Continuous Testing

Productiviteitsverhoging door Continuous Testing.
Door Bart de Best

Context:
Deze blog is ontleend aan mijn ervaring als DevOps trainer, coach en auditor. Elke toepassing van Continuous Testing heeft weer meer inzichten gegeven over dit krachtige concept. Deze blog beschrijft zowel de succes verhalen als de beperkingen.

Uitdaging:
De uitdaging van de toepassing van Continuous Testing is het gegeven dat de DevOps engineer de knop in het hoofd moet omzetten om eerst een testcase te schrijven en dan pas de sourcecode. Het niet toepassen van Continuous Testing kan leiden tot een laat in de CI/CD secure pipeline vaststellen van defecten, een lage dekkingsgraad van testen, opoffering van testtijd aan programmeertijd en verlaagde prestatie door lang zoeken naar de veroorzaking van de defecten. De beloning van het toepassen van Continuous Testen kan oplopen tot 300% performance verbetering van de DevOps engineers.

Oplossing:
De oplossing voor deze uitdaging is gevonden in het concept van Continuous Testing waarin Test Driven Development verankerd is. Deze blog bespreekt de TDD aanpak van Continuous Testing aan de hand van de volgende stappen:

1. Definitie van Test Driven Development (TDD)
2. Definitie van Continuous Testing value stream
3. De werkwijze
4. De ervaringen

1. Definitie van Test Driven Development (TDD)

TDD is de kern van Continuous Testing, de value stream die binnen DevOps invulling geeft aan test management. TDD is een aanpak die gericht is op het integreren van testen en programmeren. Hiertoe wordt uitgegaan van de volgende principes:

- Shift left
- Testcase first
- Incrementeel Iteratief
- Unit testcase
- Code driven testing
- Test automation
- Isolatie

Shift left
De werkwijze van TDD is gericht op het uitvoeren van 80% van de testcases in de ontwikkelomgeving teneinde 80% van de defecten in de ontwikkelomgeving (O) te vinden.

Daardoor is er nog maar 20% van de testinspanning in de testomgeving (T) en de acceptatieomgeving (A) over. De testinspanning verschuift in de O-T-A-P-straat dus van rechts naar links.

Testcase first
Het principe 'testcase first' verwijst naar het gegeven dat eerst een testcase wordt geschreven en dan een klein deel van de sourcecode die net genoeg is om de testcase succesvol uit te kunnen voeren. Daardoor moet er eerst over de sourcecode worden nagedacht voordat deze wordt geschreven.

Incrementeel Iteratief
In plaats van een unit (functie et cetera) in één keer te schrijven en dan in één keer te testen wordt een unit stap voor stap gebouwd waarbij elke stap begint met een extra testcase. Het incrementele en iteratieve karakter van Agile Scrum wordt dus doorgetrokken tot op het schrijven van sourcecode. Een nieuw increment van de unit vereist dat alle testcases succesvol zijn afgesloten.

Unit testcases
TDD richt zich op de kleinste unit van programmeren hetgeen vaak een functie is die bijvoorbeeld geschreven wordt in Python of Java. Daardoor is de integratie van testen en programmeren ook mogelijk geworden.

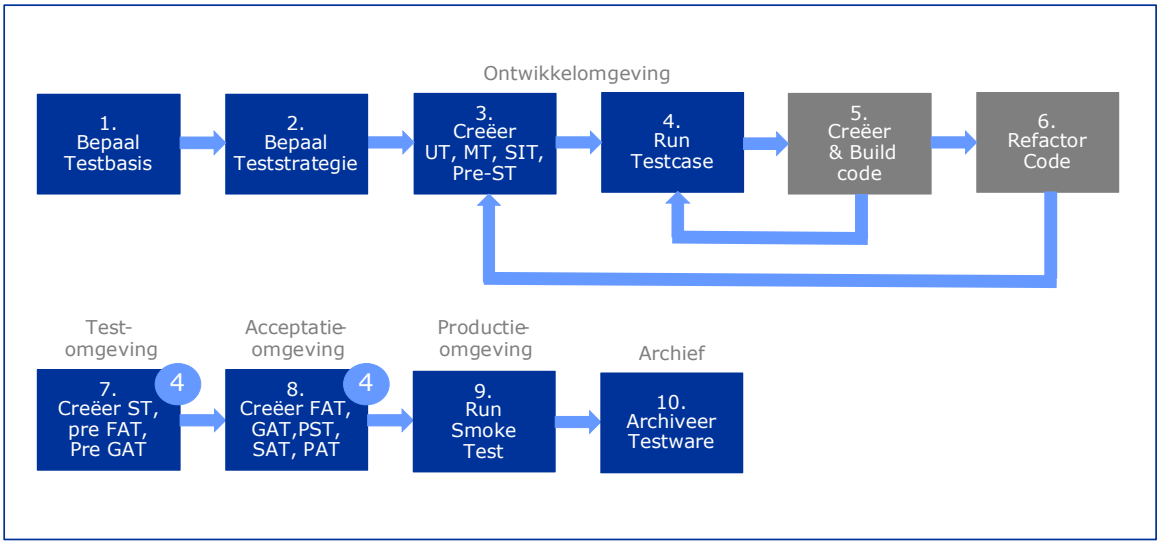
Code driven testing
Unit testcases worden geprogrammeerd in de taal van de sourcecode. Hierdoor zijn de testcases ook in GIT op te nemen en te voorzien van versiebeheer. Tevens is in GIT de relatie tussen de unit testcases en de sourcecode items te leggen. Dit leidt ertoe dat het uitchecken van de sourcecode in de meeste editors ook de testcase uit checkt zodat beide tegelijker tijde aangepast kunnen worden.

Test automation
De uitvoer van de unit testcases is verplicht bij elk increment door de unit testcases na de build direct af te trappen. De testrun van unit testcases mag niet meer dan 5 minuten kosten.

Isolatie
De unit testcase moet in isolatie worden getest. Dit wil zeggen dat deze geen interfaces mag hebben met databases, andere applicaties, netwerkverkeer, message queues, e-mail services et cetera. De consequentie is dat soms gewerkt moet worden met mocking dan wel faking om de unit onder test aan de tand te voelen.

2. Definitie van de Continuous Testing value stream
Een voorbeeld van een Continuous Testing value stream is weergegeven in figuur 1. TDD start in stap 3 waarin de unit testcase (UT) wordt gecreëerd. Die testcase wordt in stap 4 uitgevoerd zonder dat er sourcecode is geschreven en gaat dus in de eerst testrun van het betrokken increment fout.

Dan wordt de sourcecode geschreven in stap 5 en wordt net zolang stap 4 en 5 uitgevoerd tot de testcase succesvol is doorlopen. Daarna wordt de sourcecode opgeschoond en wordt een nieuw increment van de unit gestart door in stap 3 de nieuwe unit testcase te schrijven.



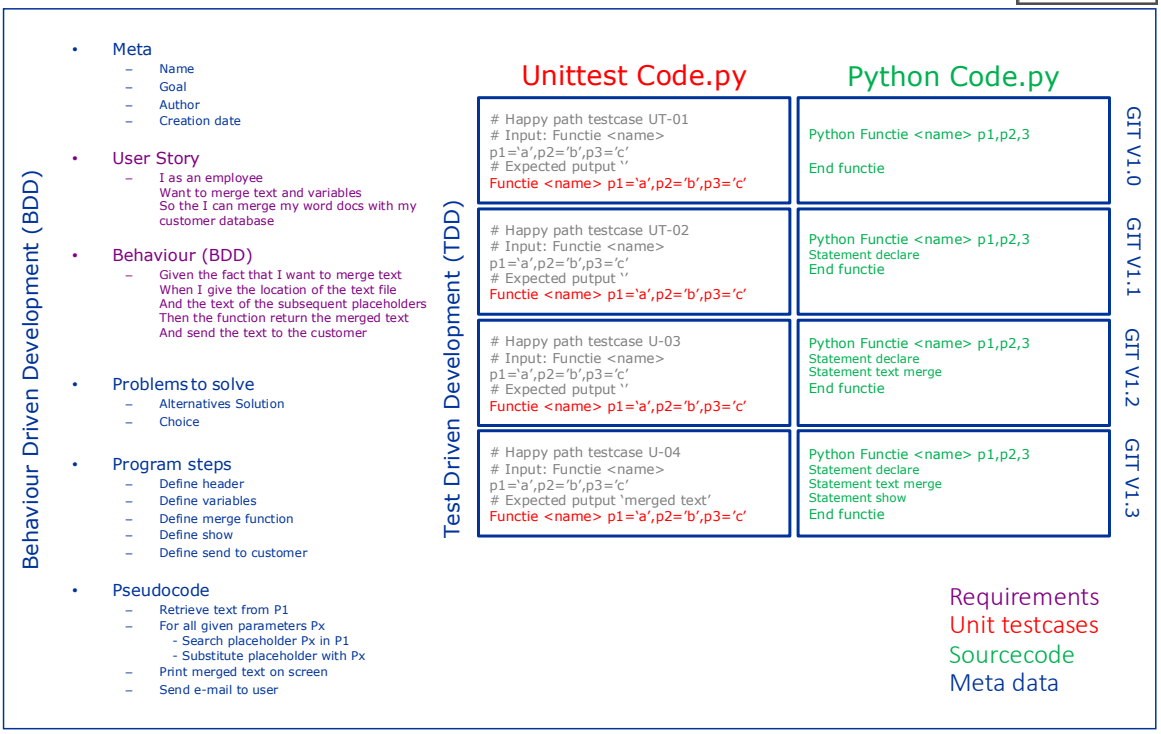
Figuur 1. Continuous Testing value stream.

De afkortingen in figuur 1 zijn:

- MT = Module Test
- SIT = System Integration Test
- ST = System Test
- FAT = Functionele Acceptatie Test
- GAT = Gebruiker Acceptatie Test
- SAT = Security Acceptatie Test
- PAT = Productie Acceptatie Test
- PST = Performance Stress Test

3. De werkwijze

In figuur 2 is een overzicht gegeven van de TDD aanpak. Links is eerst de meta data van de functie geschreven. Daarna is de userstory en het requirement (BDD) benoemd. Deze zijn de basis voor de sourcecode.



Figuur 2, Voorbeeldtoepassing van TDD.

Voordat begonnen wordt aan de sourcecode wordt eerst een lijstje gemaakt van de problemen die opgelost moeten worden. Het aantal problemen hangt af van de ervaring van de DevOps engineer. Een ervaren DevOps engineer benoemt alleen de overwegingen voor oplossingen en de keuze die is gemaakt voor de implementatie. Vervolgens worden de stappen waarin de functie worden beschreven opgenoemd gevolgd door de pseudocode die beschrijft hoe de functie moet gaan werken.

In het algemeen kost deze definitie ongeveer 15 minuten. Het kan echter vele malen meer tijd besparen tijdens de aanpassing van de sourcecode en het oplossen van bugs.

Rechts is in het rood de unit testcase te zien en in het groen de sourcecode. De syntax van de testcase en sourcecode klopt natuurlijk niet en is alleen ter indicatie van de werkwijze zo geschreven.

4. De ervaringen

In de loop van de jaren heb ik verschillende ervaringen mogen opdoen met TDD in het kader van Continuous Testing die ik graag met u deel.

Eigen ervaring

In een half jaar tijd heb ik de taal Python leren programmeren op basis van het boek ‘Think python’ op basis van een vrijwillige training buiten werktijd bij een klant. De oefeningen waren best pittig en het direct programmeren van de oplossing leidde vaak tot het niet kunnen afronden van de opdracht op een avond. Totdat ik TDD toepaste en stap voor stap de opdracht schreef op basis van eerst geschreven testcase en dan de sourcecode. Dit scheelde mij in de meeste gevallen een factor 3 qua tijd.

De fout die ik in het begin maakte was alle unit testcases proberen te schrijven en dan pas de sourcecode. Het was niet alleen erg moeilijk om vooraf alle unit testcases te definiëren maar het leverde ook niet de feedback op tijdens het programmeren die ik nodig had om een fout te vinden. Toch was ik erg eigenwijs en probeerde nog menig keer zonder TDD de opdracht af te ronden. Na een aantal weken heb ik definitief de oude manier van programmeren achter mij gelaten en kon veel sneller mijn leercurve doorlopen.

Training ervaringen

Het voordeel van TDD wordt wel eens ter discussie gesteld in een training. Dit zijn altijd heel fijne discussies en geven een scherper beeld van de toepasbaarheid van TDD. Lastige toepassingen zijn:

1. Userinterface ontwikkeling.
   Hierbij is het lastig om testcase te schrijven omdat de userinterface vaak niet een losse functie is maar direct leidt tot een End-2-End test. TDD kan wel gebruikt worden maar dan is het zaak om te kijken of de interface tussen de front-end en de back-end van de applicatie te gebruiken. Als de front-end een functie aanroept van de back-end dan kan daar de unit testcase tegen geschreven worden.

2. Software die sterk afhangt van infrastructurele services.De software moet in isolatie worden getest. In dit geval moet dus mocking of faking worden gebruikt. Het ontwikkelen daarvan kan een significatie tijdsbesteding omvatten. Er moet dan ook een business case moet afgewogen.

3. Legacy software.
   Dit is software waarvan de code vaak een monoliet is. Dit wil zeggen dat er geen losse functies te onderkennen zijn en er eigenlijk alleen systeemtesten en dergelijke mogelijk zijn. Refactoring van de applicatie moet dan overwogen worden zoals het ombouwen tot microservices. Dit kan een kostbare exercitie zijn die vooral voor applicaties wordt gedaan die de primaire business value streams ondersteunen. Het is wel zo dat AI hier een enorme versnelling in kan geven.

4. Data intensieve applicaties.
   Het toepassen van TDD bij applicaties waarvan de logica sterk afhankelijk is van de data zoals bij data-analyse en machine learning is lastig. Er zal in dat geval wellicht wat afbreuk moeten worden gedaan aan een aantal TDD principes.

Coach ervaringen

De vraag bij consultancy is vaak of TDD optineel kan worden toegepast, dus indien de DevOps engineer dit nodig acht. Dit is een listige stelling omdat de kans op verwatering van TDD dan continue op de loer ligt. Daarom adviseer ik om altijd TDD te doen, tenzij bijvoorbeeld de mocking te duur is, dan kan nog steeds TDD worden gedaan maar moet er flexibel met de TDD principes worden omgegaan.

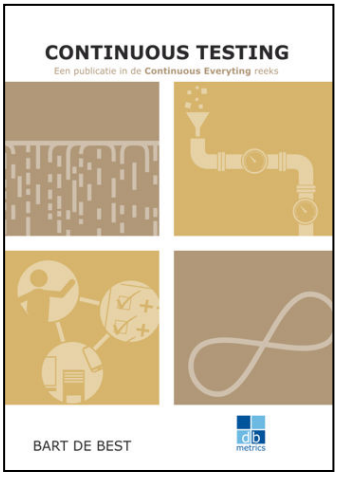
Een andere ervaring is dat de test management termen niet conform de common sense worden gehanteerd. Zo worden nog wel een Selenium testen van de user interface als unit testcases gedefinieerd. Dit is niet verstandig omdat hierbij de hele applicatie doorlopen wordt. Dit vervagen van terminologie is verraderlijk in de communicatie.

Audit ervaringen

Ik heb diverse organisaties mogen auditen op Continuous Testing. Hierbij ontstaat er nog wel eens de discussie waarom TDD op niveau 2 van volwassenheid moet worden gezien op de schaal van 5 (CMM). Waarom is dit niet niveau 3? De reden hiervoor is dat niveau 2 van het CMM model aangeeft dat de flow is ingeregeld, zijnde de stappen van de value streams. TDD hier voor Continuous Testing het anker omdat het invulling geeft aan de shift left organisatie. Het weglaten op niveau 2 veroorzaakt een heel andere flow zonder fast feedback.

Met TDD kan continu de kwaliteit van de software ontwikkeling worden gevolgd. De unit testcases kunnen ook gebruikt worden bij regressietesten. Daarom is TDD is een goed voorbeeld van het invulling geven aan Continuous Testing.

Door Bart de Best
DutchNordic.Group



https://www.dbmetrics.nl/ce-nl/continuous-testing-nl/