# Deel 2: Testautomatisering is testen

In deel 1 van deze blogserie hebben we stilgestaan bij wat je kunt verwachten van testautomatisering en wat je wel en niet kunt (test)automatiseren. In deze blog zullen we zien dat testautomatisering niet anders is dan gewoon testen.

Testautomatisering is testen, maar dan op een geautomatiseerde manier. Doordat geautomatiseerd testen in essentie niet verschilt van handmatig testen (controleren op verwacht resultaat) gaat veel van de bestaande theorie rond testen ook op voor testautomatisering.

We zijn gewend om handmatig testen uit te voeren op verschillende testniveaus (bv. systeemtest, integratietest, ketentest). Het principe van testniveaus geldt ook voor geautomatiseerd testen. Je kunt geautomatiseerd testen toepassen op allerlei testniveaus (zie sectie “De testautomatiseringspiramide” voor meer details)

Met testautomatisering zijn we in staat om in korte tijd veel tests uit te voeren zonder dat het tijd kost van mensen. Voor het uitvoeren van de tests is dit waar, maar het maken en beheren van tests kost wel tijd en inspanning van mensen. Ook bij geautomatiseerd testen moeten we er dus voor zorgen dat we met zo min mogelijk tests een zo groot mogelijke testdekking verkrijgen. We moeten dus goed nadenken over wat de efficiënte tests zijn voordat we tests gaan bouwen. Voor het ontwikkelen van efficiënte tests zijn twee zaken belangrijk:

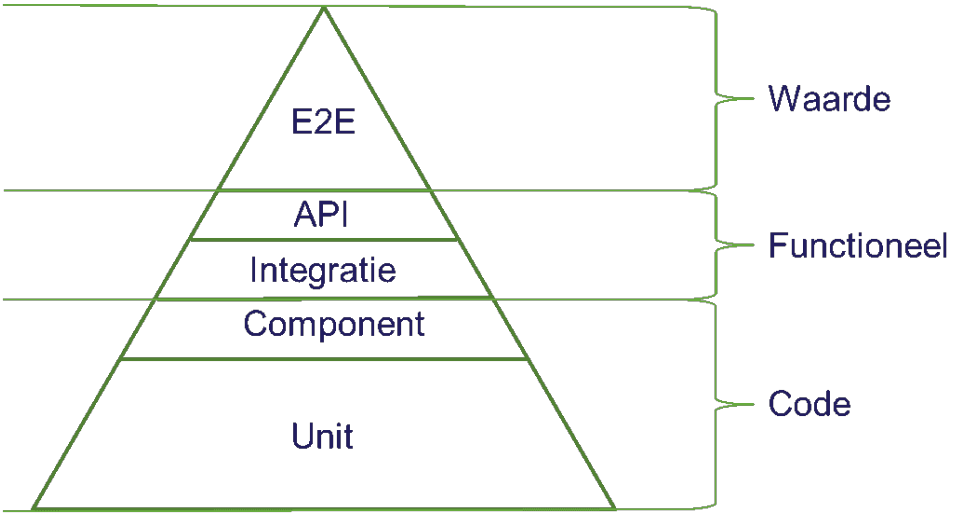
- No risk-no test. Een test mag alleen bestaan als hij een onderkend risico afdekt. Hierbij krijgen grote risico’s meer testdekking en lage risico’s minder testdekking.
- Door gebruik te maken van testtechnieken als equivalentieklassen, grenswaarden en slimme datacombinaties kunnen we de hoeveel tests beperken terwijl we toch een goede testdekking creëren.

Tests, dus ook geautomatiseerde tests, hebben als doel informatie te verschaffen over het geteste (sub)systeem. Dit doen we door het opstellen van inzichtelijke testrapportage. We zullen zien dat testautomatisering op verschillende testniveaus kan worden toegepast, ook op sterk technische niveaus (whitebox tests). Het is belangrijk om de geautomatiseerde tests zoveel als mogelijk te beschrijven in herkenbare termen die aansluiten bij de belanghebbende van het testniveau. Dan slaagt testautomatisering erin waardevolle informatie te leveren waar de afnemers (ontwikkelaar, tester, business) ook echt iets aan hebben.

# De testautomatiseringspiramide

Een *good practice* voor testautomatisering is de test(automatiserings)piramide. Dit is een heuristiek voor het correct opbouwen van de testdekking van geautomatiseerd testen. Een groot voordeel van testautomatisering ten opzichte van handmatig testen is dat je het product/systeem ook whitebox kunt testen. Er ontstaan nieuwe testniveaus zoals unit test en component tests, op diepere lagen in de code.

**De testpiramide ziet er als volgt uit:**




De piramide is onderverdeeld in drie soorten testdekking: codekwaliteit, functionele kwaliteit en waarde voor de eindgebruiker.

Unittests zijn kleine compacte tests op code units (een klasse of een methode). Ze hebben

als doel aan te tonen dat code correct werkt. Hiertoe worden de afhankelijkheden tussen verschillende units verwijderd. Elke unit wordt in isolatie van de andere units getest, dit wil zeggen dat eventuele afhankelijkheden tussen units worden vervangen door test doubles (f. Dit heeft een aantal voordelen:

- Code units zijn klein, ze bezitten maar een beperkte functionaliteit. De tests zijn dus klein en daardoor snel.
- Als een test faalt, is snel te achterhalen in welk stukje code de fout zit; de code unit is immers geïsoleerd. Debuggen is dus snel en eenvoudig.

De kracht van unittests is ook haar beperking. Correct functionerende, losse onderdelen van de code maken nog niet dat het samengestelde geheel van code correct werkt. Door het loskoppelen van afhankelijkheden geven unit tests geen garantie voor een goed werkend geheel.

Het volgende testniveau is component test. In dit testniveau worden de echte afhankelijkheden tussen units gebruikt en worden samenhangende units (een softwarecomponent) integraal getest. Afhankelijkheden met andere componenten worden op dit niveau niet meegenomen.

Op het integratietestniveau testen we componenten in samenhang. Samenhangende componenten realiseren (deel)functionaliteit, dus op dit niveau is het mogelijk om (deel)functies en/of business logica tegen de functionele requirements te testen.

Op eventueel aanwezige API’s kunnen we vaststellen of een systeem op de technische koppelvlakken correct functioneert. API’s zijn relatief eenvoudig te benaderen met testtools en geven de mogelijkheid functioneel te testen conform functionele ontwerpen. Veel business logica is dan vaak al testbaar.

Als laatste rest het E2E-testniveau. Dit is het traditionele testniveau waarop we een systeem of een keten van systemen als een black-box, via de GUI, handmatig testen. Hier testen we het complete systeem op correct geïmplementeerde user journeys, op waarde voor de eindgebruiker. Op dit niveau geeft een geslaagde test veel vertrouwen in de kwaliteit van het product, aan de andere kant raakt de test veel samengestelde code waardoor debuggen tijdsintensief is en de test veel tijd kost om uit te voeren. Daarnaast kan de test door het grote aantal externe factoren eerder ten onrechte falen, waardoor de beheerinspanning toeneemt en de waarde van de test afneemt. Daarom is het van belang dat de onderliggende testniveaus voldoende dekking bieden.

## Betekenis piramidevorm

Het vaststellen van geschikte testniveaus binnen de context van het product is onderdeel

van het bepalen van de teststrategie. De lagen in de testpiramide kunnen per context verschillen omdat ze geënt zijn op de architectuur van het systeem. Het principe blijft echter steeds overeind: hoe lager in de piramide hoe sneller de test maar hoe meer je een geïsoleerd stuk code test waardoor het minder toevoegt aan het vertrouwen dat het systeem het juiste doet. Hoe hoger in de testpiramide hoe trager de test, maar op dit niveau geeft elke geslaagde test wel veel vertrouwen in correct gedrag voor de eindgebruiker. Hierin moet een goede verdeling worden gezocht. Dit komt tot uiting in de piramidevorm. Die geeft aan dat de testdekking op de aanwezige risico's in het product (testen is risicogebaseerd) zoveel mogelijk naar beneden in de testpiramide moet worden gedrukt. Onderin de piramide borgen we testdekking op deelfunctionaliteit, edge cases, equivalentieklassen, foutafhandeling, etc. Deze testdekking vul je aan met een kleinere testdekking op integratie en API-niveau. De testdekking wordt gecompleteerd met een zeer beperkt aantal E2E gebaseerde tests. Hier zie je dus een duidelijk verschil met handmatig testen dat zwaar leunt op E2E gebaseerd testen.

## Vooruitblik volgende blog

Tot zover deze blog. In de volgende blog staan we stil bij code coverage en de aanpak van testautomatisering voor de verschillende lagen van de testpiramide.

STAY TUNED!

**Wim ten Tusscher**