Test Driven Development (TDD)

Inleiding

Bij softwareontwikkeling is Test Driven development (TDD) tegenwoordig een veel gebruikte methode. TDD is een methode waarbij, in kleine incrementele stappen, unit tests worden gemaakt, en in dezelfde kleine incrementele stappen wordt de code gemaakt om aan die tests te voldoen.

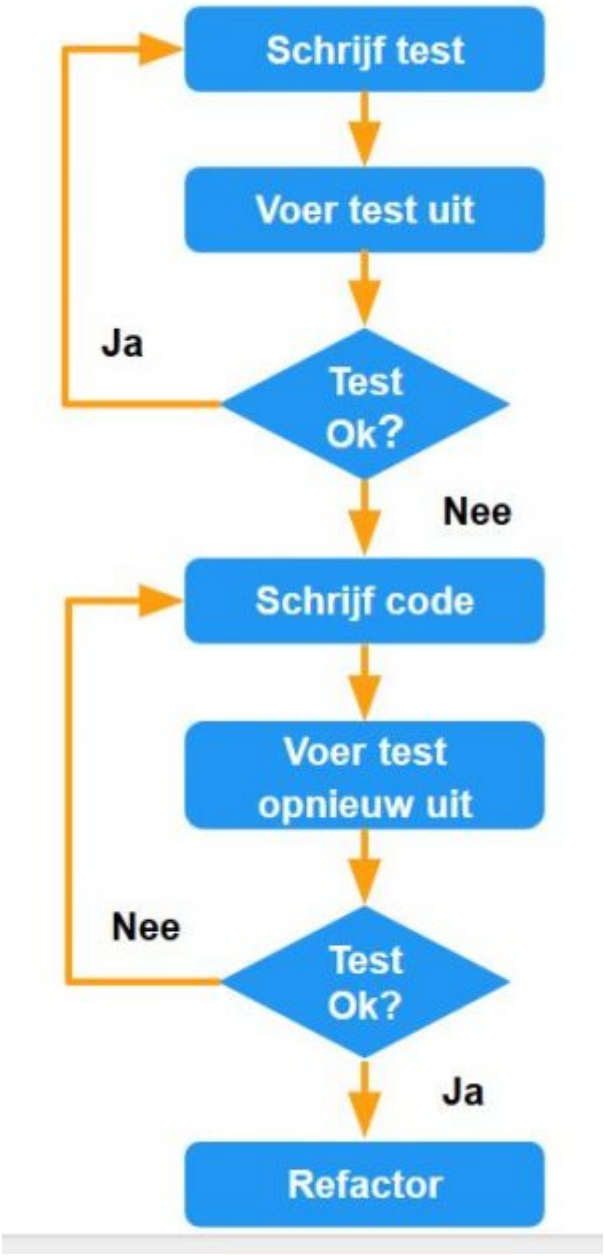
Dit is het tegenovergestelde van een traditionele aanpak van softwareontwikkeling waarbij eerst de code wordt gemaakt en er vervolgens wordt getest.

Wat is TDD?

Vooraf

De kern van TDD draait om vijf eenvoudige stappen, die tijdens de hele levenscyclus van de softwareontwikkeling herhaald worden. Het doel van deze stappen (en van TDD in het algemeen) is ervoor te zorgen dat de code eenvoudig en efficiënt is en tegelijkertijd aan alle functionele eisen voldoet.

De volgende stappen worden steeds opnieuw uitgevoerd door de ontwikkelaar:
- schrijf een unit test die een aspect van het programma test
- voer de test uit, deze moet mislukken omdat in het programma de functionaliteit nog ontbreekt
- schrijf “net genoeg” code, zo eenvoudig mogelijk, om de test te laten slagen
- voer de unit test opnieuw uit
- als de test slaagt, ga dan verder met de volgende test, en anders herschrijf / wijzig de code om de test te laten slagen
- Onderstaande flowchart is een visualisatie van deze 5 stappen:



Test driven development flowchart

Schrijf een unit test

Bij TDD begint de ontwikkeling van elke nieuwe functie met het schrijven van een unit test. Deze unit test kan niet anders dan mislukken omdat deze is geschreven voordat de code voor de betreffende functie is geschreven. Wanneer de unit test onverhoopt toch slaagt, dan is of de gewenste functionaliteit al gebouwd of is de unit test niet goed.

Voer unit test uit

Het uitvoeren van de nieuwe unit test moet aantonen dat de test niet per ongeluk slaagt zonder dat er nieuwe code is geschreven. Kortom, de test moet falen.

Schrijf code

De volgende stap is het schrijven van een stukje code die ervoor zorgt dat de unit test slaagt. De nieuwe code hoeft in dit stadium nog niet perfect te zijn en kan de unit test bijvoorbeeld op een onelegante manier laten slagen. Dat is opzich geen probleem om dat in een latere stap de code kan worden verbeterd.

Belangrijk uitgangspunt is, dat de geschreven code alleen is gemaakt om de unit test te laten slagen. Niet meer en niet minder.

Voer unit test opnieuw uit

Voer de unit test opnieuw uit en controleer of de unit test slaagt. Wanneer dat het geval is kan doorgegaan worden met de volgende stap, anders dient de code te worden aangepast om er voor te zorgen dat de unit test alsnog slaagt.

Refactor

Indien nodig kan de code nu worden opgeschoond. Hierbij speelt het verwijderen van dubbele code een essentiële rol. Na het opschonen worden alle beschikbare unit test opnieuw uitgevoerd om te controleren dat er door het schonen geen regressie is ontstaan.

Voordelen van TDD

- Het kan leiden tot eenvoudige en modulaire code. Dit bevordert de onderhoudbaarheid van de code.
- Het kan ontwikkelaars helpen om fouten eerder te vinden dan anders zou gebeuren. En zoals Boehm al in 1979 heeft aangetoond: hoe eerder fouten worden gevonden des te goedkoper het oplossen is.
- De unit testen kunnen dienen als een soort documentatie waardoor de code makkelijk te begrijpen is.
- Op de lange termijn kan het de ontwikkeltijd versnellen.
- Het helpt ontwikkelaars om gefocused te blijven op het bieden van de eenvoudigste oplossingen voor een probleem.

Nadelen van TDD

- TDD vergt in het begin van een project veel tijd en moeite waardoor het kan lijken dat de ontwikkeling langzamer gaat dan anders.
- Het is moeilijk om goede unit test te schrijven die de essentie afdekt.
- Wanneer de functionaliteit wijzigt dan moeten de unit tests ook worden aangepast. Dit kan leiden tot veel meer inspanning als er veel functionele wijzigingen zijn.