IVAR JACOBSON INTERNATIONAL

USE-CASE 2.0

Handboek voor het succesvol toepassen van Use Cases

Ivar Jacobson

Ian Spence

Kurt Bittner

Vertaald door: Eric Lopes Cardozo

July 2012

Over dit Handboek

Dit handboek beschrijft hoe je use cases op een agile en schaalbare wijze toepast. Gebaseerd op de nieuwste beproefde inzichten beschrijven we de volgende generatie van de use-case techniek die we Use-Case 2.0 noemen. Het doel is je een fundament te bieden waarmee je de use-case techniek maximaal tot zijn recht kan laten komen. Dit fundament is breed toepasbaar, van kleine agile teams op één locatie tot grote geografisch gescheiden teams, uitbesteding en complexe grootschalige systeemontwikkeling (systemen van systemen).

Use-Case 2.0 beschrijft de essentie van use-case driven development als een makkelijk toepasbare en herbruikbare aanpak. Het biedt ook een introductie van het idee achter use cases en hun toepassing. Het is met opzet geen allesomvattend handboek dat ingaat op alle facetten van use cases en het is ook geen lesboek over use-case modelleren. Het is dus mogelijk dat het niet toereikend is om de aanpak direct in praktijk te brengen. Het is bijvoorbeeld niet bedoeld om je te leren hoe je moet modelleren. Daarvoor verwijzen we je graag naar onze eerdere publicaties en boeken over dit onderwerp.

Leeswijzer

Dit handboek bevat vier hoofdstukken:

- Wat is Use-Case 2.0? – Een korte introductie van de werkwijze
- Use-Case 2.0 Basisprincipes – Een inleiding tot use cases op basis van zes principes die samen het fundament vormen van de werkwijze
- Use-Case 2.0 Werkwijze – De werkwijze zelf, beschreven in de vorm van elementaire concepten, activiteiten, werkproducten en de spelregels die zorgen voor hun samenhang
- Use-Case 2.0 Toepassen – Een samenvatting van wanneer en hoe de werkwijze toe te passen.

Deze hoofdstukken worden voorafgegaan door deze korte inleiding en afgesloten met een korte conclusie.

Wanneer dit je eerste kennismaking met use cases is, dan raden we je aan om eerst de hoofdstukken “Wat is Use-Case 2.0?”, “Use-Case 2.0 Basisprincipes” en “Use-Case 2.0 Toepassen” te lezen zodat je bekend bent met de basisconcepten. Wil je de werkwijze gaan toepassen, ga dan verder met het hoofdstuk “Use-Case 2.0 Werkwijze”.

Wanneer je al bekend bent met de basisprincipes van use cases, dan zou je kunnen beginnen met de hoofdstukken “Use-Case 2.0 Werkwijze” en “Use-Case 2.0 Toepassen” nadat je het hoofdstuk “Wat is Use-Case 2.0?” hebt gelezen. Dit helpt je om Use-Case 2.0 te vergelijken met je eigen ervaringen en de verschillen te begrijpen.

Dit zijn natuurlijk slechts suggesties - je kunt het handboek uiteraard ook gewoon van voor tot achter lezen.

Wat is Use-Case 2.0

Use Case: Een use case omvat alle manieren waarop het systeem gebruikt kan worden om een bepaald doel voor een bepaalde gebruiker te behalen. Een complete set use cases geeft je alle zinvolle manieren om het systeem te gebruiken en illustreert de waarde die dit zal opleveren.

Use-Case 2.0: Een schaalbare, agile werkwijze die use cases gebruikt voor het vastleggen van een set requirements (systeemeisen) en het aansturen van de incrementele ontwikkeling van het systeem dat hierin moet voorzien.

Use-Case 2.0 stuurt de ontwikkeling van een systeem aan door je eerst te helpen te begrijpen hoe het systeem zal worden gebruikt. Vervolgens helpt het je om een geschikt systeem te laten ontstaan dat de gebruikers daadwerkelijk ondersteunt. Het kan worden gebruikt naast je huidige management- en technische werkwijzen die de succesvolle ontwikkeling van software en andere vormen van systemen ondersteunen. Zoals je zult merken is Use-Case 2.0:

- Lichtgewicht
- Schaalbaar
- Veelzijdig
- Eenvoudig toe te passen.

Use cases maken inzichtelijk wat een systeem moet doen en, door doelbewuste weglating, wat het systeem *niet* zal doen. Ze maken ideevorming, scope management en incrementele ontwikkeling mogelijk van ieder type systeem van iedere omvang. Sinds de eerste introductie op OOPSLA in 1987 zijn use cases gebruikt voor het aansturen van systeemontwikkeling. Door de jaren heen zijn ze de basis gaan vormen voor vele verschillende methoden en een integraal onderdeel geworden van Unified Modeling Language (UML). Ze worden gebruikt in veel verschillende contexten en omgevingen en door diverse soorten teams. Use cases kunnen bijvoorbeeld zowel waardevol zijn voor kleine agile ontwikkelteams die applicaties ontwikkelen waarbij de nadruk ligt op interactie met de gebruiker als ook voor omvangrijke projecten die complexe samengestelde systemen moeten ontwikkelen, zoals enterprise systemen, productfamilies en systemen in de *cloud*.

De use-case werkwijze gaat veel verder dan alleen het vastleggen van requirements. In onze visie zouden use-cases breder moeten worden gebruikt, namelijk ook voor het aansturen van de ontwikkeling. Dit betekent dat Use-Case 2.0 ook activiteiten ondersteunt zoals analyse, ontwerp, plannen, schatten, voortgangsbewaking en testen. Het schrijft niet voor hoe je ontwikkelwerkzaamheden moet plannen of besturen en evenmin hoe je een systeem moet ontwerpen, coderen en testen. Wel biedt het een structuur voor de succesvolle toepassing van de door jou gekozen management- en ontwikkelaanpak.

Use-Case 2.0 is een goed gedefinieerde werkwijze die zichzelf in het veld heeft bewezen. Hoewel de term Use-Case 2.0 een nieuwe versie van use cases suggereert, betreft het hier geen nieuwe versie van Unified Modeling Language. Het is meer een optelsom van de veranderingen in de manieren waarop ontwikkelaars en analisten use cases hanteren. Een use case blijft gewoon een use case, maar de manier waarop een use case wordt gepresenteerd, gehanteerd en onderhouden is geëvolueerd naar een effectievere vorm. De wijzigingen zijn verre van theoretisch maar uitermate praktisch en gebaseerd op 20 jaar wereldwijde toepassing met alle soorten softwareontwikkeling.

Basisprincipes

Er zijn zes basisprincipes die ten grondslag liggen aan iedere succesvolle toepassing van use cases:

1. Houd het simpel door verhalen te vertellen
2. Begrijp het grotere geheel
3. Focus op waarde
4. Bouw het systeem op in *slices*
5. Lever het systeem incrementeel op
6. Sluit aan op de behoeften van het team.

In dit hoofdstuk gaan we nader in op deze principes waarna we ze zullen gebruiken als vertrekpunt voor de introductie van de concepten *use-case modelleren* en *use-case driven development*.

Principe 1: Houd het simpel door verhalen te vertellen

Culturen overleven en groeien door het vertellen van verhalen omdat dit de eenvoudigste en meest effectieve manier is om kennis door te geven. Het is daarom ook de beste manier om te communiceren wat een systeem moet doen, en om iedereen die aan het systeem werkt dezelfde doelen te laten nastreven.

Met use case leggen we de doelstellingen van het systeem vast. Om ze te begrijpen, vertellen we verhalen. Deze verhalen vertellen hoe doelen moet worden behaald en hoe problemen onderweg moeten worden afgehandeld. Use cases bieden de mogelijkheid om alle verschillende maar samenhangende verhalen op een simpele en begrijpelijke manier te identificeren en vast te leggen. Hiermee kunnen de requirements van een systeem eenvoudig worden verzameld, gedeeld en begrepen. Aangezien een use case zich richt op het behalen van een bepaald doel, geeft de use case ook richting aan het vertellen van de verhalen. In plaats van te proberen om het hele systeem in één keer te beschrijven, kunnen we dit use case voor use case doen. Voor iedere use case bundelen we zijn verhalen in een use-case beschrijving.

Wanneer verhalen vertellen wordt gebruikt als techniek om requirements te communiceren, dan is het essentieel om ervoor te zorgen dat verhalen zo worden vastgelegd dat ze uitvoerbaar en testbaar zijn. Een use-case beschrijving moet dan ook worden vergezeld van een set testgevallen waarmee de inhoud van de use case wordt gecompleteerd. De testgevallen zijn het belangrijkste onderdeel van een use-case. Ze zijn zelfs belangrijker dan de use-case beschrijving, omdat ze de verhalen praktisch maken en ze eenduidig kunnen aantonen dat een systeem inderdaad doet wat men ervan verwacht. Het zijn immers de testgevallen die definiëren wanneer een use case succesvol is geïmplementeerd.

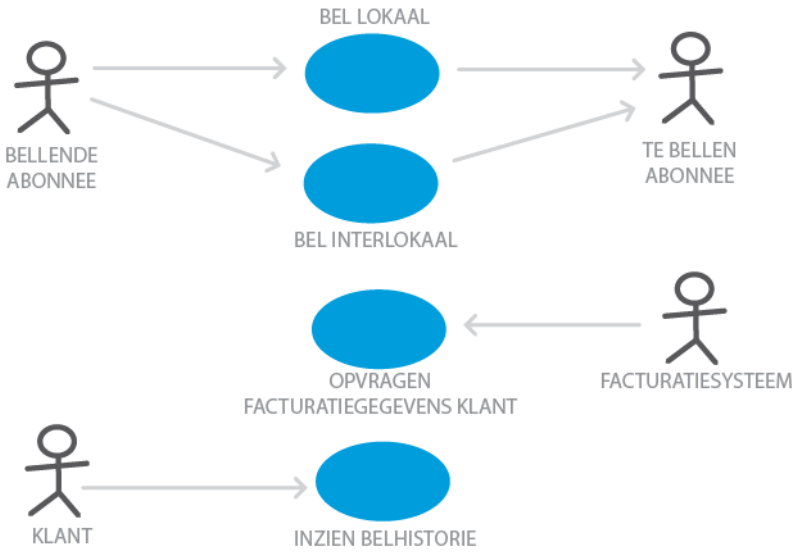
Principe 2: Begrijp het grotere geheel

Begrip van het grotere geheel is essentieel, of het nu gaat om een klein systeem of een groot systeem, een software systeem of een hardware systeem, of het opzetten van een nieuw bedrijf. Je zult merken dat het zonder begrip van het systeem als geheel onmogelijk is om goede beslissingen te nemen over de scope van het systeem, de kosten en de baten. Dit betekent echter niet dat alle requirements vooraf vastgelegd moeten worden. Je hebt iets nodig dat het gewenste systeem samenvat en je tevens in staat stelt om de scope en voortgang op systeemniveau te begrijpen.

Een use-case diagram is een eenvoudige manier om een overzicht te geven van de requirements van een systeem. Figuur 1 toont een use-case diagram van een eenvoudig telefoniesysteem. Dit figuur laat je alle manieren zien waarop het systeem kan worden gebruikt, wie de interactie start en welke andere partijen

betrokken zijn. Een Bellende Abonnee kan bijvoorbeeld een lokaal of een interlokaal gesprek aanvragen met een van de in het systeem bekende Te Bellen Abonnees.

Ook is zichtbaar dat gebruikers niet alleen mensen maar ook systemen kunnen zijn, of beide. Een antwoordapparaat kan bijvoorbeeld ook de rol van een Te Bellen Abonnee aannemen in plaats van een persoon.



FIGUUR 1: USE-CASE DIAGRAM VAN EEN EENVOUDIG TELEFONIESYSTEEM

Een use-case model wordt gevisualiseerd door een of meerdere use-case diagrammen. Use-case modellen laten zien dat systemen meerdere doelen van verschillende belanghebbenden dienen. In een use-case model worden belanghebbenden die het systeem gebruiken en bijdragen aan het vervullen van de doelstellingen gemodelleerd als actoren. De wijzen waarop het systeem kan worden gebruikt om doelstellingen te behalen, worden gemodelleerd als use cases. Op deze manier biedt het use-case model de context waarbinnen requirements kunnen worden ontdekt, gedeeld en doorgrond. Het geeft ook op een eenvoudig toegankelijke manier een overzicht van alles wat het systeem moet doen. In een use-case diagram, zoals Figuur 1, worden actoren weergegeven als draadpoppetjes en use cases als ellipsen. Met pijlen wordt aangeven wie (of wat) de use case kan starten.

Een use-case model is een model van alle waardevolle manieren om een systeem te gebruiken. Het stelt je in staat om heel snel de scope van het systeem af te bakenen en het team een duidelijk beeld te geven van wat het systeem moet doen, zonder je te hoeven verdiepen in allerlei details of de interne werking van het systeem. Al met een beetje ervaring is het eenvoudig om zelfs voor de meest complexe systemen use-case modellen op te stellen en zo alle betrokkenen een duidelijk overzicht te geven van de omvang en de doelen van het systeem.

Principe 3: Focus op waarde

Wanneer je probeert te begrijpen hoe een systeem gebruikt gaat worden, is het altijd belangrijk om te focussen op de toegevoegde waarde die het systeem zal hebben voor zijn gebruikers en andere belanghebbenden. Waarde ontstaat uitsluitend als het systeem daadwerkelijk wordt gebruikt. Het is dus veel beter om je te richten op dit gebruik, dan om lange lijsten van systeemeigenschappen of *features* op te sommen.

Use cases bieden deze focus omdat ze gericht zijn op hoe het systeem zal worden gebruikt om een specifiek doel voor een bepaalde gebruiker te behalen. Ze omvatten vele manieren om het systeem te

gebruiken; manieren waarbij daadwerkelijk doelen worden behaald, en manieren om onderweg optredende problemen af te handelen. Om de waarde eenvoudig te identificeren, te kwantificeren en te leveren, dien je de use-case beschrijving te structureren. Houd het simpel door te beginnen met de eenvoudigste manier om een doel te behalen. Leg vervolgens alternatieve manieren vast om dit doel te behalen. Richt je daarna op manieren om problemen af te handelen die het behalen van dit doel in de weg staan. Dit maakt het verband duidelijk tussen alle mogelijke manieren om het systeem te gebruiken. Het zorgt ervoor dat de meest waardevolle manieren als eerste worden geïdentificeerd en ontwikkeld. Tevens stelt het je in staat om later minder waardevolle manieren toe te voegen zonder het voorgaande te beschadigen of te wijzigen.

In sommige gevallen biedt het ontwikkelen van alleen de eenvoudigste manier om het systeem te gebruiken afdoende toegevoegde waarde. In andere gevallen maken alternatieve manieren om het systeem te gebruiken juist het verschil, bijvoorbeeld ten opzichte van de concurrentie.

Figuur 2 toont een op de bovenstaande wijze gestructureerde use-case beschrijving. De eenvoudigste manier om het doel te behalen staat in de basisstroom (Engels: basic flow). De andere manieren zijn weergegeven als alternatieve stromen (Engels: alternative flow). Op deze manier creëer je een set stromen waarmee de verhalen gestructureerd en beschreven worden. Verder helpen ze ons om de testgevallen te vinden die de verhalen compleet maken.

FIGUUR 2: DE STRUCTUUR VAN EEN USE-CASE BESCHRIJVING

Figuur 2 laat de structuur van een use-case beschrijving zien, in dit geval de use case *Opnemen Geld* voor een geldautomaat. De basisstroom bestaat uit een aantal eenvoudige stappen die de interactie tussen de gebruikers en het systeem duidelijk maakt. Onder de noemer alternatieve stromen zijn andere manieren om het systeem te gebruiken in kaart gebracht. Voorbeelden hiervan zijn het vragen om een niet-standaard bedrag, optionele faciliteiten die kunnen worden geboden zoals een ontvangstbewijs, en het afhandelen van problemen zoals een vastzittende kaart.

Het is niet nodig om alle stromen tegelijkertijd te detailleren. Het is het heel gebruikelijk om tijdens het vastleggen van de basisstroom na te denken over alternatieve manieren om het systeem te gebruiken en over wat er tijdens een bepaalde stap kan misgaan. Die manieren kun je vastleggen als alternatieve stromen, zoals in figuur 2, terwijl je je blijft concentreren op de basisstroom. De gevonden alternatieven pak je dan, indien nodig, later op.

Een dergelijke contourschets kan al genoeg zijn om de verhalen vast te leggen en de softwareontwikkeling aan te sturen. In andere gevallen heeft een team meer detail nodig om te begrijpen wat het systeem moet doen. De uitbreidbare structuur van de use-case beschrijving is hierbij van groot belang. Om de use case ooit succesvol uit te voeren, is allereerst de basisstroom nodig. Deze moet als eerste worden ontwikkeld. Alternatieve stromen kunnen worden toegevoegd aan de basisstroom wanneer nodig. Hiermee kun je je helemaal focussen op de waarde die het systeem moet leveren. Het is ook niet noodzakelijk om een hele use case ineens op te leveren want je kunt je richten op die delen die het meeste waarde toevoegen. Ook betekent dit dat je geen compleet use-case model of zelfs maar een complete use case nodig hebt om te starten met de ontwikkeling van het systeem. Met alleen de basisstroom van de belangrijkste use case kun je al beginnen toegevoegde waarde te leveren.

Deze structuur maakt het eenvoudig om verhalen vast te leggen en te valideren op compleetheid. Het is bovendien eenvoudig om minder waardevolle verhalen eruit te filteren. Deze constante focus op waarde stelt je in staat om iedere release van het systeem zo klein mogelijk te houden, en maximale waarde te leveren aan de gebruikers en de belanghebbenden die het systeem financieren.

Principe 4: Bouw het systeem op in *slices*

De meeste systemen vereisen veel werk voordat ze bruikbaar en operationeel zijn. Ze hebben veel requirements, met veel onderlinge afhankelijkheden, die het noodzakelijk maken dat eerst de onderliggende requirements zijn ontwikkeld voordat een systeem toegevoegde waarde kan leveren. Het is nooit verstandig om te proberen zo'n systeem in één keer te vervaardigen. De aanbeveling is om zo'n systeem in behapbare delen te ontwikkelen waarbij ieder deel duidelijke toegevoegde waarde heeft. Deze behapbare delen noemen we use-case *slices.*

Volg dit eenvoudige recept. Begin met de meest waardevolle functionaliteit van het systeem en focus daarop. Deel die functionaliteit vervolgens op in slices. Bepaal welke testgevallen nodig zijn voor de acceptatie van de slices. Sommige slices zullen vragen oproepen die nog niet kunnen worden beantwoord. Leg die voor nu aan de kant. Kies de meest cruciale slice, dat wil zeggen de slice die de functionaliteit van begin tot eind raakt, of daar zo dicht mogelijk tegenaan ligt. Maak met het team een schatting van die slice en begin met de ontwikkeling (bedenk hierbij: schattingen mogen afwijken want het zijn maar schattingen).

Dit is de Use-Case 2.0 aanpak, waarbij de use cases worden opgedeeld in passende eenheden van werk en waarbij het systeem slice voor slice tot stand komt.

De use case opdelen in slices

De verhalen en de wijze waarop we die vastleggen bieden de sleutels voor het vinden van de juiste slices. Ieder verhaal is een goede kandidaat-slice. Ieder verhaal wordt gedefinieerd door een deel van de use-case beschrijving en een of meer bijbehorende testgevallen. De testgevallen zijn het belangrijkste onderdeel van de beschrijving van de use-case slice omdat die een verhaal concreet maken.

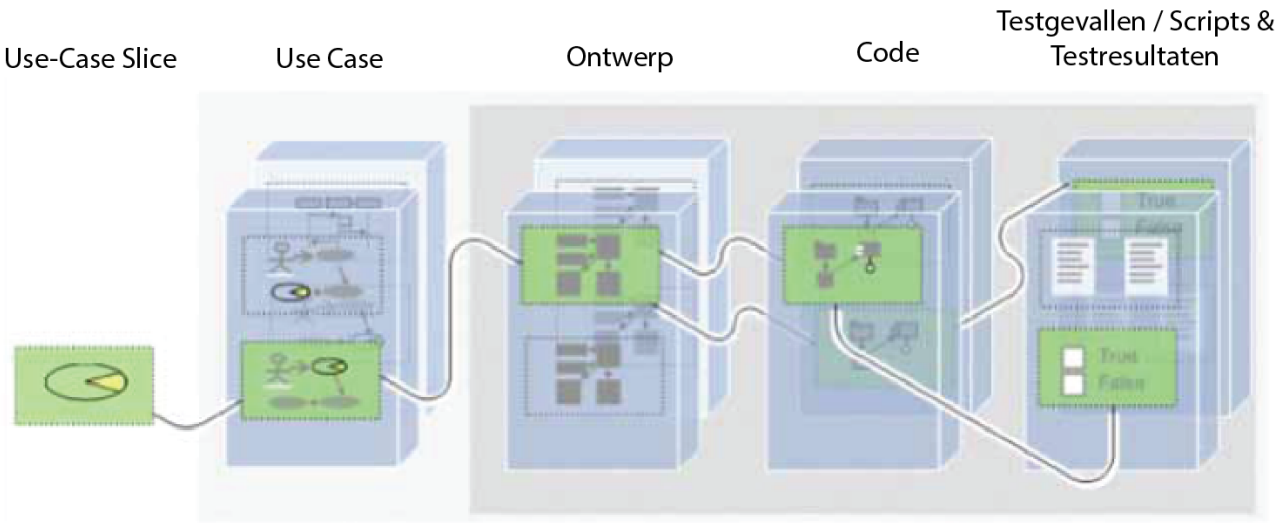
Door het bovenstaande recept toe te passen, zullen de use cases de waardevolle systeemfunctionaliteit weergeven. Kies de meest waardevolle use case om de belangrijkste systeemfunctionaliteit te vinden. Voor het vinden van de meest essentiële slice dien je vervolgens alle minder belangrijke manieren om het doel te bereiken, of problemen af te handelen, af te pellen. Dit kun je doen door te focussen op één verhaal: het verhaal dat wordt beschreven door de basisstroom. Een slice die is gebaseerd op de basisstroom raakt namelijk gegarandeerd een geheel gebruiksscenario van begin tot eind en zal de meest directe manier zijn voor de gebruiker om zijn doel te bereiken.

Schat de slice in en begin met de ontwikkeling. Vervolgens kunnen aanvullende slices van de use case worden opgepakt, net zolang tot er genoeg slices ontwikkeld zijn om een bepaalde gebruiker een bruikbare oplossing te bieden. Dit kan worden herhaald voor iedere andere use case die nodig is om een bruikbaar systeem op te kunnen leveren.

Een use-case slice hoeft geen complete stroom en al zijn testgevallen te bevatten. De eerste slice kan bestaan uit alleen de basisstroom en slechts één testgeval. Met aanvullende slices kun je dan de basisstroom compleet maken en veiligstellen dat alle testgevallen worden meegenomen. Dit is een zeer flexibel opdeelmechanisme, dat je in staat stelt om slices te maken die precies groot of klein genoeg zijn om de ontwikkeling goed te kunnen aansturen.

Slices zijn meer dan requirements en testgevallen alleen

Wanneer we een systeem bouwen in slices, dan is het niet voldoende om alleen de requirements op te delen. Hoewel use cases traditioneel werden gebruikt om requirements te begrijpen en vast te leggen, hebben ze altijd een breder toepassingsgebied gehad. Figuur 3 geeft weer dat use-case slices méér doorsnijden dan alleen de requirements. Ze doorsnijden ook alle andere aspecten van een systeem en zijn documentatie.



FIGUUR 3: EEN USE-CASE SLICE IS MEER DAN ALLEEN EEN DOORSNEDE VAN REQUIREMENTS

Figuur 3 laat aan de linkerkant een use-case slice zien. Deze slice is afkomstig uit een van de use cases uit de kolom “Use Case”. De volgende kolom laat de slice zien vanuit ontwerpperspectief waarbij de betrokken ontwerpelementen inzichtelijk worden gemaakt. De volgende kolom laat de slice zien vanuit codeerperspectief. Hier kun je zien welke delen van code nodig zijn voor het programmeren van de slice. De laatste kolom laat de slice zien vanuit testperspectief. Hierbij doorsnijdt de slice de testset. Dit zijn zowel de logische testgevallen (test cases) als de fysieke testgevallen (test scripts) die nodig zijn om de test daadwerkelijk uit te voeren, en de gegenereerde testresultaten.

Deze aanpak levert je niet alleen traceerbaarheid op van de requirements naar de code en de testset. Het helpt je ook om het juiste systeem te ontwikkelen. Wanneer je start met een slice dan is het van belang om de impact van die slice op het ontwerp en de code te doorzien. Zijn nieuwe systeemelementen noodzakelijk? Kan het worden gebouwd door alleen bestaande elementen aan te passen? Als de impact te groot is, kun je zelfs besluiten om de slice helemaal niet te bouwen! Met een basaal systeemontwerp kan zo’n analyse eenvoudig en snel worden uitgevoerd. Het is een heel goede manier om te doorzien wat het toevoegen van een slice betekent.

Door ieder aspect van het systeem slice voor slice te doorgronden, helpen use cases je bij alle verschillende aandachtsgebieden van het systeem, zoals de user interface, de architectuur, testen en de planning. Ze bieden een manier om de requirements te linken aan (1) de systeemonderdelen waarin ze gerealiseerd worden, (2) de testen om te verifiëren of ze succesvol zijn geïmplementeerd en (3) de release- en projectplannen die richting geven aan de ontwikkelwerkzaamheden. Use-Case 2.0 bevat

hiervoor een speciaal werkproduct, de use-case realisatie. Aan iedere use case kan een use-case realisatie worden toegevoegd om van de impact op andere aspecten van het systeem vast te leggen.

Use-Case Slices: Het belangrijkste onderdeel van Use-Case 2.0

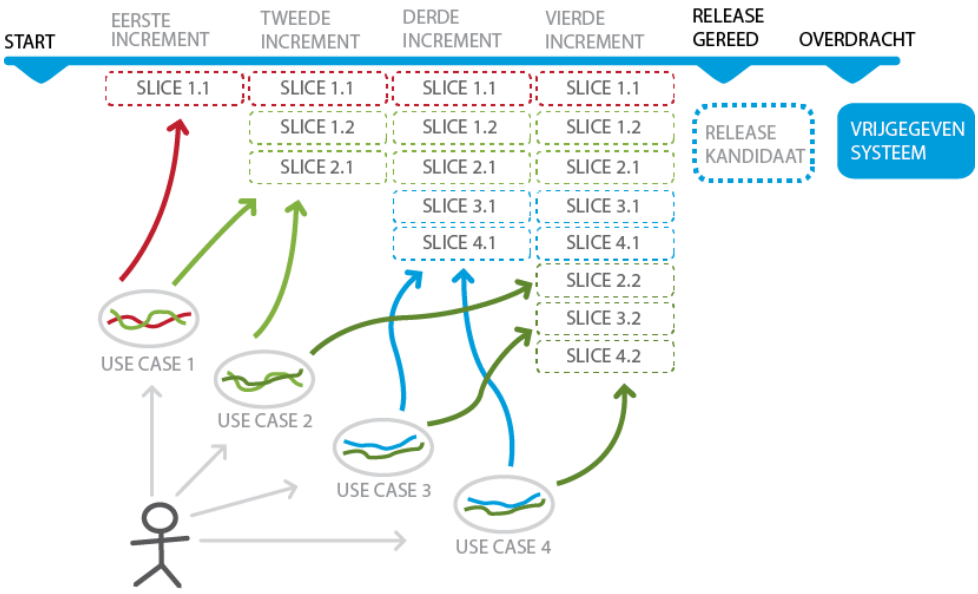
De use-case slice is een even integraal deel van Use-Case 2.0 als de use case zelf. Het zijn de slices die mogelijk maken dat use cases kunnen worden opgedeeld in passende hoeveelheden werk dat door een ontwikkelteam kan worden opgepakt. Stel je voor dat je deel uitmaakt van een klein team dat iedere twee weken werkende software oplevert. Een hele use case is waarschijnlijk te groot om binnen twee weken te kunnen ontwikkelen en testen. Met use-case slices ontstaat een heel ander verhaal, omdat je de grootte van slices kunt aanpassen aan de capaciteit van een team. Use-case slices stellen het team ook in staat om onnodige requirements achterwege te laten. Hierdoor is het team in staat om binnen de kortst mogelijke tijd een waardevol en bruikbaar systeem te leveren.

Principe 5: Lever het systeem incrementeel op

De meeste softwaresystemen evolueren via meerdere generaties. Ze worden niet in één keer ontwikkeld. De ontwikkeling gaat via een reeks van releases, waarbij iedere nieuwe release voortbouwt op de vorige. Ook de releases zelf worden meestal niet in één slag ontwikkeld, maar evolueren via een reeks van incrementen. Ieder increment levert een demonstreerbare of bruikbare versie van het systeem op. Ieder increment bouwt voort op het vorige increment om zo functionaliteit toe te voegen of de kwaliteit van het voorgaande te verbeteren. Dit is in onze visie de manier waarop alle systemen zouden moeten worden ontwikkeld.

De use cases zelf kunnen ook al te groot zijn om in één slag te worden opgeleverd. We hebben bijvoorbeeld in het eerste increment van een telefoniesysteem waarschijnlijk niet alle manieren nodig om lokaal te bellen. De meest basale faciliteiten zijn meestal genoeg om ons op gang te helpen. De meer optionele of speciale manieren om lokaal te bellen, zoals bellen op kosten van de ontvanger (collect call) of opnieuw het laatst gedraaide nummer bellen (redial), kunnen met latere incrementen worden toegevoegd. Door de use cases op te delen in slices kunnen we een fijnmazigere controle bereiken waarmee we de toegevoegde waarde van iedere release kunnen maximaliseren.

Figuur 4 laat de incrementele ontwikkeling van een systeem zien. Het eerste increment bevat maar één slice: de eerste slice van use case 1. Het tweede increment voegt een andere slice van use case 1 toe en de eerste slice van use case 2. Meer slices worden toegevoegd om zo het derde en vierde increment te vormen. Het vierde increment wordt compleet en bruikbaar genoeg geacht om te worden uitgeleverd.



FIGUUR 4: USE CASES, USE-CASE SLICES, INCREMENTEN EN RELEASES

Use cases zijn een uitstekend hulpmiddel voor releaseplanning. Werken op het use-case niveau stelt je in staat om hele clusters requirements door te schuiven naar volgende releases. Door besluiten te nemen op use-case niveau kun je snel het grotere geheel schetsen om vervolgens te focussen op die delen die moeten worden opgepakt tijdens de komende release.

Use-case diagrammen zijn een probaat middel om teamdoelstellingen inzichtelijk te maken. Je kunt dit doen door een diagram te maken met alleen de use cases die worden opgepakt in de huidige release. Use cases die in volgende releases aan bod komen, worden dan achterwege gelaten. Zulke use-case diagrammen weerspiegelen helder het thema van een release. Door ze aan de muur van een projectkamer te hangen, zijn ze voor iedereen duidelijk zichtbaar.

Use-case slices zijn een prima hulpmiddel voor het maken van kleinere incrementen op weg naar een complete release. Ze stellen je in staat om onafhankelijk implementeerbare en testbare delen op te pakken en zo het systeem increment voor increment te laten groeien.

Principe 6: Sluit aan op de behoeften van het team

Helaas is er geen pasklare oplossing die alle uitdagingen van softwareontwikkeling dekt. Verschillende teams en verschillende omstandigheden vereisen verschillende aanpakken en verschillende mate van detail. Dit impliceert dat geselecteerde werkwijzen en technieken afdoende aanpasbaar moeten zijn om te kunnen voorzien in de actuele behoeften van het team.

Dit geldt voor nagenoeg alle werkwijzen, inclusief degene die je kiest om requirements te delen en om softwareontwikkeling aan te sturen. Zo zijn bijvoorbeeld lichtgewicht requirements (zoals user stories) uiterst effectief in geval van nauwe samenwerking met gebruikers. Gebruikers lichten hun requirements dan mondeling toe en geven het ontwikkelteam tijdig antwoord op vragen die naar boven komen. Wanneer deze manier van samenwerken niet mogelijk is, bijvoorbeeld omdat de gebruikers niet afdoende beschikbaar zijn, dan zullen de requirements in meer detail moeten worden vastgelegd en onvermijdelijk zwaarlijviger worden. Zo zijn er nog vele andere omstandigheden waarin een team behoefte kan hebben aan meer gedetailleerde requirements. Het is echter goed om je te realiseren dat het onbelangrijk is om alle mogelijke omstandigheden te bedenken waarbij een lichtgewicht aanpak niet past. Belangrijker is te erkennen dat werkwijzen schaalbaar moeten zijn.

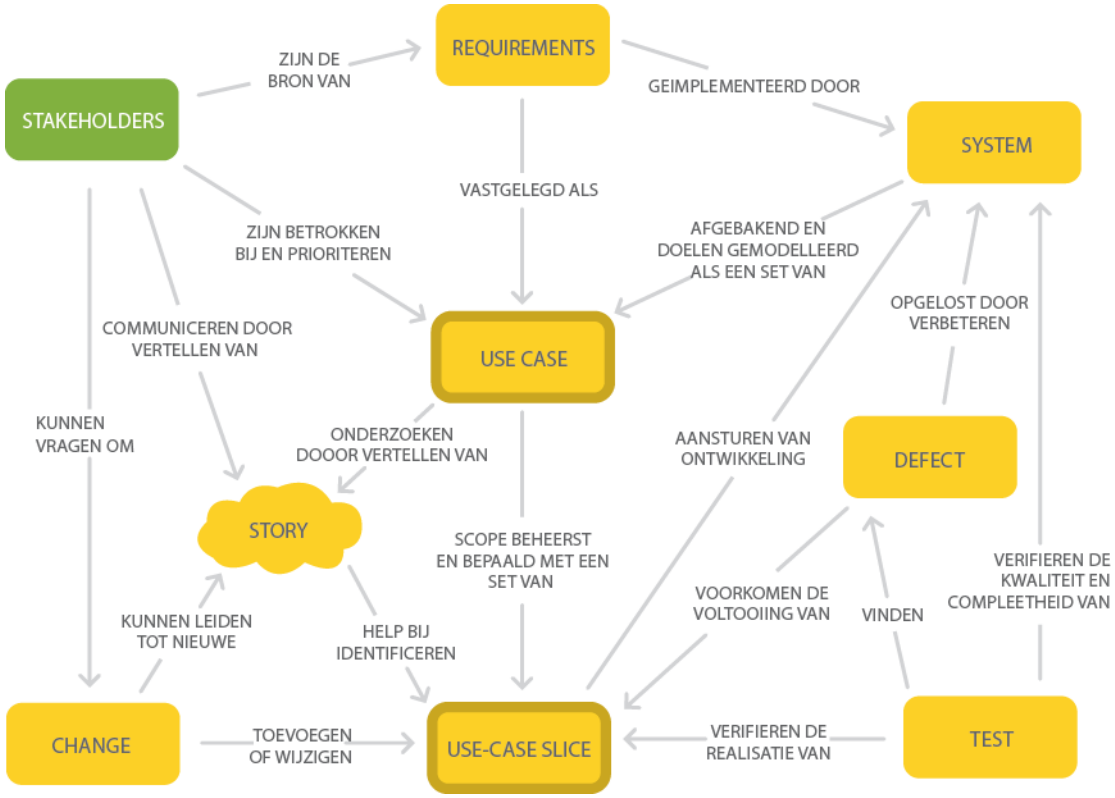
Use-Case 2.0 is ontworpen met dit gegeven in het achterhoofd, en het is net zo lichtvoetig als je zelf wilt. Kleine intensief samenwerkende teams kunnen werken met zeer lichtvoetige use-case beschrijvingen die alleen de essentie van verhalen bevatten. Deze kunnen zijn vastgelegd op handgeschreven indexkaartjes. Daarentegen kunnen grote geografisch gespreide teams meer gedetailleerde use-case beschrijvingen in de vorm van documenten gebruiken. Het is aan het team om te bepalen wat het minimum detailniveau is. En het is ook aan het team om te bepalen wanneer meer detail nodig is en hoe dit wordt verkregen, bijvoorbeeld wanneer blijkt dat bepaalde problemen niet kunnen worden opgelost op basis van alleen de essentie.

Use-Case 2.0 Basisprincipes

Use-Case 2.0 bevat een set dingen om mee te werken en een set dingen om te doen.

Dingen om mee te werken

Use-Case 2.0 gaat over de requirements, over het te ontwikkelen systeem dat moet voldoen aan die requirements, en over de testen die worden gebruikt om aan te tonen dat het systeem voldoet aan die requirements. De use case, het verhaal en de use-case slice staan centraal. Zij bevatten de requirements en stellen ons in staat om de systeemontwikkeling goed aan te sturen. Figuur 5 geeft de samenhang tussen deze concepten weer. Het laat ook zien welke impact wijzigingen (changes) en bevindingen (defects) hebben op het gebruik van Use-Case 2.0.



FIGUUR 5: USE-CASE 2.0 CONCEPTEN IN SAMENHANG

In het bovenstaande figuur staan stakeholders (Nederlands: belanghebbenden) voor mensen, groepen of organisaties die invloed uitoefenen op, of worden geraakt door, het softwaresysteem. Requirements zijn wat het systeem moet doen om in de behoeften van deze belanghebbenden te voorzien. Het is belangrijk erachter te komen wat van het softwaresysteem wordt verwacht, dit begrip te delen met de belanghebbenden en de teamleden, en het te gebruiken voor de ontwikkeling van het nieuwe systeem. In Use-Case 2.0 worden de requirements vastgelegd in de vorm van een set use cases. Hiervan wordt de scope bepaald en beheerst met behulp van een set use-case slices. Iedere door de belanghebbenden gevraagde wijziging heeft uitbreidingen of wijzigingen op de set use cases en use-case slices tot gevolg.

Het systeem representeert het te ontwikkelen systeem. Het is meestal een softwaresysteem. Use-Case 2.0 kan echter ook worden gebruikt voor de ontwikkeling van een nieuw bedrijf (waarbij het bedrijf wordt

gezien als een systeem) of een combinatie van hardware- en softwaresystemen (embedded systemen). Het is het systeem dat invulling geeft aan de requirements. Het systeem is daarom ook het onderwerp van het use-case model. De kwaliteit en de compleetheid van het systeem worden geverifieerd door een set testen. Met deze testen wordt ook geverifieerd dat use-case slices succesvol zijn ontwikkeld. Mochten er tijdens het testen bevindingen (defects) worden ontdekt, dan blokkeren die de voltooiing van use-case slices, totdat de bevindingen zijn verholpen en het systeem is verbeterd.

Het vertellen van verhalen vormt de brug tussen belanghebbenden, de use cases en de use-case slices. Het is de wijze waarop belanghebbenden hun requirements uiten en use cases verkennen. Het doorgronden van de verhalen is het mechanisme om de juiste use-case slices te vinden en de ontwikkeling van het systeem aan te sturen.

Use Cases

Een use case representeert alle manieren waarop het systeem kan worden gebruikt om een bepaald doel te behalen voor een bepaalde gebruiker. De verzameling use cases geeft ons alle zinvolle manieren om het systeem te gebruiken.

Een use case is:

- Een opeenvolgende reeks van acties die een systeem uitvoert om een waarneembaar waardevol resultaat te leveren aan een bepaalde gebruiker
- Het gedrag van het systeem dat in samenwerking met een gebruiker iets waardevols oplevert voor die gebruiker
- De kleinste eenheid van handelingen dat een betekenisvol resultaat oplevert aan de gebruiker
- De context voor een set samenhangende requirements.

FIGUUR 6: DE LEVENSCYCLUS VAN EEN USE CASE

Om een use case te begrijpen vertellen we verhalen. De verhalen gaan over hoe doelen succesvol worden behaald en hoe problemen op de weg daarnaartoe worden afgehandeld. Ze zorgen ervoor dat we begrijpen waar een use case over gaat en dat we ze slice voor slice kunnen ontwikkelen.

Zoals Figuur 6 laat zien, doorloopt een use case verschillende stadia, van zijn identificatie tot zijn uiteindelijke implementatie door het systeem. De verschillende stadia zijn belangrijke mijlpalen gedurende de begripsvorming en implementatie van de use case:

1) Doel vastgesteld: Wanneer het doel van de use case is vastgesteld.
2) Verhaallijn helder: Wanneer de structuur van de use-case beschrijving afdoende duidelijk is voor het team om werk te identificeren en de eerste use-case slices te implementeren.
3) Eenvoudigste verhaal beschikbaar: Wanneer het eenvoudigste verhaal geïmplementeerd is waarmee de gebruiker zijn doel kan behalen.
4) Afdoende verhalen beschikbaar: Wanneer afdoende verhalen zijn geïmplementeerd zodat gebruikers een bruikbare oplossing wordt geboden.
5) Alle verhalen beschikbaar: Wanneer alle verhalen van de use case zijn geïmplementeerd.

Dit wordt bereikt door de use case slice voor slice te implementeren. De stadia bieden een eenvoudige manier om de voortgang te bepalen gedurende het doorgronden en ontwikkelen van de use case.

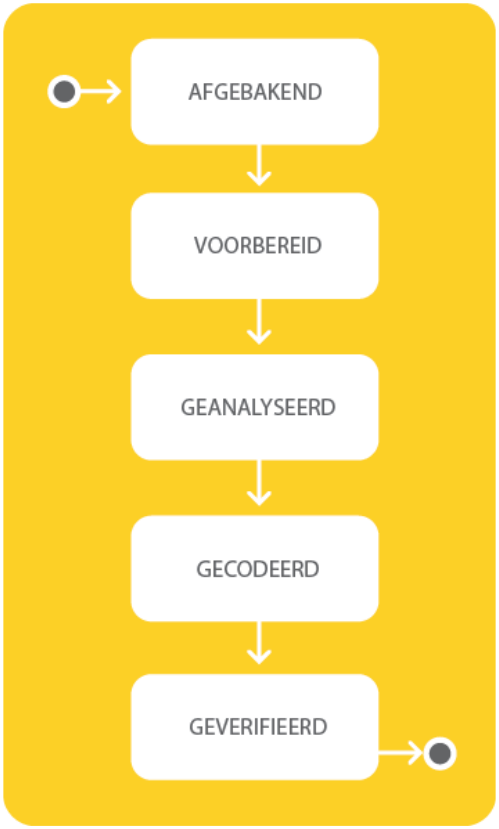
Use-Case Slices

Use cases omvatten veel samenhangende verhalen die verschillen in belang en prioriteit. Er zijn vaak te veel verhalen om in één release te kunnen opleveren, en meestal ook te veel om in een enkel increment aan te kunnen werken. Daarom hebben we behoefte aan een manier om de use cases op te delen in kleinere eenheden die:

- Ons in staat stellen om te bepalen welke delen van de use case op welk moment moeten worden opgeleverd,
- Het team passende werkeenheden bieden voor ontwikkeling en testen,
- Ons kleine en gelijksoortige werkeenheden geven die snel te ontwikkelen zijn.

Een use-case slice bevat een of meer verhalen uit een use case, die samen een werkeenheid vormen met een duidelijke toegevoegde waarde voor de klant. De slice fungeert als kapstok voor al het werk dat nodig is voor de ontwikkeling van de geselecteerde verhalen. Zoals eerder aangegeven omvat een use-case slice meer dan alleen requirements en testgevallen. Een use-case slice doorloopt alle stadia van de ontwikkeling, van identificatie tot en met acceptatie.

De use-case slice is het belangrijkste element van Use-Case 2.0. Dit komt doordat ze zowel worden gebruikt voor het vaststellen van requirements als voor de aansturing van de ontwikkeling van het bijbehorende systeem.



FIGUUR 7: DE LEVENSCYCLUS VAN EEN USE-CASE SLICE

Use-case slices:

- Zorgen ervoor dat use cases kunnen worden opgedeeld in kleinere en onafhankelijk opleverbare werkeenheden.
- Zorgen ervoor dat requirements binnen een set use cases kunnen worden geordend, geprioriteerd en parallel kunnen worden ontwikkeld.
- Zorgen voor samenhang tussen de verschillende systeemmodellen (requirements, analyse, ontwerp, code en test) zoals onderkend in use-case driven development.

Zoals Figuur 7 laat zien doorloopt een use-case slice diverse stadia van zijn initiële identificatie tot acceptatie. De stadia zijn belangrijke mijlpalen gedurende de begripsvorming, de ontwikkeling en het testen van de use-case slice:

1) Afgebakend: wanneer omvang en bereik van de behandelde verhalen duidelijk zijn.
2) Voorbereid: wanneer de slice dusdanig is uitgewerkt in de use-case beschrijving en de testgevallen dat duidelijk is wat het succesvol ontwikkelen van de slice betekent.
3) Geanalyseerd: wanneer de impact op de systeemcomponenten duidelijk is en de geraakte delen gereed zijn voor coderen en unit-testen.
4) Gecodeerd: wanneer de slice onderdeel is geworden van het softwaresysteem en gereed is om getest te worden.
5) Geverifieerd: wanneer de slice gereed is bevonden en kan worden opgenomen in een release.

Deze stadia bieden een eenvoudige manier om de voortgang te bepalen tijdens het doorgronden en de ontwikkeling van use-case slices. Ze markeren tevens perioden van inactiviteit waarop de slice potentieel

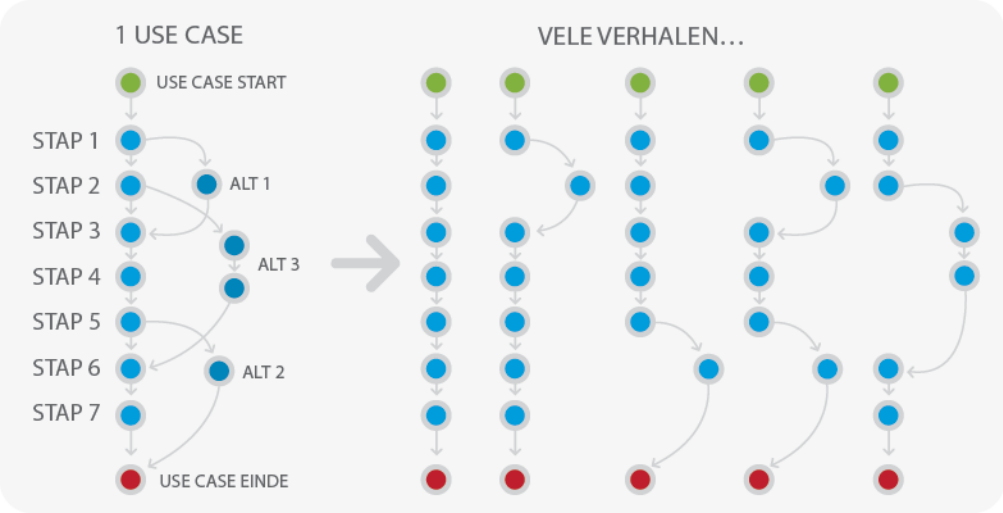
kan worden overgedragen naar een ander persoon of team. Een toevallige waarnemer zou dit, op basis van de gegeven stadia, als een watervalproces kunnen opvatten: afbakenen> voorbereiden> analyse> coderen> verifiëren. Het grote verschil is echter dat in een watervalproces alle requirements worden voorbereid voorafgaand aan de analyse en dat alle analysewerkzaamheden zijn afgerond voorafgaand aan het coderen en dat alle codeerwerkzaamheden zijn afgerond voorafgaand aan de verificatie. Realiseer je dat we hier te maken hebben met een individuele use-case slice. Bij een set use-case slices kunnen al deze activiteiten parallel worden uitgevoerd.

Terwijl de ene use-case slice wordt geverifieerd, kan een andere slice worden gecodeerd. Tegelijkertijd kan een derde slice wordt gecodeerd en een vierde geanalyseerd. In het volgende hoofdstuk, waarin we Use-Case 2.0 zullen positioneren ten opzicht van verschillende ontwikkelaanpakken, zullen we hier dieper op ingaan.

Verhalen

We verkennen een use case met onze belanghebbenden door middel van het vertellen van verhalen. Ieder verhaal dat van waarde is voor de gebruikers of andere belanghebbenden is een route door één van de use cases. Een verhaal kan functioneel van aard zijn of juist een kwaliteitsaspect belichten.

Een verhaal wordt weergegeven door een deel van de use-case beschrijving, bestaande uit een of meerdere stromen en eventueel aanvullende requirements, aangevuld met een of meer testgevallen. Begrip van de structuur van de use-case beschrijving is de sleutel tot het vinden van effectieve verhalen. Het netwerk van stromen kun je zien als een routekaart die alle verhalen opsomt die tot de use case behoren. Figuur 8 laat de samenhang zien tussen de stromen binnen de use-case beschrijving en de verhalen die ze beschrijven.



FIGUUR 8: DE SAMENHANG TUSSEN STROMEN EN VERHALEN

Aan de linkerkant van de figuur is de basisstroom als een lineaire set stappen weergegeven. Alternatieve stromen kun je zien als aftakkingen van de basisstroom, en worden altijd gedefinieerd als variatie op de basisstroom. Aan de rechterkant van de figuur worden enkele verhalen van de use case getoond. Ieder verhaal doorloopt een of meerdere stromen, beginnend bij de start van de basisstroom en eindigend bij het einde van de basisstroom. Dit zorgt ervoor dat alle verhalen gerelateerd zijn aan hetzelfde doel. Het zorgt er ook voor dat verhalen compleet, betekenisvol en complementair zijn omdat ze voortborduren op het eenvoudige verhaal zoals weergegeven door de basisstroom. Tot slot zal ieder verhaal een of meerdere testgevallen hebben.

Er zijn twee algemene aanpakken voor het identificeren van verhalen en het opstellen van de use-case beschrijving:

Top-down: Sommige mensen houden van een top-down aanpak waarbij ze 1) de use case identificeren, 2) de basisstroom schetsen en 3) op basis van de basisstroom brainstormen over alternatieven. Dit structureert de use-case beschrijving en stelt hen in staat om de verhalen te identificeren.

Bottom-up: Andere mensen gebruiken een bottom-up aanpak. Hierbij wordt gestart met een brainstorm van enkele verhalen die vervolgens worden geclusterd naar een thema met als doel de use cases te identificeren. De set verhalen wordt vervolgens bestudeerd om de basisstroom en alternatieven te bepalen. De use-case structuur helpt dan bij het identificeren van ontbrekende verhalen en zorgt ervoor dat alle verhalen goed zijn vormgegeven en complementair zijn.

Het beste is de aanpak te kiezen die het best past bij de belanghebbenden. De aanpakken kunnen ook worden gecombineerd; eerst top-down en dan vervolgens bottom-up voor alle nieuwe verhalen.

De verhalen vormen een handzaam middel om de juiste use-case slices en, nog belangrijker, de juiste testgevallen te vinden. Het is niet nodig om de status van de verhalen zelf te bewaken. Het uitvoeren van testgevallen maakt namelijk de compleetheid van het systeem inzichtelijk, evenals de voortgang van de use case en de use-case slices. De use-case slices worden gebruikt om de ontwikkeling aan te sturen.

Bevindingen en Wijzigingen

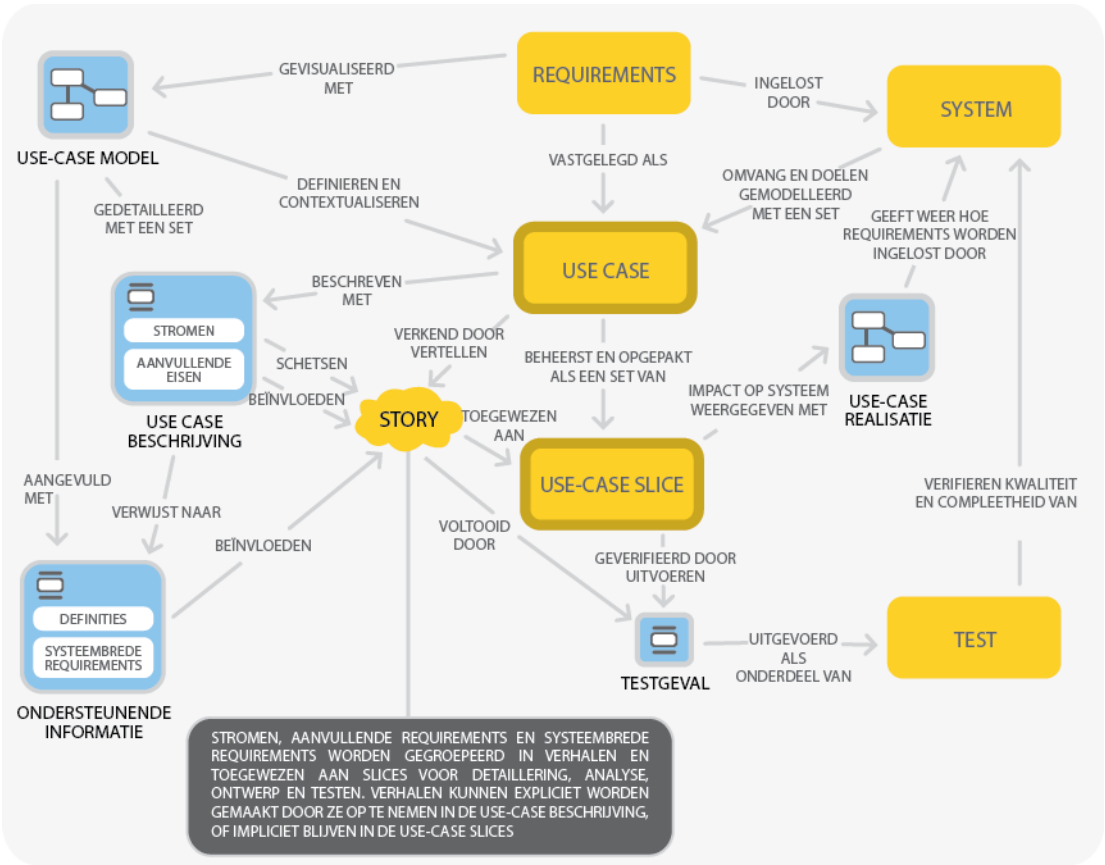
Hoewel geen direct onderdeel van Use-Case 2.0, is het belangrijk te begrijpen hoe bevindingen en wijzigingen samenhangen met use cases en de use-case slices.

Door belanghebbenden voorgestelde wijzigingen worden geanalyseerd in het licht van het huidige use-case model, de use cases en use-case slices. Dit zorgt ervoor dat de impact van de wijziging snel inzichtelijk wordt. Een nieuwe use case toevoegen aan het systeem is bijvoorbeeld een grote wijziging omdat het de doelstellingen en de toegevoegde waarde van het systeem raakt. Een wijziging op een bestaande use case is daarentegen meestal veel kleiner. Dit geldt des te meer als het een verhaal betreft dat nog niet is toegewezen aan een slice en nog niet is voorbereid, geanalyseerd, gecodeerd en geverifieerd.

Bevindingen worden afgehandeld door te bepalen met welke use-case slices, en hieruit voortvloeiend, met welke testgevallen deze zijn ontdekt. Als ze gevonden zijn tijdens het coderen of de verificatie van een use-case slice, dan mag de slice niet ‘door’ totdat de bevinding is opgelost en de testgevallen positief zijn uitgevoerd. Mochten ze later tijdens regressietesten worden gevonden, dan biedt de relatie tussen het gefaalde testgeval en de use case uitkomst. Deze traceerbaarheid zorgt ervoor dat de impact van de bevinding op de gebruikers en de bruikbaarheid van het systeem snel kan worden vastgesteld.

Werkproducten

De use cases en de use-case slices worden ondersteund door een aantal werkproducten die het team helpt de slices onderling te delen, te doorgronden en te documenteren. Figuur 9 laat de vijf werkproducten binnen Use-Case 2.0 zien (in blauw) en hun samenhang met de requirements, de use cases, de use-case slices, verhalen, testen en het systeem (in geel).



FIGUUR 9: DE USE-CASE 2.0 WERKPRODUCTEN

Het use-case model visualiseert de requirements in de vorm van een set use cases. Het geeft een overzichtelijk beeld van het te bouwen systeem als geheel. Het model definieert de use cases en biedt een context voor het uitwerken van individuele use cases. De use cases worden verkend door verhalen te vertellen. Iedere use case wordt beschreven door 1) een use-case beschrijving die zijn verhalen schetst en 2) een set testgevallen die de verhalen compleet maken. De verhalen worden beschreven in de vorm van een set stromen. Deze kunnen worden aangevuld met een set aanvullende requirements die de verhalen beïnvloeden, je helpen om de juiste verhalen toe te wijzen aan de use-case slices voor ontwikkeling, en allerbelangrijkst, de juiste testgevallen bepalen.

Het use-case model wordt aangevuld met aanvullende informatie. Dit werkproduct bevat de begrippen (jargon) die worden gebruikt in het use-case model en bij het schetsen van verhalen in de use-case beschrijvingen. Het bevat ook systeembrede requirements die van toepassing zijn op alle use cases. Ook die beïnvloeden de verhalen die worden gekozen uit de use-case en die worden toegewezen aan use-case slices om te worden ontwikkeld.

Je zou verontrust kunnen raken door het gebrek aan expliciete werkproducten voor het vastleggen van verhalen en use-case slices. In onze visie zijn die niet nodig omdat ze volledig worden beschreven door

andere werkproducten. Desgewenst kun je de aan een use case gerelateerde verhalen opsommen in een extra onderdeel in een use-case beschrijving, maar dat is in de meeste gevallen overbodig.

Werken met de use case en use-case slices

Naast het opstellen en bewaken van de status van werkproducten, dien je ook de status en eigenschappen van de use cases en use-case slices te bewaken. Dit kan op vele manieren worden gedaan en met vele hulpmiddelen (tools). De stadia kunnen worden bewaakt met Post-its© of spreadsheets. Indien meer formaliteit vereist is, kunnen ook commercieel beschikbare hulpmiddelen voor requirements management, wijzigingenbeheer en bevindingbeheer worden gebruikt.

Figuur 10 laat een use case en enkele van zijn slices zien, vastgelegd met Post-Its©.

FIGUUR 10: EIGENSCHAPPEN VAN EEN USE CASE EN ZIJN SLICES OP POST-ITS©

De getoonde use case is use case '7 Zoek en koop' van een webwinkelsysteem. Slices 1 en 2 van de use case zijn afgeleid van individuele verhalen van de basisstroom: 'Kies en Koop 1 Product' en 'Kies en Koop 100 Producten'. Slice 3 is gebaseerd op meerdere verhalen die te maken hebben met de beschikbaarheid van de verschillende ondersteunende systemen die betrokken zijn bij de use case. Deze verhalen omvatten een aantal alternatieve stromen.

De essentiële eigenschappen van een use case zijn naam, status en prioriteit. In dit geval is het populaire MoSCoW (Must, Should, Could, Would) prioriteringsschema gebruikt. De use case moet ook worden geschat. In dit voorbeeld is een eenvoudig mechanisme gebruikt dat werkt op basis van relatieve omvang en complexiteit.

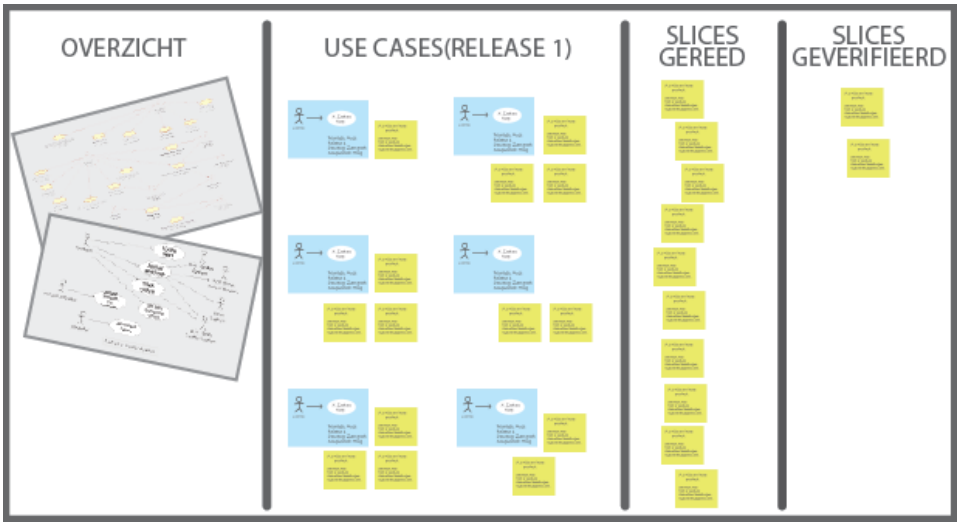
De essentiële eigenschappen van een use-case slice zijn:

1) Een opsomming van zijn verhalen.
2) Verwijzingen naar de use case en de stromen die de verhalen vormen.
3) Verwijzingen naar de testen en de testgevallen die zullen worden gebruikt om de ontwikkeling te verifiëren.
4) Een inschatting van het werk dat nodig is om de slice te ontwikkelen en te testen.

In dit voorbeeld zijn de verhalen gebruikt om de slices een naam te geven. De referenties naar de use case zijn impliciet bekend door de nummering van de slices en de opsomming van stromen. De

schattingen zijn later toegevoegd nadat het team hierover is geraadpleegd. Dit zijn de grote getallen rechtsonder in iedere slice. In dit geval heeft het team met behulp van planning poker een relatieve schatting gemaakt op basis van *story points*; 5 story points voor slices 7.1 en 7.2 en 13 story points voor slice 7.3 waarvan het team denkt dat het twee keer zo veel werk vereist als de andere slices. De schatting had ook kunnen worden gemaakt met ideale mandagen, T-shirt maten (XS,S,M,L,XL,XXL) of iedere andere populaire schattingstechniek.

De use cases en de use-case slices dienen ook te worden gesorteerd, zodat de belangrijkste als eerste worden opgepakt. Figuur 11 laat zien hoe met Post-Its© een eenvoudige backlog (werkvoorraad) kan worden gemaakt op een whiteboard. Wanneer je van links naar rechts leest dan zie je 1) een overzicht van het grotere geheel , weergegeven door de use-case diagrammen die de scope van het hele systeem en de eerste release inzichtelijk maken, 2) de use cases die zijn toegekend aan de eerste release en enkele van hun use-case slices die zijn geïdentificeerd maar nog niet gedetailleerd en gesorteerd, 3) de gesorteerde lijst van slices die gereed zijn om te worden ontwikkeld tijdens de release en tot slot 4) die slices die het team succesvol heeft ontwikkeld en geverifieerd.



FIGUUR 11: USE CASES EN USE-CASE SLICES GEBRUIKEN OM EEN PRODUCT BACKLOG OP TE ZETTEN

Figuur 11 is louter illustratief want er zijn vele andere manieren om een backlog vorm te geven en te werken met requirements. Er zijn ook teams die zich zorgen maken dat Post-Its© van het whiteboard vallen. Deze teams gebruiken vaak een eenvoudige spreadsheet om de toestand van hun use-cases en use-case slices te bewaken. Figuur 12 en 13 geven daar een voorbeeld van.

USE CASE	NAAM	RELEASE	PRIORITEIT	STATUS	OMVANG	COMPLEXITEIT
7	ZOEK EN KOOP	1	1 - MUST	VERHAALLIJN HELDER	ZEER GROOT	HOOG
14	KRIJG SPECIALE AANBIEDINGEN	1	1 - MUST	VERHAALLIJN HELDER	MIDDEL	MIDDEL
17	ONDERHOUDEN PRODUCTEN	1	1 - MUST	VERHAALLIJN HELDER	GROOT	HOOG
12	VOLG BESTELLINGEN	1	2 - SHOULD	VERHAALLIJN HELDER	GROOT	LAAG
13	ZOEK WINKEL	1	2 - SHOULD	VERHAALLIJN HELDER	KLEIN	LAAG
16	MAAK AANBIEDINGEN	1	2 - SHOULD	VERHAALLIJN HELDER	MIDDEL	HOOG
11	INZIEN BESTELHISTORIE	1	3 - COULD	VERHAALLIJN HELDER	KLEIN	MIDDEL
1	ONTWERPEN HUIS			DOEL VASTGESTELD		
2	ONTWERPEN INTERIEUR			DOEL VASTGESTELD		
3	ONTWERPEN TUIN			DOEL VASTGESTELD		
4	BEPLANTEN TUIN			DOEL VASTGESTELD		

FIGUUR 12: HET USE-CASE WERKBLAD VAN EEN EENVOUDIG USE-CASE VOLGSYSTEEM

USE CASE		SLICE	STORIES	STROOM	TEST CASES	STATUS	VOLGORDE	SCHATTING (STORY POINTS)	DOEL RELEASE
7	ZOEK EN KOOP	7.1	KOOP 1 PRODUCT	BF	7.1.1	VOORBEREID	1	13	1
7	ZOEK EN KOOP	7.2	KOOP MEERDERE PRODUCTEN	BF	7.2.1, 7.2.3	VOORBEREID	2	13	1
7	ZOEK EN KOOP	7.4	AFHANDELEN BETALING EN LEVERING	A4, A5, A6	7.3.1, 7.3.2	AFGEBAKEND	3	3	1
17	ONDERHOUDEN PRODUCTEN	17.1	TOEVOEGEN NIEUWE PRODUCT	BF	17.1.1, 17.1.2	AFGEBAKEND	4	20	1
1	ONTWERPEN HUIS	1.1	TOEVOEGEN EERSTE HUIS	BF	1.1.1, 1.1.2, 1.1.3	AFGEBAKEND	5	5	1
2	ONTWERPEN INTERIEUR	2.1	ONTWERP KAMER	BF, A3		GEVERIFIERD	6	5	1
2	ONTWERPEN INTERIEUR	2.4	KOOP BENODIGDHEDEN	A6		AFGEBAKEND	7	3	1
7	ZOEK EN KOOP	7.5	AFHANDELEN HULPSYSTEMEN NIETBESCHIKBAAR	A9, A11, A12		AFGEBAKEND	8	5	1
7	ZOEK EN KOOP	7.6	PRODUCT NIET OP VOORRAAD	A7		AFGEBAKEND	9	13	1
17	ONDERHOUDEN PRODUCTEN	17.2	AFHANDELEN INVALIDE PRODUCTIDETAILS	A2, A3		AFGEBAKEND	10	5	1
7	ZOEK EN KOOP	7.7	STOPPEN MET WINKELEN	A14		AFGEBAKEND	11	20	1
5	BEOORDELEN ONTWERP	5.1	NAVIGEER DOOR ONTWERP	BF, A1		AFGEBAKEND	12	8	1
5	BEOORDELEN ONTWERP	5.2	AFHANDELEN NAVIGATIEFOUTEN	A2		AFGEBAKEND	13	2	1
1	ONTWERPEN HUIS	1.2	TOEVOEGEN OF VERWIJDEREN VLOER	A2, A5	1.2.1, 1.2.2, 1.2.3	AFGEBAKEND	14	1	1

FIGUUR 13: HET USE-CASE SLICE WERKBLAD VAN EEN EENVOUDIG USE-CASE VOLGSYSTEEM

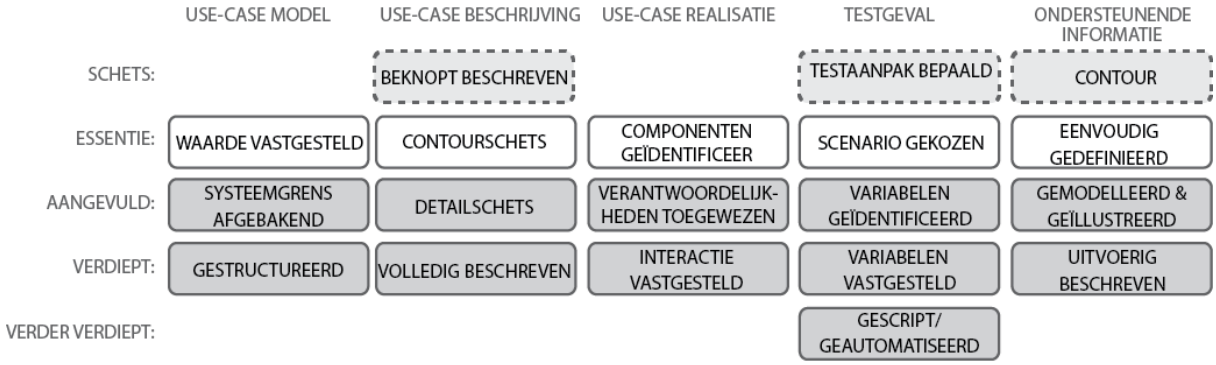
Dit zijn voorbeelden om je op weg te helpen. De use cases en de use-case slices staan centraal bij alles wat het team doet, dus ongeacht je aanpak, zorg ervoor dat ze tastbaar en zichtbaar zijn voor het team. Voel je vrij om attributen toe te voegen waar je dat nodig acht. Denk bijvoorbeeld aan de bron van de use case of zijn eigenaar, of het increment binnen de release waarin een slice moet worden opgepakt.

Voltooien van de werkproducten

Behalve het bewaken van use cases en use-case slices zou je, op zijn minst, ook de ondersteunende werkproducten moeten schetsen en delen met belanghebbenden.

Alle werkproducten zijn voorzien van een aantal detailniveaus. Het eerste detailniveau definieert de essentie, de minimale hoeveelheid informatie die nodig is om de werkwijze te laten werken. De meer gedetailleerde niveaus helpen het team om te kunnen gaan met bijzondere omstandigheden waar ze mee te maken kunnen krijgen. Dit stelt bijvoorbeeld kleine intensief samenwerkende teams in staat om zeer lichtgewicht use-case beschrijvingen te hebben in de vorm van indexkaartjes. Grote geografisch gescheiden teams kunnen dan meer gedetailleerde use-case beschrijvingen in de vorm van documenten gebruiken. Het team kan zo zelf het juiste detailniveau bepalen, benodigd voor louter overleg of het nauwgezet vaststellen van belangrijke requirements of kritieke veiligheidseisen. Het is aan het team te bepalen of ze meer nodig hebben dan de essentie, om zo op een natuurlijke manier detail toe te voegen wanneer ze problemen tegenkomen waarbij alleen de essentie niet afdoende is.

Figuur 14 geeft een overzicht van de detailniveaus van de Use-Case 2.0 werkproducten. Het laagste detailniveau is bovenin de tabel weergegeven. Het detailniveau neemt toe naarmate je verder naar beneden gaat in de kolommen, waarbij de inhoud steeds verder verrijkt en verdiept wordt.



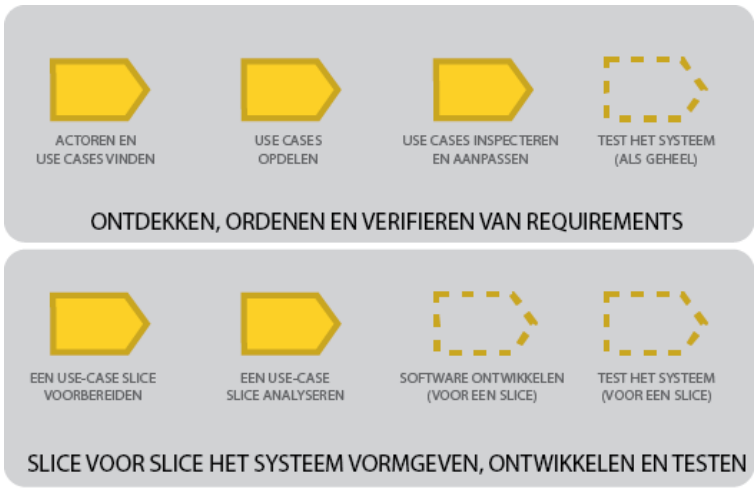
FIGUUR 14: DETAILNIVEAUS VAN WERKPRODUCTEN

Het goede nieuws is dat je altijd op dezelfde manier start, namelijk met het minimale. Het team kan dan steeds, afhankelijk van hun behoefte, het juiste detailniveau opzoeken. Het detailniveau kan ook worden aangepast om verspilling te reduceren; alles voorbij de essenties zou een helder bestaansrecht moeten hebben of anders moeten worden geëlimineerd. Om (naar zeggen) met Einstein te spreken: “Alles moet zo eenvoudig mogelijk worden gemaakt, maar niet eenvoudiger”.

Zie voor meer informatie over werkproducten en hun detailniveau Appendix 1: Werkproducten.

Dingen om te doen

Use-Case 2.0 splitst het werk op in een aantal essentiële activiteiten die moeten worden uitgevoerd om de use cases waarde te laten hebben voor het team. Deze activiteiten zijn weergegeven in Figuur 15, waarin ze gegroepeerd zijn in activiteiten die benodigd zijn voor het ontdekken, ordenen en verifiëren van requirements, en activiteiten die benodigd zijn voor het vormgeven, ontwikkelen en testen van het systeem. De gele blokpijlen geven de activiteiten weer die expliciet zijn gedefinieerd binnen de werkwijze. De gestreepte blokpijlen representeren andere activiteiten waar het succes van de werkwijze van afhangt. Use-Case 2.0 zegt dat deze laatste activiteiten nodig zijn, maar niet hoe ze moeten worden uitgevoerd.



FIGUUR 15: DE ACTIVITEITEN BINNEN USE-CASE 2.0

Lees Figuur 15 van links naar rechts om een idee te krijgen van de volgorde waarin de activiteiten voor de eerste keer worden uitgevoerd. De activiteiten zullen allemaal vele malen worden uitgevoerd gedurende een project of onderhoudsperiode. Zelfs een eenvoudige activiteit zoals ‘Vind Actoren en Use

Cases' moet mogelijkerwijs meerdere keren worden uitgevoerd om alle use cases te vinden en wellicht parallel met andere activiteiten worden uitgevoerd of erna. Terwijl bijvoorbeeld wordt verder gegaan met 'Vind Actoren en Use Cases', kan tegelijkertijd worden gewerkt aan de realisatie van enkele slices van use cases die eerder zijn gevonden.

De rest van dit hoofdstuk gaat nader in op deze activiteiten waarbij we een use-case slice volgen van de initiële identificatie van de use case waartoe hij behoort, tot zijn finale test en inspectie. Het volgende hoofdstuk gaat in op hoe we deze activiteiten kunnen organiseren voor verschillende soorten ontwikkelaanpakken, zoals Scrum, Kanban, Iteratief en Waterval.

Vind Actoren en Use Cases

Je dient eerst enkele actoren en use cases te vinden om je te helpen:

- Overeenstemming te verkrijgen over de doelen van het systeem.
- Overeenstemming te verkrijgen over het nieuwe systeemgedrag.
- De inhoud van releases te bepalen.
- Overeenstemming te verkrijgen over de toegevoegde waarde van het systeem.
- Manieren te identificeren om het systeem te gebruiken en te testen.

De beste manier om dit te doen is om een use-case modelleer-workshop te houden met belanghebbenden. Het is niet nodig om in één keer alle use cases te vinden. Focus gewoon op die use cases die de belanghebbenden de meeste toegevoegde waarde bieden. Je zult de overige actoren en use cases vinden tijdens het inspecteren en aanpassen van de use cases.

TIP: MAAK SNEL EEN USE-CASE MODEL MET EEN *MODEL STORM*

De formele aard van het use-case model en zijn gebruik binnen Unified Modeling Language kan een drempel zijn bij het betrekken van belanghebbenden bij het modelleerwerk.

Een goede manier om dit op te lossen is om belanghebbenden samen te laten brainstormen over de verschillende gebruikers en hun doelen met behulp van Post-Its© (gebruikers verticaal en hun doelen horizontaal). Faciliteer vervolgens het groeperen van het resultaat naar actoren en use cases. Belanghebbenden vinden het dan meestal eenvoudig om deze te kwantificeren, te schetsen en te sorteren.

Zodra de use cases geïdentificeerd zijn, moeten ze worden geordend als leidraad voor de releaseplanningen van het team. Een prettige eigenschap van use cases is dat ze op hoog niveau scope management mogelijk maken zonder de noodzaak om vooraf alle verhalen te kennen of uit te werken. Mocht een use case niet nodig zijn, dan is er ook geen reden om nader in te gaan op zijn verhalen. Mocht de use case onderdeel zijn van de scope, dan moet deze zodanig worden geschetst dat er genoeg informatie is om de use case op te delen in use-case slices.

Herhaal deze activiteit zo vaak als nodig om je model te laten evolueren en ontbrekende actoren en use cases te vinden.

Use Cases Opdelen

Vervolgens dien je je eerste use-case slices te maken. Dit doe je om:

- Passende werkeenheden voor het team te creëren.
- Binnen budget en doorlooptijd te blijven.
- Zoveel mogelijk waarde te leveren aan de gebruikers en andere belanghebbenden.
- Aantoonbaar kritische projectvoortgang te behalen of behoeften van belanghebbenden te begrijpen.

Zelfs de meest eenvoudige use case zal meerdere verhalen omvatten. Je dient de use case op te delen in slices om verhalen te selecteren die moeten worden ontwikkeld. Je kunt het maken van slices het beste samen doen met belanghebbenden om ervoor te zorgen dat ze de moeite waard zijn om te ontwikkelen. Alle use cases in één keer opknippen is onnodig. Waar het om gaat, is te kunnen voorzien in de directe behoefte van het team.

Het is niet nodig om een use-case helemaal op te delen. Richt je op de slices die nodig zijn om voortgang te maken en laat de andere verhalen voor wat ze zijn totdat ze nodig zijn. Je kunt zelfs een zogenaamd *pull model* introduceren waarbij ontwikkelaars vragen om nieuwe slices zodra ze capaciteit hebben om deze te ontwikkelen.

TIP: VOOR JE EERSTE SLICE HEB JE ALLEEN DE BASISSTROOM VAN DE BELANGRIJKSTE USE CASE NODIG

Veel mensen denken dat ze eerst alle use cases moeten schetsen voordat ze kunnen beginnen met het maken van slices. Dit leidt geheel onnodig tot een watervalaanpak die het vervaardigen van waardevolle werkende software vertraagt.

Een enkele slice van een use case is genoeg om het team te laten starten met het ontwikkelen en testen van het systeem.

De eerste slice zou altijd moeten worden geënt op de basisstroom. Voor sommige complexe systemen omvat de slice mogelijk niet eens de volledige stroom. Het is prima om een deel van de stroom te nemen waarbij je details van stappen kunt weglaten en *placeholders* maakt voor andere, zodat je de grootste risico's voor de realisatie bij de kop kunt pakken en achterhaalt of de use case al dan niet kan worden ontwikkeld.

De gecreëerde slices moeten worden gesorteerd zodat het team ze in de juiste volgorde zal ontwikkelen. Nogmaals, doe dit bij voorkeur samen met belanghebbenden en andere teamleden om er zeker van te zijn dat de sortering helpt om het kleinst mogelijke bruikbare systeem te bepalen. De beste manier om dit te doen is door een combinatie van prioriteit, toegevoegde waarde, risico en noodzaak in ogenschouw te nemen.

Herhaal deze activiteit net zo vaak als er nieuwe slices nodig zijn.

Een Use-Case Slice Voorbereiden

Zodra een slice is uitgekozen voor ontwikkeling is er meer werk nodig om:

- De slice gereed te maken voor de ontwikkeling.
- Duidelijk te maken wat een succesvolle ontwikkeling inhoudt.
- Vereiste karakteristieken te definiëren (bijvoorbeeld niet-functionele eisen).
- Te focussen op het ontwikkelen van software die de testen kan doorstaan.

Het voorbereiden van een use-case slice is teamwork. Je hebt inbreng nodig van belanghebbenden, ontwikkelaars en testers om de use-case beschrijving te verhelderen en testen te definiëren.

TIP: EEN SLICE ZONDER TESTGEVALLEN IS SLECHT VOORBEREID

Wanneer je een slice voorbereidt, definieer dan ook de testgevallen die nodig zijn voor de verificatie van de ontwikkeling. Want het zijn immers de testgevallen die ons laten weten wat daadwerkelijk moet worden gedaan.

De testgevallen bieden de ontwikkelaars een norm voor wat het systeem moet doen. Dit betekent dat slices pas klaar zijn als de software alle testgevallen doorstaat.

Focus bij het voorbereiden van een use-case slice op de behoeften van de ontwikkelaars en testers die de slice gaan ontwikkelen en verifiëren. Denk na over hoe ze toegang krijgen tot de informatie die ze nodig hebben. Zijn ze in staat om met materiedeskundigen te praten om de verhalen uit te werken of moet alles voor ze worden gedocumenteerd? Je dient ook balans aan te brengen tussen het detailleren van de use-case beschrijving en het detailleren van testgevallen. Hoe meer detail je aanbrengt in de use-case beschrijving, des te eenvoudiger het wordt om de testgevallen te maken. Aan de andere kant, hoe lichtvoetiger de use-case beschrijving, des te minder dubbele vastlegging en herhaling van informatie uit de use-case beschrijving in de testgevallen. Het beste is om de use-case beschrijving en de testgevallen tegelijkertijd uit te werken zodat de auteurs in staat zijn om balans aan te brengen tussen hun eigen behoeften en die van hun belanghebbenden. Het kan zijn dat er werk overblijft, bijvoorbeeld wanneer de testgevallen moeten worden geautomatiseerd, maar er mag geen twijfel bestaan over waar een succesvolle ontwikkeling aan moet voldoen.

Voer deze activiteit minstens één keer uit voor iedere slice en herhaal deze activiteit in geval van wijzigingen op de slice.

Een Use-Case Slice Analyseren

Voordat je begint met coderen, moet je een slice analyseren om:

- Te bepalen wat de impact is op systeemelementen die nodig zijn voor de ontwikkeling van de slice.
- De verantwoordelijkheden van de geraakte systeemelementen vast te stellen.
- Te definiëren hoe de systeemelementen samenwerken om de use case uit te voeren.

Wanneer een team een nieuwe slice krijgt aangereikt, is het eerste dat ze doen nagaan hoe deze het systeem raakt. Hoeveel delen van het systeem moeten worden aangepast en in welke mate? Hoeveel nieuwe elementen zijn nodig en hoe moeten die worden ingepast?

Het analyseren van slices gaat vaak vooraf aan het plannen van ontwikkelactiviteiten. Het stelt een team in staat om de ontwikkeling van de slice te benaderen als een set van kleinere aanpassingen op de code in plaats van een groot, onoverzichtelijke brok werk. De slice zelf kan ook gebruik worden als werkeenheid. Het analyseren van de slice is dan gewoon de laatste activiteit van de ontwikkelaar voorafgaand aan het coderen.

TIP: DOE DE ANALYSE SAMEN EN WEES LICHTVOETIG MET DOCUMENTATIE

De eenvoudigste manier om een use-case slice te analyseren is het team samenbrengen om te bespreken hoe de slice de verschillende systeemelementen raakt.

Als het team het ontwerp verkent, maken ze hiervan een tekening op een whiteboard, meestal in de vorm van een eenvoudig *sequence diagram* of *collaboration diagram* (UML). Dit kan vervolgens worden gefotografeerd of worden vastgelegd in het door het team gekozen modelleertool.

Naarmate het team meer begrip krijgt van het systeem en zijn architectuur, zullen ze merken dat het analyseren van slices steeds makkelijker wordt en dat ze dat vaak in gedachten kunnen doen. Het is nog steeds waardevol om met enkele collega's de analyse in schetsvorm te doen voorafgaand aan het coderen. Hiermee kunnen ontwerpbeslissingen worden gevalideerd en kan worden nagegaan of alles goed is begrepen. Ook biedt het traceerbaarheid bij het bepalen van de impact van wijzigingen en bevindingen. Het resultaat van deze analyse is bekend als een *use-case realisatie*: het geeft weer hoe de elementen van het systeem een use case realiseren.

Voer deze activiteit minstens één keer uit voor iedere slice en herhaal deze activiteit in geval van wijzigingen op de slice.

Software Ontwikkelen (voor een Slice)

Je kunt nu beginnen met het ontwerpen, coderen, unit-testen en integreren van de softwarecomponenten die nodig zijn voor de ontwikkeling van een use-case slice.

De software zal slice voor slice worden ontwikkeld, waarbij diverse teamleden parallel aan verschillende slices werken. Iedere slice leidt tot een wijziging van een of meerdere delen van het systeem. Om de ontwikkeling van een slice te voltooien, zullen de diverse stukken software eerst een unit-test moeten ondergaan om vervolgens te worden geïntegreerd in de rest van het systeem.

Test het Systeem (voor een Slice)

Test vervolgens de software onafhankelijk van de ontwikkeling om te verifiëren of de use-case slices succesvol zijn ontwikkeld. Iedere slice dient te worden getest om compleet en geverifieerd te worden bevonden. Dit wordt gedaan door van iedere slice zijn testgevallen uit te voeren. Omdat slices onafhankelijk zijn, kun je ze testen zodra ze zijn ontwikkeld zodat je ontwikkelaars zo snel mogelijk terugkoppeling kunt geven.

Use-Case 2.0 is prima te gebruiken in combinatie met populaire testwerkwijzen. Het kan worden gezien als een vorm van *test driven development,* want het creëert testgevallen voor iedere slice voordat een slice wordt vrijgegeven voor ontwikkeling.

Test het Systeem (als geheel)

Ieder increment van het softwaresysteem dient te worden getest om te verifiëren dat de nieuwe use-case slices correct zijn ontwikkeld zonder andere delen van het systeem onderuit te halen. Het is niet afdoende om alleen de slices te testen zodra ze voltooid zijn. Het team dient ook het systeem als geheel te testen om zeker te weten dat alle slices ook in samenhang correct werken, en dat eerder geverifieerde slices ook nog steeds correct functioneren.

Use-Case 2.0 resulteert in robuuste en veerkrachtige testgevallen, omdat de structuur van de use-case beschrijving leidt tot onafhankelijk executeerbare, op scenario's gebaseerde testgevallen.

Use Cases Inspecteren en Aanpassen

Je dient de use cases en hun use-case slices ook continu te evalueren en aan te passen om:

- Wijzigingen af te kunnen afhandelen.
- De voortgang vast te kunnen stellen.
- Het werk te laten passen binnen de beschikbare doorlooptijd en budget.
- Het use-case model actueel te houden.
- De omvang van de slices aan te passen om de productiesnelheid te vergroten.

Naarmate het project vordert, is het essentieel dat je steeds je use-case model, use cases en use-case slices bijwerkt. Prioriteiten veranderen, er zijn leerpunten en je krijgt te maken met wijzigingsvoorstellen. Deze kunnen allemaal impact hebben op de use cases en use-case slices die ofwel al zijn ontwikkeld ofwel klaar staan om ontwikkeld te gaan worden. Deze activiteit leidt veelal tot de ontdekking van nieuwe use cases en wijzigingen op bestaande use cases en use-case slices.

TIP: ONDERHOUD REGELMATIG DE BACKLOG MET USE-CASE SLICES

Door je slices op volgorde te zetten en hun toestand te bewaken (afgebakend, voorbereid, geanalyseerd, gecodeerd, geverifieerd) creëer je een werkvoorraad van te ontwikkelen requirements. Deze lijst dient steeds te worden bewaakt en bijgesteld aan de hand van de door het team geboekte voortgang en de wensen van de belanghebbenden.

Als het project loopt, bewaak en stel dan zo nodig de slice-grootte bij om verspilling tegen te gaan en de effectiviteit van het team te verbeteren.

De verschillende omstandigheden binnen een project maken dat je de wijze waarop je Use-Case 2.0 gebruikt moet bijstellen. Dit doe je bijvoorbeeld door het aanpassen van de omvang van slices, of het detailniveau van use-case beschrijvingen, de ondersteunende informatie en de testgevallen. Het is belangrijk dat je continu je werkwijze onder de loep neemt en aanpast, net zoals de use cases en use-cases slices waar je aan werkt.

Voer deze activiteit zo vaak uit als nodig is om je use cases te onderhouden en wijzigingsvoorstellen af te handelen.

Use-Case 2.0 toepassen

Use-Case 2.0 kan worden toegepast in verschillende contexten en voor verschillende soorten systemen. In dit hoofdstuk gaan we dieper in op het toepassen van use-cases voor verschillende soorten systemen, verschillende soorten requirements en verschillende manieren van ontwikkelen.

Use-Case 2.0: Geschikt voor alle soorten systemen

Veel mensen denken dat use cases alleen kunnen worden gebruikt voor zogenaamde gebruikersintensieve systemen ofwel systemen met veel interactie tussen gebruiker en systeem. Dat is vreemd want use cases zijn oorspronkelijk bedacht voor telefooncentrales waarbij zowel mensen (bellers, onderhoudsmedewerkers) als machines in de vorm van gekoppelde systemen betrokken zijn. Use cases zijn zonder meer toepasbaar op alle systemen die worden gebruikt, en dat betekent inderdaad alle systemen.

Use-Case 2.0: Meer dan alleen gebruikersintensieve applicaties

Feitelijk zijn use-cases net zo goed bruikbaar voor embedded systemen met nauwelijks tot geen gebruikersinteractie als voor systemen met juist veel gebruikersinteractie. Tegenwoordig worden use cases gebruikt voor de ontwikkeling van allerlei soorten embedded software in diverse omgevingen: motoren, consumentenelektronica, de luchtvaart, en de militaire en medische industrie. Zelfs systemen voor real-time procesbesturing kunnen worden beschreven met use cases, waarbij iedere use case zich richt op een specifiek deel van het proces en de automatiseringsbehoefte.

Use cases zijn toepasbaar voor alle systemen die samenwerken met de buitenwereld ongeacht of de gebruikers mensen of machines zijn. De toepasbaarheid is veel breder dan de meeste mensen denken.

Use-Case 2.0: Meer dan alleen softwareontwikkeling

Use-cases zijn breder toepasbaar dan alleen voor software. Ze kunnen ook helpen om *business requirements* te doorgronden, een bestaand bedrijf door te lichten, nieuwe en betere processen te ontwerpen, en de kracht van IT te benutten om een bedrijf te transformeren. Door use cases recursief toe te passen om 1) een bedrijf en zijn interactie met de buitenwereld te modelleren en 2) de systemen te modelleren die nodig zijn om het bedrijf te ondersteunen en te verbeteren, kun je naadloos identificeren waar systemen impact hebben en welke systemen je nodig hebt om het bedrijf te ondersteunen.

De use cases die worden gebruikt om een bedrijf te modelleren worden veelal business use cases genoemd. Zij leveren de context voor de ontwikkeling van IT-systemen waardoor de ontwikkeling van het bedrijf en IT perfect kunnen worden gesynchroniseerd. Niet alleen kun je IT-systemen slice voor slice ontwikkelen, je kunt ook je bedrijfsmodel slice voor slice ontwikkelen. Dit is zeer krachtig, want het stelt je in staat om je bedrijf en zijn ondersteunende systemen tegelijk te laten evolueren. Dit maakt zowel incrementele systeemontwikkeling als incrementele bedrijfsontwikkeling mogelijk.

Tegenwoordig kunnen, en moeten, business en ondersteunende IT hand in hand worden ontwikkeld, want de één kan niet zonder de ander. Het gebruik van use cases en use-case slices aan de grenzen van zowel organisatie als IT, kan de brug slaan tussen beide werelden en ervoor zorgen dat ze daadwerkelijk kunnen samenwerken.

Use-Case 2.0: Geschikt voor alle soorten requirements

De use-case techniek is een van de populairste technieken om systeemfunctionaliteit te beschrijven. Het kan echter ook gebruikt worden voor het verkennen van non-functionele karakteristieken of kwaliteitseisen. De eenvoudigste manier om dit te doen is door deze vast te leggen als deel van de use case zelf. Relateer bijvoorbeeld performance requirements aan de tijd tussen bepaalde stappen van een use case, of geef een opsomming van de verwachte serviceniveaus voor een use case als onderdeel van de use case zelf.

Sommige kwaliteitseisen zijn subtieler en zijn van toepassing op meerdere, zo niet alle use cases. Dit geldt in het bijzonder voor het ontwikkelen van gelaagde architecturen inclusief infrastructuurcomponenten zoals beveiliging, transactiemanagement, messaging services en data management. De eisen binnen deze aandachtsgebieden kunnen nog steeds worden vastgelegd met use cases. Dit betreft dan additionele use cases die focussen op het technisch gebruik van het systeem. We noemen deze additionele use cases ook wel *infrastructuur use cases* omdat ze requirements bevatten die betrekking hebben op de ontwikkeling van de infrastructuur waarop een applicatie kan draaien. Deze use cases en hun slices kunnen worden gezien als *cross-cutting concerns* die het gedrag van het systeem beïnvloeden wanneer de meer traditionele functionele use cases worden uitgevoerd. Je kunt bijvoorbeeld een use case maken om te verkennen hoe het systeem om dient te gaan met databasetransacties inclusief alle verschillende gebruiksscenario's zoals schema's voor data locking, data caching, commit en roll-back. Deze use case is steeds van toepassing zodra een andere use case data ophaalt uit, of wegschrijft naar het systeem.

Het combineren van deze infrastructuur use cases met andere technieken zoals *separation of concerns* en *aspect oriented programming* maakt het mogelijk om ook deze algemene eisen af te handelen zonder de realisatie van de bestaande functionele use cases te hoeven wijzigen.

Use-Case 2.0: Geschikt voor alle ontwikkelaanpakken

Use-Case 2.0 is toepasbaar in combinatie met alle populaire ontwikkelaanpakken, waaronder:

- *Backlog-gedreven* iteratieve aanpakken zoals Scrum, EssUP en OpenUP.
- *Stuksgewijs-gedreven* aanpakken zoals Kanban.
- *Alles-ineens* aanpakken zoals het traditionele watervalmodel.

In de volgende drie korte paragrafen zullen we illustreren hoe Use-Case 2.0 en in het bijzonder use-case slices deze aanpakken kunnen ondersteunen. Deze paragrafen staan minder op zichzelf dan de andere in dit e-book omdat ze ervan uit gaan dat de lezer een basaal begrip heeft van de besproken ontwikkelaanpakken. We raden aan alleen de paragrafen te lezen van de ontwikkelaanpakken waar je bekend mee bent.

Use-Case 2.0 en backlog-gedreven iteraties

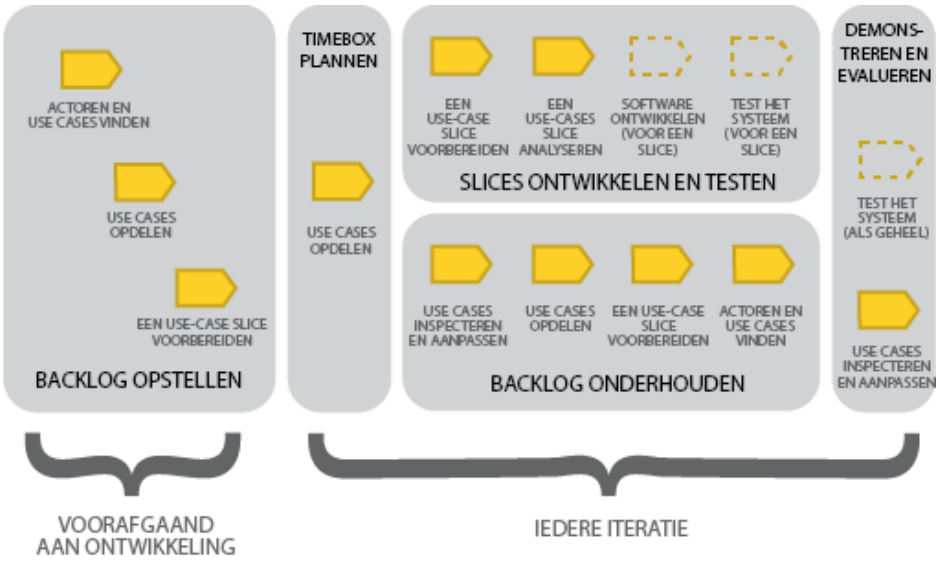
Voordat je aan de slag kunt gaan met een backlog-gedreven aanpak, dien je te begrijpen wat een backlog kan bevatten. Backlog kun je vrij vertalen met werkvoorraad. Er zijn verschillende soorten backlogs die teams kunnen gebruiken om hun werkzaamheden te sturen, zoals product backlogs, release backlogs en project backlogs. Ongeacht de gebruikte terminologie zijn ze allemaal gebaseerd op dezelfde principes. Een backlog is een geordende lijst van alles wat nodig kan zijn en het is de enige bron van requirements voor iedere door te voeren wijziging. Figuur 16 geeft het basisconcept van een backlog weer.

FIGUUR 16: EEN BASALE BACKLOG

Wanneer je Use-Case 2.0 gebruikt, dan zijn de use-case slices de primaire backlog-items. Het gebruik van use-case slices zorgt ervoor dat de backlog-items goed zijn vormgegeven, omdat ze van nature onafhankelijk zijn, toegevoegde waarde hebben en testbaar zijn. De structuur van de use-case beschrijving waarmee ze worden gedefinieerd, zorgt ervoor dat ze te schatten en onderhandelbaar zijn. Het use-case slice mechanisme zorgt ervoor dat ze net zo klein zijn als nodig zodat het ontwikkelteam er iets mee kan.

De use cases zelf stop je niet in de lijst omdat het onduidelijk is wat ze dan betekenen. Wordt hiermee bijvoorbeeld de eerste slice bedoeld? Of de laatste, of allemaal? Wanneer je een hele use case in de lijst wilt opnemen voorafgaand aan het maken van slices, maak dan een dummy slice die de hele use case vertegenwoordigt en voeg die dan toe aan de lijst.

Wanneer je een backlog-gedreven aanpak gaat gebruiken, weet dan dat de backlog niet van tevoren volledig wordt gedefinieerd maar juist steeds wordt bijgewerkt. Dit wordt ook wel *backlog grooming* genoemd. Figuur 17 laat de kenmerkende volgorde van activiteiten zien van een backlog-gedreven iteratieve aanpak.



FIGUUR 17: USE-CASE 2.0 ACTIVITEITEN VOOR ITERATIEVE ONTWIKKELING

De backlog dient te worden opgesteld voorafgaand aan ontwikkeling. ‘Vind Actoren en Use Cases’ wordt uitgevoerd om het initiële use-case model op te zetten en het systeem af te bakenen. De activiteit ‘Use Cases Opdelen’ wordt uitgevoerd om de backlog te vullen met de initiële set van belangrijkste use-case slices. ‘Een Use-Case Slice Voorbereiden’ wordt uitgevoerd om een of meer slices gereed te maken om te worden ontwikkeld in de eerste iteratie.

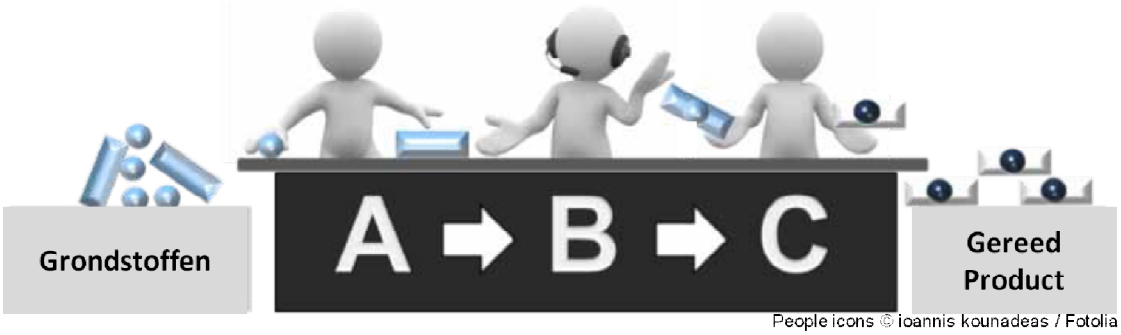
Zodra de backlog beschikbaar is, kun je de eerste iteratie starten. Iedere iteratie start met iets van planning. Tijdens deze planning kan het zijn dat je met de activiteit ‘Use Cases Opdelen’ de gekozen use-case slices kleiner maakt zodat ze afdoende klein zijn om te kunnen worden voltooid binnen de iteratie. Het ontwikkelteam voert dan de activiteiten ‘Een Use-Case Slice Voorbereiden’, ‘Een Use-Case Slice Analyseren’, ‘Software Ontwikkelen (voor een slice)’ en ‘Test het Systeem (voor een slice)’ uit om de geïdentificeerde slices te ontwikkelen en ze toe te voegen aan het systeem.

Tijdens de ontwikkeling gaat het team door met de activiteiten ‘Use Cases Inspecteren en Aanpassen’, ‘Use Cases Opdelen’ en ‘Een Use-Case Slice Voorbereiden’ om de backlog bij te werken, wijzigingen af te handelen en ervoor te zorgen dat er genoeg backlog-items klaar staan om opgepakt te worden in de volgende iteratie. Het team moet mogelijk zelfs de activiteit ‘Vind Actoren en Use Cases’ uitvoeren om grote wijzigingsvoorstellen af te handelen of om meer use cases te vinden voor het team. Bij Scrum wordt aanbevolen om teams 5 tot 10 procent van hun tijd te laten spenderen aan het onderhouden van hun backlog. Dit is een behoorlijke inspanning voor het team en Use-Case 2.0 biedt de benodigde werkproducten en activiteiten om dit eenvoudig en efficiënt te doen.

Tot slot dient een team aan het einde van een iteratie het systeem te demonstreren en terug te blikken op de, tijdens de iteratie, geleverde prestatie. Het team zou de activiteit ‘Test het Systeem (als geheel)’ moeten uitvoeren om de voortgang vast te stellen en de activiteit ‘Use Cases Inspecteren en Aanpassen’ om de kwaliteit en effectiviteit van hun use cases en use-case slices te bepalen.

Use-Case 2.0 en stuksgewijs ontwikkelen

Stuksgewijs ontwikkelen is een aanpak die voorkomt dat er batches of voorraden van requirements ontstaan zoals in een iteratieve- of watervalaanpak. In een stuksgewijze of *one-piece flow* aanpak gaan requirements stuk voor stuk door het ontwikkelproces. One-piece flow is een techniek die is ontleend aan *lean manufacturing*. Figuur 18 laat zien hoe een klein team werkpakketten door het proces laat gaan, van werkstation A naar B naar C.



FIGUUR 18: BASISCONCEPT STUKSGEWIJZE AANPAK

Om dit effectief te laten werken heb je kleine werkpakketten van soortgelijke omvang nodig die snel door het systeem kunnen gaan. Binnen softwareontwikkeling zijn de requirements de grondstoffen en is de werkende software het eindproduct. Use cases zijn te onregelmatig van vorm en te groot van omvang om snel door het systeem te gaan. De benodigde tijd op werkstation A, B en C wordt daardoor te onvoorspelbaar waardoor het systeem kan vastlopen. Use-case slices kunnen echter klein genoeg

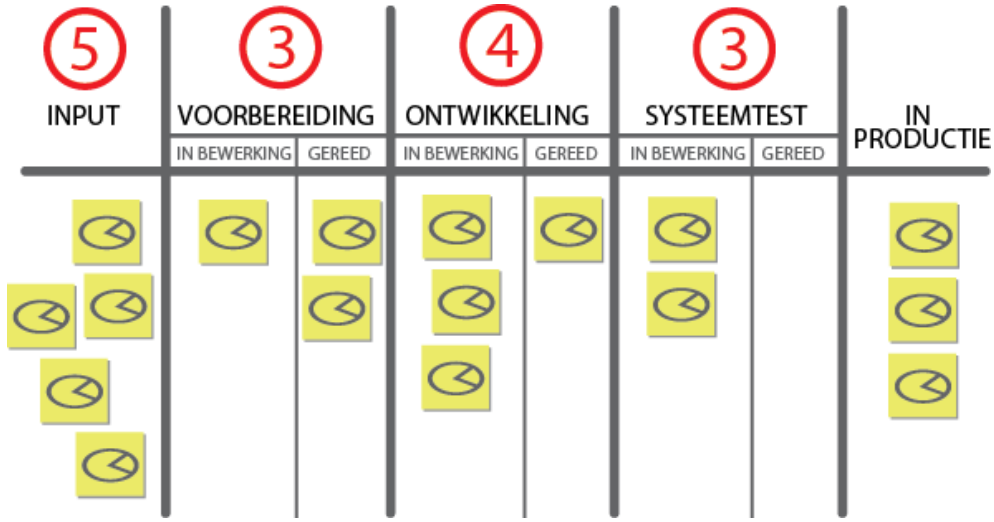
worden gemaakt, en afgestemd worden op de behoeften van het team. Figuur 19 laat zien hoe stuksgewijs ontwikkelen eruit ziet met use-case slices.

FIGUUR 19: STUKSGEWIJS SOFTWARE ONTWIKKELEN MET USE-CASE SLICES

Behalve dat ze snel door het systeem moeten kunnen gaan, moeten er ook genoeg werkpakketten in het systeem aanwezig te zijn om het systeem aan de gang te houden. Stuksgewijs ontwikkelen betekent niet dat er steeds maar aan één eis tegelijk wordt gewerkt of dat er steeds maar één werkpakket wordt doorgegeven van het ene naar het volgende werkstation. De stroomsnelheid wordt op niveau gehouden door het zetten van limieten op de hoeveelheid gelijktijdig werk per werkstation. Hiermee wordt tevens verspilling (Engels: waste) tegengegaan ten gevolge van grote werkvoorraden.

Stuksgewijs ontwikkelen betekent niet dat individuen maar één ding tegelijk doen en maar op één werkstation werkzaam mogen zijn. Er kunnen bijvoorbeeld meer mensen werkzaam zijn op Ontwikkeling dan op Test en wanneer het systeem dreigt vast te lopen, moeten mensen daar worden ingezet waar nodig om het systeem vlot te trekken. Als er geen use-case slices zijn die gereed zijn om getest te worden, maar er wel slices zijn die vastzitten in analyse, dan kunnen testers initiatief tonen door te helpen met het analysewerk. Op dezelfde manier ben je niet beperkt tot één persoon per werkstation of zelfs tot één instantie van ieder werkstation.

Kanban-borden zijn een techniek om de doorstroming van een productielijn te visualiseren. Een Kanban is een teken, vlag of signaal binnen een productieproces om de productie of levering van een product te starten binnen een *just-in-time* en *lean manufacturing*-proces. Op een Kanban-bord worden Kanban-kaarten gebruikt om werkpakketten in het systeem weer te geven. Figuur 20 geeft een eenvoudig Kanban-bord weer van een ontwikkelteam dat eerst iedere slice analyseert om de impact te doorgronden. Vervolgens ontwikkelt en unit-test het team de software en tot slot wordt onafhankelijk het resulterende systeem getest alvorens het in productie wordt genomen.

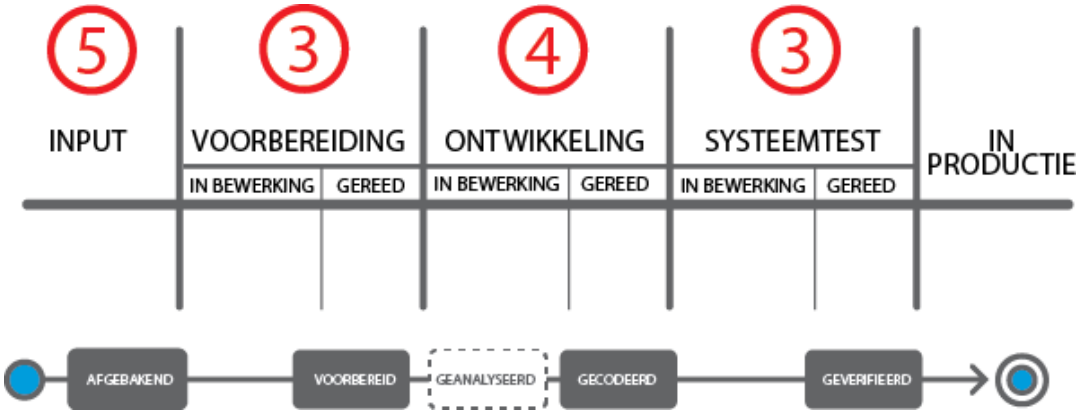


FIGUUR 20: USE-CASE SLICES OP EEN KANBAN-BORD

De limieten voor werk-in-uitvoering zijn rood omcirkeld weergegeven. Wanneer je van links naar rechts leest, dan zie je dat slices eerst geïdentificeerd en afgebakend worden voordat ze als input kunnen fungeren voor het team. Hier heeft de limiet de waarde 5 en de klanten, producteigenaren of het business team dat de bron is van de requirements moeten er voor zorgen dat te allen tijde 5 slices klaar staan om te worden ontwikkeld.

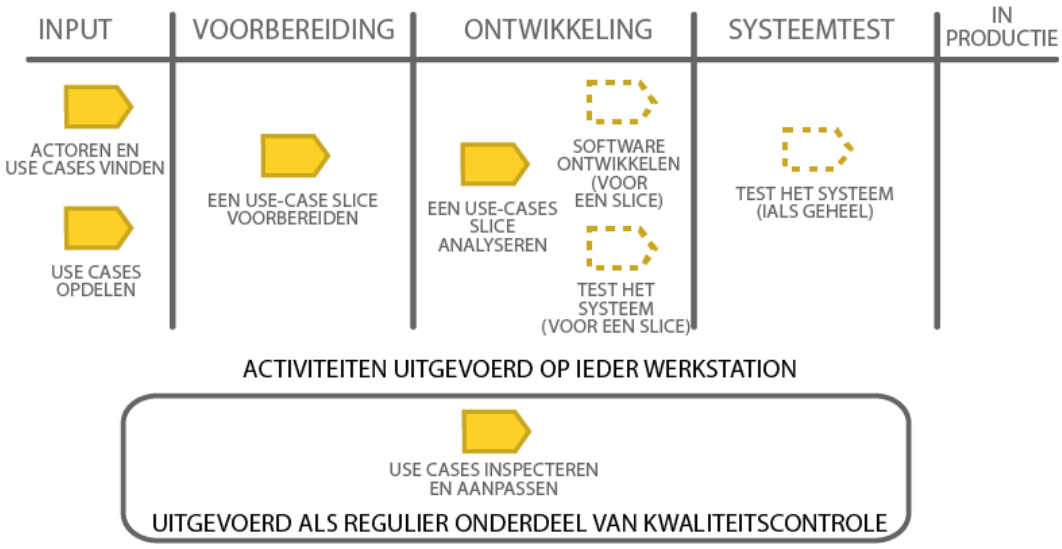
Slices worden afgenomen van de 'input'-wachtrij en verplaatst naar 'voorbereiding' alwaar de impactanalyse wordt uitgevoerd, de verhalen worden verhelderd, en de testgevallen worden voltooid. Hier heeft de werk-in-uitvoering limiet de waarde 3. De kolom 'in bewerking' bevat slices waar daadwerkelijk aan wordt gewerkt. De kolom 'gereed' bevat slices die gereed staan om te worden opgepakt door een ontwikkelaar. Op deze manier gaan de slices door het systeem vervolgens naar ontwikkeling, test en na een succesvolle onafhankelijke systeemtest in productie. Een werk-in-uitvoering-limiet omvat al het werk binnen een werkstation. Dit zijn zowel de slices die in behandeling zijn als de slices die gereed zijn voor de volgende bewerkingsslag. Er is geen limiet op het aantal slices dat tegelijk in productie kan worden genomen.

Het is belangrijk te realiseren dat er geen ultiem Kanban-bord is of een algemene set van werk-in-uitvoering-limieten. De structuur van het bord hangt af van de structuur van je team en de gehanteerde werkwijzen. Je dient het bord en de limieten aan te passen zodra je je werkwijze aanpast. De stadia van de use-case slices zijn een geweldig hulpmiddel om deze manier van werken uit te denken. Figuur 21 laat de samenhang zien tussen de stadia en het Kanban-bord zoals weergegeven in Figuur 20. De stadia zijn zeer krachtig omdat ze duidelijk definiëren welke toestand een slice moet hebben om te kunnen worden overgedragen aan de volgende schakel in de keten.



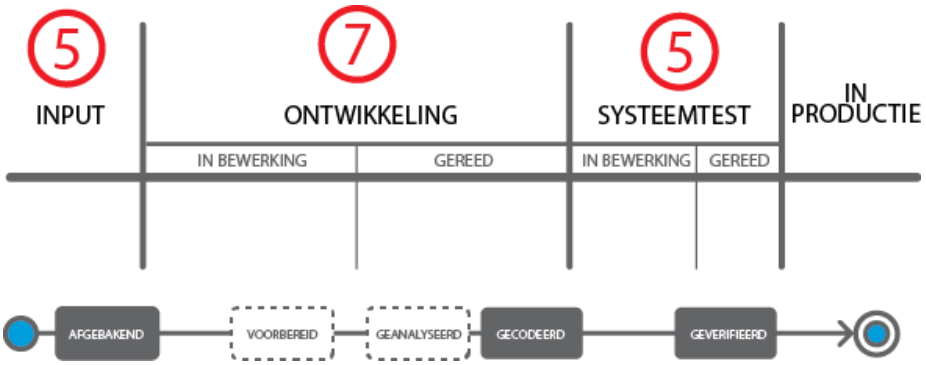
FIGUUR 21: SAMENHANG USE-CASE SLICE STADIA EN HET KANBAN BORD

Figuur 22 laat zien waar de verschillende Use-Case 2.0 activiteiten worden toegepast. Interessant is hier dat de activiteit 'Use Cases Inspecteren en Aanpassen' niet een verantwoordelijkheid is van een bepaald werkstation maar dat dit wordt gedaan als onderdeel van de reguliere kwaliteitscontrole door het gehele team. Deze activiteit helpt het team om zowel het aantal en type werkstations als hun werk-in-uitvoering-limieten te bepalen.



FIGUUR 22: USE-CASE 2.0 ACTIVITEITEN VOOR STUKSGEWIJZE ONTWIKKELING

Je kunt bijvoorbeeld naar aanleiding van een evaluatie van de effectiviteit van het team besluiten om het werkstation 'voorbereiding' te elimineren en de werk-in-uitvoering-limieten van ontwikkeling en systeemtest te verhogen. Opnieuw gebruik je de stadia van de use-case slices om voor een werkstation te definiëren wanneer hun werk gereed is, met als resultaat het Kanban-bord weergegeven door Figuur 23.

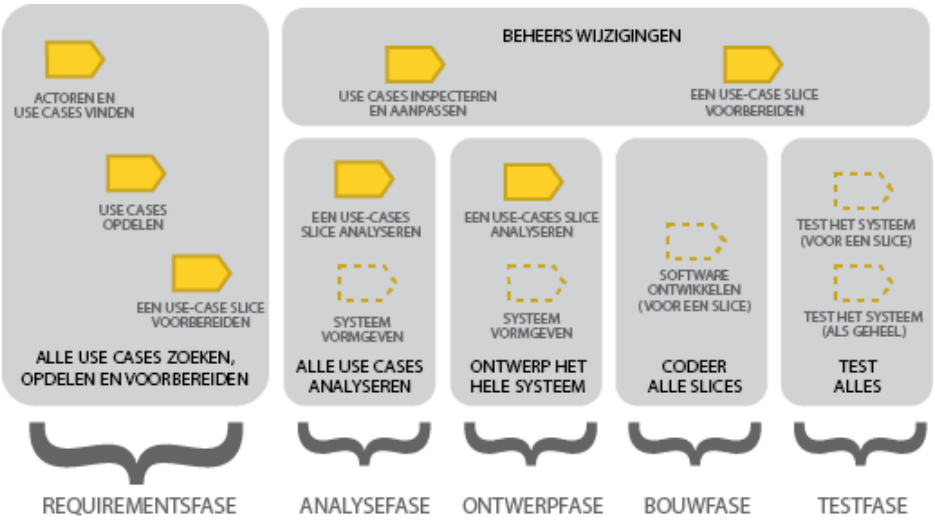


FIGUUR 23: AANGEPAST KANBAN BORD MET STADIA VAN GEREEDHEID

Use-Case 2.0 en waterval

Er zijn meerdere redenen te bedenken waardoor je gedwongen kunt worden software te ontwikkelen binnen de beperkingen van een vorm van het watervalbesturingsmodel. Dit betekent dat wordt getracht om alle requirements ineens vast te leggen voordat ze worden overgedragen aan een derde partij voor ontwikkeling.

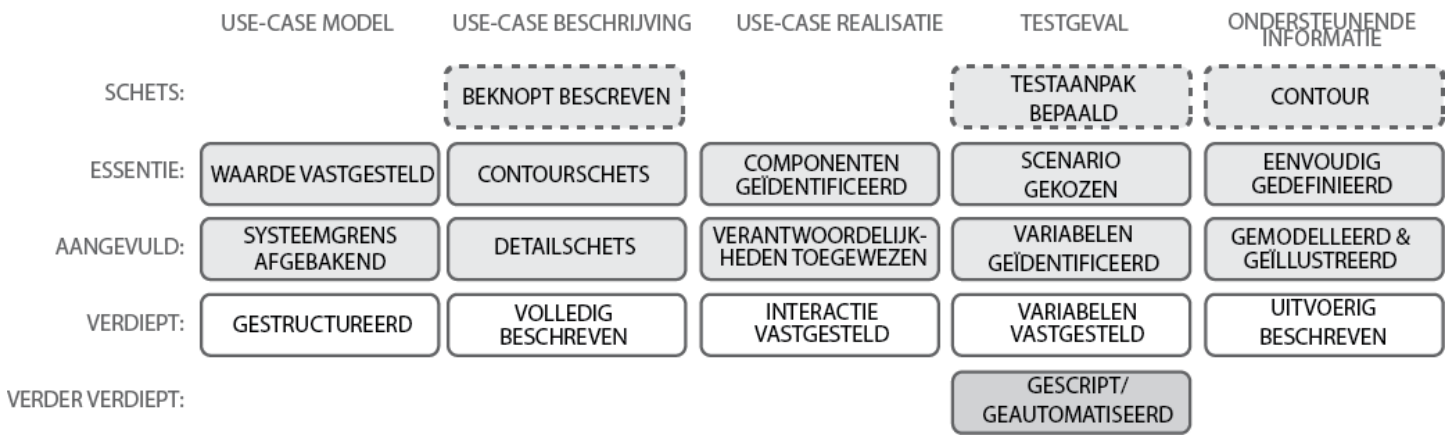
Wanneer je voor een watervalaanpak kiest, worden use cases niet continu bewerkt en gedetailleerd om zo het uiteindelijke systeem te laten ontstaan. Ze worden allemaal ineens gedetailleerd bij aanvang van de werkzaamheden. Vervolgens doorlopen ze in perfecte synchronisatie de volgende fasen van ontwikkeling, met steeds de focus op één bepaalde activiteit. Figuur 24 laat de kenmerkende volgorde van activiteiten van een watervalaanpak zien.



FIGUUR 24: USE-CASE 2.0 ACTIVITEITEN VOOR EEN WATERVALAANPAK

Zelfs binnen de meest strikte watervalomgeving zullen er tijdens de ontwikkeling van de software steeds wijzigingen naar voren komen. In plaats van wijzigingen te omarmen en het indienen van wijzigingsvoorstellen aan te moedigen, zullen watervalprojecten trachten het wijzigen onder controle te houden. Zo nu en dan zal de activiteit 'Use Cases Inspecteren en Aanpassen' worden uitgevoerd wanneer een wijzigingsvoorstel niet kan worden tegengehouden. In dat geval zullen additionele use-case slices moet worden voorbereid om geaccepteerde wijzigingen door te kunnen voeren. Het is onwaarschijnlijk dat er use cases zullen worden gevonden na de requirementsfase omdat dit als een te grote verandering zal worden beschouwd.

De 'één stap tegelijk' aard van de watervalaanpak betekent dat de teamsamenstelling verandert met iedere faseovergang. Hierdoor is de mogelijkheid tot directe communicatie om informatie over verhalen te delen minimaal. Om dit te ondervangen is het noodzakelijk om de werkproducten van meer detail te voorzien, ver voorbij de ondergrens. Figuur 25 laat de detailniveaus zien die doorgaans worden gebruikt bij watervalprojecten.



FIGUUR 25: DETAILNIVEAUS VAN WERKPRODUCTEN BINNEN EEN WATERVALAANPAK

Binnen iedere fase van ontwikkeling worden een of meer werkproducten uitgewerkt tot een zeer hoog detailniveau om ervoor te zorgen dat ze 1) compleet zijn en 2) een antwoord bieden op iedere vraag die in een latere fase naar voren zou kunnen komen. In de requirementsfase wordt het use-case model uitgewerkt totdat alle use cases daarin verwerkt zijn. Verder worden alle use-case beschrijvingen volledig beschreven en de ondersteunende informatie wordt uitvoerig vastgelegd.

In dit stadium wordt enige aandacht gegeven aan testen en wordt de teststrategie bepaald. De testgevallen worden dan in de wacht gezet totdat de testfase is bereikt.

De use cases en hun ondersteunende informatie worden overgedragen aan het analyse- en ontwerpteam dat met behulp van use-case realisaties eerst de verantwoordelijkheden van de diverse componenten in kaart brengt en vervolgens alle interacties bepaalt. Uiteindelijk wordt gestart met coderen waarbij alle use-case slices van alle use cases zullen worden ontwikkeld. Tot slot worden de testers erbij betrokken en worden alle testgevallen in detail uitgewerkt zodat daarna met de testuitvoering kan worden begonnen.

De sequentiële aard van deze manier van werken zou je ertoe kunnen verleiden te denken dat er hier geen rol is weggelegd voor use-case slices en dat het uitwerken van de use cases als geheel afdoende houvast biedt. Dit is echter niet waar omdat de use-case slice een fijnmazigere controle biedt waarmee het requirementsteam veel meer grip krijgt op de afbakening van het systeem. Want zelfs bij watervalprojecten is het onwaarschijnlijk dat je alle verhalen van alle use cases nodig hebt. Use-case slices helpen je ook om *last minute* wijzigingen, veroorzaakt door problemen met de doorlooptijd of kwaliteit, af te handelen.

Use-Case 2.0 – Geschikt voor alle soorten teams

Een andere belangrijke eigenschap van Use-Case 2.0 is de aanpasbaarheid aan bestaande teamstructuren en functies, terwijl teams worden aangespoord om verspilling te elimineren en efficiëntie te vergroten. Om die reden schrijft Use-Case 2.0 geen specifieke rollen of teamstructuren voor, maar definieert het een set van stadia voor ieder essentieel element (de use case en de use-case slice).

Zoals toegelicht in de discussie over Use-Case 2.0 en stuksgewijs ontwikkelen, geven de stadia aan wanneer werkpakketten klaar staan om te worden overgedragen van de ene persoon naar de andere. Dit maakt het mogelijk om de werkwijze te gebruiken voor alle soorten en maten van teams, van kleine multidisciplinaire teams met weinig tot geen overdrachtsmomenten tot grote netwerken van specialistische teams waarbij iedere toestandsovergang de verantwoordelijkheid is van een andere specialist. De doorstroming van werkpakketten van team naar team kan worden bewaakt door de status van use-case slices en hun overdrachtsmomenten te volgen. Met de hiermee verkregen inzichten kunnen teams hun werkwijze continu verbeteren.

Use-Case 2.0: Schaalbaar naar behoefte: verdiepen, verbreden en opschalen

Er bestaat geen voorgedefinieerde aanpak die in iedere behoefte voorziet en daarom moet onze toepassing van Use-Case 2.0 kunnen schalen in verschillende dimensies:

1. Verdieping om meer ondersteuning te bieden aan minder ervaren beoefenaars (ontwikkelaars, analisten, testers, etc.) of beoefenaars die meer richting willen of nodig hebben.
2. Verbreding om de gehele levenscyclus af te dekken; behalve analyse, ontwerp, coderen en testen ook operationeel gebruik en onderhoud.
3. Opschalen om grote en zeer grote systemen zoals systemen van systemen te ondersteunen: enterprise systemen, productlijnen en gelaagde systemen. Zulke systemen zijn complex en worden typisch door meerdere parallelle teams ontwikkeld, op verschillende locaties en mogelijk door verschillende bedrijven en waarbij tevens gebruik wordt gemaakt van reeds bestaande systemen (legacy) of pakketten.

Ongeacht de complexiteit van het systeem dat je ontwikkelt, begint je altijd op dezelfde manier door eerst de meest belangrijke use cases te bepalen en vervolgens de grote lijn te bepalen door middel van een grafische samenvatting van wat moet worden gebouwd. Vervolgens kun je Use-Case 2.0 zo aanpassen dat het voorziet in de ontluikende behoeften van het team. Sterker nog, Use-Case 2.0 staat erop dat je continue evalueert en verspilling voorkomt, de productiviteit verhoogt en in de pas blijft lopen met de steeds veranderende behoeften van het team.

Conclusie

Use-Case 2.0 is beschikbaar als een beproefde en goed gedefinieerde werkwijze en werkt prima samen met vele verschillende andere werkwijzen voor softwareontwikkeling zoals: Continuous Integration, Intentional Architecture en Test-Driven Development. Het werkt ook in combinatie met alle populaire besturingsmodellen. Het heeft de lichtvoetigheid en de flexibiliteit om teams te ondersteunen die op een *agile* manier werken. Ook kan het voorzien in de mate van compleetheid en mate van formaliteit die wordt vereist door een meer formele setting of waterval omgeving.

Use-Case 2.0 is:

- Lichtgewicht – In zowel zijn beschrijving als toepassing.
- Schaalbaar – en geschikt voor alle soorten en maten van teams en systemen.
- Veelzijdig – en geschikt voor alle soorten systemen en ontwikkelaanpakken.
- Eenvoudig toepasbaar – use-case modellen kunnen snel worden opgezet en de slices voorzien in de behoeften van het team.

Use-Case 2.0 is gratis en publiek beschikbaar door middel van dit handboek. In onze visie zijn de binnen Use-Case 2.0 onderkende 'dingen om mee te werken' en 'dingen om te doen' niet onderhandelbaar. Want hoewel het best mogelijk is om alleen onderdelen van Use-Case 2.0 te gebruiken, zal het resultaat onvoorspelbaar zijn en zal het geen Use-Case 2.0 zijn.

Dit handboek is opzettelijk beknopt gehouden en biedt mogelijk onafdoende informatie om er direct mee aan de slag te kunnen. Aanvullende informatie is beschikbaar in de vorm van een volledig gedocumenteerde werkwijze (practice) via www.ivarjacobson.com. De practice wordt aangeboden in de vorm van een stand-alone website of als een plug-in voor EssWork.

Dit handboek is de eerste in een reeks van publicaties over Use-Case 2.0. Je kunt vele andere artikelen, whitepapers en blogs verwachten over dit onderwerp op www.ivarjacobson.com.

Appendix 1: Werkproducten

Deze appendix bevat definities en aanvullende informatie over de werkproducten die worden gebruikt binnen Use-Case 2.0

Deze werkproducten zijn:

- Ondersteunende Informatie.
- Testgeval.
- Use-Case Model.
- Use-Case Beschrijving.
- Use-Case Realisatie.

Ondersteunende Informatie

Het doel van het werkproduct 'Ondersteunende Informatie' is het vastleggen van belangrijke termen die gebruikt worden om het systeem te beschrijven, en van alle requirements die niet passen binnen het use-case model.

De ondersteunende informatie:

- Helpt bij het verkrijgen van een gemeenschappelijk begrip van de gespecificeerde oplossing.
- Focust op concepten en termen die duidelijk moeten zijn voor iedereen die betrokken is bij de werkzaamheden, in het bijzonder die termen waaraan wordt gerefereerd door de use cases.
- Legt belangrijke globale eisen en kwaliteitsaspecten vast die individuele use cases overstijgen, zoals te ondersteunen platformen en de beschikbaarheid van het systeem.
- Benoemt iedere standaard waaraan moet worden voldaan. Bijvoorbeeld voor coderen, presentatie, taal, veiligheid en iedere andere van toepassing zijnde industriestandaard.
- Helpt bij het identificeren van aanvullende verhalen die niet direct kunnen worden afgeleid van use cases, zoals verhalen die benodigd zijn voor het demonstreren van de verschillende platformen of de gewenste mate van beschikbaarheid.

Ondersteunende informatie dient de evolutie van de use cases en de realisatie van de use-case slices te ondersteunen. Gebruik het om je use-case model aan te vullen en spraakverwarring tussen teamleden te voorkomen. De informatie kan afkomstig zijn uit vele bronnen, zoals andere requirements, specificaties, en discussies met belanghebbenden en materiedeskundigen. Zolang ze nuttig zijn voor het begrip van het use-case model en het systeem dat het beschrijft , vallen ook domein-, proces- en andere *business modellen* onder de noemer ondersteunende informatie.

De ondersteunende informatie kan met diverse mate van diepgang worden beschreven, van een eenvoudige set definities tot een uitvoerig en volledig beschreven set definities, standaards en kwaliteitseisen. De ondersteunende informatie kan worden beschreven op de volgende detailniveaus:

Contour: Een tussenliggend detailniveau dat aangeeft wat inbegrepen is en alleen een contour geeft van de belangrijkste termen en aandachtsgebieden.

Meer detail is vereist wanneer de informatie het succesvol identificeren en voorbereiden van de juiste use-case slices moet ondersteunen.

Eenvoudig gedefinieerd: Alle termen waarnaar wordt verwezen vanuit use-case beschrijvingen moeten gedefinieerd zijn en de algemene kwaliteitseisen dienen duidelijk gespecificeerd te zijn. Op dit detailniveau zijn ze vastgelegd in de vorm van eenvoudige lijsten van declaraties, bijvoorbeeld zoals de termenlijst in dit e-book.

Dit is de lichtste mate van detail die de realisatie van use-case slices ondersteunt. Het verheldert ook de globale systeemeisen, zodat het team kan vaststellen of de realisatie van de slices daadwerkelijk een systeem oplevert dat bruikbaar is, en niet alleen demonstreerbaar. Het is geschikt voor de meeste teams, in het bijzonder die teams die nauw samenwerken met hun gebruikers en in staat zijn om eventueel ontbrekende informatie mondeling aan te vullen.

Gemodelleerd en Geïllustreerd: Meer detail kan worden aangebracht door de basale definities om te zetten naar modellen die heel precies de definities beschrijven, inclusief hun attributen en relaties, eventueel verhelderd met concrete gebruiksvoorbeelden.

Op dit niveau gaan we voorbij aan eenvoudige definities en beginnen we aanvullende technieken te hanteren zoals een business rule catalogus, informatiemodellering en domeinmodellering. Het is vooral nuttig voor use-case modellen waarbij misverstanden over

de requirements kunnen leiden tot ernstige consequenties op het gebied van veiligheid, financiën of wetgeving.

Uitvoerig beschreven: Soms is het noodzakelijk om de informatie te verhelderen door middel van meer gedetailleerde uitleg en ondersteunende materialen zoals uitvoerig beschreven voorbeelden, bronnen en kruisverwijzingen.

Op dit detailniveau wordt de ondersteunende informatie meer gecompliceerd, met meer precisie en kruisverwijzingen. Ook kan gebruik worden gemaakt van formele specificatietechnieken.

De ondersteunende informatie biedt een centrale plek voor termen en afkortingen die nieuw zijn voor het team en ook voor algemene kwaliteitsattributen die ieder teamlid behoort te begrijpen. Het is een essentiële aanvulling op het use-case model. Zonder ondersteunende informatie is het onmogelijk om te weten wanneer het systeem bruikbaar en gebruiksklaar is.

De ondersteunende informatie wordt doorgaans weergegeven in de vorm van een eenvoudige lijst van termen en hun definities. De lijst is vaak opgedeeld in secties zoals definities van termen, bedrijfsregels, operationele beperkingen, ontwerpbeperkingen, standaarden en systeembrede requirements. De lijst kan worden gepubliceerd als onderdeel van een Wiki-site om toegankelijkheid en onderhoud te vereenvoudigen.

Testgeval

Een testgeval heeft als doel een duidelijke definitie te geven van wat het betekent om een deel (slice) van de requirements te voltooien. Een testgeval definieert een set van test inputs en verwachte resultaten met als doel te kunnen bepalen of het systeem al dan niet correct werkt.

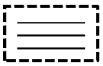
Testgevallen:

- Bieden de bouwstenen voor het ontwerpen en vervaardigen van testen.
- Bieden een mechanisme voor het voltooien en verifiëren van requirements.
- Zorgen ervoor dat testen kunnen worden gespecificeerd voorafgaand aan de realisatie.
- Bieden een manier om de kwaliteit van het systeem te beoordelen.

Vaak wordt vergeten dat testgevallen een belangrijk onderdeel zijn van een use case. De testgevallen bieden de echte definitie van wat een systeem behoort te doen om een use case mogelijk te maken. De test cases zijn vooral belangrijk bij het opknippen van de use case in slices omdat ze de ontwikkelaars een duidelijke definitie geven van de succesvolle realisatie van een use-case slice.

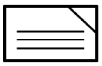
Testgevallen kunnen worden gebruikt voor verschillende soorten requirements werkwijzen, waaronder use cases, user stories en declaratief beschreven requirements. In alle gevallen moet een tester weten welk deel van de requirements moet worden getest, met een duidelijk begin en eind van waaruit een uitvoerbaar testscenario kan worden afgeleid.

Testgevallen kunnen worden uitgewerkt op de volgende detailniveaus:



Testaanpak bepaald: Het laagste detailniveau beschrijft alleen het initiële idee achter het testgeval. Bij het bepalen van een testgeval moet duidelijk zijn wat het idee achter het testgeval is en op welk deel van de requirements het van toepassing is.

Er moet meer detail worden toegevoegd om het testgeval daadwerkelijk te kunnen uitvoeren.

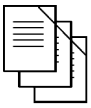


Scenario gekozen: Een tester heeft een testscenario nodig om een testgeval te kunnen uitvoeren. De structuur van de use-case beschrijving zorgt ervoor dat iedere use-case slice de tester een of meerdere kandidaat-testscenario's aanreikt. De kunst van het opstellen van effectieve testgevallen bestaat uit het kiezen van de juiste set potentiële testscenario's die invulling geven aan de testaanpak en die duidelijk aangeven wanneer een slice klaar is.

Dit is het laagste detailniveau dat nodig is voor een uitvoerbaar testgeval. Zodra het scenario is gekozen, is het testgeval afdoende gedetailleerd voor *exploratory* en *investigative testing*. Deze testsoorten kunnen zeer nuttig zijn en de inzichten die ze bieden kunnen zelfs van onschatbare waarde zijn, vooral wanneer een project zich nog in een vroeg stadium bevindt en de specificaties (en de oplossing) niet stabiel genoeg zijn voor formele, gescripte testsoorten.

Variabelen geïdentificeerd: Een testgeval neemt input, manipuleert toestanden binnen het systeem, en produceert een resultaat. Deze variabelen verschijnen als input, interne toestanden en output in de requirements. Op dit detailniveau zijn de acceptabele bereiken voor de belangrijkste variabelen binnen het scenario expliciet vastgesteld.

Dit detailniveau is geschikt voor die testgevallen waarbij het oordeel van de tester een essentieel onderdeel is van de test, bijvoorbeeld in geval van het testen van de bruikbaarheid. Het kan ook worden gebruikt wanneer meer structuur nodig is voor *exploratory* en *investigative*



Variabelen vastgesteld: Het testgeval kan verder worden uitgewerkt door alle variabelen binnen het testgeval expliciet van specifieke waarden te voorzien.

Deze mate van detail is geschikt voor handmatig testen omdat alle informatie beschikbaar is die een weldenkende tester nodig heeft om een test herhaaldelijk en consistent uit te voeren.

Gescript/Geautomatiseerd: Het is de moeite waard om een testgeval te automatiseren wanneer deze meerdere keren moet worden uitgevoerd of vele soorten testen ondersteunt.

Op dit detailniveau kan de test worden uitgevoerd zonder tussenkomst van een tester.

Een testgevallen is het belangrijkste aan een use case gerelateerd werkproduct. Onthoud dat het de testgevallen zijn die definiëren wat het betekent om de realisatie van een use case te voltooien, niet de use-case beschrijving. Op een andere manier bekeken zijn de testgevallen de beste manier waarop je requirements aangereikt kunt krijgen.

De testgevallen zullen worden gebruikt zolang het systeem in gebruik is. Ze worden niet alleen gebruikt tijdens de ontwikkeling van de use cases, maar ook voor regressietesten en andere kwaliteitscontroles. Het goede nieuws is dat de structuur van de use cases en de use-case beschrijvingen op een natuurlijk wijze leidt tot goed gedefinieerde, robuuste en veerkrachtige testgevallen; testgevallen die net zolang meegaan als het systeem invulling geeft aan de use cases.

Use-case beschrijvingen zijn verzamelingen verhalen waar het systeem in moet voorzien. Ieder in de use-case beschrijving beschreven verhaal dient minstens één testgeval te hebben. Je stelt de testgevallen gelijktijdig op met de use-case beschrijvingen als onderdeel van het voorbereiden van een use-case slice voor ontwikkeling.

Use-Case Model

Een use-case model is een model van alle waardevolle manieren om een systeem te gebruiken en de baten die dat oplevert. Een use-case model heeft als doel om alle waardevolle manieren om een systeem te gebruiken in een toegankelijk formaat vast te leggen om de requirements van het systeem vast te leggen en deze te gebruiken voor de aansturing van de ontwikkeling en verificatie.

Een use-case model:

- Stelt een team in staat om overeenstemming te krijgen over de vereiste functionaliteit en karakteristieken van een systeem.
- Bakent duidelijk de systeemgrenzen en de scope van het systeem af door een overzicht te geven van zijn actoren (buiten het systeem) en zijn use cases (binnen het systeem).
- Maakt agile requirements management mogelijk.

Een use-case model bestaat hoofdzakelijk uit een set actoren en use cases, en diagrammen die hun relaties illustreren. Use-case modellen kunnen op diverse manieren worden vastgelegd, bijvoorbeeld als onderdeel van een Wiki, op een whiteboard of flip-over, als een set MS-Powerpoint©dia's, in een MS-Word© document of in een modelleertool.

Use case modellen kunnen met de volgende detailniveaus worden uitgewerkt:

Waarde vastgesteld: De eerste stap op weg naar een compleet use-case model is het identificeren van de belangrijkste actoren en primaire use cases. Dit zijn diegene die werkelijk waarde leveren.

Dit is het laagste detailniveau. Het is geschikt voor de meeste projecten, vooral diegene die nieuwe functionaliteit toevoegen aan bestaande systemen waarbij het niet nuttig is om alles te modelleren wat het systeem al doet.

Systeemgrens afgebakend: De primaire actoren en use cases leggen de essentie vast van de reden waarom het systeem wordt ontwikkeld. Ze laten zien hoe de gebruikers waarde krijgen van het systeem. Het kan zijn dat dit onvoldoende inzicht geeft over hoe het systeem moet worden gestart en in leven moet worden gehouden. In dat geval kunnen secundaire actoren en use cases worden toegevoegd die nodig zijn voor het effectief onderhouden van het systeem.

Dit detailniveau is nodig voor het modelleren van fonkelnieuwe systemen of nieuwe generaties van bestaande systemen. Op dit detailniveau zijn alle actoren en use cases geïdentificeerd en gemodelleerd.

Gestructureerd: Het use-case model bevat vaak redundante informatie, zoals veel voorkomende stappen of interactiepatronen. Het structureren van het use-case model is een manier om deze redundante informatie weg te nemen.

Voor grote en complexe systemen, vooral degene die worden gebruikt om dezelfde functionaliteit te leveren binnen veel verschillende contexten, kan het structureren van het use-case model helpen bij de begripsvorming, het wegnemen van redundantie (verspilling) en het vinden van herbruikbare elementen.

Het use-case model dient zijn doel zolang het duidelijk maakt welke waarde belanghebbenden zullen ervaren van jouw nieuwe of aangepaste systeem. Wees terughoudend met het detailleren van het model. Ga alleen naar de niveaus 'Systeemgrens Afgebakend' of 'Gestructureerd' wanneer deze mate van detail duidelijk toegevoegde waarde biedt en je helpt om het nieuwe systeem efficiënter te ontwikkelen.

Use-Case Beschrijving

Het doel van de use-case beschrijving is om het verhaal te kunnen vertellen van hoe het systeem en zijn actoren samenwerken om een bepaald doel te behalen.

Use-case beschrijvingen:

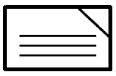
- Geven een contour van de verhalen waarmee requirements kunnen worden verkend en use-case slices kunnen worden geïdentificeerd.
- Beschrijven de volgorde van handelingen, inclusief varianten, die een systeem en zijn actoren uitvoeren om een bepaald doel te behalen.
- Worden vormgeven als een set van stromen die beschrijven hoe een actor het systeem gebruikt om een doel te behalen en wat het systeem moet doen voor de actor om dit mogelijk te maken.
- Leggen de requirements vast die nodig zijn voor andere ontwikkelactiviteiten.

Use-case beschrijvingen kunnen op vele verschillende manieren worden vastgelegd, zoals als onderdeel van een Wiki, op indexkaartjes, als MS-Word documenten of met behulp van een van de vele beschikbare commerciële tools voor taakbeheer, requirements management of modellering.

Use-case beschrijvingen kunnen worden opgesteld in verschillende mate van detail, variërend van een eenvoudige schets waarmee de basisstroom en de belangrijkste variaties worden geïdentificeerd tot de hoogste mate van detail waarbij alle acties, inputs en outputs worden gedefinieerd die zijn betrokken bij het uitvoeren van de use-case. Use-case beschrijvingen kunnen met de volgende mate van detail worden uitgewerkt:

Beknopt beschreven: De laagste mate van detail, die alleen het doel van de use case vastlegt en de actor die de use case start.

Dit detailniveau is geschikt voor die use cases waarvan je besluit ze (nog) niet te ontwikkelen. Meer detail is vereist zodra de use case moet worden opgedeeld (in use case slices) voor ontwikkeling.



Contourschets: De contouren van een use case moeten worden geschetst om gevoel te krijgen voor zijn omvang en complexiteit. Deze mate van detail maakt ook effectieve scopebeheersing mogelijk, omdat de schets je in staat stelt om de diverse delen van de use-case onderling te prioriteren en ze, indien nodig, toe te wijzen aan verschillende releases.

Dit is de lichtste mate van detail die je in staat stelt om een use case op te delen in use-case slices en deze te ontwikkelen. Het is geschikt voor teams die nauw samenwerken met hun gebruikers. Ontbrekende details worden dan conversationeel en door het verder uitwerken van de bijbehorende testgevallen verkregen.

Detailschets: Soms is het noodzakelijk om de verantwoordelijkheden van het systeem en zijn actoren te verduidelijken. Een contourschets legt hun verantwoordelijkheden vast, maar maakt niet duidelijk welke delen van de use case zijn toebedeeld aan het systeem en welke aan de actor(en).

Op dit detailniveau geeft de beschrijving een dialoog weer tussen het systeem en zijn actoren. Het is vooral geschikt voor het vaststellen van de architectuur voor een nieuw systeem of een nieuwe gebruikerservaring.

Volledig beschreven: Use-case beschrijvingen kunnen worden gebruikt om een zeer gedetailleerd pakket van eisen samen te stellen door de use-case op de hoogste mate van diepgang uit te werken: Volledig beschreven. Deze extra diepgang kan nodig zijn om een gebrek aan expertise binnen het team of een te grote afstand tot belanghebbenden te ondervangen, of om effectief complexe requirements over te dragen.

Deze mate van detail is vooral geschikt voor die use cases waar een misverstand over de inhoud ernstige consequenties kan hebben op het gebied van veiligheid, financiën of wetgeving. Het kan ook nuttig zijn in geval van *off-shoring* of *outsourcing* van softwareontwikkeling.

De use-case beschrijving is een zeer flexibel werkproduct waarvan het detailniveau benodigd om succesvol te zijn, kan worden aangepast aan de omstandigheden. Wanneer je deel uitmaakt van een klein team dat nauw samenwerkt met de gebruiker aan een project waarin ruimte is om samen dingen te ontdekken, dan bieden contourschetsen een lichtvoetige manier om dat te doen. Detailschetsen of volledig beschreven use cases kunnen worden gebruikt om kennisgaten binnen een team te dichten. Dergelijke kennisgaten zijn vaak aanwezig in een omgeving met weinig tot geen bewegingsruimte en zonder toegang tot de echte experts.

Niet iedere use-case beschrijving hoeft op hetzelfde detailniveau te worden uitgewerkt. Het is heel normaal dat de meest belangrijke en risicovolle use cases een hogere mate van detail kennen dan andere. Hetzelfde geldt voor de hoofdstukken van de use case beschrijving. Ook hier zijn de meest belangrijke, complexe of risicovolle delen vaak met meer detail uitgewerkt.

Use-Case Realisatie

Het doel van een use-case realisatie is om weer te geven hoe systeemelementen zoals componenten, programma's, stored procedures, configuratiebestanden en databasetabellen samen een use case uitvoeren. Use-case realisaties:

- Identificeren de systeemelementen die betrokken zijn bij de use case.
- Leggen de verantwoordelijkheden van systeemelementen vast bij het uitvoeren van de use case.
- Beschrijven hoe de systeemelementen samenwerken om de use case uit te voeren.
- Vertalen het businessjargon in de use-case beschrijvingen naar de taal die de ontwikkelaar spreekt bij het beschrijven van de technische realisatie.

Use-case realisaties zijn zeer nuttig en kunnen worden gebruikt om richting te geven aan het vervaardigen en valideren van de diverse invalshoeken die teams gebruiken om hun systemen te ontwerpen en op te bouwen. User-interface ontwerpers gebruiken bijvoorbeeld use-case realisaties (in de vorm van storyboards) om de impact van use cases op de gebruikersinterface te onderzoeken. Architecten gebruiken use-case realisaties bij het analyseren van use-cases die bepalend zijn voor de architectuur en om de geschiktheid van de architectuur te beoordelen. Use-case realisaties kunnen diverse verschijningsvormen hebben. De gekozen vorm is volledig afhankelijk van de door het team gekozen ontwikkelwerkwijzen. Veel voorkomende verschijningsvormen zijn: eenvoudige tabellen, storyboards, sequence diagrammen, collaboration diagrammen en data-flow diagrammen. Het belangrijkste is dat het team iets maakt waarmee duidelijk wordt welke systeemelementen betrokken zijn bij de uitvoering van de use case, en hoe deze worden geraakt.

Maak een use-case realisatie voor iedere use case, om de systeemelementen te bepalen die nodig zijn om de use cases uit te voeren en, nog belangrijker, bepaal in hoeverre deze moeten worden aangepast. Je kunt een use-case realisatie ook zien als een aanvulling of verdieping van een use-case beschrijving. De use-case beschrijving is dan het "wat" en de use-case realisatie is dan het "hoe".

Use-case realisaties kunnen op de volgende detailniveaus worden weergegeven:

Componenten geïdentificeerd: Het laagste detailniveau dat alleen de systeemelementen laat zien, zowel bestaande als nieuwe, die moeten samenwerken om invulling te geven aan de use case.

Dit detailniveau is geschikt voor kleine teams, die nauw samenwerken en die eenvoudige systemen vervaardigen waarvan de architectuur bekend is. Je kunt meer detail nodig hebben wanneer jouw systeem complex is of wanneer je onderdeel bent van een groot of gedistribueerd team.

Verantwoordelijkheden toegewezen: Om het team in staat te stellen om de geraakte systeemelementen in parallel aan te passen, of om meerdere slices tegelijk op te pakken, dienen ontwikkelaars de verantwoordelijkheden te kennen van de individuele elementen. De verantwoordelijkheden definiëren op hoog niveau wat ieder element zal doen, zal vastleggen en zal bewaken.

Dit detailniveau is geschikt in situaties waarbij iedere use-case slice meerdere systeemelementen raakt of waarbij de slices door meerdere ontwikkelaars parallel worden ontwikkeld. Het zou ook moeten worden gebruikt wanneer de systeemarchitectuur onvolwassen is en de basale verantwoordelijkheden van de systeemelementen nog niet helder zijn.

Interactie vastgesteld: Om van ieder bij een use case betrokken systeemelement volledig en eenduidig de wijzigingen vast te stellen, dient een use-case realisatie alle details van interfaces en interacties te bevatten die nodig zijn om de use case uit te voeren.

Deze mate van detail is vooral nuttig voor die use cases waarvan het ontwerp complex en uitdagend is. Het kan ook van pas komen wanneer systeemelementen moeten worden ontwikkeld door ontwikkelaars die weinig tot geen kennis hebben van het systeemontwerp, niet kunnen terugvallen op ervaren ontwerpers, en geen toestemming hebben om het ontwerp aan te passen. Het is ook nuttig voor onervaren ontwikkelaars die het vak nog moeten leren.

De use-case realisatie is een zeer flexibel werkproduct. Teams kunnen details toevoegen waar en wanneer ze daar behoefte aan hebben. Als een klein team gezamenlijk alle analyse- en ontwerpactiviteiten uitvoert, dan zijn eenvoudige, lichtvoetige use-case realisaties afdoende. Als een groot team niet in staat is om gezamenlijk veel analyse- en ontwerpsessies te houden voor de ontwikkeling van een complex systeem, dan zullen meer gedetailleerde realisaties helpen om het ontwerp over de bühne te krijgen. Ook zullen ze ervoor zorgen dat alle ontwikkelaars over alle informatie beschikken die nodig is om hun systeemelementen succesvol op te leveren en hun use-case slices te ontwikkelen.

Gebruikte Termen

Actor: Een actor definieert een rol die een gebruiker kan spelen die met het systeem werkt. Een gebruiker kan een persoon zijn of een ander systeem. Actoren hebben een naam en een korte omschrijving en worden geassocieerd met de use cases waarmee ze interactie hebben.
Alternatieve stroom (Engels: alternative flow) Een variatie op de basisstroom van een use-case beschrijving. Er kunnen meerdere alternatieve stromen zijn.
Applicatie: Computer software ontworpen om actoren te helpen bepaalde taken uit te voeren.
Aspect Oriented Programming: Een programmeertechniek die zich richt op het meer modulair maken van software door afzondering van *cross-cutting concerns* mogelijk te maken (zie http://en.wikipedia.org/wiki/Aspect-oriented_programming).
Basisstroom (Engels: basic flow)**:** Een beschrijving van een route door een use-case beschrijving. Vaak de eenvoudigste route die de meeste toegevoegde waarde biedt voor een gebruiker. Er is altijd maar één basisstroom.
Belanghebbende (Engels: stakeholder)**:** Een persoon, groep of organisatie die invloed uitoefent op, of wordt geraakt door het softwaresysteem.
Gebruiker: Een belanghebbende die met het systeem werkt om zijn doelen te behalen.
Klant: De belanghebbende die de ontwikkeling van het systeem financiert of waarvan wordt verwacht dat die het systeem koopt zodra het voltooid is.
Requirements: Wat het softwaresysteem moet doen om te voorzien in de behoeften van de belanghebbenden.
Separation of Concerns: Het proces van het opdelen van een systeem om overlap in functionaliteit te minimaliseren (zie http://en.wikipedia.org/wiki/Separation_of_Concerns).
Softwaresysteem: Een systeem bestaande uit software, hardware en digitale informatie, dat zijn toegevoegde waarde vooral levert door middel van zijn software. Een softwaresysteem kan onderdeel uitmaken van een grotere software, hardware, business of sociale oplossing.
Stroom (Engels: flow) Een beschrijving van een gehele of gedeeltelijke route door een use-case beschrijving. Er is altijd minstens een basisstroom, mogelijk aangevuld met een of meer alternatieve stromen.
Systeem: Een groep van dingen of delen die samenwerken of onderling verbonden zijn om samen een geheel te vormen. Meestal gebruikt om te verwijzen naar het onderwerp van het use-case model: het te bouwen product.
Systeemelement: Lid van een reeks elementen die samen een systeem vormen (15288:2008 ISO/IEC).
Testgeval (Engels: test case): Een testgeval definieert een set testcondities en verwachte resultaten met als doel te bepalen of een systeem al dan niet correct werkt.
Use Case: Een use case omvat alle manieren waarop het systeem gebruikt kan worden om een bepaald doel voor een bepaalde gebruiker te behalen.
Use-Case 2.0: Een schaalbare, agile werkwijze die use cases gebruikt voor het vastleggen van een set requirements (systeemeisen) en het aansturen van de incrementele ontwikkeling van het systeem dat hierin moet voorzien.
Use-Case Diagram: Een diagram waarop een aantal actoren, use cases en hun relaties inzichtelijk wordt gemaakt.
Use-Case Model: Een model van alle zinvolle manieren om een systeem te gebruiken, en de waarde die dat oplevert. Een use-case model bestaat hoofdzakelijk uit een set actoren en een set use cases, en diagrammen waarmee hun relaties worden geïllustreerd.

Use-Case Beschrijving (Engels: use-case narrative)**:** Een beschrijving van een use case waarmee wordt verteld hoe het systeem en zijn actoren samenwerken om een bepaald doel te behalen. Het bevat een volgorde van handelingen en varianten daarop, die een systeem en zijn actoren kunnen uitvoeren om een bepaald doel te behalen.
Use-Case Slice: Een use-case slice is de selectie van een of meer verhalen uit een use case om een werkpakket te vormen dat een duidelijke toegevoegde waarde heeft voor een klant.
Verhaal (Engels: story): Een voor een gebruiker of andere belanghebbende zinvolle manier om het systeem te gebruiken. Binnen Use-Case 2.0 wordt een verhaal beschreven als onderdeel van een use-case beschrijving, een of meer stromen en aanvullende eisen, en een of meer testgevallen.

Dankbetuiging

Algemeen

Use-Case 2.0 is gebaseerd op binnen de industrie geaccepteerde best practices die al tientallen jaren in gebruik zijn en hun waarde bewezen hebben. We willen de tienduizenden mensen bedanken die use cases dagelijks gebruiken in hun projecten en in het bijzonder diegenen die hun ervaringen binnen en buiten hun eigen organisatie hebben gedeeld. Zonder hun harde werk en enthousiasme hadden we nooit de motivatie en de kennis gehad om de volgende evolutie van deze techniek mogelijk te maken. We hopen dat je dit korte e-book handig vindt, en dat je blijft kijken naar de manier waarop je use cases toepast en je aanpak waar nodig bijstelt.

Voor de Engelse versie

We willen ook iedereen bedanken die een directe bijdrage heeft geleverd aan het vervaardigen van (de oorspronkelijke Engelse versie van dit e-book, in willekeurige volgorde: Paul MacMahon, Richard Schaaf , Eric Lopes Cardozo, Svante Lidman, Craig Lucia, Tony Ludwig, Ron Garton, Burkhard Perkens-Golomb, Arran Hartgroves, James Gamble, Brian Hooper, Stefan Bylund and Pan-Wei Ng.

Voor de Nederlandse versie

Wij willen iedereen bedanken die een directe bijdrage heeft geleverd aan het vervaardigen van deze Nederlandse versie, in willekeurige volgorde: Marco Balvers, Sander Daniels , Dennis Geluk, Rob den Hollander, Stefan Jansen, Pieter Mandemaker, Nicole de Swart, Guido Zwiebel-Klaeijsen,Selmar van Egmond, Marc Faber, Remko van der Mei en Jack Westman

Bibliografie

Object Oriented Software Engineering: A Use Case Driven Approach

Ivar Jacobson, Magnus Christerson, Patrik Jonsson, Gunnar Overgaard

Het oorspronkelijke boek waarmee use cases werden geïntroduceerd.

- Uitgever: Addison-Wesley Professional; Herziene uitgave (10 juli 1992)
- ISBN-10: 0201544350
- ISBN-13: 978-0201544350

The Object Advantage: Business Process Reengineering With Object Technology

Ivar Jacobson, Maria Ericsson, Agneta Jacobson

Het definitieve handboek over het gebruik van use cases voor business process reengineering.

- Uitgever: Addison-Wesley Professional (30 september 1994)
- ISBN-10: 0201422891
- ISBN-13: 978-0201422894

Software Reuse: Architecture, Process and Organization for Business Success

Ivar Jacobson, Martin Griss, Patrik Jonsson

Een uitvoerig handboek over software-hergebruik, inclusief gedetailleerde aanwijzingen voor het gebruik van use cases voor het ontwikkelen van productfamilies en systemen-van-systemen.

- Uitgever: Addison-Wesley Professional (1 juni 1997)
- Language: English
- ISBN-10: 0201924765
- ISBN-13: 978-0201924763

Use-Case Modeling

Kurt Bittner and Ian Spence

Het ultieme handboek over het opstellen van use-case modellen en het schrijven van goede use cases.

- Uitgever: Addison-Wesley Professional; 1e editie (30 augustus 2002)
- ISBN-10: 0201709139
- ISBN-13: 978-0201709131

Aspect-Oriented Software Development with Use Cases

Ivar Jacobson and Pan-Wei Ng

Het boek waarmee use-case slices werden geïntroduceerd onder hun vorige naam: use-case modules.

- Uitgever: Addison-Wesley Professional; 1e editie (9 januari 2005)
- ISBN-10: 0321268881
- ISBN-13: 978-0321268884

Over de Auteurs

Ivar Jacobson

Dr. Ivar Jacobson is de vader van componenten en op componenten gebaseerde architecturen, use cases, aspect-oriented softwareontwikkeling, moderne business engineering, Unified Modeling Language en het Rational Unified Process.Zijn meest recente bijdrage aan de software-industrie is een formeel *practice* concept dat practices (werkwijzen, procescomponenten) promoot als 'eersteklas burgers' van de softwareontwikkeling en waarmee processen worden beschouwd als een samenstelling van practices. Hij is de belangrijkste auteur van zes invloedrijke bestsellers. Hij is keynote spreker op vele grote conferenties over de gehele wereld en heeft verscheidene consultants op het gebied van procesverbetering getraind.

Ian Spence

Ian is Chief Technology Officer van Ivar Jacobson International waar hij is gespecialiseerd in het *agile* toepassen van het Unified Process. Hij is certified RUP practitioner, Scrum Master en een ervaren coach die honderden projecten heeft geholpen met het toepassen van iteratieve en agile technieken. Hij heeft meer dan 20 jaar ervaring in de software-industrie, waaronder de gehele ontwikkel-levenscyclus, inclusief het vastleggen van requirements, architectuur, analyse, ontwerp, coderen en project management. Zijn specialisaties zijn iteratief project management, agile teamwork en requirements management met use cases. Vanuit zijn rol als CTO geeft Ian richting aan de technologische ontwikkelingen binnen Ivar Jacobson International en werkt hij met IJI's Technology Office aan de volgende generatie van slimme, actieve werkwijzen (practices) voor softwareontwikkeling. Hij is de projectleider en procesarchitect voor de ontwikkeling van het Essential Unified Process en de practices die het bevat. Wanneer hij niet werkt aan het onderzoeken, vastleggen en definiëren van practices, is hij actief met het helpen van bedrijven bij het vormgeven en uitvoeren van veranderprogramma's voor het verbeteren van softwareontwikkeling. Hij is mede-auteur van de Addison-Wesley boeken "Use Case Modeling" en "Managing Iterative Software Development Projects".

Kurt Bittner

Kurt is Chief Technology Officer van Ivar Jacobson International, America. Hij heeft meer dan 25 jaar werkervaring in de software-industrie en gewerkt in tal van rollen, waaronder: ontwikkelaar, teamleider, architect, projectmanager en bedrijfsleider. Hij heeft agile projecten geleid, is verantwoordelijk geweest voor een grote divisie van een softwareontwikkelbedrijf, heeft diverse start-ups helpen overleven en opbloeien, heeft een overname begeleid, en gewerkt in tal van branches waaronder luchtvaart, financiën, energie en elektronica. Hij had een sleutelrol bij de eerste ontwikkeling van het Rational Unified Process en meer recent van IBM's Jazz project. Zijn ervaring omvat belangrijke werkzaamheden in de bancaire en financiële wereld, op het gebied van architectuur en ontwerp van relationele databasesystemen, en consultancy en mentoring van een grote schare klanten op het gebied van strategieën en aanpakken voor het verbeteren van softwareontwikkeling. Hij is mede-auteur van "Use Case Modeling", "Managing Iterative Software Development Projects" en "The Economics of Iterative Software Development"

Eric Lopes Cardozo

Eric is Principal Consultant van Ivar Jacobson International. Hij heeft meer dan 20 jaar ervaring in de software-industrie en gewerkt in tal van rollen, van programmeur tot programma manager en ook die van coach, docent en ondernemer. Eric heeft al vele projecten en organisaties geholpen met het toepassen van iteratieve en agile technieken. Zijn ervaring omvat de gehele ontwikkel-levenscyclus, van idee tot en met uitrol, en vele branches, waaronder bancair, verzekeringen, energie, voedingsmiddelen,

(petro)chemie, telecom en (semi)overheid. Hij was projectleider van het team dat verantwoordelijk was voor het herschrijven van de RUP® Business Modeling discipline. Hij is ook de auteur van “the seven habits of effective iterative development”, een artikel dat door IBM is verheven tot standaardwerk voor de introductie van iteratief ontwikkelen.

Over Ivar Jacobson International

IJI helpt wereldwijd softwareontwikkelteams met het verbeteren van hun prestaties en met het wegnemen van barrières die de adoptie van nieuwe werkwijzen in de weg staan. Met behulp van zeer ervaren mensen, innovatieve werkwijzen en bewezen oplossingen bieden wij onze klanten een optimale samenwerking tussen business en IT, sterk presterende teams en projecten die resultaten leveren.

www.ivarjacobson.com

Zweden
+46 8 515 10 174
info-se@ivarjacobson.com

Nederland
+31(0) 20 654 1878
info-nl@ivarjacobson.com

Groot-Brittannië
+44 (0)1189 001 460
info-uk@ivarjacobson.com

Azië
+8610 82486030
info-asia@ivarjacobson.com

Amerika
+1 703 338 5421
info-usa@ivarjacobson.com