Toelichting ‘Test-First’ principe

Het toepassen van het ‘test-first’ principe voor een Agile Release Train (ART), bijvoorbeeld voor de bouw van een portal, wordt in een SAFe traject door Infocon als volgt ingericht:

	TDD	BDD	ATDD
Stakeholders	Ontwikkelaar	Gebruiker, ontwikkelaar, tester	Gebruiker, ontwikkelaar, tester
Focus	Unit testing	Requirements	Acceptatietests
Moeilijkheidsgraad	Gemakkelijk	Moeilijk	Moeilijk
(Mogelijke) tooling	Gherkin, Cucumber, Easy, JDave, Concordion, JBehave, FitNesse, BeanSpec, Specflow	Gherkin, Cucumber, Easy, JDave, Concordion, JBehave, FitNesse, BeanSpec, Specflow	TestNG, FitNesse, Cucumber, JBehave, Concordion, EasyB
Toegevoegde waarde	Nieuwe features worden gefocust op de requirements	Nieuwe features worden gealigned met business behoeftes	Maakt unit testing gemakkelijker te implementeren

Om misverstanden te voorkomen wordt onderstaand duidelijk gemaakt wat bedoeld wordt met Test-Driven Development (TDD), Behavior Driven-ontwikkeling (BDD) en Acceptance Test-Driven Development (ATDD). Wat is het? Hoe werkt het? Bovendien, wat zijn de gevolgen daarvan voor het proces van ontwikkeling/testen?

Test-Driven Development: “Is de code correct?”

Test-Driven Development (TDD) richt zich op het "inside-out" perspectief, wat betekent dat een ontwikkelaar tests maken vanuit perspectief van een ontwikkelaar. "Is deze code juist?" Dat is de vraag achter TDD.

De methode richt zich specifiek op "unit tests." De ontwikkelaar neemt een requirement en zet deze vervolgens om in een specifieke testcase. Vervolgens schrijft de ontwikkelaar de code om alleen die bepaalde testgevallen te laten slagen. Deze praktijk is bedoeld om onnodige updates, die niet bijdragen aan de requirements, te voorkomen. Test-Driven Development (TDD) dwingt ontwikkelaars om zich te concentreren op de product requirements voordat de code wordt geschreven. Dit is een fundamenteel verschil met de traditionele programmering waar ontwikkelaars unit tests schrijven *na* het schrijven van de code.

TDD is een eenvoudig proces dat zes stappen volgt. Deze worden uitgeschreven aan de hand van het voorbeeld: “Laten we Spotify gebruiken om een specifiek album te streamen”.

Stap 1: Voeg een test toe.

De eerste stap is om een requirement om te zetten in een test die duidelijk is, zodat de ontwikkelaar volledig begrijpen de specificatie van het feature kan begrijpen.

In het voorbeeld van Spotify. Hier willen we een functie maken waarmee een abonnee kan zoeken naar een specifieke album om deze daarna te streamen. De test zou zijn om te controleren of een gebruiker een betaalde abonnee is en of het specifieke album zowel in de catalogus van Spotify is en een licentie heeft van de kunstenaar. Indien aan alle voorwaarden is voldaan, dan mag het album worden gestreamd.

Stap 2: Voer de test uit en kijk of de nieuwe test mislukt.

De volgende stap is het uitvoeren van de test om na te gaan of deze mislukt. Als de nieuwe test slaagt, betekent dat, dat het vereiste gedrag al bestaat en nieuwe code niet nodig is, of dat de nieuwe test gebrekkig is en moet worden gewijzigd.

In ons voorbeeld. Nu moeten we de test draaien om te kijken of deze mislukt. De gebruiker moet niet een specifiek album kunnen zoeken en streamen.

Stap 3: Schrijf de code.

De ontwikkelaar schrijft de code die de test laat slagen. Kwalitatief goede code is niet zo belangrijk in dit stadium; het doel is om de code schrijven die de test laat slagen.

In ons voorbeeld. De ontwikkelaar schrijft de code, die waarmee Spotify abonnees een specifiek album kunnen zoeken en streamen.

Stap 4: Uitvoeren van de test.

De vierde stap is het uitvoeren van de test. Als test slaagt voldoet de code aan de eisen en heeft de code geen invloed op bestaande features. Zolang de test niet slaagt moet de ontwikkelaar de code (blijven) aanpassen.

In ons voorbeeld. De tester moet kunnen zoeken naar een album, laten we zeggen “Because The Internet” by Childish Gambino en moet dit dan daarna kunnen streamen.

Stap 5: Refactor de code.

Nadat de test is geslaagd moet de nieuwe code eventueel nog worden aangepast om te voldoen aan de geldende programmeer standaards.

Stap 6: Herhaal.

"Herhaal" is niet noodzakelijkerwijs een volgende stap, maar een herinnering dat de ontwikkelaar het proces moet herhalen als er nieuwe eisen zijn of de functionaliteit van de software wil verbeteren.

Teams, die de Test-Driven ontwikkeling toepassen, resulteren meestal in aanzienlijke vermindering van defect aantallen, ondanks een stijging van de kosten van de initiële ontwikkeling. Bovendien leidt Test-Driven ontwikkeling tot verbetering van de kwaliteit van de code, die meer modulair, flexibel en uitbreidbaar is.

Behavior Driven-Development (BDD) – “Is dit wat we moeten testen?”

Behavior Driven Development (BDD) richt zich op het "outside-in" perspectief, wat betekent dat we gedrag testen, dat gerelateerd is aan business behoeftes. Het proces is zeer vergelijkbaar met TDD. Echter wordt het "Five Why’s"[1] principe toegepast op elke user story om ervoor te zorgen dat de business behoeftes gerelateerd is aan het doel. BDD vereist begeleiding van ontwikkelaars, testers en gebruikers om antwoorden op de "Why’s" achter een user story te vinden. Dit alles leidt tot de drijvende vraag achter BDD oftewel "Is dit wat wij moeten testen?"

[1] Zie: https://en.wikipedia.org/wiki/5_Whys

BDD is grotendeels een uitbreiding van de TDD-methodologie. De ontwikkelaar definieert een testcase, test de code om te verifiëren dat de testcase zal mislukken en schrijft vervolgens de code die nodig is om testcase te laten slagen en test vervolgens de code om juistheid van de code te controleren. Waar BDD verschilt van TDD is hoe de testcase wordt gespecificeerd. BDD test cases bedchrijven het gewenste gedrag. De duidelijke taal van BDD testcases maakt het eenvoudig voor alle stakeholders in een ontwikkelingsproject om het te begrijpen. Het volgende is een voorbeeld van een BDD test case voor een gebruiker die probeert een specifiek album op Spotify te streamen.

Voorbeeld
Story: Stream specifiek album op Spotify – *beschrijving van een user story*
Als een Spotify abonnee – *"wie": primaire stakeholder van een user story*
Wil ik om mijn favoriete album te kunnen afspelen – *"wat": effect van een user story op primaire stakeholder*
Kunnen zoeken en mijn favoriete album kunnen streamen uit de catalogus van Spotify – *"waarom": waarde primaire stakeholder*

Scenario
Gegeven dat Childish Gambino een licentie heeft – *conditie*
En de gebruiker een abonnement heeft op de Spotify service – *conditie*
Als de Spotify abonnee “Because The Internet” album van Childish Gambino selecteert uit de Spotify catalogus – *als het event wordt getriggerd*
Dan zal Spotify het “Because The Internet” album van Childish Gambino van het begin af aan streamen voor de abonnee – *het te verwachten resultaat*

Acceptance Test-Driven Development (ATDD) – “Doet de code wat het moet doen?”

De Acceptance Test-Driven ontwikkeling (ATDD) is bedoeld ter bevordering van de samenwerking tussen de gebruiker, ontwikkelaar en tester om ervoor te zorgen dat acceptatie tests bestaan voordat de code wordt geschreven. De acceptatietest is geschreven vanuit het perspectief van de gebruikers en dient als een requirement voor hoe de software moet functioneren. "Werkt de code zoals bedoeld?" Dat is de drijvende vraag achter ATDD.

Gegeven een album met licentie in de Spotify catalogus – *setup, gespecificeerde status*
En een gebruiker is heeft een betaald abonnement – *vervolg setup*
Als een gebruiker een album selecteert uit de Spotify catalugus – *trigger, een actie of event treedt op*
Dan wordt een album voor een gebruiker gestreamd – verificatie; output wordt geproduceerd, of status wijzigt