IMPLEMENTEREN VAN BEHAVIOUR DRIVEN DEVELOPMENT

Door Roy de Kleijn • roy.dekleijn@polteq.com • @TheWebTester

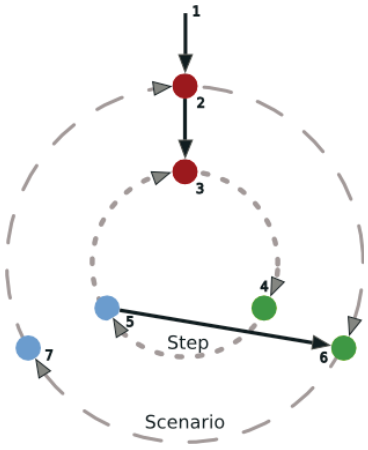
Een veel voorkomend probleem bij traditionele software ontwikkeling is dat de betrokken partijen geen gemeenschappelijke taal spreken en iedereen haar eigen jargon hanteert. Dit resulteert in verkeerd geïnterpreteerde requirements, verkeerd ontworpen, ontwikkelde en geteste producten die vervolgens – uiteraard – niet aan de verwachtingen voldoen. Een andere veel voorkomend fenomeen is dat functionaliteit en features worden ontwikkeld die niet (meteen) toegevoegde waarde leveren. Dit is zonde van de geïnvesteerde tijd en het geld. Hoe kan Behaviour Driven Development een oplossing zijn voor dit probleem? Ben je na het lezen van dit artikel echt nieuwsgierig geworden, dan kan je direct zelf aan de slag met een kant-en-klaar voorbeeld.

Behaviour Driven Development

Behaviour Driven Development is een Agile-softwareontwikkelaanpak, waarbij in korte iteraties een waarde-vermeerdering van het product wordt gerealiseerd en opgeleverd. Deze methode is gericht op het verbeteren van de communicatie tussen de betrokken partijen, zodat een gemeenschappelijk begrip ontstaat over wat gemaakt moet worden en aan welke eisen het opgeleverde product moet voldoen. Het is belangrijk dat de volgende betrokken partijen in dit proces goed samenwerken: business, ontwerpers, ontwikkelaars, testers en eventueel mensen van andere afdelingen. Een manier om tot een gemeenschappelijk begrip te komen is het organiseren van een interactieve sessie met alle betrokken partijen om gezamenlijk acceptatiecriteria te definiëren. Het is belangrijk dat alle betrokken partijen hun inbreng kunnen doen, zodat geen verrassingen ontstaan bij oplevering.

Test Driven Development maar dan anders...

Het proces van Behaviour Driven Development is vergelijkbaar met het proces van Test Driven Development, met als belangrijkste verschil dat BDD Acceptance Test Driven is en TDD Unit Test Driven is. Een kenmerk van Behaviour Driven Development is dat het een 'outside-in' aanpak is. Dat wil zeggen dat het product vanaf de buitenkant wordt ontworpen, ontwikkeld en getest. Dit in tegenstelling tot Test Driven Development ('inside-out' aanpak) waarbij het product van binnenuit wordt ontworpen, ontwikkeld en getest, door middel van unittests. Het slagen van de unittests vertelt dat het testobject op de goede manier is gemaakt, maar niet dat het juiste is gemaakt. De 'outside-in' aanpak maakt het geschikt om zowel nieuwbouw als legacy-systemen te testen, omdat je de applicatie vanaf de buitenkant benadert en je geen notie hoeft te hebben van de applicatiecode en structuur.



Op hoog niveau is het proces van Behaviour Driven Development als volgt te beschrijven. Eerst worden functionele (automatisch) uitvoerbare acceptatietests geschreven. Voor elk scenario wordt een aantal stappen doorlopen.

De binnenste cirkel stelt het implementeren van de stappen van het scenario voor, waarbij de volgende stadia worden doorlopen: falen van de stap, omdat er nog geen testobject is (**rood**); daarna wordt precies genoeg applicatiecode geschreven om de stap te laten slagen (**groen**) en het verbeteren van de geïmplementeerde code (**blauw**). ➔

Zodra alle stappen zijn geïmplementeerd val je weer terug in de buitenste cirkel waarbij de laatste stadia worden doorlopen: doordat alle stappen zijn geïmplementeerd slaagt het scenario (**groen**); en kan nogmaals worden gewerkt aan het verbeteren van de geïmplementeerde code, zodat beter onderhoudbare en robuuste, beheerbare code ontstaat (**blauw**). Dit proces wordt herhaald voor alle scenario's (acceptatiecriteria).
Met deze aanpak wordt het minimale (optimale) product gerealiseerd en opgeleverd dat voldoet aan de gestelde eisen. Het is verstandig de applicatiecode, ondanks de BDD aanpak, volgens de TDD aanpak te ontwikkelen en testen en dus unit tests te schrijven.

Stories / acceptatiecriteria

Bij het opstellen van de acceptatiecriteria blijkt onmiddellijk dat Behaviour Driven Development gericht is op de bewustwording van de waardevermeerdering die in het product wordt gerealiseerd.

Een Behaviour Driven Development story wordt beschreven met de volgende syntax:

```
In order to <receive benefit> as a <role>, I want <goal/desire>
```

Deze functionele beschrijving wordt door en/of namens vertegenwoordigers van de business geschreven. Kenmerkend is dat het beoogde doel als eerste wordt beschreven, omdat het hierom draait. Op basis van de stories worden de acceptatiecriteria, in de vorm van scenario's, door het team geschreven. Het is belangrijk dat de juiste betrokken partijen hierbij betrokken zijn, zodat alle eisen en wensen boven tafel komen en je niet vertrouwd op één persoon.

Scenario's hebben een eenduidige syntax:

```
Scenario: <titel van het scenario>
Given <pre-conditie>
When <actie>
And <eventuele vervolg actie>
Then <post-conditie>
```

Deze format heeft als voordeel dat het eenvoudig, voor alle betrokken partijen, te lezen en te begrijpen, is zodat er gericht over gediscussieerd kan worden om consensus te bereiken. Daarnaast maakt deze notatie het mogelijk om de scenario's eenvoudig door een tool te laten parsen. Waar nodig worden de scenario's voorzien van 'examples'. Dit zijn tabellen met testdata. Elke regel uit een tabel is één test en kan er als volgt uit zien:

```
Examples:
|username|password|
|testerA|passwordA|
|testerB|passwordB|
```

Uitvoerbare specificaties

Groot voordeel van Behaviour Driven Development is dat de scenario's, geschreven met behulp van de Given/When/Then stappen, aan uitvoerbare code worden gelinkt. Op deze wijze ontstaan uitvoerbare specificaties. De meeste frameworks om behaviour driven tests uit te voeren bieden de functionaliteit om de geschreven stories meteen uit te voeren, zodat een leeg skelet ontstaat en je alleen nog de daadwerkelijke testcode hoeft te maken.
➔

Living documentation

Bij iedere testuitvoer worden letterlijk de specificaties als uitgangspunt gebruikt. Iedere regel van de specificatie kleurt groen of rood naar gelang het resultaat. Op deze manier ontstaat 'Living documentation' en heb je op ieder moment de beschikking over de actuele staat van het testobject. In theorie zou het alle andere documentatie die de functionele beschrijving van een applicatie geeft kunnen vervangen.

Het beste resultaat behaal je als je Behaviour Driven Development toepast in combinatie met Continuous Integration, zodat bij iedere doorgevoerde wijziging onmiddellijk de actuele staat van het product (testobject) duidelijk wordt weergegeven. Zo weet je binnen afzienbare tijd welk effect de doorgevoerde veranderingen en toegevoegde functionaliteiten en features heeft op de rest van de producten. Het mooie is dat je met Behaviour Driven Development deze resultaten op een uniforme en begrijpelijke manier beschikbaar stelt voor de hele organisatie.

Voorbeeldproject

Op github (software versiebeheersysteem) heb ik een voorbeeldproject aangemaakt dat gebruikt kan worden om te beginnen met Behaviour Driven Development. Het voorbeeldproject kan worden gebruikt voor het testen van een website en ondersteunt parallelle testuitvoer en maakt screenshots wanneer iets niet volgens verwachting verloopt.

Vooraf te installeren

1. Eclipse IDE for Java Developers – www.eclipse.org/downloads/
   Dit is een ontwikkelomgeving waarin de tests gemaakt kunnen worden.
2. Maven - In eclipse: Help -> Eclipse Marketplace -> zoek: Maven Integration for Eclipse
   Maven maakt het mogelijk om project afhankelijkheden te configureren.
3. JBehave Plugin – volg de instructies op https://github.com/Arnauld/jbehave-eclipse-plugin
   De JBehave plugin maakt het eenvoudiger om stories te schrijven.

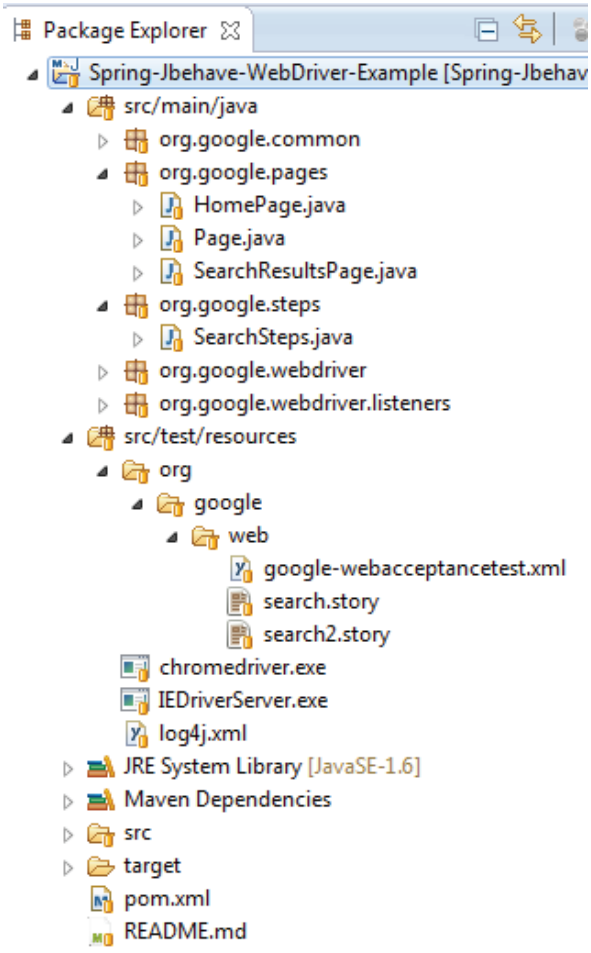
Het voorbeeldproject importeren

1. Download de broncode vanaf de volgende locatie: http://roydekleijn.github.io/Spring-Jbehave-WebDriver-Example/
2. Pak het ZIP-bestand uit naar een zelfgekozen locatie
3. (In Eclipse) Klik rechtermuisknop in het 'Package Explorer' venster en kies achtereenvolgens: Import -> Existing Maven Projects.
4. Kies als 'root Directory' de map waar de bestanden zijn opgeslagen.
5. Doorloop de wizard en klik op Finish. ➔

Structuur van het voorbeeldproject

Het project is verdeeld in drie belangrijke onderdelen:

- org.google.pages – Dit package bevat de abstractie (vertaling van het testobject naar code) van de applicatie die getest wordt.
- `org.google.steps` – Dit package bevat de mapping tussen de tekstuele stappen en uitvoerbare testcode.
- `org/google/web` – Deze map bevat de tekstuele story bestanden.

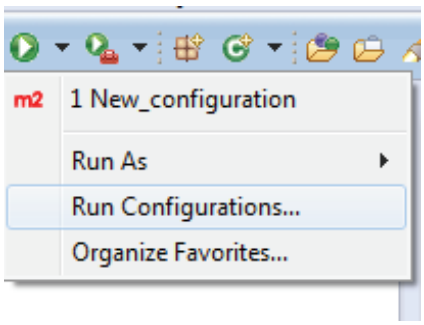


Uitvoeren van de specificaties

Het uitvoeren van de stories kan als volgt:

1. Navigeer naar Run -> Run Configurations
2. Klik rechtermuisknop op Maven Build en kies New
3. Selecteer het project door op Browse Workspace te klikken.
4. Vul bij Goals het volgende command in:

```
integration-test -Dgroup=google -Dbrowser=firefox
```



←