Eén van de grootste uitdagingen in software-ontwikkeling is hoe en wanneer de benodigde inspanning voor een project kan worden geschat. In de meer dan dertig jaar dat de software-ontwikkeling bestaat is dit een lastig aspect gebleken. Aangezien slechts zestien procent van alle projecten binnen hun oorspronkelijke budget en tijd blijven, is het in vooral moeilijk vooraf een goede schatting af te geven. Een vriend vertelde onlangs dat in zijn organisatie schattingen nog altijd natte-vinger-werk zijn, ondanks dertig jaar ervaring in Cobol. En als schattingen al lastig zijn voor traditionele, lineaire projecten, wat zegt dit over het plannen van projecten met moderne technieken en technologieën, zoals agile, iteratieve methodieken, UML, object-oriëntatie en web development? Kunnen we dit überhaupt?

thema

Schatten met use cases

Voorspellen van hoeveelheid werk in projecten

Tijd om de mogelijkheden te onderzoeken hoe je (ruwweg) correcte schattingen kunt leveren in webgerelateerde of objectgeoriënteerde projecten. Dit artikel behandelt een aantal onderwerpen die hiermee samenhangen. Dit is een zoektocht naar een zinvolle eenheid voor het schatten van tijd en complexiteit. Het artikel motiveert waarom en hoe use cases deze eenheid kunnen zijn, dit in tegenstelling tot user stories zoals voorgesteld in Extreme Programming. Vervolgens wordt een model voor schatten uitgewerkt, dat deze use cases als eenheid gebruikt. Tenslotte wordt ingegaan op het iteratieve en incrementele karakter van agile methodologieën en hoe dit behulpzaam is bij het optimaliseren van het schattingsmodel.

UITDAGINGEN BIJ PROJECTPLANNING Waarom is het zo moeilijk een inschatting te maken van het werk in een project? Twee aspecten zijn in hoge mate bepalend. Het eerste aspect wordt het beste verwoord door Frank Zappa. In een interview zei de componist: '*What something is, is more influenced by when it is then anything else*'.

Dit geldt zeker voor het schatten van een project. De meeste projecten vereisen op voorhand een nauwkeurige schatting van het benodigde werk. Dat wil zeggen, een inschatting die wordt gebruikt om te beslissen of het project al dan niet doorgang vindt. Ieder project is immers anders. Verschillende teams werken op verschillende manieren samen. Ook wanneer hetzelfde team gedurende twee opeenvolgende projecten in dezelfde samenstelling samenwerkt, zal het nooit twee keer op dezelfde manier te werk gaan. Een project begint weliswaar zonder veel kennis, gedurende het project groeit de opgedane kennis, krijgt de samenwerking binnen het team gestalte en wordt pas de complexiteit van het probleemdomein ontdekt. De beste inschatting kan dus pas geleverd worden zodra het project voltooid is. Tot die tijd is het niet mogelijk een perfecte inschatting te maken. Dit mag een weinig bruikbare constatering zijn, het maakt wel duidelijk dat het op voorhand inschatten en plannen van het werk riskant is, gezien de zeer minimale informatie die beschikbaar.

> Noch functiepuntanalyse, noch het tellen van classes en entiteiten is afdoende voor schattingen van het werk

Laten we eens kijken naar de manier waarop planningen zijn samengesteld. Een planning kent doorgaans twee variabelen. De eerste variabele is de complexiteit van het systeem dat gebouwd wordt. De tweede variabele is de snelheid waarmee het team opereert, ofwel de mate van complexiteit die een team kan behappen in een gegeven tijdsbestek. U kent zonder

twijfel uitspraken als "We doen zes uur per functiepunt." De inspanning voor een project wordt berekend als het product van deze twee variabelen. Dit resulteert in de totale hoeveelheid werk die nodig is om het project te voltooien. Hierbij dienen zich twee uitdagingen aan. Projectteams hebben een waarde nodig om de totale complexiteit van het systeem te duiden. Daarnaast is er een goede indicator om de snelheid van het team aan te geven. In Smart en Extreme Programming wordt dit de velocity van het team genoemd. Laten we eens nauwkeuriger kijken naar deze twee uitdagingen.

Een eenheid van complexiteit Hoe kan een project de totale complexiteit van het systeem bepalen? Met name in de vroege fasen van een project is informatie schaars. Hoeveel kennis van het domein is beschikbaar bij de start van een project? Het beste is een behoorlijke meeteenheid te vinden voor deze complexiteit en deze vervolgens gedurende het hele project te hanteren. Vervolgens is het zaak het domein van het systeem uit te drukken in de gekozen meeteenheid. Maar welke meeteenheid is beschikbaar als de eerste tijdsplanning wordt gemaakt?

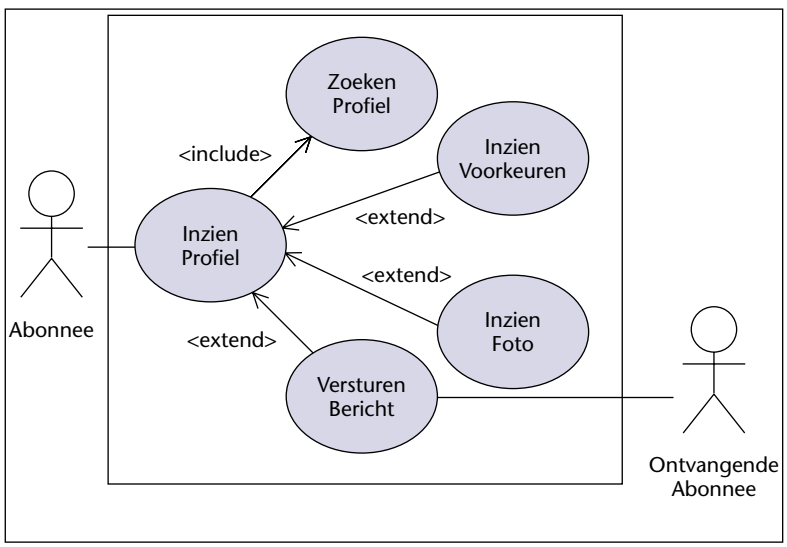
> In plaats van te kijken naar de technische componenten beschrijven use cases de bedrijfsprocessen

Traditioneel wordt de techniek functiepuntanalyse (FPA) gebruikt. Deze techniek meet software door de functionaliteit waarin een systeem voorziet te kwantificeren. FPA verdeelt de functionaliteit van het systeem in een aantal onderdelen, zoals externe inputs, externe outputs, externe inquiry's, interne logische bestanden en externe interface-bestanden. Deze opdeling legt de tekortkomingen van FPA bloot als het aankomt op een eerste schatting. FPA is oorspronkelijk bedoeld om de totale hoeveelheid *programmeerwerk* in te schatten in een project. Dit gebeurt dus pas nadat het functioneel ontwerp is gemaakt en niet al aan het begin van een project. Dat blijkt uit de vereiste ontwerpkennis bij de identificatie van de onderdelen. To the point: geen enkel teamlid beschikt over dergelijke diepgaande kennis bij de start van een project. Bovendien ontbreekt een goede mapping van de FPA- onderdelen naar webgerelateerde of objectgeoriënteerde projecten. Functiepunt-analyse is verre van ideaal voor een eerste inschatting.

Met enige regelmaat maak ik projecten mee die business classes of entiteiten tellen. Maar om dezelfde redenen als welke gelden voor FPA zijn deze technieken evenmin toepasbaar aan het begin van een project. Ook hier is diepgaande kennis vereist van het systeem. Bovendien zijn noch business classes noch entiteiten een goede meeteenheid voor complexiteit. Er gebeurt in systemen namelijk aanzienlijk meer dan alleen het construeren van entiteiten of business classes. Hoe zit het bijvoorbeeld met requirements-analyse, web design, of het creëren van interfaces naar andere systemen?

Use cases als eenheid voor complexiteit Maar welke eenheid voor complexiteit is nu wel bruikbaar? Tegenwoordig is er een techniek die in heel veel projecten wordt toegepast wordt. Deze techniek beschrijft de requirements en wordt vroeg genoeg in het project toegepast om van nut te zijn. Deze techniek is de use case.

Use cases worden afgebeeld als ellipsen in use case diagrammen. Iedere use case modelleert requirements die worden geïmplementeerd in het systeem. Iedere use case wordt uitgevoerd door één of meer actoren. Een actor modelleert de rol die gebruikers, andere systemen of organisaties hebben in relatie tot het gemodelleerde systeem. Iedere use case heeft een korte en soms uitgebreide beschrijving, afhankelijk van de methodologie of expert die wordt geraadpleegd. Deze beschrijving wordt echter pas later in het project opgesteld, meestal gedurende ontwerp. Figuur 1 toont zo'n use case diagram.



Figuur 1. Een use case diagram

Laat we dit eens onderzoeken. Er pleit veel voor het gebruik van use cases. Use cases worden gebruikt om requirements vast te leggen in de vroege fasen van een project. Indien op tijd gemodelleerd zijn ze op het juiste moment beschikbaar om een eerste inschatting te maken. Use cases kunnen worden gemodelleerd op een

vervolg op pagina 49.

vervolg van pagina 8.

'hoog' abstractieniveau. Dat betekent dat analyse van de use cases kan worden uitgevoerd zonder diepgaande kennis van het systeem. Bij het modelleren van use cases worden technische details achterwege gelaten. In plaats van te kijken naar de verschillende technische componenten waaruit de functionaliteit is opgebouwd, beschrijven use cases de bedrijfsprocessen die door het systeem ondersteund worden. Use cases zien er dus veelbelovend uit als eenheid van complexiteit in projecten, in het bijzonder in de vroegere fasen van een project.

Om een lang verhaal kort te maken: use cases maken hun belofte waar. Schatten met use cases is een techniek die is geïntroduceerd in de agile methodiek Smart. Ervaring in dit soort projecten leert dat use cases niet alleen een goede eenheid zijn voor het bepalen van complexiteit, maar ook zeer goed functioneren als de primaire eenheid van werk in een project. In Smart worden use cases derhalve gebruikt voor het vaststellen van prioriteiten voor iteraties en incrementen. Daarnaast gelden ze als basis voor het modelleren van het ontwerp, en om het modelleren van testscenario's en test cases aan te sturen.

Het schatten van de complexiteit van use cases Hoe wordt dit Zwitserse zakmes gebruikt voor het schatten van complexiteit? Ook al handelt dit artikel niet over de vraag hoe use cases worden gemodelleerd, ontkom ik niet aan een nadere beschouwing. Iedere use case beschrijft een bedrijfsproces dat wordt uitgevoerd door één of meerdere actoren. Als use cases tot op afdoende detailniveau zijn gemodelleerd, beschrijven ze de manier waarop een actor het bedrijfsproces uitvoert. Uitgedrukt in business modeling-terminologie kun je zeggen dat elke use case op dit niveau overeenkomt met exact één elementair bedrijfsproces (EBP). Zo'n EBP of task geldt als een enkele interactie met de bijbehorende actor. De interactie bij het uitvoeren van een use case verloopt vrijwel altijd volgens hetzelfde patroon:

- *Starten*. De task wordt gestart
- *Opbouwen*. Er wordt een webpagina opgebouwd die visuele elementen combineert met de gegevens die worden getoond. Deze gegevens worden eerst door de task onttrokken uit business classes, componenten, of via interfaces naar andere systemen.
- *Tonen*. De webpagina wordt teruggestuurd naar de browser van de actor. Deze kan nu activiteiten ondernemen.
- *Interacteren*. De actor interacteert met de webpagina, voert activiteiten uit en bevestigt of annuleert deze, op wat voor manier dan ook.
- *Uitvoeren*. De task voert de activiteiten van de actor uit. Vaak worden gewijzigde of nieuwe gegevens opgeslagen in de database, soms ook teruggekoppeld naar componenten of andere applicaties.

Dit patroon geeft een prima indicatie van de elementen die nodig zijn voor het uitvoeren van een use case, en dus voor het inschatten van de complexiteit van deze use case. Een aantal klassen wordt geconstrueerd, aangepast of hergebruikt om de use case te implementeren. Een task implementeert het bedrijfsproces. Er wordt een webpagina ontworpen. Business classes, componenten en interfaces naar andere applicaties behandelen de gegevens. Ik merk hierbij op dat al deze klassen netjes op hun plaats vallen in de lagen van vrijwel iedere gangbare meerlagen-architectuur. Systemen worden volgens zo'n architectuur gebouwd om verantwoordelijkheden afdoende te kunnen scheiden.

> Er is maar één manier beschikbaar voor de greenfield inschatting van de velocity: gokken

Tijd om deze kennis te gebruiken voor het ramen van de complexiteit van een use case. Dit kan gelukkig op een abstract en weinig gedetailleerd detailniveau. In de vroegere fasen van een project volstaat immers een ruwe schatting. Zelfs zonder veel ervaring kan de complexiteit van use cases al worden vastgesteld als slechts de naam en het doel van de use case bekend zijn.

Een voorbeeld: de use case **Inzien Profiel** uit het use case diagram Figuur 1. Dit voorbeeld is ontleend aan een online dating service die Dare2Date heet. Hierbij kan een abonnee op de dating service het profiel van een andere abonnee bekijken. Toepassen van het patroon levert onderstaande klassen:

- De task **InzienProfiel** wordt gestart. Hierbij worden bij voorkeur de precondities van de use case gevalideerd.
- Een webpagina **PageInzienProfiel** voor het tonen van het profiel
- Een business class **Profiel** die de gegevens van het profiel beheert. Waarschijnlijk ook een business class **Abonnee** die de huidige abonnee toont op de webpagina.
- Een stored procedure **GetProfiel** die door de business class wordt gebruikt om de database te raadplegen.

De totale ontwikkelinspanning die geleverd wordt voor de use case is eenvoudig af te leiden uit boven-

staande kennis. Stel de complexiteit per klasse vast. Geef hiervan een grove schatting, bijvoorbeeld op een schaal van 1 tot 5. Hierbij staat 1 voor 'eenvoudig' en 5 voor 'complex'. In meer detail treden is onnodig aangezien de meeste systemen bestaan uit een groot aantal use cases. Eventuele schoonheidsfouten zijn tegen elkaar weg te strepen vanwege het grote aantal use cases. Schroom niet om de complexiteit van een use case volgens dit recept binnen de minuut vast te stellen.

De task uit het voorbeeld wordt ingeschat als een 4, de webpagina krijgt waarde 4. De business classes **Profiel** en **Abonnee** worden uitsluitend gebruikt om gegevens op te halen. Dit is eenvoudige functionaliteit, allebei waarde 2. Ook de stored procedure is betrekkelijk eenvoudig en krijgt waarde 3. Use case **Inzien Profiel** heeft een totale complexiteit van 4 + 4 + 2 x 2 + 3= 15.

Dit patroon voor het schatten van complexiteit op basis van use cases en een meerlagenarchitectuur, blijkt in de praktijk gemakkelijk toepasbaar. Sinds de introductie van deze techniek in Smart is dit patroon frequent gebruikt met overtuigend resultaat. Een organisatie meldt dat hun initiële inschattingen slechts tien procent van het uiteindelijke projectresultaat afwijken.

Vaststellen en ijken van velocity Eén probleem is nu uit de weg. De complexiteit van het systeem is vastgesteld. Nu de tweede variabele: de snelheid of velocity van het team. Zoals gezegd is een groot probleem dat ieder project anders is, in een andere omgeving wordt uitgevoerd, met andere mensen, die bovendien gebruik maken van andere tools en andere technieken. Is het dan nog wel mogelijk één waarde af te leiden om de velocity van het team te duiden? Een eerlijk antwoord? Nee, althans niet totdat het project is voltooid, of zoals in agile methodieken totdat een deel van het werk is voltooid. Geen enkele waarde waarmee een team of projectmanager bij de start van een project op de proppen komt, vertegenwoordigt de exacte velocity.

Schroom niet om de complexiteit van een use case volgens dit recept binnen de minuut vast te stellen.

Dit lijkt een groot probleem. Het is waar dat projecten nooit in staat zijn een exacte inschatting te geven van de velocity van een project. Projecten kunnen echter behoorlijk in de buurt komen door de voortgang steeds opnieuw te ijken. In agile methodieken, zoals extreme programming, Smart en Crystal, wordt gewerkt in korte iteraties. Iedere iteratie levert een stuk voor de klant waardevolle software op. De activiteiten in de iteraties komen steeds overeen. Zo kent Smart binnen haar iteraties een nagenoeg dagelijkse cyclus waarin een use case wordt ontworpen, gebouwd en getest. Nu kan de velocity van het team aan het eind van iedere iteratie worden gemeten, puur gebaseerd op basis van het tot dan toe opgeleverde werk.

Als gedurende een iteratie zes use cases zijn geaccepteerd, zeg met een totale complexiteit van 90, en de iteratie duurt drie weken, dan is vast te stellen dat het team op één werkdag een gemiddelde complexiteit van 6 realiseert. De velocity wordt voor volgende iteraties vastgesteld op 6. Dit geeft de mogelijkheid om meer nauwkeurige inschattingen af te geven voor latere iteraties. Sterker nog, de velocity is mede bepalend voor het aantal use cases dat tijdens volgende iteraties wordt uitgevoerd. Overigens geldt natuurlijk ook dat hoe meer iteraties voltooid zijn, des te beter de schattingen zijn.

Terug naar het voorbeeld. Use case **Inzien Profiel** had een complexiteit van 15. Het ligt in de lijn der verwachtingen dat het bovenstaande team zo'n twee à drie dagen nodig heeft voor het realiseren van deze use case.

De initiële velocity Eén onderwerp blijft onbesproken. Hoewel het herijken van de velocity een duidelijke activiteit is, is het vaststellen van de initiële velocity een ander verhaal. Het is immers de eerste keer dat het team samenwerkt in precies deze configuratie aan een nieuw project. Er is maar één manier beschikbaar voor de greenfield inschatting van de velocity: gokken. Gebruik de ervaring die de teamleden uit eerdere projecten in vergelijkbare rollen meenemen. Gebruik de velocity van eerdere projecten die in een vergelijkbare configuratie zijn uitgevoerd. Probeer een realistische velocity vast te stellen waarin de teamleden zich kunnen vinden. Wees redelijk en vermijd overmatig optimisme.

Helaas is het gebruik van use cases voor schattingen een redelijk nieuwe discipline. Er is weinig literatuur over dit onderwerp beschikbaar. Nog niet. De ervaring totnogtoe in Smart-projecten toont een voortgang variërend van drie tot zes dagen per use case. Ik geef u nog een paar aanvullende tips.

Meet voortdurend Projecten binnen uw eigen organisatie worden in vergelijkbare configuraties uitgevoerd, gebruikmakend van dezelfde ontwikkelomgeving, dezelfde tools, en met name dezelfde methodiek. Dus alhoewel ieder project nog steeds uniek is, kennen uw projecten een vergelijkbare werkwijze. Het meten van de velocity in projecten lijkt een nogal omvangrijke klus, gegeven de benodigde administratie van de uren waarin aan use cases is gewerkt. Reden voor veel projecten om te stoppen met meten. Desondanks is er geen alternatief voor het verkrijgen van een betrouwba-

re velocity. Meet daarom voortdurend. Zorg daarbij wel voor dat de administratie minimaal is. Noteer bijvoorbeeld alleen het totaal aantal bestede uren in een iteratie en verdeel dit over de gerealiseerde use cases. Ook dit geeft al snel bevredigend resultaat.

Wees consistent In de literatuur verschillen de meningen over het optimale detailniveau van use cases. Maakt u zich hierover geen zorgen. Wanneer u use cases gebruikt voor het meten van de complexiteit van projecten, wees dan consistent in het gekozen detailniveau. Dit is essentieel om de velocity in iteraties en over projecten te vergelijken en te hergebruiken.

Deze consistentie is één van de uitdagingen bij het plannen binnen extreme programming. Hier hanteert men user stories voor het inschatten van het werk. Deze user stories worden geschreven door de klant. Het werk wordt ingeschat door de ontwikkelaars. Dit biedt de klant grote flexibiliteit. De klant kan nagenoeg alles kwijt in een user story. Afhankelijk van de klant wisselt het detailniveau van user stories van projecten tot project. Het loopt uiteen van het plaatsen van het bedrijfslogo op alle webpagina's tot het maken van een batch voor een salarisadministratie. Er is geen consistentie in het detailniveau van user stories. Dit maakt het extra moeilijk de complexiteit van het systeem vast te stellen. Er is geen patroon.

User stories geven bovendien een extra uitdaging bij het maken van een initiële inschatting. De collectie user stories biedt geen garantie voor een goed overzicht over het systeem. User stories kunnen afkomstig zijn uit verschillende delen van het systeem, en niets met elkaar te maken hebben. Hierdoor is de scope en de totale complexiteit van het systeem lastig vast te stellen. Een suggestie die ik geef is om in extreme programming user stories door use cases te vervangen.

Vermijdt details Het valt mij keer op keer op dat projecten nog voor het maken van een eerste schatting de requirements van een systeem tot op zeer laag detailniveau uitwerken. Projecten hopen hiermee meer zekerheid over hun schattingen te verkrijgen. Dit is echter een schijnzekerheid. Het detailniveau van de requirements overstijgt steeds opnieuw de aanwezig kennis, waardoor aannames of diepgravend onderzoek volgt. Pas altijd op met "the big design upfront". Projecten die zich hieraan overgeven, lopen steevast tegen onfrisse verrassingen later in het project.

Geen van de agile methodieken bezoldigt zich aan the big design upfront. In Smart worden de use case tijdens de eerste fase gemiddeld geproduceerd in twee tot vier gefaciliteerde workshops die zo'n twee à drie uur duren. Dit blijkt voldoende.

Itereer kort Het toepassen van iteratieve werkwijzen is een grote stap voorwaarts om grip te krijgen op de voortgang van een project. Gedurende elke iteratie wordt een aantal use cases ontworpen, gerealiseerd, getest en geaccepteerd. Dit biedt niet alleen flexibiliteit, maar stelt projecten ook in de gelegenheid hun velocity op korte termijn te herijken. Een reële velocity is van groot belang voor het plannen van toekomstige iteraties. Zo kan het aantal use cases dat wordt gerealiseerd tijdens een volgende iteratie nauwkeuriger worden vastgesteld. Deze precisie is fundamenteel voor het vertrouwen in het project van alle betrokken partijen, en dus ook fundamenteel voor het welslagen van het project.

> Het valt op dat projecten vaak de requirements van een systeem tot op zeer laag detailniveau uitwerken

Itereer daarom kort. De meeste agile methodieken hanteren iteraties van twee tot zes weken. De voortgang wordt nu regelmatig herijkt en er is voortdurend feedback. In tegenstelling tot projecten die iteraties van zes maanden of meer hanteren. Daar kan de voortgang slechts eenmaal of tweemaal per jaar kan worden herijkt. Feedback is nagenoeg uitgesloten. Dit zijn geen iteraties maar deelprojecten.

Tenslotte Het voorspellen van de hoeveelheid werk in projecten is moeilijk. Schattingen zijn het hardst nodig in de eerste fasen van een project, terwijl de meeste beschikbare technieken kennis gebruiken die pas in een latere fase beschikbaar is. Daarom zijn noch functiepuntanalyse, noch het tellen van classes en entiteiten afdoende voor deze schattingen. Use cases worden al in een vroeg stadium van een project geproduceerd. Ze modelleren requirements, en houden technische requirements onder de motorkap. Use cases kunnen daarom ook vroeg in een project worden gebruikt voor het inschatten van het werk, op basis van het vermenigvuldigen van de complexiteit van een systeem en de velocity van het team. De complexiteit van use cases is te vinden door ze te mappen op de gehanteerde (meerlagen)architectuur. Hoewel dit een technische aangelegenheid lijkt, is deze methode toe te passen zonder veel kennis van het toekomstige systeem. De velocity wordt eenmalig vastgesteld en voortdurend verbeterd. Dit vereist voortdurend meten in projecten, bij voorkeur in korte iteraties en met het hanteren van een consistente modelleerstijl.

Sander Hoogendoorn, Partner Ordina.