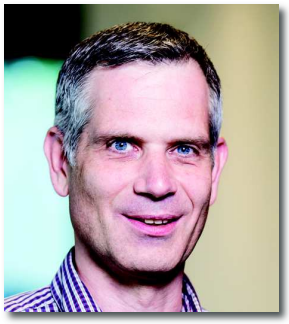
HET LEVEN VAN EEN TOPMODEL

Door Joris Meerts ● joris.meerts@improveqs.nl

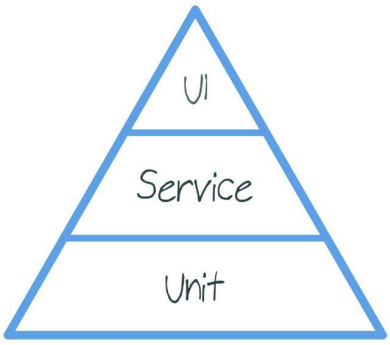


'All problems in computer science can be solved by another level of indirection' - David Wheeler

De testautomatiseringpiramide heeft een belangrijke plek in Agile software ontwikkeling. Het model—dat oorspronkelijk bedacht is door Mike Cohn in 2004 [Cohn, 2004]—heeft in de afgelopen jaren flink aan populariteit gewonnen. Dat blijkt onder andere uit het aantal variaties dat voorbij komt in talloze artikelen en weblogs. Maar deze wildgroei heeft het oorspronkelijke doel van de piramide en de kracht van het model aangetast. Zozeer dat het noodzakelijk is terug te gaan naar de basis.

Een populaire piramide

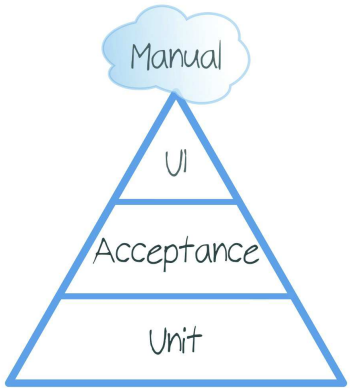
De kracht van de piramide zit in de eenvoud. Cohns model uit 2009 [Cohn, 2009], gepubliceerd in het boek *Succeeding with Agile* bestaat uit slechts drie lagen, waarbij elke laag grofweg verwijst naar een laag in de softwarearchitectuur. In elke laag is met een enkel woord aangeduid om welke laag het gaat: *Unit*, *Service* en *UI* (als afkorting voor user interface). Er wordt niet gegoocheld met aantallen; de verhoudingen tussen de hoeveelheden testen in elke laag zijn heel algemeen af te leiden uit de piramide: veel, minder, minst. Wat het model ons duidelijk wil maken, is dat we testen vooral automatiseren tegen de architectuurlaag waarop zij het meest efficiënt zijn.



Figuur 1: Cohns piramide uit Succeeding with Agile (2009)

In de wolken

Gezien de populariteit van het model zijn er verschillende aanpassingen gedaan die het doel van de oorspronkelijke piramide hebben vertroebeld. Een belangrijke aanpassing is het toevoegen van een wolk boven de piramide. Deze wolk bevat de handmatige uit te voeren testen. De wolk is toegevoegd door Lisa Crispin en Janet Gregory in het boek *Agile Testing* [Crispin en Gregory, 2009]. Crispin en Gregory maakten de wolk zodat we niet vergeten dat handmatig testen een onderdeel is van het testen van de software. Maar het geeft het de indruk dat handmatig testen een laag is bovenop de geautomatiseerde testen. Dat roept de vraag op hoe de handmatige testen zich verhouden tot de geautomatiseerde testen. Hebben we meer of minder handmatige testen dan geautomatiseerde UI testen? En zijn handmatige testen altijd minder efficiënt dan geautomatiseerde testen?



Figuur 2: Een verkorte weergave van Lisa Crispins en Janet Gregory's piramide uit het boek Agile Testing (2009)

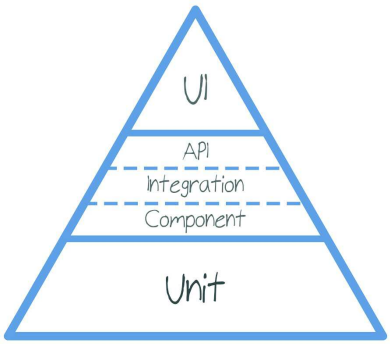
Bovendien is 'manual' geen laag van de software architectuur. We moeten ons dus afvragen tegen welke laag van de softwarearchitectuur deze testen worden uitgevoerd. Misschien wel tegen alle lagen. Maar dan hoort de wolk niet aan de bovenkant van de piramide te staan. Al met al roept de toevoeging vragen op die niet door het model kunnen worden beantwoord.

Een nieuwe visie op de middenlaag

In Crispins en Gregory's plaatje is ook de naam van de middenlaag aangepast van *Service* naar *Acceptance*. Ze hebben deze benaming overgenomen van het model dat Cohn in 2004 voor ogen had. Crispin en Gregory doelen op de acceptatie testen die worden geautomatiseerd met tools zoals FitNesse, Cucumber of SpecFlow. Het gaat om voor de klant leesbare testen die doorgaans zijn opgeschreven in een 'given-when-then' structuur. Ze worden waarschijnlijk grotendeels uitgevoerd tegen architectuur lagen die zich onder de UI—en dus buiten het zicht van de klant—bevinden. Maar de term *Acceptance* zegt meer over het soort test dat wordt uitgevoerd en het doel daarvan, dan over de laag in de architectuur. De acceptatietesten hoeven niet uitsluitend te worden uitgevoerd tegen de middenlaag van de architectuur. Bovendien kunnen er grote verschillen ontstaan tussen acceptatietesten wat betreft omvang, complexiteit en breekbaarheid. De piramide met de term Acceptance in het midden geeft geen inzicht in hoe we acceptatie testen moeten organiseren. Daardoor verliest dit model aan waarde ten opzichte van Cohns model uit 2009.

Een heet hangijzer

De middenlaag van de piramide is in meerdere opzichten een heet hangijzer. In allerlei variaties wordt aangetoond dat mensen worstelen met de toepassing van de laag in de praktijk. In een prominent voorbeeld van deze worsteling wordt de middenlaag opgesplitst in drie delen, te weten *Component*, *Integration* en *API*. Deze variatie is bedacht door Alister Scott in 2011 [Scott, 2011]. Scott splitst de laag op uit onvrede over de vage term 'Acceptance' (gepopulariseerd door Crispin en Gregory) en probeert de laag beter te definiëren. Hij richt zich daarbij op wat (welke ingangen) de geautomatiseerde testen gebruiken om te kunnen testen. Met de opdeling van de middenlaag voegt Scotts poging een dimensie toe aan de piramide.



Figuur 3: De opsplitsing van de middenlaag, naar het model van Alister Scott (2011)

Cohn zegt dat we de testen zo efficiënt mogelijk moeten verdelen over verschillende architectuurlagen waarbij we op laag niveau veel kleine, snelle testen maken die weinig afhankelijkheden bevatten. Naarmate we hoger in de piramide komen, worden de testen groter, trager en zijn ze breekbaar omdat ze meer afhankelijkheden bevatten. Cohn heeft het niet over de functionaliteit die de software bevat voor het ontsluiten van testen en gaat er waarschijnlijk vanuit dat testen in alle architectuurlagen altijd met hetzelfde gemak te schrijven zijn. De piramide gaat altijd op.

Scotts model is in zekere zin realistischer. Hij zegt niet te kijken naar lagen in de software architectuur maar naar de functionaliteit die de software biedt waarop de testen kunnen inhaken. Deze visie is interessant en misschien bruikbaar maar ze breekt het piramide model. Het kan voor komen dat unittesten namelijk niet makkelijk te maken zijn. Dit is bijvoorbeeld het geval bij het testen van PL/SQL database code. Het unittesten van deze code vereist voorwerk (het vullen van bepaalde tabellen in de database) en kan alleen worden gedaan met een niet eenvoudig te hanteren framework. Het kan in zo'n situatie zo zijn dat het eenvoudiger is om API-testen te schrijven dan unittesten—dat de API laag beter ontsloten is. Dat zou betekenen dat er meer API testen zijn dan unittesten en dat is niet wat de piramide voorschrijft. Door een nieuwe dimensie te introduceren zorgt Scott ervoor dat de wereld minder makkelijk in het model te vatten is.

Vage begrippen

Ook introduceert Scott de begrippen *Component*, *Integration* en *API*. Binnen de vakgebieden softwareontwikkeling en softwaretesten zijn dit alles behalve eenduidige begrippen. Recent nog heb ik bijvoorbeeld gedurende twee dagen gediscussieerd over de betekenis van integratie testen. Uit die discussie bleek vooral dat er over dit onderwerp diepgewortelde persoonlijke meningen bestaan. Helaas is het zo dat we er in het vakgebied softwaretesten nooit vanuit mogen gaan dat iedereen dezelfde voorstelling heeft van een term, zelfs niet van een gangbaar begrip als integratie testen.

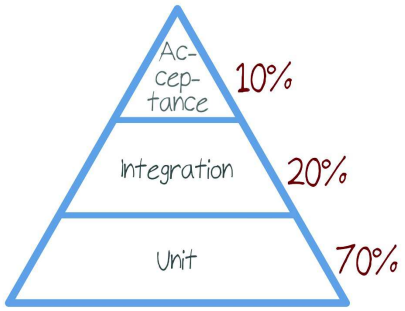
Scott vervangt de vage term *Acceptance* door drie andere onnauwkeurige termen. Neem bijvoorbeeld de benaming *Integration*. Deze term lijkt te wijzen op het test level integration test uit de ISTQB methodiek. Maar dit is waarschijnlijk iets anders dan wat Scott in gedachten had. Het is daardoor niet makkelijk om vanuit het model van de piramide antwoorden te krijgen op vragen over Scotts middenlaag. Klopt het bijvoorbeeld dat er meer integratie testen zijn dan API-testen? Integratie testen lijken, vanwege het feit dat ze complexer en breekbaarder kunnen zijn dan API-testen, juist minder voor te moeten komen. En tegen welke architectuur laag voeren we integratietesten uit? Volgens ISTQB kunnen we dit tegen meerdere lagen doen en in de praktijk zal dit ook tegen

meerdere lagen gebeuren. Tot slot kunnen we ons afvragen in welke laag de integratietesten vallen die we middels de API uitvoeren.

Cohn geeft in *Succeeding with Agile* aan dat zijn aanduiding 'Service' bedoeld is als een generieke term die niet alleen betrekking heeft op een service-oriented architecture. Volgens hem zijn alle applicaties opgebouwd uit een vorm van services. Uiteraard kan de middenlaag meerdere architectuurlagen beslaan. Maar de kracht van Cohns model zit in het niet verder uitsplitsen van die architectuurlagen. Hierdoor verzanden we niet in een discussie over definities of bijvoorbeeld over de mate van breekbaarheid van een test. Zijn API testen bijvoorbeeld breekbaarder dan integratie testen? Wie zal het zeggen?

Verhoudingen

De verhoudingen tussen de 'hoeveelheid' testen in Cohns piramide zijn niet absoluut vastgelegd. Cohn noemt geen getallen. De relatieve verhoudingen zouden afgeleid kunnen worden uit het oppervlak van elke laag. Maar ook hierover doet hij geen uitspraken. Toch zijn er interpretaties die verhoudingsgetallen koppelen aan de verschillende lagen. In een artikel in 2011 publiceert James Crisp een piramide waarin hij de verhoudingen tussen de verschillende lagen kwantificeert.



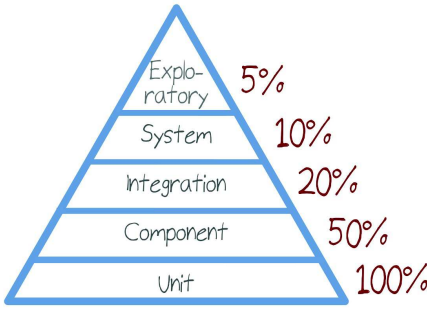
Figuur 4: James Crisp (2011) kwantificeert de verhoudingen tussen de lagen van de piramide

Het totaal van de geautomatiseerde testen bedraagt uiteraard 100%. Crisp laat de verdeling zien van het totaal aantal geautomatiseerde testen tussen de verschillende lagen. In zijn artikel geeft hij aan dat de genoemde percentages niet meer dan een ruwe schatting zijn. Toch noemt Crisp getallen. Het gebruik ervan roept vragen op over wat een acceptatietest dan precies is en hoe we de omvang van een acceptatietest meten. Deze informatie hebben we namelijk nodig om twee dingen vast te stellen. Ten eerste willen we weten dat de ene acceptatietest even groot is als de andere, zodat we ze bij elkaar op kunnen tellen. En ten tweede moeten we weten hoe bijvoorbeeld de unittesten zich verhouden tot de acceptatietesten. Als ik tien acceptatietesten heb gemaakt en honderd unittesten, heb ik dan veel of weinig unittesten in verhouding tot de acceptatietesten? Opnieuw is het niet makkelijk om hier een antwoord op te geven. Een methode voor het kwantificeren van acceptatietesten is er bij mijn weten niet. Een getalsmatige benadering van de piramide zal dus altijd op voor een project specifieke schattingen en aannames gebaseerd zijn. Cohn omzeilt deze handig door geen getallen te noemen.

Dekkingsgraad

Een andere benadering voor het kwantificeren van de testen in de piramide vinden we in het boek *The Clean Coder* van Robert Martin [Martin, 2008]. Zijn invalshoek ligt voor de hand en toch is deze niet verder gepopulariseerd. Martin geeft in zijn bekende boek aan wat de *dekkingsgraad* per laag zou moeten zijn. De unit testen hebben een

dekkingsgraad van 100% en gaandeweg naar de top van de piramide loopt de dekkingsgraad terug. Zo hebben de integratietesten een dekkingsgraad van 20%. Martin voegt dus de dimensie *dekkingsgraad* toe aan de piramide en ook deze benadering roept vragen op. De belangrijkste vraag is wat de testen dan afdekken. Voor unittesten kan nog wel een dekkingsgraad berekend worden en daarvoor hebben we ook tools. Maar wat betekent een dekkingsgraad van 20% voor de integratietesten? Worden met die integratietesten andere zaken afgedekt dan de unittesten? Dat ligt voor de hand, omdat we in de hogere lagen niet opnieuw zaken willen testen die al afgedekt zijn door 'lagere' testen. Het is aan te raden om herhaling voorkomen, maar Martin laat in het midden hoe we dit doen.



Figuur 5: Robert Martin introduceert bij elke laag de dekkingsgraad als extra dimensie

En er gebeurt nog veel meer

Een eenvoudige zoekactie in Google Images levert voor de zoekterm 'test automation pyramid' een breed scala aan piramides op. Zo zijn er varianten die non functionals zoals performance of security opnemen in de piramide. Ook wordt er een onderscheid gemaakt tussen functionele testen en technische testen, waarbij wordt gesuggereerd dat de functionele testen de bovenkant van de piramide beslaan en de technische testen de onderkant. Er wordt gestrooid met begrippen zoals *system test*, *session based testing* of *user journeys*; begrippen waaraan alleen in een specifieke context invulling gegeven kan worden. En als klap op de vuurpijl wordt de piramide ondersteboven gekeerd. Er lijkt kortom geen einde te komen aan de variaties op Cohns piramide.

Zoals ik hierboven heb beschreven is elke variatie in feite een nieuw model, met andere dimensies *en een ander doel*. De piramides lijken veel op elkaar en worden om die reden als uitwisselbaar of vergelijkbaar beschouwd. Ogenschijnlijk kleine aanpassingen worden zonder kritische beoordeling overgenomen. Maar de keuze voor het gebruik van het ene of het andere model kan alleen gemaakt worden na een uitgebreide studie van de dimensies van het model. Daarbij stellen we vragen zoals ik hierboven gesteld heb en beoordelen we of die vragen naar tevredenheid beantwoord kunnen worden.

Conclusie

De testautomatiseringpiramide is waarschijnlijk niet in elke context bruikbaar. Als er gewerkt wordt met een monolithische architectuur of als de tester uitsluitend het eindproduct kan testen door een GUI, dan is een discussie over de efficiëntie van het automatiseren van testen tegen verschillende architectuurlagen vrijwel zinloos. Maar als de mogelijkheid bestaat om testautomatisering tegen verschillende lagen uit te voeren, dan biedt Cohns piramide een eenvoudige en eenvoudig te hanteren heuristiek.

De kracht van zijn model is dat het genoeg houvast biedt voor een discussie over testautomatisering, zonder gelijk in allerlei ingewikkelde details te treden. De belangrijkste boodschap is dat we testautomatisering op kunnen

bouwen zodat we breekbaarheid en lange doorlooptijden tot een minimum beperken. En zoals we hebben gezien ontneemt elke toevoeging of aanpassing aan Cohns model een stukje van het zicht op dit oorspronkelijke doel.

Referenties

- [Cohn, 2004] Cohn, M. (2004). *Moving from test-last to test-driven*. Gepresenteerd op Scrum Gathering
- [Cohn, 2009] Cohn, M. (2009). *Succeeding with Agile* - Software Development Using Scrum. Pearson Education
- [Crisp, 2011] Crisp, J. (2011). *Automated Testing and the Test Pyramid* http://jamescrisp.org/2011/05/30/automated-testing-and-the-test-pyramid/
- [Crispin en Gregory, 2009] Crispin, L. and Gregory, J. (2009). *Agile Testing: A Practical Guide for Testers* and Agile Teams. Pearson Education
- [Martin, 2008] Martin, R. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.
- [Scott, 2011] Scott, A. (2011). *Yet another software testing pyramid* https://watirmelon.blog/2011/06/10/yet-another-software-testing-pyramid/ ←