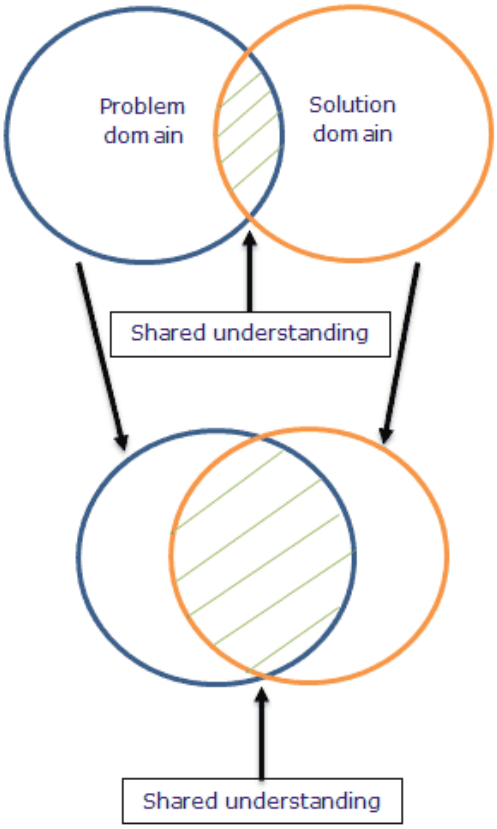
Behaviour Driven Development (BDD) in Cucumber is dé oplossing. BDD in Cucumber werkt voor geen meter. Of is er een genuanceerdere benadering? Matt Wynne, auteur van “The Cucumber Book ”, was 22 september in Eindhoven om ons te vertellen wat de toegevoegde waarde en de beperkingen van BDD zijn. Ik neem je mee in mijn weergave van zijn redevoering, met enkele redactionele toevoegingen.

Het probleem dat zo oud is als softwareontwikkeling zelf, is het verschil tussen wens en implementatie. De klant zegt dat hij een portal wil, maar bedoelt een intranet. Aan de kant van softwareontwikkeling wordt dit opgevat als een portal en voilà, daar is het eerste misverstand. Nu hebben we natuurlijk Agile en als we de principes goed toepassen komen we een heel eind. Echter, de praktijk is dat veel (Agile-)projecten in de loop der tijd tot een technisch feestje verworden met als gevolg dat het gezamenlijke begrip afneemt.

BDD heeft als doel om de business en softwareontwikkeling tot een gezamenlijk begrip te laten komen over het gewenste gedrag van software. Anders verwoord, het probleemdomein (business) en het oplossingsdomein (softwareontwikkeling) komen bij goed gebruik van BDD dichter bij elkaar te liggen.

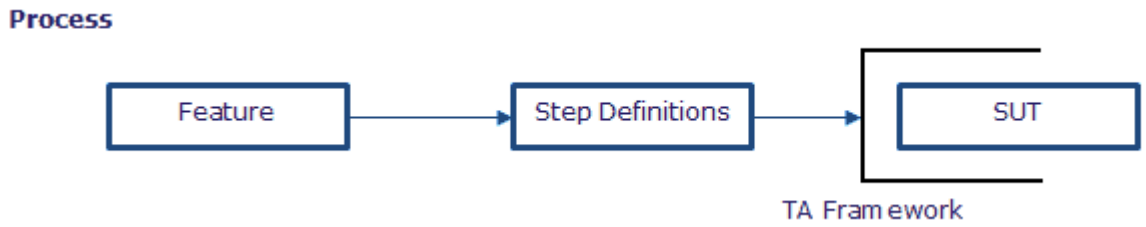


Cucumber

Cucumber is een samenwerkingstool dat faciliteert in de gezamenlijke begripsvorming betreffende het gewenste gedrag van software. Cucumber kan specificaties, geschreven in natuurlijke taal, uitvoeren. Deze uitvoerbare specificaties heten features en fungeren ook als geautomatiseerde tests en (levende) documentatie. Dit maakt BDD een Test-First benadering.
Het schrijven van features met daarin scenario's behoort in overleg tussen de drie amigo's te gebeuren. Dit zijn: de product owner/business analist, de tester, de ontwikkelaar. De conversaties die tijdens dit overleg plaatsvinden, vergroten het gezamenlijke begrip en helpen om "unknown unknowns" – verborgen onduidelijkheden – bloot te leggen.
De syntax die in deze features wordt gebruikt heet Gherkin. Een voorbeeld van een feature geschreven in deze syntax ziet er als volgt uit:

Feature: Login
Scenario: Successful Login
Given I am not logged in
When I login with user "test" and password "test"
Then I should be logged in and see my profile

Over de semantiek en het detailniveau van features dienen in een team afspraken gemaakt te worden, maar het voert te ver om daarover uit te weiden.



Een feature is een uitvoerbare specificatie, daarom wordt BDD ook wel Specification By Example genoemd. Als de feature wordt uitgevoerd, gebeurt er niets. Om deze specificaties daadwerkelijk iets te laten uitvoeren, moeten er step definitions geschreven worden. Een voorbeeld van de When-stap in Ruby:

When(/^I login with user "(.*?)" and password "(.*?)"$/) do |username, password|
@username = username
@password = password
…
end

Na het schrijven van de step definitions en het uitvoeren van de feature, zullen de tests falen. Als er falende tests zijn, kan de benodigde code voor het testautomatiseringsframework en

de applicatie worden geschreven. Als de tests slagen, zal de code gerefactord worden. Refactoren is het aanpassen van de structuur van de code zonder het gedrag te veranderen. Dit refactoren heeft als doel de herbruikbaarheid, leesbaarheid en efficiëntie te vergroten. Na het refactoren worden de features opnieuw uitgevoerd om zeker te stellen dat het gedrag nog correct is.

Leven na BDD

“We all know that silver bullets are bullshit”, zo begon Matt Wynne het gedeelte over het leven na BDD. BDD heeft voordelen, maar als het als dé oplossing wordt gezien, heeft het meer nadelen dan voordelen. Voordat een software-ontwikkelteam begint, moet het zich realiseren dat er leven is na BDD. Matt legde dit uit aan de hand van zes fases waar een team (meestal) door heen gaat terwijl het BDD leert toepassen.

Fase één heet: “Let’s make toast the American way! I’ll burn the toast, you scrape it”. Eerst wordt er software ontwikkeld, dan komen er testers die handmatige tests uitvoeren en uiteraard bevindingen doen. Een projectmanager kent prioriteiten toe, de programmeurs implementeren oplossingen en het verhaal begint weer bij de testers.

Fase twee heet: “I’ll burn the toast, you’ll automatically scrape it”. Eerst wordt er software ontwikkeld, dan schrijven testers geautomatiseerde tests en voeren deze uit. Hier komen bevindingen uit. Een projectmanager kent prioriteiten toe, de programmeurs implementeren oplossingen en het verhaal begint weer bij de testers.

Tot nog toe is er geen sprake van BDD, ook al zou Cucumber worden gebruikt. De software wordt achterwaarts gebouwd, want het verifiëren van het juiste gedrag (de acceptatiecriteria) vindt achteraf plaats.

Fase drie heet: “Three amigo’s”. Eerst vinden er conversaties tussen de drie amigo’s plaats. Daar komen features met scenario’s uit. Ontwikkelaars schrijven code om de scenario’s te laten slagen. Testers automatiseren andersoortige tests. Testers doen bevindingen tijdens deze exploratory tests. Een projectmanager kent prioriteiten toe, de programmeurs implementeren oplossingen en het verhaal begint weer bij de testers.

Fase vier heet: “Test first”. Eerst vinden er conversaties tussen de drie amigo’s plaats. Daar komen features met scenario’s uit en worden er abstracte regels geëxtraheerd. Ontwikkelaars automatiseren deze test en testers helpen bij het bedenken van (nieuwe/andere) tests. Ontwikkelaars schrijven bijbehorende code om de scenario’s te laten slagen. Bevindingen worden nieuwe scenario’s genoemd. Dus testers vinden nieuwe scenario’s tijdens hun exploratory tests. Een projectmanager kent prioriteiten toe en het verhaal begint weer bij het toevoegen van scenario’s in de features.

Fase vijf heet: “Disillusionment”. Builds duren een eeuwigheid en slagen zelden. Er zijn scenario’s bij die de ene keer falen en de ander keer slagen. Het aantal features is behoorlijk toegenomen en op UI-niveau zijn er veel te veel features die lang duren. Dit wordt ook wel het ice cream cone anti-pattern genoemd. Het team krijgt een gedeelde vijand: Cucumber. Veel teams houden hier op en geven Cucumber de schuld van hun problemen. Maar dat is niet terecht. Als een team in fase vijf terechtkomt, moet het zich afvragen wat er met de scenario’s moet gebeuren. Heeft een bepaald scenario nog toegevoegde waarde? Zo nee, verwijder hem dan. Zo ja, kan dit scenario ook vertaald worden in een test op een lager testniveau? Als het een unit(integratie)test kan worden, maak deze en verwijder het scenario.

Fase zes heet: “Transcendence”. Het probleemdomein wordt door het team begrepen en goed in een oplossing gemodelleerd. Scenario’s zijn voor iedereen goed leesbaar. De nadruk binnen het team ligt niet meer op (het aantal) scenario’s, maar op de abstracte regels die uit deze scenario’s voortvloeien. Er is een goede strategie die bepaalt op welk niveau welke tests plaatsvinden.

Conclusie

BDD kan goed worden gebruikt in combinatie met andere good practices. BDD is geen silver bullet, maar een manier om tot een gezamenlijk begrip te komen. Cucumber faciliteert in de gezamenlijke begripsvorming. Binnen deze kaders is BDD hot en is het gebruik van Cucumber zinvol.

David Baak, testspecialist bij Polteq

[1] https://pragprog.com/book/hwcuc/the-cucumber-book