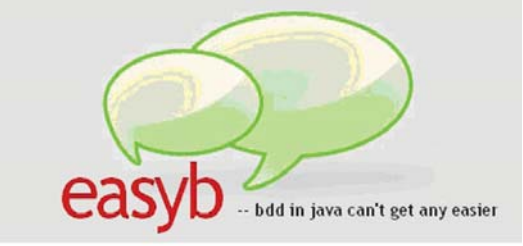
Enterprise

Behavior Driven Development (BDD) is geen nieuw framework of een compleet nieuwe methodiek voor software development. Het is een collectie van bestaande technieken die gericht is op het opstellen van duidelijke testcases die zowel door developers als gebruikers wordt begrepen. Deze bestaande technieken zijn voornamelijk afkomstig van Test Driven Development (TDD) en domain driven design/development.

Behavior Driven development

Beter dan Test Driven Development



BDD is Test Driven Development maar dan beter uitgevoerd. Om dat beter te begrijpen moeten we eerst inzichtelijk maken waar TDD zich op richt. TDD houdt zich bezig met het unit testen van een systeem. Een unit is in dit geval een onderdeel van een systeem, zoals class of een methode van een class, en deze methode of class wordt vervolgens getest.

Met als gevolg dat de structuur en denkwijze van de testcases gelijk lopen aan de structuur en denkwijze van de classes of methodes. Maar een test zou zich niet moeten richten op een technisch aspect (zoals een class of een methode, of hoe de classes en methodes elkaar aanroepen), maar op het gedrag van het systeem.

In plaats van testen te schrijven die controleren wat de code doet, moeten de tests beschrijven of specificeren wat het systeem moet doen. Het gedrag van het systeem in bepaalde situaties moet worden getest.

Dit komt ook dichter bij een complete TDD-aanpak waar begonnen wordt met het schrijven van de testcase en daarna pas de code. Het schrijven van een testcase is een stuk makkelijker als er niet van tevoren hoeft te worden vastgelegd welke class of methode iets gaat uitvoeren, maar als je het gedrag dat je verwacht beschrijft in een testcase. Vervolgens kun je aan de hand van dit gedrag de classes en methodes invullen.

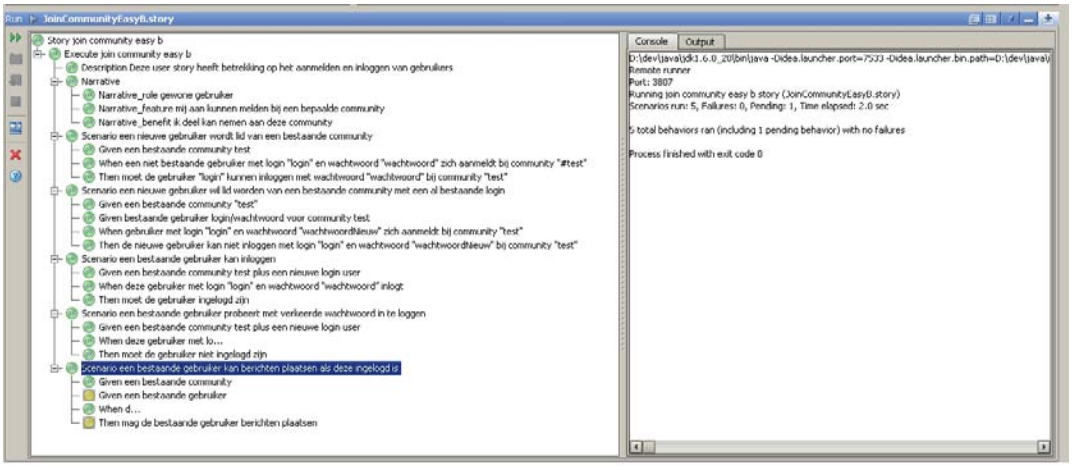
Gemeenschappelijke taal

Om het gedrag van het systeem te kunnen beschrijven is het belangrijk dat de gebruikers van het systeem erg betrokken zijn en helpen bij het beschrijven van de specificaties. De meeste gebruikers zijn hier niet ervaren in. Dit levert vaak onduidelijke en onvolledige requirements op en dus later in het traject problemen.

Om te zorgen dat de specificaties zoveel mogelijk aansluiten bij de ideeën van de gebruikers, dienen alle betrokkenen dezelfde woorden te gebruiken. Voor elke betrokkene dient er een duidelijke afbakening te zijn wat de gebruikers nu precies bedoelen met hun woorden. Deze ideeën van de gebruikers worden vervolgens als een gemeenschappelijke taal gehanteerd die door het hele team gebruikt wordt. Als een gebruiker het vervolgens heeft over een bepaald concept (zoals bijvoorbeeld een debiteur), dan dient iedereen te weten wat er met een debiteur wordt bedoeld.

De gemeenschappelijke taal (een concept afkomstig uit hoek van domain driven development) wordt vervolgens gebruikt om voorbeeldspecificaties te beschrijven die door de eindgebruikers worden

Ronald Haring
is Senior Ontwikkelaar bij iProfs.



begrepen. Deze specificaties worden op een dusdanige manier vastgelegd dat de eindgebruikers deze ook makkelijk kunnen lezen. Dus niet verstopt in de programmatuur, maar bijvoorbeeld, in een los bestand waarin in de gemeenschappelijke taal staat beschreven. De gemeenschappelijke taal kan opgesteld zijn in gewone tekstbestanden die later gekoppeld worden aan een programmeertaal (zoals JBehave met Java) of in een DSL die gemaakt is voor het opstellen van de testcases in de gemeenschappelijke taal (zoals EasyB in groovy).

User stories

Een testcase voor BDD komt over het algemeen overeen met het opstellen van een user story. Dit kan gebruikt worden om globale specificaties op te schrijven zoals:

- Wanneer een persoon een bepaalde rol heeft;
- Wil ik dat het systeem het volgende doet;
- Zodat er een bepaalde opbrengst is.

Indien een user story op deze manier wordt opgesteld, wordt meteen duidelijk wat de meerwaarde is van het systeem voor bepaalde personen.

Om vervolgens vast te stellen wat het systeem doet binnen specifieke situaties, dienen de acceptatiecriteria te worden getest. Deze acceptatiecriteria (of specificaties van het systeem) kunnen dan als volgt opgesteld worden:

- Gegeven een bepaalde situatie van het systeem;
- Als ik dan bepaalde acties uitvoer;
- Dan zou het volgende zich voor moeten doen.

Een enkele specifieke situatie van een user story wordt een scenario genoemd. Een user story kan opgebouwd zijn uit een aantal scenario's die allemaal een ander aspect van de user story belichten.

Indien de testcases op deze manier worden opgesteld, kunnen de gebruikers van het systeem beter aangeven wat het systeem moet doen in die situaties. De gebruikers helpen dan actief mee om hun systeem te bouwen aangezien ze zelf kunnen lezen wat het systeem doet.

Een alternatieve manier voor het opstellen van gedrag is door gebruik te maken van specificaties. Een specificatie staat wat meer los van de user stories. Bij een user story wordt uitgegaan van een bepaald gedrag dat zich in een bepaalde situatie voor doet, terwijl bij een specificatie driven framework het gedrag wordt gespecificeerd en dit wordt getest. De specificaties staan iets dichter bij unit testen dan de user stories. Het leest alleen wel wat makkelijker dan een unit test.

Een voorbeeld van een specificatie kan zijn:

Voor een nieuw aangemaakte gebruiker op de site geldt:

- dat de gebruiker kan inloggen;
- dat de gebruiker berichten kan posten in het forum;
- dat de gebruiker zijn wachtwoord kan aanpassen.

Een scenario voor een user story hiervoor kan zijn:

- Gegeven het feit dat een gebruiker zich geregistreerd heeft op de site;
- Wanneer de gebruiker is ingelogd;
- Dan kan de gebruiker berichten plaatsen in het forum;
- En kan de gebruiker zijn wachtwoord aanpassen.

Het verschil zit er voornamelijk in dat bij specificatietesten er wordt uitgegaan van een bepaalde situatie die verder uitvoerig wordt getest, terwijl de user story wat meer beschrijft hoe die situatie optreedt. Dit soort user stories worden ook wel 'conversatie unit test' genoemd, doordat het geheel goed te lezen is en afgeleid kan worden uit een conversatie met de eindgebruiker.

Een testcase voor BDD komt overeen met het opstellen van een user story.

Om te zorgen dat de specificaties zoveel mogelijk aansluiten bij de ideeën van de gebruikers, moeten alle betrokkenen dezelfde woorden gaan gebruiken.

Nadeel van EasyB is dat het bij een falende testcase gigantisch veel uitvoer geeft.

BDD Voorbeeld

Voor het opstellen van een voorbeeldsituatie ben ik uitgegaan van een aantal simpele user stories van een website waarop mensen zich kunnen aanmelden bij verschillende communities en waarop zij berichten kunnen plaatsen in een forum. De administratieve kanten van een dergelijke site komen in deze user stories niet te sprake. Dit is onderdeel van een aantal andere user stories die hier niet bij betrokken worden.

User story met scenarios

Als een gewone gebruiker

- wil ik mij aan kunnen melden bij een bepaalde community met een username en wachtwoord;
- zodat ik kan inloggen en deel kan nemen aan deze community.

Scenarios

Scenario: een nieuwe gebruiker wordt lid van een bestaande community

- Gegeven een bestaande community 'test'
- Als een niet bestaande gebruiker met login 'login' en wachtwoord 'wachtwoord' zich registreert
- Dan moet de nieuwe gebruiker login kunnen inloggen met wachtwoord 'wachtwoord'

Scenario: een nieuwe gebruiker wil lid worden van een bestaande community met een al bestaande login

- Gegeven een bestaande community 'test'
- Als een gebruiker met login 'login' en wachtwoord 'wachtwoordNieuw' zich aanmeldt
- En er bestaat al een gebruiker met login 'login'
- Dan moet de nieuwe gebruiker een melding krijgen
- En de nieuwe gebruiker kan niet inloggen met login 'login' en wachtwoord 'wachtwoordNieuw'

Scenario: een bestaande gebruiker kan inloggen

- Gegeven een bestaande community test;
- En een bestaande gebruiker met login 'login' en wachtwoord 'wachtwoord';
- Indien deze gebruiker met login "login" en wachtwoord "wachtwoord" inlogt;
- Dan moet de gebruiker ingelogd zijn.

Scenario: een bestaande gebruiker probeert met verkeerde wachtwoord in te loggen

- Gegeven een bestaande community test
- En een bestaande gebruiker met login 'login' en wachtwoord 'wachtwoord'
- Indien deze gebruiker met login 'login' en wachtwoord 'wachtwoordTypo' inlogt
- Dan moet de gebruiker niet ingelogd zijn

Frameworks

Er zijn veel frameworks beschikbaar voor BDD. Voor Java zijn de bekendste JBehave en JDave. JBehave richt zich op de user stories en uitwerkingen hiervan, terwijl JDave zich richt op het uitwerken van de specificaties. EasyB is een Groovy-framework dat beide vormen van BDD ondersteunt.

EasyB

EasyB is een BDD-framework geschreven als een domain specific language in Groovy. Bij EasyB wordt de user story in dezelfde file opgeschreven als de uit te voeren testcase, maar er hoeft nog geen code in te staan waardoor het initieel toch ook goed leesbaar is voor een gewone gebruiker. Een voorbeeld hiervoor is:

```
scenario 'een nieuwe gebruiker meldt zich aan
bij een bestaande community'
{
given "een bestaande community"
when "de nieuwe user zich aanmeldt"
then "dient de superuser een aanmelding
te krijgen"
}
```

Bij het uitvoeren van deze code wordt de uitvoer gemeld als pending. Dit betekent dat er nog geen test voor aanwezig is, maar op deze manier is al wel vastgelegd wat het gewenste gedrag moet zijn. Vervolgens kan de code toegevoegd worden, waarna de pending status verdwijnt en (hopelijk) de groene balk weer tevoorschijn komt om aan te geven dat deze test goed heeft gewerkt.

Aangezien de story wordt gezien als een Groovyscript, kan er binnen de code voor de testcase worden gewerkt met Groovy en dus ook met gewone Java-objecten. Een nadeel van EasyB is dat als een testcase faalt er een gigantische hoeveelheid uitvoer komt, voornamelijk gegenereerd door EasyB.

Het is dan lastig om te achterhalen wat er nu echt fout is gegaan. Het vergeten van een komma voor een accolade bijvoorbeeld, zorgt voor behoorlijke lange stracktraces zonder dat je goed kunt achterhalen wat er nu precies fout is gegaan. Ook is er standaard geen integratie met Spring en transacties. Dit dien je zelf in een eigen class te doen.

Voor elke story en/of scenario kunnen standaardmethodes worden uitgevoerd, vergelijkbaar met '@Before' en '@After' annotaties. Om goed met EasyB te werken, is wel de laatste versie van een goede IDE nodig, zodat code completion beschikbaar komt. Enige kennis van Groovy kan ook geen kwaad, alhoewel je ook prima zonder enige Groovysyntax te werk kunt gaan.

EasyB kan ook worden gebruikt in combinatie met populaire Mock-frameworks. Als je echter met Mockito werkt, dien je wel iedere keer dat je de Mockito 'when clause' wilt gebruiken deze op te schrijven met de fully qualified name, aangezien 'when' een keywoord is binnen EasyB.

Een voorbeeld zegt meer dan duizend verduidelijkingen:

```
description "Deze user story heeft betrekking op
het aanmelden en inloggen van gebruikers"

narrative "", {
 as_a "gewone gebruiker"
 i_want "mij aan kunnen melden bij een
bepaalde community"
 so_that "ik deel kan nemen aan deze community"
}

CommunityService communityService

before_each "begin met een lege set", {
CommunityService =
new CommunityServiceWithMap();
}

scenario "een nieuwe gebruiker wordt lid van
een bestaande community", {
    given "een bestaande community \"test\"",
{
  communityService.save(new Community("test"))
    }
  when "een niet bestaande gebruiker met
login \"login\" en wachtwoord \"wachtwoord\"
zich aanmeldt bij community \"test\"", {
        communityService.join("test", "login",
 "wachtwoord")
    }
    then "moet de gebruiker \"login\"
kunnen inloggen met wachtwoord \"wachtwoord\"
bij community \"test\"", {
  communityService.login("test", "login",
 "wachtwoord").shouldNotBe null
    }
}
```

Als een stuk code in een 'before_each' closure staat, dan zal deze code voor elk scenario worden uitgevoerd. Als een stuk code in een 'before' closure staat, dan zal deze code alleen aan het begin worden uitgevoerd.
EasyB kan ook parameters vervangen voor het runnen van de scripts. De variabelen die vervangen kunnen worden dienen gedefinieerd te worden in een 'before' of 'before_each' stuk en zijn dan beschikbaar in de methodes.
Voor meer voorbeelden verwijs ik graag naar de meegeleverde code waarin wat meer voorbeelden staan van hoe EasyB gebruikt kan worden. De complete sourecode is te downloaden via de website van iProfs.
Het opschrift van de EasyB website 'EasyB makes it easy' klopt wel wat mij betreft. Het opschrijven van verwacht gedrag van het systeem in de EasyB syntax leest voor zowel gebruikers als programmeurs gemakkelijk. «

EasyB kan parameters vervangen voor het runnen van scripts.

Referenties

- http://www.easyb.org site voor easyB
- http://behaviour-driven.org/ algemene site over behaviour driven development
- http://blog.dannorth.net/introducing-bdd een van de eerste uitvinders van bdd
- http://en.wikipedia.org/wiki/Behavior_Driven_Development wiki lemma over bdd
- http://jbehave.org/ java variant van BDD
- http://jdave.org/ nog een java variant van BDD maar dan alleen voor het specificatie deel