Methodology

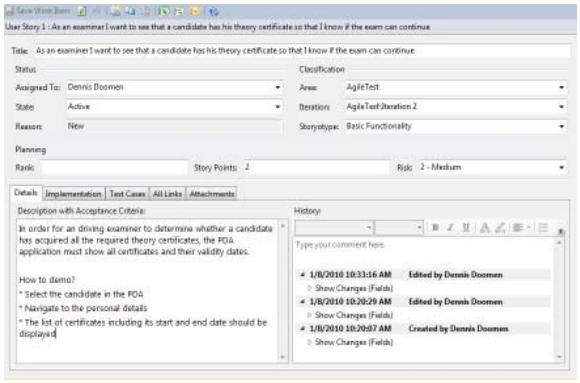
Met de introductie van Team Foundation Server 2010 haakt Microsoft ook actief aan op de best practices van Scrum en Extreme Programming. Het VSTS for Agile process template bevat bijvoorbeeld het nieuwe User Story work item type en levert bovendien rapportages en Excel workbooks die je hele ontwikkelproces kunnen ondersteunen. Reden genoeg om het user story concept nog maar eens uit te leggen.

User Stories zijn een verhaal apart

Wegwijs in de wereld van deze methodiek

Eén van de essentiële verschillen tussen stories en andere methodieken is de manier waarop grote brokken functionaliteit worden opgedeeld in kleine hapklare en schatbare blokjes. Stel je bijvoorbeeld de wens voor om in een E-Commerce website het concept product te introduceren. In veel projecten zou het realiseren daarvan worden opgedeeld in een aantal taken of use cases. Zo zou er voor de beheerder een scherm voor het opvoeren van producten moeten komen en voor de potentiële koper een scherm om naar producten te kunnen zoeken.
Elk project prioriteert zijn werk, en voor een use case georiënteerd project zou dat kunnen betekenen dat de mogelijkheid van zoeken naar producten pas veel later wordt gebouwd dan het opvoeren daarvan. Het scherm voor het opvoeren wordt uiteraard helemaal afgemaakt, inclusief alle bells and whistles die uit de review met de klant komen. Het grote nadeel hiervan is dat de gewenste functionaliteit dus eigenlijk pas af is als alle betrokken schermen zijn afgerond. Zeker bij een Agile project is het mogelijk dat door wijzigingen in de prioriteit of de scope deze functionaliteit nooit echt afgerond wordt.

User stories proberen dit op te vangen door te stellen dat deze functionaliteit pas business value heeft als het hele proces van opvoeren en zoeken werkt. Uiteraard loop je daarmee het gevaar dat dit alsnog een grote brok werk wordt. Daarom moet je al bij het opstellen expliciet onderscheid maken tussen de kern van de functionaliteit zoals de business die ziet en alle extra's die het gebruik ervan makkelijker of gebruiksvriendelijker maken. Zo zou de eerste story zich kunnen beperken tot een simpel invoerscherm en een bijbehorend scherm met een overzicht van producten waar de bezoeker uit kan selecteren. De layout laten conformeren aan de huisstijl, een mogelijkheid om producten in categorieën op te delen, en validatie van velden zouden allemaal als aparte stories kunnen worden gerealiseerd. En ja ik weet het, helemaal onafhankelijk zijn die extra stories natuurlijk niet, maar het dwingt je wel om je te focussen op de kern van de functionaliteit. Want na het realiseren van die eerste story krijgt de gebruiker al een idee van het systeem in wording en is hij beter in staat om feedback te geven. Sterker nog, in veel projecten is zo'n eerste story de katalysator voor nieuwe ideeën en inzichten.



Figuur 1: Een user story in Visual Studio 2010.

Is het ontwerp dan functioneel?

Ook dat niet. Uit een onderzoek van de Standish

Dennis Doomen
is principal consultant bij Aviva Solutions

Group waarbij meer dan 40.000 projecten zijn bekeken, kwam een aantal verklaringen boven water waarom zo veel projecten nog falen. Eén was het feit dat we onterecht aannemen dat het mogelijk is om een functionele eis ondubbelzinnig op te schrijven. En de andere was de gevaarlijke aanname dat de business al vanaf het begin precies weet wat ze wil. Vandaar dat Agile methodieken werken met korte cycli en dat de intensieve betrokkenheid van de business zo belangrijk is.
Dat is precies de reden waarom een user story moet worden gezien als een herinnering om, op het moment dat deze gerealiseerd gaat worden, opnieuw met de klantverantwoordelijke in overleg te gaan. Pas dan kun je optimaal gebruik maken van het voortschrijdend inzicht dat tijdens het lopende project is opgebouwd. En pas dan heeft de product owner het systeem in aanbouw in levenden lijve gezien, en je weet ongetwijfeld wel dat dit voor een hele hoop nieuwe wensen en eisen kan zorgen.

Requirements vangen

Maar omdat stories eindig zijn kun je ze ook niet als systeemdocumentatie zien. Met andere woorden, zolang het project loopt worden ze gebruikt als centrale plaats voor het vangen van requirements, discussies, schattingen en voortgang. Maar zodra de betreffende functionaliteit is gerealiseerd, hebben de stories onvoldoende detail om ze als referentie te gebruiken. Bovendien worden de stories niet bijgewerkt als er tijdens een demo met de klant verbeteringen en aanvullingen worden doorgevoerd.
Bovendien merkten wij tijdens het uitvoeren van het eerste project dat we nog wat structuur misten. Allereerst had het team behoefte aan een centrale plek waar de concepten en bedrijfsregels uit het business domain konden worden bijgehouden. Omdat ik zelf veel ervaring had met domeinmodellen zoals Martin Fowler dat bedacht heeft, en het laatste jaar meer en meer ben gaan doen met Domain Driven Design (een concept van Eric Evans), lag het voor de hand om hier een domeinmodel voor te gebruiken.

En ook de klantvertegenwoordiging had soms wat moeite om het overzicht te houden over de vele stories die op onze lijst van requirements stonden. Na wat gegoogled te hebben, liep ik tegen een hele set van posts en discussies aan geleid door Alistair Cockburn. Hij had hetzelfde probleem en had geëxperimenteerd met het toepassen van use case diagrammen om de context van het systeem te illustreren. Ik gebruik de use cases op zo'n diagram om schermen, externe koppelingen en processen te representeren. Dit blijkt erg goed te helpen als je met je klant in gesprek bent over een nieuwe feature. En ook al kun je een volledig functioneel ontwerp in de use case beschrijvingen kwijt, bij ons blijft het bij simpele diagrammen. Alle functionele aspecten blijven onderdeel van de stories.

Wie? Wat? Waarom?

Een ander belangrijk aspect van user stories is het feit dat het bij de meeste requirements wel duidelijk is wat de specifieke wens of eis is en welke soort gebruiker een rol speelt, maar de reden of aanleiding daarvan ontbreekt. User stories worden daarom in principe altijd geschreven in de vorm "Als wie, wil ik wat, zodat ik waarom". De wie beschrijft de stakeholder of rol die belang heeft bij de eis of wens beschreven door de wat. De waarom moet aangeven waarom die eis of wens zo belangrijk is voor het product. Een goed voorbeeld van het nut van dat laatste is dat het ontwikkelaars de mogelijkheid geeft om op een beter alternatief voor een eerder door de klant bedachte oplossing te komen. Pas wel op dat je tijdens het verzamelen van user stories niet uitkomt op een reden die een herhaling is van de wens of eis. De truc is daarbij om door te blijven vragen totdat de echte reden boven water komt.

Wie schrijft ze dan?

Dat hangt er vanaf. In Scrum bijvoorbeeld, is het de product owner rol (de term die ik in de rest van dit artikel gebruik) die de klant(en) representeert en de verantwoordelijkheid heeft over de volledige lijst van user stories, ook wel de product backlog genoemd. Buiten het feit dat de product owner de prioriteiten bepaalt binnen een project, is hij in principe ook degene die de user stories schrijft. Maar in de praktijk zal dat vaak in samenwerking gaan met het team dat de functionaliteit moet gaan realiseren. Het schrijven van goede stories is namelijk niet triviaal.

Welke eisen?

Zo moeten ze idealiter aan een aantal eisen voldoen, gevangen in het Engelse acroniem INVEST. Allereerst moeten ze zo ***independent*** mogelijk zijn. Immers, hoe onafhankelijker ze van elkaar zijn hoe makkelijker het is om de prioriteiten tussen stories te wijzigen. Dat dat soms lastig voor elkaar te krijgen is staat buiten kijf, maar soms kun je dit oplossen door de story te splitsen of twee stories juist te combineren.

De tweede eis is dat een story ***negotiable*** moet zijn, wat zoiets betekent als dat de beschrijving van de story voldoende ruimte moet bieden om tijdens de bouw discussie mogelijk te maken. Zoals ik al eerder schreef, stories zijn geen vervanging van het functioneel ontwerp, maar juist een herinnering om tijdens de bouw samen met de product owner de details uit werken. Schrijf je stories die functioneel helemaal dichtgetimmerd lijken te zijn, dan zou je daarmee de impressie kunnen wekken dat er geen discussie meer nodig is, terwijl dat nou juist de essentie is van user stories.

Bij de meeste requirements ontbreekt de reden of de aanleiding.

Storyotype	Beschrijving
New Functionality	Een brok nieuwe basisfunctionaliteit die (redelijk) onafhankelijk is van andere user stories.
Variation	Een aanvulling of uitbreiding van een eerdere user story, zoals bv. extra zoekvelden of extra kolommen.
New Business Rule	Aanvullende (complexe) bedrijfsregels of controles.
User Interface Enhancement	Het verfraaien van de look-and-feel zoals het toepassen van kleuren of een bepaalde layout of het verbeteren van de gebruiksvriendelijkheid.

Tabel 1: Storyotypes volgens Gerard Meszaro.

De derde eis, ***value***, geeft aan dat de user story waarde moet hebben gezien vanuit de business. Een voorbeeld hiervan is wederom de eerder genoemde productcatalogus. Alleen het scherm voor het opvoeren van producten heeft geen business value. Immers, zolang een potentiele klant die producten nog niet kan bekijken, heb je daar niets aan. Beide schermen volledig realiseren is ook geen optie, omdat dat juist weer heel erg lang kan duren. Maar voor de product owner zou het voldoende kunnen zijn als de klant in een eerste versie alleen nog een beschrijving en een prijs kan zien.
De mogelijkheid om er een foto bij te plaatsen zou wellicht in de volgende versie gebouwd kunnen worden en is dus een aparte user story. Helaas is deze manier van opsplitsen niet altijd triviaal, dus is het zaak om samen met de product owner te bekijken wanneer een story nog business value heeft.

De vierde eis stelt dat elke story ***estimable*** oftewel schatbaar moet zijn. Dat klinkt misschien flauw, maar zodra je merkt dat het moeilijk is om de omvang van een user story te schatten, dan is het zeer waarschijnlijk dat de scope van die story te breed is. Overigens mogen de schattingen voor stories die pas in een later stadium van het project worden opgepakt nog best grof zijn. Elke poging om hier een gedetailleerde schatting aan te hangen creëert de illusie dat alle belangrijke aspecten al bekend waren ten tijde van de schatting. Het is veel belangrijker om later met alle kennis en ervaring die dan beschikbaar is een realistischer schatting te maken.

Als je de omvang van de user story niet kunt inschatten, is de scope te breed.

Een maandje of zo?

Een belangrijke middel of te zorgen dat user stories schatbaar zijn is door ze klein te houden, oftewel ***small***. Een mooi voorbeeld dat ik ergens in een blog gelezen heb is om iemand in het team te vragen een schatting te geven van de hoeveelheid werk voor een redelijk grote story.
Als hij of zij deze vraag beantwoord met: “Ehm, iets van een maandje of zo” dan kun je er vergif op innemen dat de ontwikkelaar nog helemaal geen beeld heeft van de inhoud en de scope van die story. In zo’n geval is het niet onverstandig om te proberen deze story op te splitsen in een aantal kleinere stories met business value en een beperkte scope. En daarmee zijn we aangekomen bij de laatste letter van het acroniem, de ‘t’ van ***testable***. En dit heeft niets met geautomatiseerde tests of iets dergelijks te maken, maar juist met het feit dat een user story voorzien moet zijn van een aantal meetbare criteria waarmee ondubbelzinnig kan worden bepaald of de interpretatie van het team overeenkomt met de story zoals die door de product owner bedoeld was. Omdat de product owner het resultaat natuurlijk nog niet gezien heeft, krijg je dit niet altijd even makkelijk boven water. Maar een hulpmiddel dat kan helpen is om te vragen hoe hij de gewenste functionaliteit zou demonstreren aan andere gebruikers. Dit wordt ook wel de how-to-demo van een story genoemd.

Hoe beperk je de scope?

Uit eigen ervaring weet ik dat het bepalen van de juiste scope erg lastig is, zeker in het bijzijn van de product owner of een andere vertegenwoordiger van de business. Een bijzonder handig hulpmiddeltje om deze zo scherp mogelijk te definiëren is door gebruik te maken van zogenaamde storyotypes. Het idee hierachter is dat een goede story altijd voldoet aan exact één storyotype. Lukt dat niet, dan is de story te groot of te klein en is het zaak om ze te combineren of juist op te splitsen.
Het concept van stereotypes voor user stories is voor het eerst door Gerard Meszaro beschreven in zijn artikel Using Storyotypes to Split Bloated XP Stories. Daarin onderkend hij de vier storyotypes uit tabel 1.

Stel je nu eens een scherm voor waarin je kunt zoeken op producten uit een catalogus. In een aanpak gebaseerd op fish level use cases wordt het gehele scherm, inclusief alle bedrijfsregels, mogelijke zoekvelden en de volledige look-and-feel van de resultaten, meestal als één use case gezien. Maar wil je dit scherm opdelen in een aantal kleinere use cases die nog steeds voldoen aan INVEST, dan kunnen deze vier storytypes hierbij helpen.

Zo zou je als eerste story het zoeken kunnen beperken op alleen de naam van een product en presenteer je het resultaat in de standaard layout. Dat biedt nog steeds business value en is dus in feite New Functionality. Alle aanvullende zoekmogelijkheden kun je later toevoegen als één of meerdere Variations en het opleuken van de look-and-feel als UI Enhancement. Kom je door tijdnood niet meer toe aan die extra mogelijkheden,

Figuur 2: Het storyotype veld bij een user story.

dan is je basisfunctionaliteit nog steeds zinvol. Als je al in Visual Studio 2010 naar het User Story work item hebt gekeken dan zal het je wellicht zijn opgevallen dat deze geen veld voor een storyotype kent. Maar niet getreurd, het Agile process template is vrij makkelijk aan te passen. En als de TFS 2010 Power Toys beschikbaar komen wordt dat helemaal een makkie. In de tussentijd kun je een aangepast Work Item Template (een WIT) via mijn blog downloaden (zie de links).

Wat is het waard?

In de praktijk hebben de storytypes mij enorm geholpen in de discussie met de product owner, maar ik merkte wel dat er een aantal soorten activiteiten zijn die niet helemaal passen op deze vier storyotypes en de eisen uit het INVEST acroniem. Vaak zijn er eisen en wensen in het project die niet van de product owner komen, maar die wel belangrijk zijn om het betreffende systeem in productie te kunnen zetten. Zo zou de IT-afdeling bepaalde eisen kunnen stellen aan het systeem om uitgerold te kunnen worden op hun infrastructuur. Of misschien willen ze de mogelijkheid hebben om realtime logging op te kunnen vragen. Als architect wil je dat het systeem op een bepaalde manier gebouwd wordt, zodat de onderhoudbaarheid gegarandeerd is, en als teamleider wil je mogelijk een automatische nachtelijke build inrichten om zo de kwaliteit in de gaten te kunnen houden.

Je zou dit soort taken kunnen oppakken door ze als onderdeel van een normale user story op te pakken. Maar je zou ook een vast percentage van je capaciteit kunnen reserveren. Ik ben van mening dat het beter is alle niet-functionele zaken expliciet zichtbaar te maken in je project. Dat helpt ook in de discussie met de product owner als je hem moet uitleggen waarom je maar een beperkt aantal van zijn user stories gaat oppakken. Om die reden moeten user stories in mijn project project value hebben, en niet per se business value.

Waar begin je?

Stel nou dat je na veel overleg een lijst van user stories hebt en je begint met het realiseren van een nieuw systeem. De eerste story zal dan buiten de gewenste functionaliteit ook het bouwen van het skelet van je architectuur omvatten. Hoe voorkom je dan dat je daarmee een onacceptabele hoeveelheid van je tijd bezig bent? Het artikel Managing the Bootstrap Story van Jennitta Andrea behandelt deze uitdaging in meer detail en biedt een aantal alternatieve oplossingen.

Eén van die oplossingen is om in overleg met de product owner een story te definiëren die weliswaar minimale functionaliteit biedt, maar toch project value heeft. Zo'n story wordt vaak de backbone story genoemd omdat je daarmee de ruggengraat van je systeem realiseerd. Een vaak voorkomend compromis is om de backbone story te gebruiken als proof-of-concept van de gekozen architectuur. Omdat hij daarmee het vertrouwen verkrijgt dat het team in staat is om een dergelijk product te bouwen, zou dat voor de product owner al voldoende project value kunnen bieden.

Storyotype	Beschrijving
Compound Epic	Een samengestelde user story die dient om een aantal stories logisch te groeperen.
Complex Epic	Een user story waarvan de inhoud en impact pas later in het project bepaald kan worden, maar waarvan wel bekend is dat de hoeveelheid werk omvangrijk is.
Setup	Een story die dient om de projectomgeving in te richten. Bv. een source control omgeving, een project website, een build omgeving.
Technical	Een user story die als doel heeft om een technische aanpassing of verbetering door te voeren. Bv. voldoen aan een coding standard, het refactoren van een slecht ontwerp, het uitvoeren van een performance test.
Documentation	Een story voor het schrijven van bv. een gebruikers handleiding, de installatiehandleiding, etc.
Training	Het geven van trainingen aan bv. functioneel beheerders, of het organiseren van workshops voor eindgebruikers.
Quality Improvements	Een story die als doel heeft om een aantal gerelateerde bugs op te lossen.
Spike	Een story met als doel een technisch onderzoek te doen naar bv. de toepassing van een bepaald stuk technologie, of het proberen van een alternatieve technische oplossing.

Tabel 2: Nuttige storyotypes volgens Dan Rawsthorne.

Nog meer storyotypes?

De bovengenoemde backbone story is natuurlijk ook gewoon een storyotype. Sterker nog, nadat ik op zoek ging naar oplossingen voor het vastleggen van de niet-functionele eisen, liep ik tegen een presentatie aan met een hele set van additionele storyotypes. Dan Rawsthorne, de auteur, heeft gepoogd om voor virtueel elke mogelijk activiteit een storyotype te definiëren. Ik vind zelf dat hij daarbij wat ver gegaan is, maar een kleine set van additionele storyotypes blijkt in de praktijk toch wel nuttig te zijn.

Als de ontwikkelaar niet kan inschatten hoeveel werk hij aan een user story heeft, kun je er gif op innemen dat de user story te groot is.

Wanneer is het verhaal af?

Maar hoe weet je nu of een user story succesvol is afgerond? Nou, als het goed is conformeren alle stories aan INVEST en zijn ze voorzien van een aantal criteria (de how- to-demo) opgesteld door de product owner. Daarmee is het feit of een gerealiseerde story functioneel correct is al bepaald. Wat je dan nog mist is een aantal afspraken, zodat alle stakeholders, inclusief de product owner, weten wanneer het team vindt dat de story af is. Wat dat precies inhoud kan per team wisselen, maar meestal worden verscheidene van de onderstaande criteria aangehouden.

- De code compileert en er zijn geen warnings of errors;
- De code voldoet aan de coding standards van het project of organisatie;
- De code is gereviewed;
- Alle unit- en integratietesten zijn geslaagd;
- De Code Analysis van Visual Studio geeft geen waarschuwingen;
- ReSharper geeft geen meldingen over potentiële fouten (alles is 'groen');
- De dagelijkse integratiebuild heeft succesvol gedraaid;
- De functionaliteit is getest door een ander lid van de team dan de ontwikkelaar;
- De oplossing is gecontroleerd aan de hand van een interne checklist;
- De functionaliteit is getest door een systeemtester;
- De look-and-feel is gecontroleerd door een medewerker van de afdeling communicatie.

Het totaal van die afspraken en de how-to-demo van de story wordt de definition-of-done genoemd.

En de planning dan?

Stories zijn ook uitstekend geschikt om te gebruiken als eenheid in je planning. Puristisch gezien hebben Agile projecten geen lange termijn planning en wordt de functionaliteit iteratief gerealiseerd waarbij de prioriteit continu kan worden bijgesteld. Maar in de realiteit ontkom je er vaak niet aan om een planning te maken. Hoe los je dat dan op?

Wat ik vaak doe is alle stakeholders in aantal workshops te krijgen en met behulp van de use case modellen als context alle stories op papier te krijgen. Daarbij moet je waken dat je niet te veel in de detail treedt. Dat zou alle betrokkenen een vals gevoel van precisie kunnen geven waardoor ze de stories toch als functioneel ontwerp gaan zien. Zijn er brokken functionaliteit waarvan niemand nog precies weet hoe dat zou moeten gaan werken, maak er dan een epic story van en introduceer een spike om later in het project tijd te reserveren om die epic verder uit te werken.

Organiseer daarna een aantal kortere meetings met het team, of, als dat er nog niet is, met een aantal ervaren ontwikkelaars. Laat hier alle stories de revue passeren en probeer de omvang te schatten in zogenaamde story points. Sommige mensen uit de Agile community zeggen dat je hier alleen relatief moet schatten. Dus een story die gevoelsmatig twee keer zo veel werk is als een andere story moet ook dubbel zoveel story points krijgen. Ik werk zelf met de insteek dat elke story point overeenkomt met de ideale dag van een ervaren senior software ontwikkelaar. Met andere woorden, één story point betekent dat een ervaren ontwikkelaar die bekend is met de gekozen architectuur, aanpak en technologie 8 uur lang moet werken, zonder gestoord te worden door telefoon, email en andere afleidingen. Mike Cohn, schrijver van het boek User Stories Applied, heeft hier een groot aantal hoofdstukken aan gewijd. Idealiter is elke story tussen de 1 en 8 story points, maar vooral aan het begin van het projecten kunnen er nog epics zijn die nog niet op te splitsen vallen.
Na die meetings heb je een schatting van de totale omvang van het systeem. Om nu tot een aantal uren te komen moet je een schatting maken van de productiviteit van het verwachte team ten opzichte van die ideale senior ontwikkelaar. Mike Cohn doet dit door een tabel te maken met de verwachte rollen, hun inzet, en hun productiviteit in een percentage af te zetten. Het gemiddelde percentage van het team vermenigvuldig je met het aantal story points om op het verwachte aantal manuren uit te komen.
Het resultaat van dit alles is dus een planning om een vast aantal story points te realiseren. Natuurlijk is het maar een schatting, want zowel de productiviteit kan tegenvallen als de schatting in story points kunnen afwijken, maar het geeft je wel een initiële schatting die gebruikt kan worden voor een globale planning en budgetbesprekingen. Uiteraard is het van belang om te zorgen dat je de daadwerkelijke productiviteit continu blijft meten.

Tjonge, en nu?

Het moge duidelijk zijn dat een user story geen onafhankelijk concept is, maar nauw samenhangt met heel veel van de facetten van ons werk in de software industrie. In dit artikel heb ik een poging gedaan om een aantal van die facetten toe te lichten en de relatie ertussen te verduidelijken. Maar ondanks dat ik niet heel erg de diepte in kan gaan hoop ik dat ik je zo enthousiast heb gemaakt dat je staat te springen om meer over de wondere wereld van de stories te weten te komen. «

Voor vragen of opmerkingen is Dennis Doomen bereikbaar op dennis.doomen@avivasolutions.nl of op Twitter ID ddoomen.
Dennis blogt op www.dennisdoomen.net

Links

- De blogs van Dennis Doomen en Aviva Solutions http://www.dennisdoomen.net en http://blog.avivasolutions.nl
- Managing The Bootstrap Story http://www.agilealliance.org/system/article/file/886/file.pdf
- Using Storyotypes to Split Bloated XP Stories http://www.springerlink.com/content/h53td341yagcc80w/
- Storyotypes for Agile Development http://www.sdex-po.ru/data/speakers_2007/dan_rawsthorne_storyotypes.pdf
- User Stories Applied for Agile Software Development http://www.amazon.com/User-Stories-Applied-Software-Development/dp/0321205685
- Domain Driven Design Quickly http://www.infoq.com/mini-books/domain-driven-design-quickly