly replacing traditional methods at Nokia: A survey of opinions on agile transformation. *Information and Software Technology*, *53*(3), 276–290. https://doi.org/10.1016/j.infsof.2010.11.010

Lindvall, M., Muthig, D., Dagnino, A., Wallin, C., Stupperich, M., Kiefer, D., … Kahkonen, T. (2004). Agile software development in large organizations. *Computer*, *37*(12), 26–34.

Maalej, W., Happel, H.-J., & Rashid, A. (2009). When users become collaborators: towards continuous and context-aware user input. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications* (pp. 981–990). ACM.

Neely, S., & Stolt, S. (2013). Continuous delivery? easy! just change everything (well, maybe it is not that easy). In *Agile Conference (AGILE), 2013* (pp. 121–128). IEEE.

Nygaard, K. (1990). Program Development as a Social Activity. *PDC*, 4–13.

Ohno, T. (1988). *Toyota Production System: Beyond Large-Scale Production*. CRC Press.

Olsson, H. H., Alahyari, H., & Bosch, J. (2012). Climbing the" Stairway to Heaven"–A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software. In *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on* (pp. 392–399). IEEE.

Perry, D. E., Sim, S. E., & Easterbrook, S. M. (2004). Case studies for software engineers. In *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on* (pp. 736–738). IEEE.

Perry, W. E. (2006). *Effective methods for software testing* (3rd ed). Indianapolis, IN: Wiley.

Petersen, K., & Wohlin, C. (2009). A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. *Journal of Systems and Software*, *82*(9), 1479–1490.

Petersen, K., & Wohlin, C. (2010). The effect of moving from a plan-driven to an incremental software development approach with agile practices: An industrial case study. *Empirical Software Engineering*, *15*(6), 654–693. https://doi.org/10.1007/s10664-010-9136-6

Pinto, J. K., & Slevin, D. P. (1988). Project success: definitions and measurement techniques. *Project Management Journal*, *XIX*(1), 67–72.

Ramesh, G., & Devadasan, S. R. (2007). Literature review on the agile manufacturing criteria. *Journal of Manufacturing Technology Management*, *18*(2), 182–201. https://doi.org/10.1108/17410380710722890

Remi Jullian, A., & Sangeetha, M. (2016). From Dev to Ops – Introduction to Devops on understanding Continuous Integration and Continuous Delivery. *International Journal of Innovative Research in Computer and Communication Engineering*, *4*(6), 12567–12572.

Rico, D. F. (2008). What is the ROI of Agile vs. Traditional Methods? *TickIT International*, *10*(4), 9–18.

Rook, P. (1986). Controlling software projects. *Software Engineering Journal*, *1*(1), 7.

Royce, W. W. (1970). Managing the development of large software systems. In *proceedings of IEEE WESCON* (Vol. 26, pp. 1–9). Los Angeles.

Salah, D., Paige, R. F., & Cairns, P. (2014). A systematic literature review for agile development processes and user centred design integration. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering* (p. 5). ACM.

Serrador, P., & Pinto, J. K. (2015). Does Agile work? — A quantitative analysis of agile project success. *International Journal of Project Management*, *33*(5), 1040–1051.

Sharma, S. (2014). *Continuous Delivery vs. Continuous Deployment*. Retrieved from https://www.youtube.com/watch?v=igwFj8PPSnw

Solinski, A., & Petersen, K. (2016). Prioritizing agile benefits and limitations in relation to practice usage. *Software Quality Journal*, *24*(2), 447–482. https://doi.org/10.1007/s11219-014-9253-3

Ståhl, D., & Bosch, J. (2014). Modeling continuous integration practice differences in industry software development. *Journal of Systems and Software*, *87*, 48–59.

Stolberg, S. (2009). Enabling Agile Testing through Continuous Integration (pp. 369–374). IEEE.

Turk, D., France, R., & Rumpe, B. (2014). Limitations of agile software processes. *arXiv Preprint arXiv:1409.6600*. Retrieved from https://arxiv.org/abs/1409.6600

Verschuren, P. J. M., & Doorewaard, M. (2007). *Het ontwerpen van een onderzoek*. Lemma.

VersionOne. (2016). VersionOne 10th Annual State of Agile Report. Retrieved from https://versionone.com/pdf/VersionOne-10th-Annual-State-of-Agile-Report.pdf

Vijayasarathy, L., & Turk, D. (2008). Agile software development: A survey of early adopters. *Journal of Information Technology Management*, *19*(2), 1–8.

Voss, C., Tsikriktsis, N., & Frohlich, M. (2002). Case research in operations management. *International Journal of Operations & Production Management*, *22*(2), 195–219.

Waller, J., Ehmke, N. C., & Hasselbring, W. (2015). Including Performance Benchmarks into Continuous Integration to Enable DevOps. *ACM SIGSOFT Software Engineering Notes*, *40*(2), 1–4.

Womack, J. P., & Jones, D. T. (2010). *Lean thinking: banish waste and create wealth in your corporation.* (Simon and Schuster).

Womack, J. P., Jones, D. T., & Roos, D. (1990). *Machine that changed the world*. Simon and Schuster. Retrieved from http://web.mit.edu/esd.83/www/notebook/machine.pdf

Yin, R. K. (2013). *Case study research: Design and methods. Sage publications.* Sage Publications.

BIJLAGE A: OVERZICHT CASES

Organisatie	Rijkswaterstaat		Centric		ING-bank		Coolblue
Sector	Overheid		ICT-dienstverlening		Financiële dienstverlening		Retail
# medewerker	ca 8.700		ca 4.300		12.000+		1.500+
Afzetgebied	Nederland		Europe		Wereldwijd		Benelux
Case	**VCM**	**IVS Next**	**S4O**	**Storeworld**	**mijnING**	**CDaaS**	**CD-backoffice**
# betrokken teams in case	1	2	2	2	1	2	2
Agile-methodiek	SCRUM	SCRUM	SCRUM	SCRUM	SCRUM	SCRUM	SCRUM
Continuous Delivery	**Agile-only**	**CD**	**Agile-only**	**CD**	**Agile-only**	**CD**	**CD**
Inzet procesuitbreiding	○○○	●●●	○○○	●●●	○○○	●●●	●●●
Inzet procesautomatisering	○○○	●●○	●○○	●●●	○○○	●●●	●●●
Inzet Sociaal/organisatorische fact.	●○○	●●○	●○○	●●●	●○○	●●●	●●●
Continuous Deployment	n.v.t.	nee	n.v.t.	nee	n.v.t.	nee	ja
CD-pipeline (trajectspecifiek/centraal	n.v.t.	specifiek	n.v.t.	specifiek	n.v.t.	Centraal	Centraal
# op pipeline aangesloten teams	n.v.t.	4	n.v.t.	2	n.v.t.	800+	65

BIJLAGE B: OVERZICHT INTERVIEWS

Organisatie	Interview	Rol
Rijkswaterstaat	Omwille van privacy zijn de namen van de respondenten niet opgenomen in de publicatie maar opvraagbaar bij de onderzoeker	Project manager VCM
		Vervangend projectmanager VCM
		Project manager IVS Next
		Product owner IVS Next
		Vervangend product owner IVS Next
		Projectleider implementatie CD
Centric		Scrummaster S4O
		Unit manager Storeworld
		Consultant
		Manager R&D
Coolblue		Cloud Settler, member CD-team
		Delphi-team
ING		Programma manager Power
		Integrator Continuous Delivery
		Integratie consultant MijnING
		Product Owner CDaaS Continuous Integration & Deployment

BIJLAGE C: CODERINGSMATRIX

Continuous Delivery Element	Score	Toelichting
Procesuitbreiding	*Een procesuitbreiding waarbij het software-ontwikkelproces niet ophoudt bij de oplevering van de software aan Operations, maar wordt verlengd totdat de software ook daadwerkelijk herhaalbaar en betrouwbaar in productie ontsloten is voor de gebruikers.*	
	○○○	Softwarebouw en -release twee volledig gescheiden processen
	●○○	Softwarebouw en -release twee processen, maar met informele integratie
	●●○	Softwarebouw en -release twee processen, maar met formele integratie
	●●●	Softwarebouw en -release één proces
Procesautomatisering	*Het verregaand automatiseren van de processen rondom het samenstellen, deployen en testen van software.*	
	○○○	Geen Continuous Integration ,test- of deployment geautomatiseerd
	●○○	1 van de 3 geautomatiseerd
	●●○	2 van de 3 geautomatiseerd
	●●●	Continuous Integration ,test- én deployment geautomatiseerd.
Sociaal/organisatorische factoren	*De voor een Continuous Delivery implementatie benodigde aanpassingen in de organisatie, de organisatiecultuur en de wijze van samenwerking.*	
	○○○	Geen formele sociaal/organisatorische CD-factoren geïmplementeerd
	●○○	1 van de 3 factoren geïmplementeerd
	●●○	2 van de 3 factoren geïmplementeerd
	●●●	Verregaande formele en informele samenwerking én een cultuur van frequent releasen.