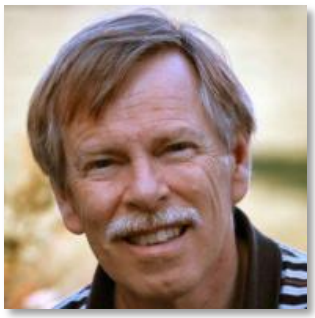
TESTNET NIEUWS

Mei 2013 • Jaargang 17 • Nummer 1 • Voorjaarsspecial

www.testnet.org secretaris@testnet.org

VAN DE REDACTIE

Door Hans van Loenhoud ● tnn@testnet.org ● @hansvanloenhoud



Een hele TestNetNieuws over testautomatisering! Ik denk altijd maar: wat is er nou zo bijzonder aan het automatiseren van een testafdeling in vergelijking met, pak 'm beet, een verkoopafdeling? Een goede businesscase maken, eisen en wensen van de gebruikers analyseren, mooi systeempje ontwerpen en bouwen (of kopen natuurlijk), hele zaakje in context testen totdat de opdrachtgever gerust is over de risico's en dan gáán met die banaan. Kortom, een gewoon, recht-toe-recht-aan automatiserings-project, zoals we dat al honderd-en-één keer hebben gedaan (en de mist in hebben zien gaan …)

Maar ik ben waarschijnlijk veel te naïef, te onnozel. Testautomatisering is blijkbaar iets heel aparts en, gelukkig, dat wordt van alle kanten belicht, haarfijn uit de doeken gedaan. Wat ikzelf, als tester, dan natuurlijk het meest interessant vind: hoe test je de testautomatisering? Hebben we daar ook een leuke beslistabel voor, of een EeVeeTeetje? En als oude rot in het vak: hoe zit dat precies met de business case? Gaan we er beter door testen, meer toegevoegde waarde leveren, of juist goedkoper testen, meer testers op straat knikkeren?

Veel vragen dus, aan het begin van deze TNN. En tegen de tijd dat je hem hebt uitgelezen, hopelijk veel antwoorden! ←

COLOFON

Redactie
Paul Beving
Hans van Loenhoud
Kees Blokland
Johan Vink
Guido Dulos
Astrid Hodes
Rob van Steenbergen
tnn@testnet.org

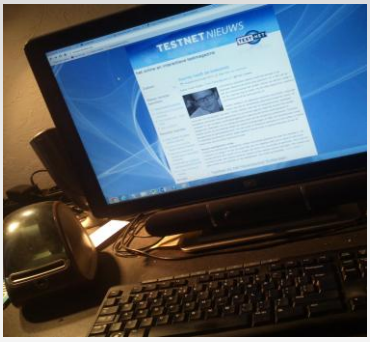
Bestuur
Michiel Vroon — *Voorzitter*
John de Goey — *Penningmeester*
Rik Marselis — *Evenementen & thema-avonden, vice-voorzitter*
Kees Blokland — *Informatievoorziening & beheer*
Bernd Beersma — *Marktverkenning & werkgroepen*
Harro Philip — *Secretaris & ledenadministratie*

Opzeggen lidmaatschap: http://www.testnet.org/algemeen/algemene-voorwaarden.html#opzeggen

In dit nummer

Nieuws.testnet.org – TestNetNieuws wekelijks online

Naast de vertrouwde publicatievorm als PDF, die de afgelopen tijd steeds vaker was gekoppeld aan het voorjaars- of najaarsevenement, is TestNet Nieuws dit voorjaar een nieuwe weg ingeslagen. De TestNetNieuws 'Weekly' verschijnt iedere week in de vorm van één artikel op de website. Daarnaast blijft deze 'Special', gewijd aan het voor- of najaarsevenement, gewoon bestaan.
Surf eens naar onze nieuwe TestNetNieuws op http://nieuws.testnet.org!



VAN DE VOORZITTER

Door Michiel Vroon ● voorzitter@testnet.org

Met deze TNN slaat TestNet een nieuwe weg in waarbij ons vertrouwde huisblad nog beter aansluit op wat binnen de vereniging plaatsvindt. Een ontwikkeling die ons bekende statement 'voor en door testers' nog beter uit de verf laat komen, helemaal gevuld met artikelen van onze leden voor onze leden over het thema testautomatisering. Nu ligt het thema mij nauw aan het hart, want eind jaren '90 was ik betrokken bij de ontwikkeling van een aanpak voor testautomatisering en daar heb ik de toolkoorts voor het eerst mogen aanschouwen. In deze roerige jaren, waar de angst voor regressie in de IT-systemen door de vele aanpassingen overheerste, werden de tools gezien als de haarlemmerolie voor alle problemen. Offshoring stond nog in de kinderschoenen en werd hoogstens toegepast voor de aanpassing van de code; het testen van de aanpassing werd natuurlijk gewoon in het lage land zelf gedaan.

En wat was er dan beter dan een tool te gebruiken die de handeling van een tester één keer afkeek en daarna tot in de eeuwigheid kon na-apen? Bij een doordachte opzet van deze schoolse gedachte zou de toekomst van de tester er donker uitzien: tools nemen het werk over! Grote bedragen werden gespendeerd aan de aanschaf, opzet en inrichting van deze hulpmiddelen tot complete testsuites en menig testadviseur nam een voorschot op het toekomstige werkeloze bestaan door nu alvast een financiële voorraad bijeen te harken. De tool werd het punt op de horizon waar naartoe werd gewerkt en alles moest daarvoor wijken.

Hoe anders zag het er een paar jaar later uit! Veel van de opgebouwde testsuites waren verworden tot coderuïnes waar niemand meer naar omkeek en iedereen omheen stapte. Dozen, dongles, handleidingen en licenties werden massaal aan de wilgen gehangen en een toolgedesillusioneerde testgeneratie ontstond. Tooling werd uit, bestempeld als toy voor de techie, speeltje van de manager en totaal niet rendabel te maken. Juist deze ontwikkeling zorgde voor een groep 'believers' die in deze tooldonkere tijd nog sterker werden in hun geloof!

Want nu we een dikke decennium verder zijn, kan de conclusie getrokken worden dat tooling weer helemaal terug is. De huidige stand van zaken in de techniek en de aanpak van IT hebben testtools een prominente en vooral ook aanwijsbare toegevoegde plek gegeven. Het is niet alleen hip om een tool te gebruiken, het is gewoon pure noodzaak geworden. De temperatuur is weer aan het stijgen. maar deze keer zal, naar ik verwacht, het koortsig niveau niet worden bereikt. Het grote verschil zit in het feit dat alles meer volwassenen is geworden: de tester, de tool en de organisatie waarbinnen ze opereren. Dit biedt de kans om een voorwaarde voor het succesvol gebruik van een tool te borgen, namelijk het besef dat een tool altijd een middel is… nooit een doel.

Heel veel plezier en wijsheid toegewenst bij het lezen van deze TTN en wellicht tot ziens op het voorjaarsevenement. ←

SILVER BULLETS AND MAGIC BULLETS

Door Alan Richardson ● alan@compendiumdev.co.uk ● @eviltester

Silver Bullets are bad. We should be wary of Silver Bullets. These are the lessons we dimly remember after reading Fred Brook's The Mythical Man Month. These are the lessons we learn from the oral history of the testers and software engineers that came before us.
But I refuse to believe that.

I love the myth of Silver Bullets. I have delighted in the notion of silver bullets since I was a child. If I packed a pistol and some silver ammunition, then I could be safe when werewolf hunting. The Lone Ranger uses Silver Bullets, thereby adding to the romanticism of The Wild West.
The Silver Bullet has a mythic resonance that I don't ever want to lose. I remain enamored with Silver Bullets.

As a child, having learned that werewolves don't exist, I started to think about the origin of the myth. Being someone who does the minimal of research, and drawing upon everything I knew from dimly remembered black and white horror films, I assumed that if I were a wealthy landowner in the 18th century and wanted to shoot peasants without repercussion, then by using silver bullets, I could claim that the peasants were actually werewolves, and I was acting in self-defense.
I imagined that an 18th century landowner could literally get away with murder. What 18th century landowner wouldn't love Silver Bullets?

My early understanding of Silver Bullets was to interpret them as an excuse, a device to justify and cover up a human failing. In the case of my early theory, a failing that consists of psychopathic tendencies, but we can generalize this to any human failing.
The danger is that we still do this. We still use Silver Bullets as an excuse for not thinking through decisions, hoping that our single decision to adopt something external, that one tool, that one process, that one metric, will make the ultimate boost to our effectiveness. ➔

But that doesn't make Silver Bullets bad. It just reminds us that we, ourselves, have a tendency to get overexcited about new toys. As a child, I learned that The Lone Ranger used Silver Bullets. How could I not love Silver Bullets? Lone Ranger books were hard to get hold of back then, so I had to rely on infrequent comic and TV appearances to educate me in the ways of The Wild West.
I don't remember a single instance of The Lone Ranger going up against a werewolf, so I suspect he was just overly superstitious.

Over the years I've come to realize that The Lone Ranger's Silver Bullets were there to remind him of the cost of a human life. Every bullet was outrageously expensive because, with every bullet fired it might deprive the Universe of a human life. Granted that doesn't communicate well when most of the people The Lone Ranger fought were actually murdering criminal scum. But the sentiment remains.
The mythic resonance of the Silver Bullet continues to remind me that people in the process are important, regardless of the tool, regardless of the process. It is the people that will make it work, it is the people that will discover how to work around the weaknesses of the tools. And yet we still seem to consider Silver Bullets as 'bad things'. It might even be, that Silver Bullets stem from an earlier mythical tradition.
Paul Ehrlich, the German immunity scientist, wanted to find Magic Bullets, cures that would target a specific germ and kill only that, leaving the rest of the organism unharmed, perhaps imagining himself as a 'Freischütz' from Germanic folklore, without the demonic overtones of course. The 'Freischütz' being someone who could shoot, seemingly without aiming, and hit whatever target he so desired - with the required folkloric caveat that the last bullet would lead to his ruin.

Again with our etymology hat on, we might posit that that the 'Freischütz' stems from someone with incredible skill. An expert, who makes it look as though no effort is involved in the shooting that no aiming takes place because they do it better than other people. And so through the jealousy of observers, and through their decision not to practice and better themselves, a myth that to be that good, you need Magic Bullets springs up. And that if you do attempt to master the skill then that mastery will lead to your ruin, so don't do it, don't aspire. ➔

We have to be careful that we don't dismiss things too early as Silver Bullets, because we don't believe they could work, when we might not be prepared to put the work in to make them happen.
We have to be wary that we label the shiny 'new thing' as a Silver Bullet, when in reality we aren't prepared to drop our preconceived notions of how to do something and put the time in to improve.
With the addition of our Magic Bullets etymological thread, our Silver Bullets become ever more mythical. Yet still, through our oral tradition, we would try to tarnish them as something impure.

I love the mythic nature of Silver Bullets. To me, they represent 'The Quest'. I hunt for an elusive state that if I can ever just reach it, then I know I will improve my skill massively. On my quest, with each adventure, with each tool I try, and each technique I master, when I realize that they are not 'The' Silver Bullet, I grow stronger and I continue my quest.
Yes, there are dangers this quest, as there are on every quest. The biggest danger is stopping early, seduced into believing you have reached its end, proclaiming that you have found 'The' Silver Bullet, when you still have a ways to go.

A good search for mythical artifacts never ends. ←

WIE TEST DE TOOLS?

Door Patrick Duisters ● patrick.duisters@improveqs.nl ● @PatrickDuisters

Testautomatisering is weer hót. Door het iteratief werken en Agile methoden neemt de behoefte aan herhaald testen toe, mede vanwege refactoring en regressietesten. Test-automatisering is ook 'programmeren' en zou dus ook getest moeten worden. Dergelijke toolvalidatie is in enkele industrieën al gebruikelijk. Wat zijn de ervaringen om dat pragmatisch aan te pakken?

Inleiding

Testautomatisering is weer hot. Net als vijftien jaar geleden, toen ik echt met testen aan de slag ging. Destijds waren we in de aanloop naar het millennium en de euro en werden er met Computer Aided Software Testing (CAST) al veel testen geautomatiseerd. Toen ging het hoofdzakelijk om het automatiseren van de fase testuitvoering. Tegenwoordig ondersteunt CAST meerdere testfasen van het testproces en bestaat uit een uitgebreide set testtooling. Testautomatisering is nu belangrijker dan ooit. Door iteratief werken en Agile methoden is de behoefte aan herhaald testen groot en neemt nog steeds toe. Geautomatiseerd testen is dan de enige optie. Bij geautomatiseerd testen moet je echter wel rekening houden met een extra risico: de kans bestaat dat je te veel op de tools gaat vertrouwen en vergeet dat een tool en de output van het tool zelf ook fouten kan bevatten.

De testframeworks, vooral voor de testuitvoering, worden gebouwd op basis van scripting. Eigenlijk is scripting natuurlijk ook 'programmeren' en zouden we deze net zo goed moeten testen als de applicaties die we er mee controleren. Mensen maken immers fouten. Dus ook in de tools en producten van testautomatisering zitten bugs.

Toolvalidatie

Testen we onze tools en testautomatiseringsproducten? Of vertrouwen we blind op de testresultaten? Natuurlijk bekijken we telkens de finale testresultaten. Waarom zouden we anders de test uitvoeren? Maar als dat bijvoorbeeld dagelijks moet, zullen we snel alleen naar de samenvatting kijken en duiken we alleen dieper in de rapportage als er bevindingen zijn. Een kritische testhouding is dus ook hier op zijn plaats.

Er zijn industrieën die strikte eisen stellen aan tooling in het algemeen. Bijvoorbeeld in het medisch domein, waar toolvalidatie erg belangrijk is.

Mijn ervaring leert echter dat toolvalidatie in de meeste industrieën veelal niet de aandacht krijgt die het verdient. Simpelweg omdat het lastig gevonden wordt en het – extra – tijd kost om uit te voeren. Toolvalidatie kost inderdaad extra tijd. Bij toolvalidatie voer je de volgende stappen uit. Om te beginnen moet je de 'intended use' van het tool vaststellen en vastleggen, bijvoorbeeld in de vorm van requirements. Vervolgens test je tegen deze requirements om te valideren of de tooling wel voldoet. Pas als dat succesvol is geweest mag je deze tools gebruiken voor het doel waarvoor ze bedoeld en gevalideerd zijn.

Pragmatisch

Uitvoeren van toolvalidatie betekent naast risicobeperking dus ook overhead. Maar kunnen we die inspanning beperken, waarbij je toch een goede check doet op de betrouwbaarheid van het gevalideerde tool? ➔

Kan je toolvalidatie 'lean' uitvoeren? Natuurlijk wel! Ik ben meerdere keren betrokken geweest bij toolvalidatie en vanuit die ervaring heb ik een aantal tips voor de aanpak van een pragmatische toolvalidatie.
Het is goed om te beginnen met een toolvalidatie-evaluatie: maak eens een inventarisatie van de aanwezige tools. Je zult versteld staan hoeveel tooltjes je gebruikt: een veelvoud van wat je verwachtte. Denk aan testmanagement, testontwerp, templates met macro's, rekensheets, bevindingenbeheer, stubs & drivers, testuitvoer, etc.
Evalueer vervolgens per tool of er testen of controles worden uitgevoerd op de testresultaten, of dat je blind vertrouwt op de resultaten die het tool genereert. In het laatste geval is het goed om het risico wat je daardoor loopt in te schatten: light weight nagaan wat de faalkans en schade kan zijn, bijvoorbeeld op basis van de vier kwadranten van PRISMA®.
Als het risico laag is, kan een exploratieve testsessie of een inspectie van de toolresultaten voldoende zijn. Als het risico hoog is, stel je uiteraard hogere eisen aan de detaillering van de requirements en de bijbehorende testen en testresultaten waarmee je je tool valideert. In alle gevallen moet je de resultaten van je validatie onderbouwen, al is het maar op één A4-tje.
Zouden we vervolgens voor het selecteren van de tools het zogenoemde 'ISTQB toolselectieproces' volgen, dan hebben we dit door de eerder genomen stappen al voor een groot gedeelte uitgevoerd en kan je volstaan met het aantonen van het geschikt zijn voor intended use (validatie, bijvoorbeeld door de tool te testen).

Ervaringen

Mijn ervaring leert dat toolvalidatie weinig prioriteit krijgt en dat de organisatie het steeds maar vooruitschuift: dat komt later wel. Maar later blijkt dat er geen requirements waren opgesteld, laat staan dat de tools getest zijn of dat scripts zijn gereviewd. Er moet dan een inhaalslag worden gedaan die eigenlijk alleen maar extra pijn doet.
Uiteraard zijn er ook voorbeelden van een meer volwassen testattitude. Meestal start een organisatie met een volwassen testattitude kleinschalig met een tool. Dit geldt vooral voor 'self made' tools. Vervolgens gaan er steeds meer mensen, teams of projecten gebruik van maken. Ze stellen eigen requirements op, specificeren de testen voor de tools en voeren ze uit. Toolvalidatie wordt dan bij voorkeur centraal ingericht zodat projecten de resultaten kunnen hergebruiken, mits de intended use hetzelfde is. Indien dat deels het geval is, kan de organisatie volstaan met een formele evaluatie van het hergebruikte deel en hoeven de tools alleen voor de verschillen in intended use gevalideerd te worden.

Conclusie

Mensen maken fouten. Ook in de ontwikkeling en toepassing van tools worden fouten gemaakt. We kunnen dan ook niet zonder meer de uit testtools verkregen testresultaten vertrouwen. Een gedegen toolselectieproces zoals in het medische domein gebruikelijk is, legt de basis voor toolvalidatie. Toolvalidatie kan echter, ook in het medische domein, risico gebaseerd en pragmatisch worden ingericht. ←

TESTAUTOMATISERING: GEEN KWESTIE VAN GELUK MAAR VAN WIJSHEID

Door Ruud Teunissen ● ruud.teunissen@polteq.com

Door onze gezamenlijke inspanning in de afgelopen decennia staan het belang en de rol van testen terecht niet meer ter discussie. Om dit vast te houden moet testen inspelen op de continue vraag naar kwaliteitsverbetering, kostenreductie, verkorten van de doorlooptijd en het verhogen van het aantal releases. Dit kan niet zonder een succesvolle implementatie en toepassing van testautomatisering. Succes mag geen kwestie van toeval of geluk meer zijn, maar van bewuste keuzes en wijsheid.

Het verleden en het heden

Als we eerlijk zijn, passen we testautomatisering tot nu toe niet overal met succes toe. Het is natuurlijk verleidelijk om uitgebreid stil te staan bij het hoe en waarom dat zo is. 'Naming and blaming' is volgens velen immers een populair tijdverdrijf van testers. Laten we onze energie echter steken in het bepalen hoe we implementatie en toepassing van testautomatisering wel succesvol kunnen aanpakken. En leren van de fouten die onmiskenbaar zijn gemaakt.

In ons enthousiasme over de schijnbaar onbeperkte mogelijkheden van de beschikbare tooling en technologie, beginnen we als 'boys met toys' met testautomatisering, zonder stil te staan bij wat we willen en kunnen bereiken. Sterker nog, het tool is vaak al geselecteerd en geïnstalleerd door relatieve buitenstaanders zoals management, inkoop, ontwikkeling, voordat een tester heeft gekeken of het tool überhaupt bijdraagt.

Het tool wordt vaak gezien als dé oplossing. Maar voor welk probleem? Belangrijke les uit het verleden en een van de essenties van succesvolle automatisering: 'Wees doelgericht, niet tool-gericht'. En dat doel is nooit testautomatisering zelf.

Overigens, tot nu toe zijn we als testers schijnbaar ongestraft weggekomen met het tekortschieten van test-automatisering. Misschien wel omdat testautomatisering door zowel testers als niet-testers tot nog toe te veel gezien is als een speeltje en te weinig als een hulpmiddel. In de huidige context is deze opstelling echter niet meer houdbaar. Softwareontwikkeling wordt meer en meer Agile, projectmanagement gaat volgens Scrum en 'continuous delivery' is steeds vaker een feit. Logisch gevolg: om in deze continu veranderende en veeleisende context goed en gedegen te blijven testen en toegevoegde waarde te blijven leveren is succesvol toepassen van testautomatisering noodzakelijk. Laten we dus bij het begin beginnen en testautomatisering behandelen zoals het hoort: het gaat om het automatiseren van het testen.

Wanneer is testautomatisering succesvol?

Het eerste dat moet gebeuren is het gezamenlijk vaststellen van de 'acceptatiecriteria'. Wanneer is testautomatisering succesvol? Welke doelen willen we bereiken? Uitermate belangrijk is hierbij dat de verwachtingen en doelstellingen realistisch zijn. ➔

Om een parallel te trekken met automatiseren van de boekhouding: na selectie en implementatie van de boekhouding is er nog steeds een boekhouder nodig. De logica, vastlegging en standaardcontroles kunnen worden geautomatiseerd, de intelligentie en interpretatie blijft mensenwerk. Met andere woorden, na het automatiseren van het testen is nog steeds een tester nodig. Gelukkig wordt dit onderschreven door een van de vele uitspraken van 'orakel' en 'goeroe' James Bach.

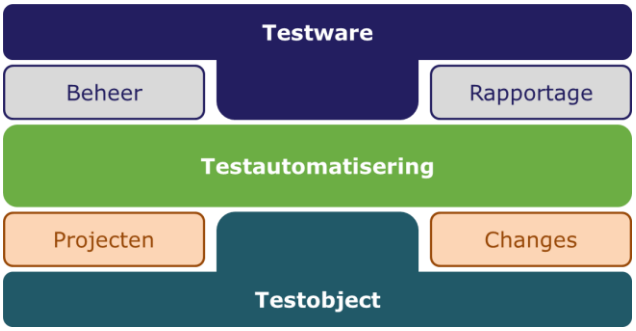
> *'I love test automation, but I rarely approach it by looking at manual tests and asking myself 'how can I make the computer do that?'. Instead, I ask myself how I can use tools to augment and improve the human testing activity. I also consider what things the computers can do without humans around, but again, that is not automating good manual tests, it is creating something new.'* - ***James Bach***

Alles automatiseren en handmatig testen overbodig maken is geen realistisch doel. Het geautomatiseerd uitvoeren van een groot deel – bijvoorbeeld 80 procent - van de regressietest wel. Net als het automatiseren van de unittest binnen een Agile aanpak of het geautomatiseerd testen van veelvoorkomende handelingen binnen een business proces, zoals het vastleggen en/of wijzigen van klantgegevens.
Een van de belangrijke kenmerken van bijvoorbeeld Agile en Scrum is dat we bij iedere stap of activiteit nadenken over de toegevoegde waarde ofwel de business case. Laten we dat ook doen bij testautomatisering. Stel dus steeds de vraag in hoeverre automatiseren van deze testen bijdraagt aan het bereiken van de afgesproken doelen. Ervaring leert dat ook hier de 80 – 20 regel van toepassing is: 80 procent van de inspanning gaat zitten in 20 procent van de automatiseerbare testen. Belangrijk dus om regelmatig vast te stellen of de inspanning van mensen en inzet van middelen in balans blijft met de opbrengst. Niet alleen op korte termijn, maar ook in de toekomst. Geautomatiseerde systemen vragen na implementatie immers om onderhoud.

Context driven

Yes, de veelgehoorde kreet 'context driven' is gelukkig ook op testautomatisering van toepassing. Succesvolle testautomatisering sluit immers nauw aan bij de gehanteerde test-, ontwikkel- en beheeraanpak en past in het systeemlandschap. Dus testautomatisering past in de context, is daarmee context driven. Maar welke aspecten van de context zijn daadwerkelijk van belang voor testautomatisering?

De verleiding is groot om te focussen op de (bestaande) testware en de te testen objecten. Hoe 'automatiseerbaar' zijn de huidige en toekomstige testgevallen? En hoe 'benaderbaar' zijn de te testen objecten voor de gekozen tooling? Ervaring heeft geleerd dat deze twee aspecten een 'gegeven' zijn. De impact op testautomatisering is dan ook veeleer dat zij in grote mate bepalend zijn voor wat bereikt – of niet bereikt – kan worden. Vergelijkbaar met de 'enablers' in veelgebruikte verbetermodellen.



Testautomatisering is hooguit een trigger voor verbetering van het testproces en de testware, het softwareproces en de software, maar zal daarbij nooit een sturende rol krijgen. ➔

In de praktijk is gebleken dat – binnen deze context – de mate waarin testautomatisering kan bijdragen vooral wordt bepaald door de architectuur voor testautomatisering, de wijze waarop de –geautomatiseerde– testscripts worden ontwikkeld en onderhouden en hoe wordt geprogrammeerd. Testautomatisering is immers automatisering. En succesvolle automatisering past binnen de context en focust op de bijdrage die het kan leveren: 'wat is nodig' staat centraal, niet 'wat kan'.

Testautomatisering moet bijdragen aan testdoelen

Laten we nog eens gaan kijken bij de boekhouder. Wat was veelal het oorspronkelijke doel van automatisering van de boekhouding? Zorgen dat de boekhouder zijn werk beter of sneller kan doen en zich kan concentreren op de kern van zijn werk: nadenken en interpreteren. Met andere woorden, automatisering helpt de boekhouder bij het behalen van zijn doelstellingen. En dit is precies wat als eerste met testautomatisering bereikt kan en moet worden. De tester kan zich concentreren op zijn belangrijkste taak: op een slimme manier testen, resultaten analyseren en inzicht geven in de kwaliteit. Automatisering neemt het routinematig werk uit handen en ondersteunt de tester. Het eerste niveau van testautomatisering is hiermee bereikt.

Testen bereikt meer met testautomatisering dan zonder

Gericht toepassen van de mogelijkheden die een gedegen architectuur en de bijbehorende tooling biedt, stelt testen in staat meer te bereiken met automatisering dan zonder. Denk hierbij niet alleen aan het uitvoeren van meer testgevallen door middel van het toepassen van data-driven technieken en het vaker draaien van geautomatiseerd testen buiten de normale werktijden van de testers, maar ook door geautomatiseerde integratie met andere tooling – zoals defectmanagement en configuratiemanagement – die gebruikt wordt binnen projecten en organisaties. Een logisch vervolg op het uit handen nemen van routinematig werk en daarmee het tweede niveau van testautomatisering.

Schaalbaar en toekomstvast

In een wereld waar Agile Development, projectmanagement volgens Scrum en 'continuous integration/delivery/deployment/improvement/…' worden toegepast en nagestreefd is het noodzakelijk testautomatisering naar het –op dit moment– hoogst mogelijke niveau te tillen. Dit betekent dat testautomatisering zelf ook de principes toepast die passen bij de ontwikkel- en beheeraanpak in de aangegeven context. Geautomatiseerde testscripts worden object georiënteerd ingericht om onderhoudsinspanning te reduceren. Testautomatisering is continu op zoek naar mogelijkheden voor verbetering. Het team werkt nauw samen met ontwikkeling om kennis, ervaringen, kunde en vaardigheden te delen. Bestaande – werkende – scripts worden 'gere-factord'. De testarchitectuur sluit nauw aan op de systeemarchitectuur en geautomatiseerd testen vindt, waar mogelijk, plaats op alle testlevels. Van unittest tot het testen van end-to-end business processen. Met andere woorden, Fit-for-purpose en Fit-for-context, en daarmee het derde niveau van testautomatisering. ➔

In de wereld van Tolkien is sprake van: 'One ring to rule them all'. In de praktijk is duidelijk gebleken dat 'One tool to rule them all' vrijwel altijd leidt tot ongewenste complicaties. Vandaar dat bewust consequent gesproken wordt over 'tooling' – een verzameling tools – en niet over één tool.

De tien pijlers van testautomatisering

Meer dan achttien jaar geleden hebben vier pijlers, fasering, organisatie, technieken en infrastructuur, de Nederlandse testwereld veroverd. Zij vormen, hoewel regelmatig in een nieuw jasje gestoken, aangepast, uitgebreid en verbeterd, de basis voor het testen binnen veel organisaties.

Het selecteren van tooling is bekend terrein, reden om hier nu niet al te lang bij stil te staan. Essentieel is wel dat breder gekeken wordt dan de normale selectiecriteria, zoals systeemvereisten, ondersteunde omgevingen, gebruikersgemak. De geselecteerde tooling moet – in lijn met de doelstellingen van testautomatisering – 'fit for purpose' en 'fit for context' zijn.

De vraag is nu: welke pijlers vormen de basis voor succesvol automatiseren van het testen? En hoeveel zijn er nodig? Zoals al eerder aangegeven wordt de kern gevormd door de gekozen architectuur voor testautomatisering, de wijze waarop de geautomatiseerde testscripts worden ontwikkeld en onderhouden en welke programmeerstandaarden gehanteerd worden. De vierde pijler is tooling. Deze vormt het platform waarop testautomatisering wordt ontwikkeld, onderhouden en draait, en bestaat uit een selectie van tools en – zelf ontwikkelde – software. Uiteraard zijn de testomgeving en de testdata cruciaal. De testomgeving en testdata moeten niet alleen beschikbaar, bereikbaar en (her)bruikbaar zijn. In lijn met het doel en de scope van testen en testautomatisering, moeten zij end-to-end testen over verschillende platforms ondersteunen. Integratie met andere tooling, zoals defect- en configuratiemanagement, is de zevende pijler. Gedegen testautomatisering staat of valt met een goed testautomatiseringsteam. Dat vormt dan ook pijler nummer acht. Samen met een strategie om te borgen dat de gestelde doelen realistisch zijn en op een transparante en efficiënte wijze worden geïmplementeerd, en een doordachte manier om alle activiteiten te plannen, begroten en bewaken, staat de teller op tien.

Key area		Contributing				Adding value				Optimizing			
1	Automation architecture	1	2	3		1	2	3	4	1	2	3	
2	Automation scripts	1	2	3	4	1	2	3	4	1	2	3	
3	Automation standards	1	2	3		1	2	3	4	1	2	3	
4	Tooling	1	2	3	4	1	2	3	4	1	2	3	4
5	Test environment	1	2	3	4	1	2	3	4	1	2	3	4
6	Test data	1	2	3		1	2	3	4	1	2	3	4
7	Tool integration	1	2	3		1	2	3	4	1	2	3	
8	Test automation team	1	2	3		1	2	3	4	1	2	3	
9	Automation strategy	1	2	3	4	1	2	3	4	1	2	3	
10	Planning & Estimation	1	2	3	4	1	2	3	4	1	2	3	
		Enable testing to achieve its goals				Mitigate more risks				Scalable, fit for purpose and context			

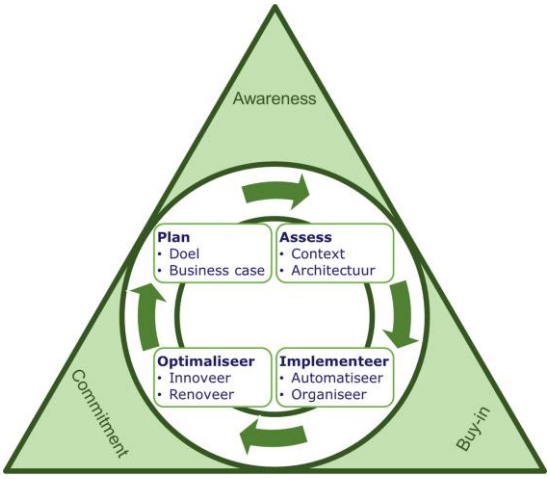
Volwassenheidsmatrix

Om testautomatisering die bijdrage te laten leveren die nodig is, is per pijler gedetailleerd en objectief aangegeven aan welke eisen moet worden voldaan om die bijdrage te leveren. Het resultaat is een volwassenheidsmatrix die het mogelijk maakt om testautomatisering van scratch af aan stap voor stap in te richten. Maar ook het naar een hoger niveau tillen van reeds bestaande testautomatisering wordt op transparante wijze ondersteund. In beide gevallen ligt de focus op de toegevoegde waarde die testautomatisering kan leveren en geldt: 'doelgericht, niet toolgericht'.

Stapsgewijze implementatie

Het vaststellen van het doel, realistisch en haalbaar, inzicht verkrijgen in de context en een analyse van het huidige en vereiste niveau, vormen de cruciale eerste stappen van succesvolle implementatie en verbetering van testautomatisering. Een goed doordachte architectuur en een pragmatisch implementatieplan zijn een logisch vervolg. ➔

Hierbij worden alle verzamelde ingrediënten samengevoegd tot een consistent geheel: test-automatisering die voldoet aan de verwachtingen en een organisatie die in staat is dit niet alleen neer te zetten, maar ook te onderhouden en optimaliseren.
In de huidige context is het uiteraard geen toeval dat de voorkeur uitgaat naar een Agile/Scrum-aanpak. Doel van de eerste sprint is het inrichten van de benodigde omgevingen, aanschaf en installatie van de benodigde tooling en het technisch 'aan de praat' krijgen van de omgeving. De architectuur, het ontwerp, wordt omgezet in een werkend testautomatiseringsplatform.



Tijdens de volgende sprint automatiseert het team een beperkt aantal representatieve scripts en voeren zij een soort 'Proof of Concept' van de gekozen oplossing uit. Aan het eind van deze sprint 'staat' het platform. De afsluitende demo geeft alle betrokkenen inzicht en vertrouwen in de gekozen oplossing. Hoeveel sprints uiteindelijk nodig zijn voor het behalen van de doelstellingen is uiteraard afhankelijk van de omvang en de context. Hierbij komen de testware en het testproces, de software en het softwareproces nadrukkelijk in beeld. Bij testware gaat het om de intrinsieke kwaliteit: mate van detail, consistentie, inzicht in wat getest wordt en gestelde prioriteiten. Bij software gaat het veeleer om de toegankelijkheid en mate waarin men zich aan standaarden heeft gehouden: input, output, herkenbare en benaderbare objecten, elementen, controls en eigenschappen. Testautomatisering is pas succesvol als het duurzaam is. Daarom is het van belang al vanaf het begin aandacht te besteden aan de aansluiting op de test-, ontwikkel- en beheeraanpak. De business blijft immers niet stil staan, de wijzigingen en projecten gaan door en testautomatisering dient hier naadloos op aan te sluiten. Dit zie je dan ook terug op diverse punten binnen de tien pijlers. Maar het is ook goed om na te denken waar testautomatisering gaat 'landen' in de organisatie. Parallel aan de implementatie moet de organisatie worden voorbereid op de nieuwe en uitdagende taak.

De toekomst

'A fool with a tool is still a fool' is en blijft waar. Maar het nadrukkelijke advies 'Structure then tool', dat Testen volgens TMap® (1995, pagina 441) gaf, is echt achterhaald. Sterker nog, ik denk dat succesvolle implementatie en toepassing van testautomatisering, inclusief de inzet van tooling, juist helpen bij het bereiken van de optimale balans tussen structuur en vrijheid van handelen die nodig is om in de toekomst als testers een essentiële bijdrage te blijven leveren. Testautomatisering met de huidige beschikbare tooling biedt ongekende mogelijkheden en het is uitdagend voor testers en ontwikkelaars. Wie weet gaan ontwikkelaars testen nu wel een leuk vak vinden. Ik neem in ieder geval de uitdaging aan om automatisering van het testen tot een succes te maken. Wie weet ga ik ontwikkeling ook wel een leuk vak vinden… ←

IMPLEMENTEREN VAN BEHAVIOUR DRIVEN DEVELOPMENT

Door Roy de Kleijn ● roy.dekleijn@polteq.com ● @TheWebTester

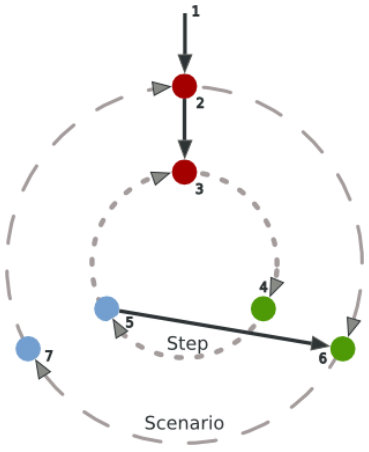
Een veel voorkomend probleem bij traditionele software ontwikkeling is dat de betrokken partijen geen gemeenschappelijke taal spreken en iedereen haar eigen jargon hanteert. Dit resulteert in verkeerd geïnterpreteerde requirements, verkeerd ontworpen, ontwikkelde en geteste producten die vervolgens – uiteraard – niet aan de verwachtingen voldoen. Een andere veel voorkomend fenomeen is dat functionaliteit en features worden ontwikkeld die niet (meteen) toegevoegde waarde leveren. Dit is zonde van de geïnvesteerde tijd en het geld. Hoe kan Behaviour Driven Development een oplossing zijn voor dit probleem? Ben je na het lezen van dit artikel echt nieuwsgierig geworden, dan kan je direct zelf aan de slag met een kant-en-klaar voorbeeld.

Behaviour Driven Development

Behaviour Driven Development is een Agile-softwareontwikkelaanpak, waarbij in korte iteraties een waarde-vermeerdering van het product wordt gerealiseerd en opgeleverd. Deze methode is gericht op het verbeteren van de communicatie tussen de betrokken partijen, zodat een gemeenschappelijk begrip ontstaat over wat gemaakt moet worden en aan welke eisen het opgeleverde product moet voldoen. Het is belangrijk dat de volgende betrokken partijen in dit proces goed samenwerken: business, ontwerpers, ontwikkelaars, testers en eventueel mensen van andere afdelingen. Een manier om tot een gemeenschappelijk begrip te komen is het organiseren van een interactieve sessie met alle betrokken partijen om gezamenlijk acceptatiecriteria te definiëren. Het is belangrijk dat alle betrokken partijen hun inbreng kunnen doen, zodat geen verrassingen ontstaan bij oplevering.

Test Driven Development maar dan anders...

Het proces van Behaviour Driven Development is vergelijkbaar met het proces van Test Driven Development, met als belangrijkste verschil dat BDD Acceptance Test Driven is en TDD Unit Test Driven is. Een kenmerk van Behaviour Driven Development is dat het een 'outside-in' aanpak is. Dat wil zeggen dat het product vanaf de buitenkant wordt ontworpen, ontwikkeld en getest. Dit in tegenstelling tot Test Driven Development ('inside-out' aanpak) waarbij het product van binnenuit wordt ontworpen, ontwikkeld en getest, door middel van unittests. Het slagen van de unittests vertelt dat het testobject op de goede manier is gemaakt, maar niet dat het juiste is gemaakt. De 'outside-in' aanpak maakt het geschikt om zowel nieuwbouw als legacy-systemen te testen, omdat je de applicatie vanaf de buitenkant benadert en je geen notie hoeft te hebben van de applicatiecode en structuur.



Op hoog niveau is het proces van Behaviour Driven Development als volgt te beschrijven. Eerst worden functionele (automatisch) uitvoerbare acceptatietests geschreven. Voor elk scenario wordt een aantal stappen doorlopen.

De binnenste cirkel stelt het implementeren van de stappen van het scenario voor, waarbij de volgende stadia worden doorlopen: falen van de stap, omdat er nog geen testobject is (**rood**); daarna wordt precies genoeg applicatiecode geschreven om de stap te laten slagen (**groen**) en het verbeteren van de geïmplementeerde code (**blauw**). ➔

Zodra alle stappen zijn geïmplementeerd val je weer terug in de buitenste cirkel waarbij de laatste stadia worden doorlopen: doordat alle stappen zijn geïmplementeerd slaagt het scenario (**groen**); en kan nogmaals worden gewerkt aan het verbeteren van de geïmplementeerde code, zodat beter onderhoudbare en robuuste, beheerbare code ontstaat (**blauw**). Dit proces wordt herhaald voor alle scenario's (acceptatiecriteria).
Met deze aanpak wordt het minimale (optimale) product gerealiseerd en opgeleverd dat voldoet aan de gestelde eisen. Het is verstandig de applicatiecode, ondanks de BDD aanpak, volgens de TDD aanpak te ontwikkelen en testen en dus unit tests te schrijven.

Stories / acceptatiecriteria

Bij het opstellen van de acceptatiecriteria blijkt onmiddellijk dat Behaviour Driven Development gericht is op de bewustwording van de waardevermeerdering die in het product wordt gerealiseerd.

Een Behaviour Driven Development story wordt beschreven met de volgende syntax:

```
In order to <receive benefit> as a <role>, I want <goal/desire>
```

Deze functionele beschrijving wordt door en/of namens vertegenwoordigers van de business geschreven. Kenmerkend is dat het beoogde doel als eerste wordt beschreven, omdat het hierom draait. Op basis van de stories worden de acceptatiecriteria, in de vorm van scenario's, door het team geschreven. Het is belangrijk dat de juiste betrokken partijen hierbij betrokken zijn, zodat alle eisen en wensen boven tafel komen en je niet vertrouwd op één persoon.

Scenario's hebben een eenduidige syntax:

```
Scenario: <titel van het scenario>
Given <pre-conditie>
When <actie>
And <eventuele vervolg actie>
Then <post-conditie>
```

Deze format heeft als voordeel dat het eenvoudig, voor alle betrokken partijen, te lezen en te begrijpen, is zodat er gericht over gediscussieerd kan worden om consensus te bereiken. Daarnaast maakt deze notatie het mogelijk om de scenario's eenvoudig door een tool te laten parsen. Waar nodig worden de scenario's voorzien van 'examples'. Dit zijn tabellen met testdata. Elke regel uit een tabel is één test en kan er als volgt uit zien:

```
Examples:
|username|password|
|testerA|passwordA|
|testerB|passwordB|
```

Uitvoerbare specificaties

Groot voordeel van Behaviour Driven Development is dat de scenario's, geschreven met behulp van de Given/When/Then stappen, aan uitvoerbare code worden gelinkt. Op deze wijze ontstaan uitvoerbare specificaties. De meeste frameworks om behaviour driven tests uit te voeren bieden de functionaliteit om de geschreven stories meteen uit te voeren, zodat een leeg skelet ontstaat en je alleen nog de daadwerkelijke testcode hoeft te maken.
➔

Living documentation

Bij iedere testuitvoer worden letterlijk de specificaties als uitgangspunt gebruikt. Iedere regel van de specificatie kleurt groen of rood naar gelang het resultaat. Op deze manier ontstaat 'Living documentation' en heb je op ieder moment de beschikking over de actuele staat van het testobject. In theorie zou het alle andere documentatie die de functionele beschrijving van een applicatie geeft kunnen vervangen.

Het beste resultaat behaal je als je Behaviour Driven Development toepast in combinatie met Continuous Integration, zodat bij iedere doorgevoerde wijziging onmiddellijk de actuele staat van het product (testobject) duidelijk wordt weergegeven. Zo weet je binnen afzienbare tijd welk effect de doorgevoerde veranderingen en toegevoegde functionaliteiten en features heeft op de rest van de producten. Het mooie is dat je met Behaviour Driven Development deze resultaten op een uniforme en begrijpelijke manier beschikbaar stelt voor de hele organisatie.

Voorbeeldproject

Op github (software versiebeheersysteem) heb ik een voorbeeldproject aangemaakt dat gebruikt kan worden om te beginnen met Behaviour Driven Development. Het voorbeeldproject kan worden gebruikt voor het testen van een website en ondersteunt parallelle testuitvoer en maakt screenshots wanneer iets niet volgens verwachting verloopt.

Vooraf te installeren

1. Eclipse IDE for Java Developers – www.eclipse.org/downloads/
   Dit is een ontwikkelomgeving waarin de tests gemaakt kunnen worden.
2. Maven - In eclipse: Help -> Eclipse Marketplace -> zoek: Maven Integration for Eclipse
   Maven maakt het mogelijk om project afhankelijkheden te configureren.
3. JBehave Plugin – volg de instructies op https://github.com/Arnauld/jbehave-eclipse-plugin
   De JBehave plugin maakt het eenvoudiger om stories te schrijven.

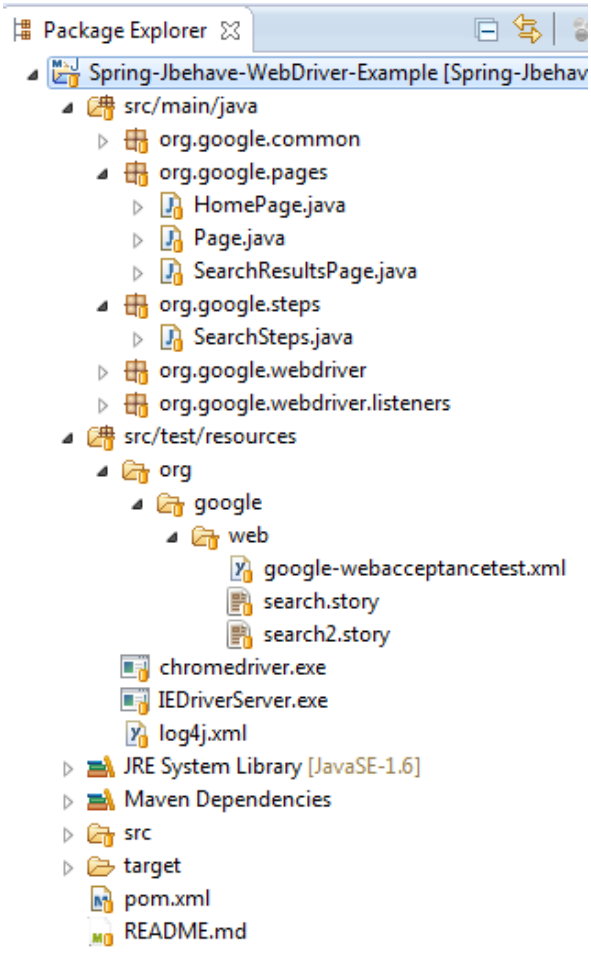
Het voorbeeldproject importeren

1. Download de broncode vanaf de volgende locatie: http://roydekleijn.github.io/Spring-Jbehave-WebDriver-Example/
2. Pak het ZIP-bestand uit naar een zelfgekozen locatie
3. (In Eclipse) Klik rechtermuisknop in het 'Package Explorer' venster en kies achtereenvolgens: Import -> Existing Maven Projects.
4. Kies als 'root Directory' de map waar de bestanden zijn opgeslagen.
5. Doorloop de wizard en klik op Finish. ➔

Structuur van het voorbeeldproject

Het project is verdeeld in drie belangrijke onderdelen:

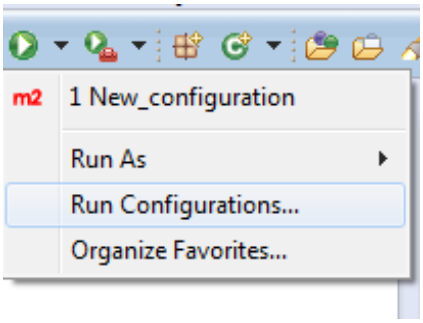
- org.google.pages – Dit package bevat de abstractie (vertaling van het testobject naar code) van de applicatie die getest wordt.
- `org.google.steps` – Dit package bevat de mapping tussen de tekstuele stappen en uitvoerbare testcode.
- `org/google/web` – Deze map bevat de tekstuele story bestanden.



Uitvoeren van de specificaties

Het uitvoeren van de stories kan als volgt:

1. Navigeer naar Run -> Run Configurations
2. Klik rechtermuisknop op Maven Build en kies New
3. Selecteer het project door op Browse Workspace te klikken.
4. Vul bij Goals het volgende command in:
   `integration-test -Dgroup=google -Dbrowser=firefox`



←

DE GEINDUSTRIALISEERDE TESTER ADEMT MODELIZATION

Door Ben Visser ● ben.visser@sogeti.nl

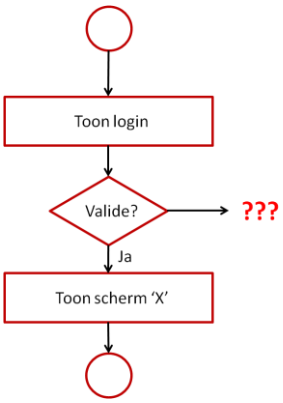
Testers kunnen niet langer voornamelijk 'ongeïndustrialiseerd' blijven werken. Ondersteuning door tools van alle testactiviteiten is noodzakelijk. Modellen vormen door hun eenduidigheid een uitstekend startpunt voor testindustrialisatie. De mindset van testers moet dan wel veranderen, modelization moet het uitgangspunt worden!
In dit artikel staat het begrip 'model' centraal. Onder model versta ik het volgende: 'elke vereenvoudigde, formele weergave van de werkelijkheid, die eenduidig uit te leggen is'. Veel voorkomende modellen zijn bijvoorbeeld stroomschema's en pseudocode. Wat wordt verstaan onder de term 'industrialisatie', wat is 'modelization' en waarom is dat zo'n groot en integraal onderdeel van de industrialisatie van testen?

De geïndustrialiseerde tester doet het liefst NIETS handmatig...

Herken je de volgende situatie? Als tester begin je aan een nieuw project. De oplevering van de functionele documentatie is weliswaar een beetje vertraagd en men verwacht nog 'een paar' aanpassingen, maar de projectleider wil toch dat je 'alvast begint'. Omdat niets doen geen reële optie is, neem je het functionele ontwerp (FO) ter hand en bent prettig verrast als je ontdekt dat het een goede basis vormt voor de Proces Cyclus Test. Hoewel de analisten geen zin hebben in een echte intake – immers het is nog niet klaar! – voer je toch een snelle review uit en passen de analisten het FO op een paar plaatsen aan. Na deze korte intake leid je met noeste arbeid enige tientallen logische testgevallen af (testmaat 2). Maar tijd om genieten heb je niet, de volgende (nog steeds niet definitieve) versie van het FO is intussen opgeleverd en eigenlijk kan je helemaal opnieuw beginnen, want er zijn midden in het proces een stuk of wat paden bij gekomen, waardoor het vrijwel niet te doen is voort te borduren op het bestaande testontwerp. En terwijl je flink bezig bent met het herschrijven van je testontwerp, komt het volgende, nog steeds niet definitieve concept FO langs... Zou het niet heerlijk zijn als we met één druk op de knop een FO op bepaalde aspecten kunnen beoordelen? En om daarna, als het FO aan de juiste criteria voldoet geautomatiseerd testgevallen te kunnen genereren!? Met andere woorden, zou het niet fijn zijn als we onze testwerkzaamheden zouden kunnen industrialiseren? Industrialiseren is het paraplubegrip waaronder standaardiseren, automatiseren en bijhouden van metrieken vallen.

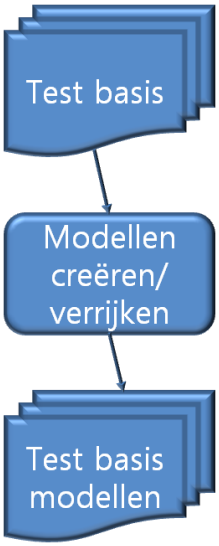
Van testbasis naar model naar geautomatiseerde testspecificatie

Ik zal aan de hand van een voorbeeld toelichten hoe we het testen kunnen (moeten!) industrialiseren.



Als testbasis voor onze test zijn een aantal use cases opgeleverd. De use cases beschrijven helder en goed hoe het proces loopt. Op diverse plekken staan alternatieve situaties beschreven en ook de foutafhandeling is opgenomen. Het is een vrij kritisch proces en sommige onderdelen moeten dan ook volgens de teststrategie met de Proces Cyclus Test testmaat-2 getest worden. De scope van de test is 'het proces van begin tot eind'. Wat het lastig maakt, is dat bij de use cases geen processtroomdiagrammen zijn opgenomen. De geïndustrialiseerde tester zorgt ervoor dat het 'van begin tot eind' stroomschema er komt: dat kan door het aan de analisten te vragen die het dan alsnog maken, maar als de analisten daar geen tijd voor hebben of krijgen, dan maakt de tester tijdens de intakefase het model zelf in een teken- dan wel procesmodelleertool. ➔

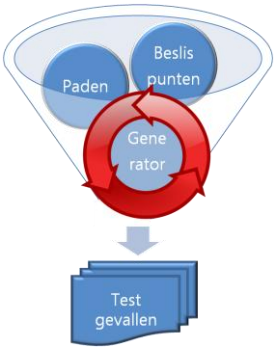
Het (grote!) voordeel hiervan is dat het maken van het model een vorm van review/intake betekent. Elke onduidelijkheid of onvolledigheid in de use cases geeft de tester expliciet in het model aan. Deze manier van werken noemen we Model Based Review (MBR).



Als het model, het stroomdiagram, gemaakt is en aangevuld met de ontbrekende informatie kan de tester met behulp van een tool op basis van dit diagram, met één druk op de knop, de testgevallen genereren.

Het tool en het model bieden je ook de mogelijkheid om als dat nodig is te variëren in testzwaartes. Zo kun je, bijvoorbeeld als je te geringe testcapaciteit hebt om de test uit te voeren met testmaat-2, in overleg met analisten en eindgebruikers de minder kritische onderdelen in het model markeren voor 'niet testen' of 'testen met testmaat-1'. Daarna kun je weer met één druk op de knop een nieuwe set testgevallen genereren die je wel met de beschikbare capaciteit kunt testen. Dit voorbeeld geeft misschien wel het grootste voordeel aan van modelizati-on: we beheren geen individuele testgevallen meer, maar modellen waaruit we testgevallen genereren.

Er zijn wel twee kanttekeningen te plaatsen.



Ten eerste: Model Based Review kost in het algemeen meer tijd dan we gewend zijn voor deze fase. Maar deze extra inspanning winnen we dubbel en dwars terug tijdens de 'testgevallen met één klik' testspecificatiefase!

Ten tweede: vanuit een stroomdiagram krijg je logische testgevallen, geen fysieke testgevallen. Om direct uitvoerbare, fysieke testgevallen te genereren zullen we het model moeten verrijken met concrete testdata. En wellicht moeten we ook aangeven welke eisen we aan de testomgeving stellen. Bijvoorbeeld eisen ten aanzien van te simuleren koppelingen en eisen aan de initiële gegevens in de testdatabases.

De testuitvoering

De industrialisering van de testuitvoering is al vaak en met wisselend succes toegepast. Een belangrijk criterium bij het opnemen van een testgeval in een geautomatiseerde testset is de afweging hoe vaak het testgeval in de toekomst 'nodig' is. Oftewel, is het een kandidaat voor een regressietestset? Ook hier kan het processchema een belangrijke ondersteunende rol spelen: welke paden moeten wel en welke niet in de regressietestset opgenomen worden? De uitdaging van testindustrialisatie ligt hem in het integreren van testspecificatie en testuitvoering op basis van uit modellen gegenereerde testgevallen.

Modelization is een kernbegrip binnen industrialisatie

Het in dit artikel geschreven voorbeeld geeft het voordeel aan van modelization, het geïndustrialiseerd gebruik van modellen tijdens testen. Modelization is het denkkader waarbij men actief streeft zoveel mogelijk te ontwikkelen, te testen en te communiceren op basis van geïntegreerde modellen. Deze zin bevat, naast de term modellen, nog een paar kernbegrippen:

Het is een denkkader. Het is geen methode of standaard, het is geen model op zichzelf. Het betreft de mindset van de tester die zich bij alles wat hij/zij doet vooraf afvraagt of handmatige activiteiten voorkomen of geminimaliseerd kunnen worden. ➔

Het is een actief streven, het gaat (nog!) niet vanzelf.
Het heeft betrekking op integratie van modellen door het gehele software voortbrengingsproces. Hiermee bedoel ik dat de verschillende disciplines in de ICT niet alleen elkaars modellen gebruiken, maar juist elkaars modellen delen en verrijken. Oftewel, verschillende disciplines werken samen aan gemeenschappelijke modellen en gebruiken modellen als ondubbelzinnig communicatiemiddel.
Verlies hierbij niet uit het oog dat modellen een middel zijn en nooit een doel op zichzelf mogen worden.

Ten slotte wil ik benadrukken dat de werkwijze in het voorbeeld niet slechts een optimistische toekomstvisie is. Diverse bedrijven kunnen dit al in meer of mindere mate! Onze uitdaging als testgemeenschap is ervoor te zorgen dat we binnenkort ook de overgebleven handmatige activiteiten kunnen industrialiseren! Het is mijn stellige overtuiging dat modelization hierbij zowel een randvoorwaarde als een katalysator is. ←

TESTAUTOMATISERING IN EEN ETL-OMGEVING

Door John Kronenberg ● John.Kronenberg@bartosz.nl ● @johnkronenberg ● Edward Crain ● Edward.crain@divetro.nl

Welke groeifasen werden doorlopen in testautomatisering bij een internationale bank voor een datawarehouseproject; van testautomatisering met batchfiles tot en met testautomatisering op basis van Specification by example met Fitnesse?

Bij het testen van onze Informatica PowerCenter applicatie testen we nu nog steeds handmatig. Kunnen we de testen niet op de een of andere manier automatiseren?' Dit vroeg een tester tijdens een projectoverleg anderhalf jaar geleden. PowerCenter is een veelgebruikte Extractie, Transformatie en Laden tool (ETL) dat gebruikt wordt voor het bouwen van bedrijfsdatawarehouses. Ons project had als doel om met behulp van Informatica PowerCenter overzichten te digitaliseren, en digitaal te distribueren.

Projectcontext

Het project, dat binnen een internationale bank plaatsvond, werkt volgens een Agile (Scrum) aanpak. Een probleem waar het ontwikkelteam tegenaan liep, was dat elke twee weken incrementeel kwalitatief goede software met nieuwe functionaliteiten opleveren, een probleem wordt als alle tests handmatig worden uitgevoerd. Na een aantal maanden zou de regressietest de volledige twee weken aan testresources in beslag nemen. Dit zou de ontwikkeling van nieuwe functionaliteiten in komende sprints in gevaar brengen.
Om zonder extra resources iedere twee weken software op te kunnen leveren waren we dus genoodzaakt om ons testproces met geautomatiseerd testen te ondersteunen.
Geautomatiseerd testen is bij Agile projecten redelijk standaard aan het worden. Deze projecten zien in dat de investering in testautomatisering resulteert in het sneller en op een reproduceerbare wijze kunnen valideren van software. Voordelen waar we voor ons project ook naar streefden.
In onze groei naar een 'volwassen' testautomatisering doorliepen we een aantal fasen. Van testautomatisering met batchfiles tot en met testautomatisering op basis van Specification By Example.

Gekozen ETL-teststrategie

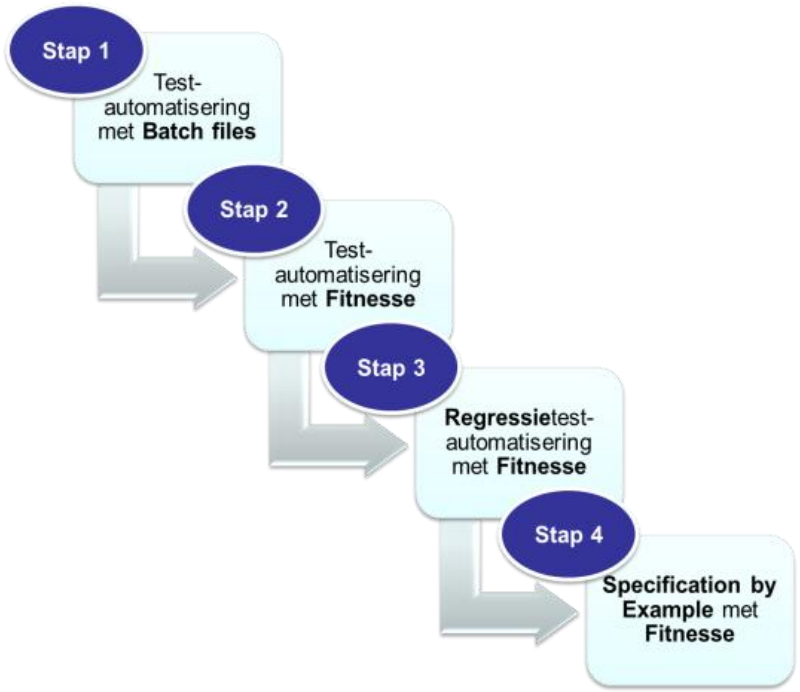
Voordat we de doorlopen groeifasen, onze roadmap, uitleggen is het belangrijk te begrijpen op welke aspecten we testten en welke teststrategie we kozen.
Bij het systeemtesten van ETL zijn de volgende aspecten belangrijk:

- Correcte transformatie van data, van source- naar targettabellen;
- Correcte lading van 'in-scope' data zonder data verlies;
- Correcte verwerking van foutieve data.

Om invulling te geven aan bovenstaande aspecten kozen we als teststrategie om voor iedere business requirement en business rule één of meerdere testgevallen te definiëren. Voor elk testgeval werd een zo klein mogelijke test-dataset gebruikt. ➔

Doordat de sourcetabellen met een relatief kleine dataset gevuld waren, werden de targettabellen ook met een relatief kleine dataset gevuld. Hierdoor was het handmatig valideren van het resultaat van een kleine set van testgevallen nog werkbaar.

Fasen van handmatig testen naar geautomatiseerd testen



De doorlopen stappen voor onze ETL testautomatisering

Stap 0 - Handmatig testen

In het begin van het project deden we al onze ETL-testen handmatig. We hadden een gevulde Excel sheet met SQL statements en de volgorde waarin de PowerCenter workflows moesten worden uitgevoerd.

We controleerden de output visueel, maar legden de outputvoorspelling niet vast. Naast het feit dat dit een arbeidsintensieve manier van testen is, is het reproduceren van een defect erg lastig.

Daarnaast kwam het in ons project regelmatig voor dat een developer de oorzaak van een gevonden defect zocht in het foutief uitgevoerd zijn van het testgeval. Het was in zo'n geval tijdrovend om dit vermoeden te ontkrachten, of te bevestigen.

In deze stap namen we het besluit om testautomatisering in ons testproces in te zetten. Achteraf gezien kun je stellen dat de hieronder beschreven stappen onze roadmap voor testautomatisering zijn geweest.

Stap 1 - Gedeeltelijke testautomatisering met batchfiles

Als eerste plaatsten we de SQL-statements, die we voorheen handmatig uitvoerden, in een tekstbestand en regelden we de automatische verwerking van dit bestand met een batchfile.

Een Oracle Architect ontwikkelde een script voor het geautomatiseerd uitvoeren van een PowerCenter workflow. In deze fase voerden we testgevallen uit door het opstarten van een batchfile. ➔

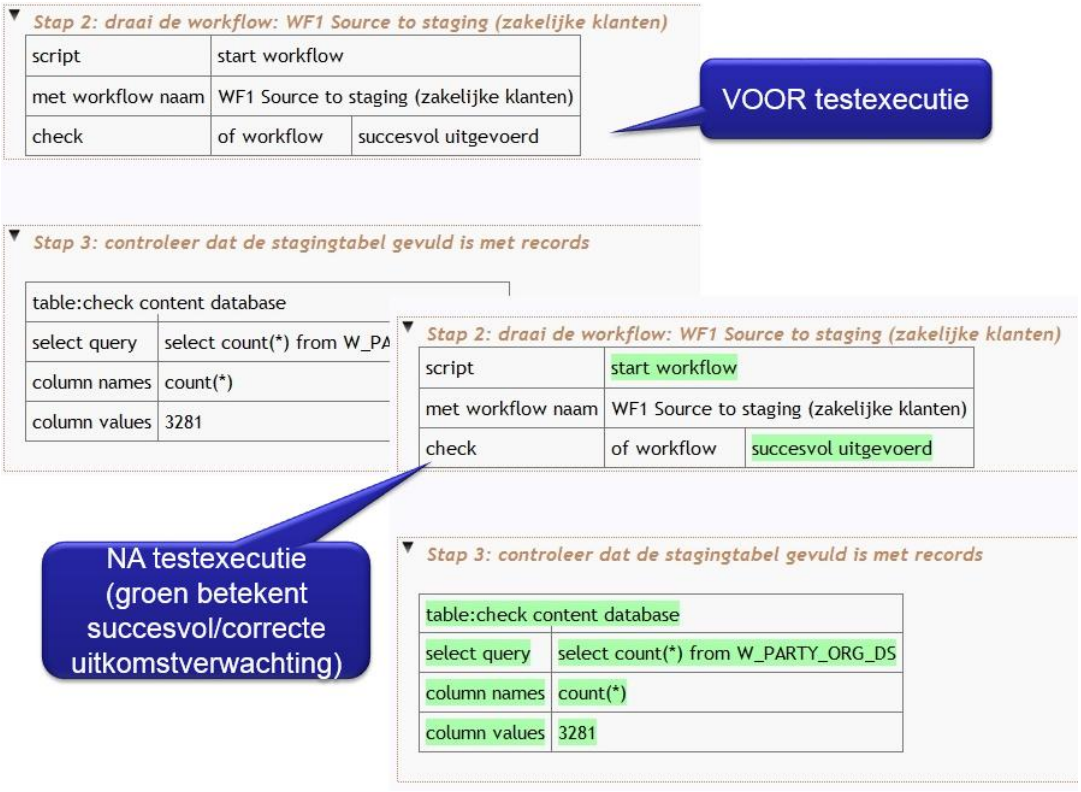
We hadden daarmee een belangrijk nadeel van handmatig testen verholpen. Defects waren reproduceerbaar. We misten alleen nog een oplossing om de testuitvoer geautomatiseerd te controleren en om op een eenvoudige wijze testsets te definiëren.

Stap 2 - Testautomatisering met Fitnesse

We onderzochten of de testtool Fitnesse geschikt was voor geautomatiseerde outputcontrole. Dit was het geval. Om Fitnesse te kunnen gebruiken moesten we wel programmeercode maken om het systeem dat getest moest worden (SUT) te koppelen. Deze programmeercode heet binnen Fitnesse een 'fixture'.

Voor ondersteuning van onze teststrategie hadden we in ieder geval de volgende fixtures nodig:

- Eén om testdata in sourcetabellen te plaatsen;
- Eén om workflows in Informatica PowerCenter te starten;
- Eén om outputverwachting met gevonden records te vergelijken;
- Eén voor het leegmaken van een database-tabel.



Voorbeeld van hoe een tabel er in de tool Fitnesse uitziet, voor en na het uitvoeren van een 'Key Example'.

Tijdens het aanmaken van de fixtures werd goed afgestemd wat de fixtures moesten doen en wat voor de tester de verwachte syntax was. De fixture voor het starten van workflows baseerde we op het script dat in een eerder stadium (stap 1) al was ontwikkeld.

Stap 3 – Regressietest automatisering met Fitnesse

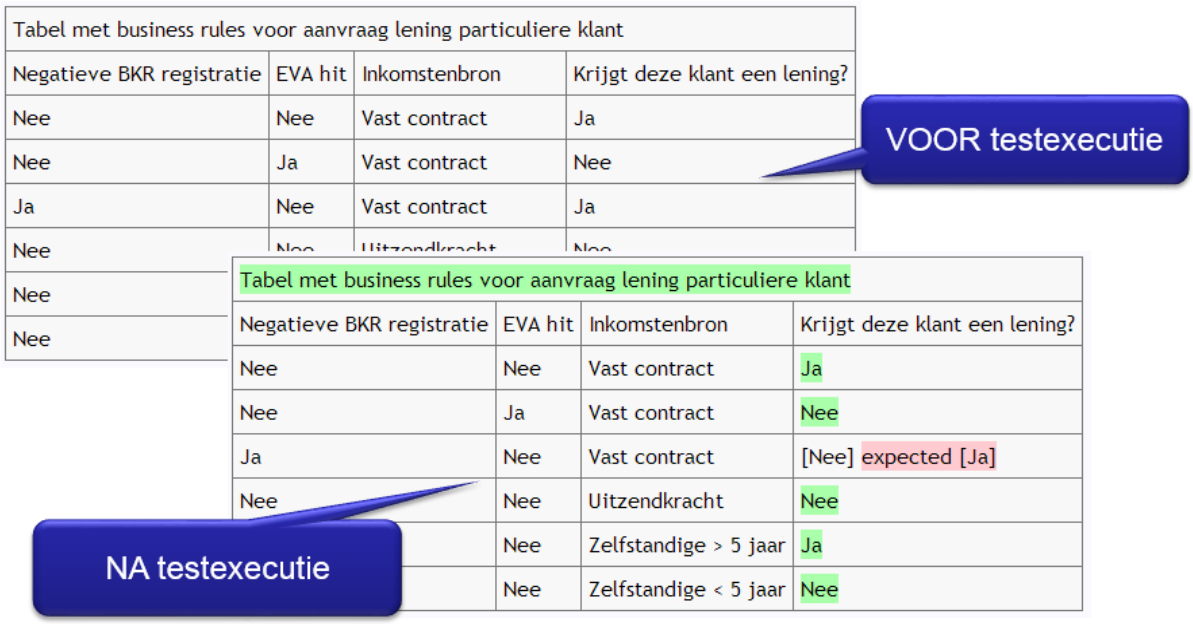
We plaatsten met terugwerkende kracht alle testgevallen van voorgaande sprints in Fitnesse. We promoveerden deze testgevallen tot regressietestgevallen. Zo creëerden we met relatief weinig inspanning een geautomatiseerde regressietestset. ➔

Een nadeel van onze inrichting van Fitnesse was dat de testen behoorlijk technisch van aard waren. Het was moeilijk voor de business stakeholders om te begrijpen hoe we de business requirements en business rules valideerden. In de volgende stap vonden we hiervoor een oplossing.

Stap 4 - Specification by Example met Fitnesse

Fitnesse is behalve een testtool ook een stand-alone wiki dat je kunt gebruiken voor de documentatie. Je kunt hier zelfs je requirements in vastleggen. Een aparte set met requirements documenten in bijvoorbeeld MS-Word wordt dan overbodig. Je hebt dan als voordeel dat de kans groter is dat de documentatie synchroon loopt met de gebouwde software. De testtool (en dus de Fitnesse wiki) moet namelijk altijd synchroon lopen met de gebouwde software. Dit zorgt ervoor dat de regressietestset in de tijd gezien altijd blijft slagen.

Als je voorbeelden gebruikt om de requirements te beschrijven (ook wel 'Key Examples' genoemd), kun je deze voorbeelden ook automatisch laten valideren door Fitnesse. De methode om requirements in voorbeelden te beschrijven wordt 'Specification by Example' genoemd.



Voorbeeld van hoe een tabel er in de tool Fitnesse uitziet, voor en na het uitvoeren van een 'Key Example'.

Door in de voorbeelden de taal te spreken van onze business stakeholders, konden we de communicatie met onze stakeholders aanzienlijk verbeteren. De business stakeholders kregen een beter begrip hoe de tests waren uitgevoerd en welke functionaliteiten waren gebouwd. Het werd daardoor gemakkelijk(er) om hierop terugkoppeling te geven.

Ervaringen tot nu toe

We zijn erg blij dat een vraag van een tester anderhalf jaar later tot een succesvolle implementatie van testautomatisering heeft geleid. Uiteindelijk is het ons gelukt om, weliswaar met vallen en opstaan, onze tests binnen onze ETL-omgeving te automatiseren met Fitnesse. Niet onbelangrijk is het vertrouwen dat we van het management en de business sponsor hebben gekregen. ➔

Het heeft geleid tot reproduceerbare defects, hogere kwaliteit en snelheid, en een betere alignment tussen de business stakeholders en ons team.
Ons succes bleef niet onopgemerkt. Ook andere projecten wilden op deze wijze gaan werken! Gelukkig waren de door ons ontwikkelde fixtures in Fitnesse flexibel en herbruikbaar. Hierdoor was het mogelijk, zonder al te veel moeite, onze Fitnesse oplossing voor de andere PowerCenter projecten binnen de desbetreffende bank te gebruiken.

Verwachting voor de toekomst

We zijn ervan overtuigd geraakt dat de combinatie van Fitnesse met Specification by Example heel krachtig is. Ook voor niet-ETL-projecten. Het heeft voor ons geleid tot een betere business-IT alignment en een toename in efficiëntie in ons ontwikkelproces. We verwachten een vergelijkbaar resultaat voor andere IT-projecten, en denken dat deze combinatie de komende tijd erg in populariteit zal stijgen. Een toekomst, waarin we ons werk vaker leuk en efficiënt kunnen uitvoeren. Wij kunnen niet wachten! ←

AUTOMATISCHE TESTTOOLS: 'HOUD HET SIMPEL!'

Door Rik Roelevink ● rik.roelevink@testwerk.nl

Automatische testtools moeilijk en duur? Dit artikel laat zien dat je 'door het gewoon te doen' met opensourcetools goede resultaten kunt behalen. Devies: Houd het simpel!

Inleiding

Graag deel ik mijn ervaring met automatische testtools met jullie. Ik ben meer dan zes jaar actief in het testvak en heb meerdere testopdrachten uitgevoerd bij verschillende klanten. Tijdens deze opdrachten heb ik kennis gemaakt met verschillende tools waaronder een aantal automatische testtools. Ik heb nog niet gewerkt met commerciële tools zoals bijvoorbeeld Quick Test Pro van HP. De tools die ik heb gebruikt zijn voornamelijk 'open source' (gratis) tools. Ik kan dus geen vergelijking maken tussen de open source en de commerciële (vaak dure) testtools Toch denk ik dat ik je kan overtuigen dat het allemaal helemaal niet zo duur hoeft te zijn en dat testautomatisering helemaal niet zo veel tijd of inspanning zal kosten.

Starten met automatische testtools

Wanneer je met automatische testtools wilt gaan werken ben je van een aantal dingen afhankelijk:

- De beschikbare tools. Sommige klanten hebben commerciële tools op de plank liggen die voor iedere tester beschikbaar zijn. Sommige klanten stellen geen tools beschikbaar. Je bent dan aangewezen op opensourcetesttools.
- Je eigen kennis en ervaring met testtools.
- De kennis en ervaring van je collega-testers met testtools.
- De applicatie die getest moet worden. De meeste automatische testtools zijn uitermate geschikt voor front-end applicaties (bijvoorbeeld websites), maar zijn lastig te gebruiken bij applicaties waarbij geen grafische gebruikers interface (GUI) aanwezig is.
- De beschikbare tijd. Als de organisatie kiest om een automatische testtool te gebruiken waar nog maar weinig kennis en ervaring mee is, zal er meer tijd gaan zitten in het opbouwen van deze kennis.

Met al deze afhankelijkheden heb ik de afgelopen jaren ervaring opgedaan. Ik ben in 2006 als tester begonnen in dienst van Testwerk. Bij mijn eerste testopdrachten zat ik voornamelijk in de laatste fase van het testtraject, de SIT/FAT en de GAT. Hierbij had ik veel te maken met gebruikers maar weinig met tooling. De enige tools die ik gebruikte, waren Excel en tools om bevindingen te loggen (bijvoorbeeld Mantis).

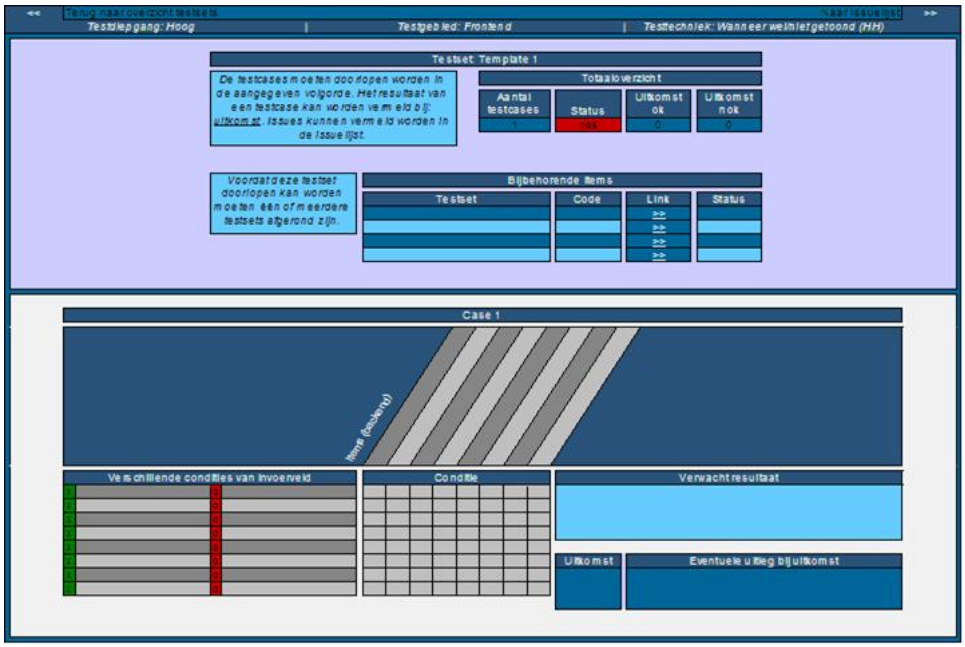
Excel als testtool

Excel lijkt een vrij basic tool, maar met al zijn beschikbare functies zijn er heel veel mogelijkheden met Excel. Zo heb ik met behulp van Excel een testset neergezet waarmee een bepaalde website door iedereen, met of zonder kennis op testgebied, getest kon worden.
Het testen zelf gebeurt hierbij nog wel handmatig, maar het tool leidt je wel door alle testgevallen heen. Excel kun je dus geen 'automatisch' testtool noemen, maar zeker wel een testtool. ➔

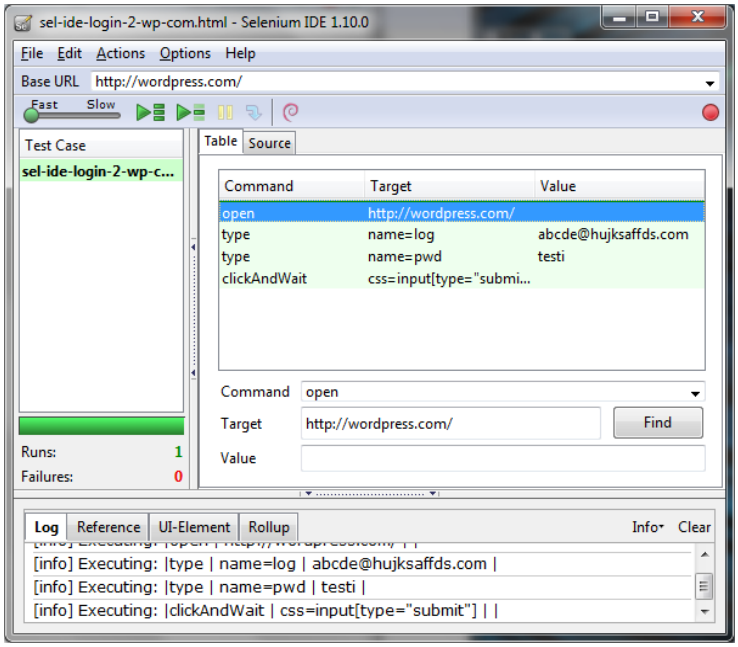
Selenium IDE en het testen van geografische kaarten

Na een aantal projecten te hebben gedaan zonder het gebruik van automatische testtools werd ik toch nieuwsgierig of het testen niet sneller en eenvoudiger kon met behulp van een automatische testtool.

Het eerste project waar ik in aanraking kwam met automatische testtools was een project waarbij een applicatie jaarlijks gedurende een kortere periode wordt gebruikt. Hierbij wordt dan wel functionaliteit gewijzigd, toegevoegd en/of verwijderd.



Deze applicatie bestaat uit een gebruikers interface die online beschikbaar wordt gesteld. In het eerste jaar dat ik bij dit project als tester werkzaam was, waren alle testgevallen in Excel aangemaakt en voerden we deze met de hand uit. Een jaar later werd ik weer bij dit project betrokken. Aangezien ik al ervaring had met de te testen applicatie en ik interesse had in automatische testtools ben ik gaan zoeken naar bruikbare tools. Er waren geen tools beschikbaar bij de klant. Dus ging ik op zoek naar opensourcetesttools. Ik kwam al snel bij Selenium IDE terecht. Selenium IDE is een add-on voor Firefox en is een 'record and playback tool'. Het werd mij al snel duidelijk dat Selenium IDE de ideale tool was voor de door mij te testen applicatie.



Met Selenium IDE kun je vrij snel aan de slag en bouw je de kennis van het tool snel op. Het is geen tool waarbij je eerst een cursus moet volgen of een theorieboek moet gaan lezen. In het begin ben je veel bezig met het bekijken en gebruiken van alle mogelijke functies binnen Selenium. Zo zal je in het begin voornamelijk gebruikmaken van de 'record and playback' functie. Ik kwam er al snel achter dat je veel sneller en overzichtelijker kan werken wanneer je de testgevallen opbouwt met behulp van bijvoorbeeld Excel of Notepad++.

Voor Selenium IDE zijn tal van extra functies beschikbaar waar je soms wel even de tijd voor moet nemen om ze te vinden op internet. Zo kan je standaard geen gebruikmaken van een if/else statement binnen Selenium IDE. Hiervoor moet je een 'flowcontrol' plugin downloaden en toepassen binnen Selenium IDE. Met behulp van deze plugin is het bijvoorbeeld mogelijk om allerlei loopjes te bouwen, zodat je bepaalde testgevallen met behulp van een klein stukje code kan laten uitvoeren. ➔

De applicatie die ik moest testen bevat naast een 'alfanumeriek' gedeelte (standaard scherm met invoervelden, dropdownlijsten, radiobuttons, etc.) ook een GIS (geografisch informatiesysteem) gedeelte. Dit GIS-gedeelte bestaat uit een kaart waarop je allerlei wijzigingen uit kan voeren. Zo kan je op de kaart bepaalde objecten intekenen, zoals de landgrenzen van een bepaald perceel. Naast de kaart is er ook een alfanumeriek gedeelte aanwezig in de GIS-schermen. Dit alfanumerieke gedeelte is afhankelijk van de gegevens die op de kaart worden ingevuld, gewijzigd en/of verwijderd. Het automatiseren van de testen voor de GIS-schermen bleek niet eenvoudig.

Uiteindelijk was de hulp van een ontwikkelaar nodig om het voor elkaar te krijgen dat ook het GIS-gedeelte getest kon worden met behulp van Selenium IDE. We voegden bepaalde functionaliteit toe die het mogelijk maakt om op de kaart te klikken en op deze manier een bepaald object te selecteren en daarna te bewerken.
Het testteam heeft uiteindelijk besloten om de testen voor het GIS-gedeelte niet uitvoerig te automatiseren aangezien dit veel tijd zou gaan kosten en de testdekking ook nooit optimaal zou worden. Wanneer je GIS-schermen met de hand gaat testen met behulp van vooraf opgestelde testgevallen, maar ook 'error guessing' en 'exploratory testing' toepast, vind je veel meer fouten dan wanneer je de testuitvoer automatiseert. De testdekking zal veel hoger liggen.

Testtool SoapUI

Naast Selenium IDE ben ik ook bezig geweest met de automatische testtool SoapUI. SoapUI is een geautomatiseerde SOAP testtool die je gebruikt voor het functioneel testen op basis van een webservice. SoapUI is net als Selenium IDE een tool die je het beste leert beheersen door ermee aan de slag te gaan. Het is een tool die je vrij eenvoudig onder de knie kunt krijgen, wanneer je niet te veel afwijkt van de standaard functionaliteit van SoapUI. Mijn besluit om deze tool te gebruiken kwam voort uit het feit dat ik twee verschillende webservices moest testen. Bij de ene webservice is het mogelijk om berichten in te schieten waarna er berichten terug komen met allerlei informatie en bij de andere webservice is het mogelijk om met behulp van bepaalde ingeschoten berichten gegevens in de database toe te voegen of te wijzigen. Ik zag geen mogelijkheid om de twee webservices handmatig te testen. Tooling was noodzakelijk.

Hulpmiddel

Een automatisch testtool is uitermate geschikt voor applicaties waarbij je veel regressietesten uitvoert. Het testtraject kost in het begin veel tijd, omdat je alle testgevallen nog moet uitdenken en aanmaken in de te gebruiken tool. Halverwege een project of soms pas tegen het einde zal de tijdsbesteding op hetzelfde niveau liggen dan wanneer je geen tools gebruikt bij de testuitvoer.
Je moet de tools wel blijven zien als een hulpmiddel, en zeker niet de behoefte hebben om alle mogelijke functionaliteit van een tool toe te passen tijdens een testtraject. Wanneer je dit doet, gaat er veel tijd zitten in het ontwerp en onderhoud van de testgevallen. Mijn conclusie is dan ook dat het gebruik van automatische testtools zeker aan te raden is maar: 'Houd het simpel!' ←

EEN KENNISMAKING MET HET TESTDOSSIERTOOL

Door Ralph Smeenk ● r.smeenk@testpeople.nl

Heb je er ook genoeg van om testgrafen en EVT's op kladblaadjes uit te werken? En van het gepuzzel om testgevallen volgens deze methodieken af te leiden? Of gebruik je zelden formele testtechnieken, omdat je ze te bewerkelijk vindt? Of ben je veel tijd kwijt met het aanpassen van een testdossier bij wijziging van de functionaliteit?
Dan is er goed nieuws, want met het Testdossiertool is een hoop van het ongemak tijdens de testspecificatie verleden tijd.

Het is inmiddels zo'n half decennium geleden dat ik met een paar ideeën en een beetje kennis van VBA besloot zelf de stoute schoenen aan te trekken om een template testdossier te creëren die de testspecificatie vergemakkelijkt. Geleidelijk aan leidde dit tot het Testdossiertool dat ik nu met gepaste trots kan presenteren aan de testwereld.

Wat is het Testdossiertool?

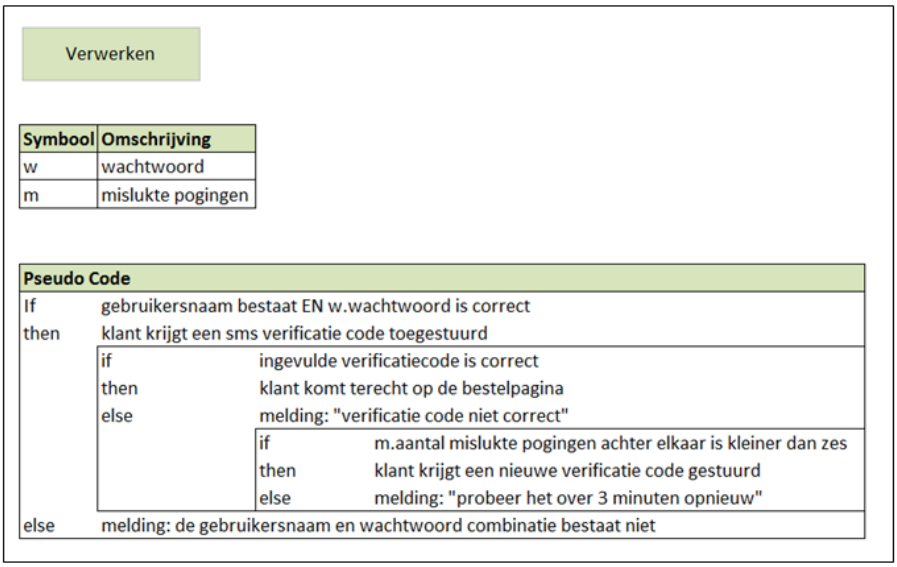
Wat is het Testdossiertool eigenlijk? Het Testdossiertool is in principe een template testdossier in Excel met verschillende macro's om de testspecificatie te vergemakkelijken. De tool bestaat uit drie hoofdniveaus:

- Het niveau van de testspecificatie;
- Het niveau van de logische testgevallen;
- Het niveau van de fysieke testgevallen.

Het niveau testspecificatie bestaat uit een EVT- en een Testgraaf-tak, waarbij de macro's helpen met de vertalingen van een hoger- naar een lager abstractieniveau. Er zijn macro's voor het afleiden van logische testgevallen volgens EVT- en de testgraaf-methodiek, een macro voor het onderhouden van je fysieke testgevallen op basis van een beslistabel en overige macro's die in het algemeen helpen met het onderhouden van het dossier.

Testspecificatie met EVT

De macro voor de elementaire vergelijkingentest (EVT) leidt logische testgevallen af volgens modified condition decision coverage. Om het te gebruiken hoef je alleen de pseudocode in een tabblad van het bestand in te vullen.
➔



Verwerken

Symbool	Omschrijving
w	wachtwoord
m	mislukte pogingen

Pseudo Code		
If	gebruikersnaam bestaat EN w.wachtwoord is correct	
then	klant krijgt een sms verificatie code toegestuurd	
	if	ingevulde verificatiecode is correct
	then	klant komt terecht op de bestelpagina
	else	melding: "verificatie code niet correct"
		if m.aantal mislukte pogingen achter elkaar is kleiner dan zes
		then klant krijgt een nieuwe verificatie code gestuurd
		else melding: "probeer het over 3 minuten opnieuw"
else	melding: de gebruikersnaam en wachtwoord combinatie bestaat niet	

Figuur1: Voorbeeld van ingevoerde pseudocode

Het aanmaken van de logische testgevallen gebeurt door te klikken op de 'Verwerken' knop. Hierdoor wordt een macro geactiveerd die de bijbehorende testgevallen wegschrijft in het EVT tabblad (zie figuur 2).

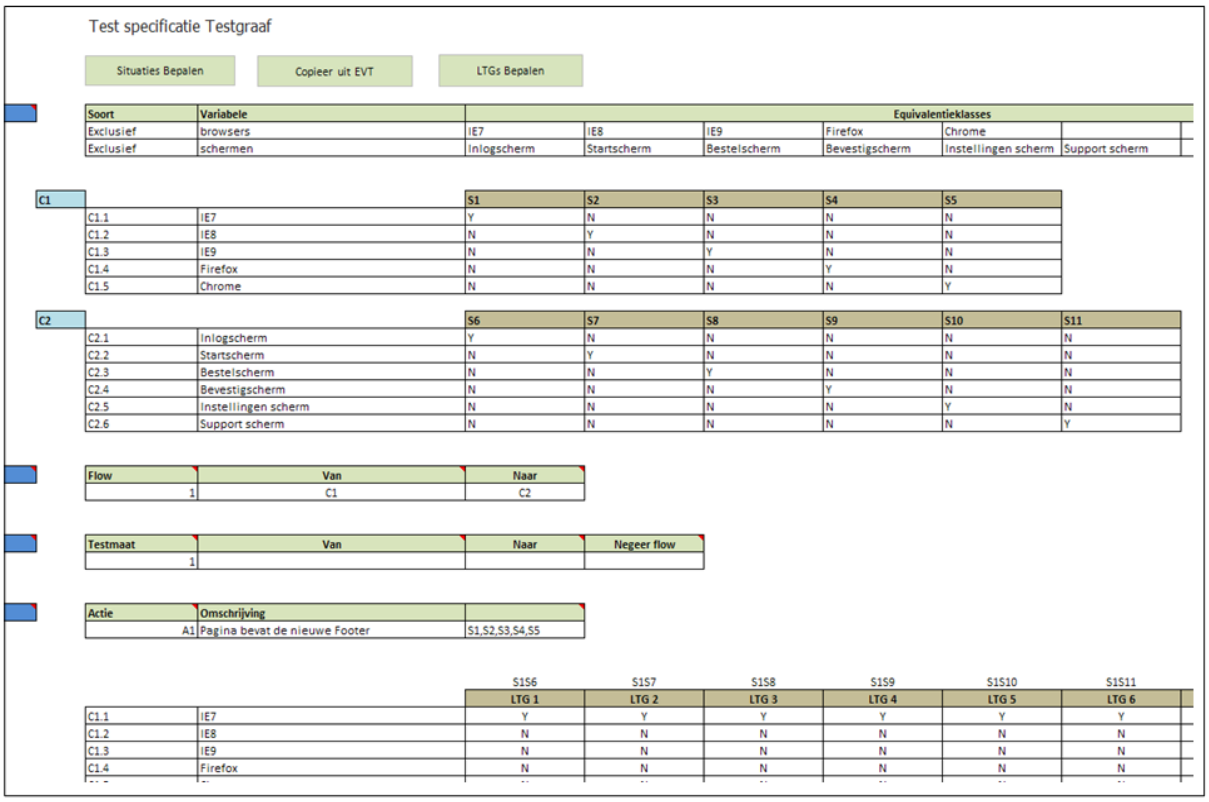
		LTG 1	LTG 2	LTG 3	LTG 4	LTG 5
C1.1	gebruikersnaam bestaat	N	Y	Y	Y	Y
C2.1	wachtwoord is correct	Y	N	Y	Y	Y
C3.1	ingevulde verificatiecode is correct			Y	N	N
C4.1	aantal mislukte pogingen achter elkaar is kleiner dan zes				Y	N
A1	klant krijgt een sms verificatie code toegestuurd			Y	Y	Y
A2	melding: de gebruikersnaam en wachtwoord combinatie bestaat niet	Y	Y			
A3	klant komt terecht op de bestelpagina			Y		
A4	melding: "verificatie code niet correct"				Y	Y
A5	klant krijgt een nieuwe verificatie code gestuurd				Y	
A6	melding: "probeer het over 3 minuten opnieuw"					Y

Figuur 2: Door macro afgeleide logische testgevallen volgens EVT

Hetzelfde EVT tabblad biedt de mogelijkheid om verschillende aspecten van het genereren van logische testgevallen te fine-tunen. Zo kan je de voor een conditie toepasselijke equivalentie-klassen nader toespitsen. En ook kan je logische testgevallen op verschillende manieren laten ordenen.
Bijzonder is dat waar klassieke EVT's bestaan uit condities met alle een Y- of N-uitkomst, dit tool kan rekenen met het gehele bereik aan equivalentieklassen in een conditie. Wat zorgt voor een meer intuïtieve specificatie. Zo verenigt het Testdossiertool dus de techniek van equivalentieklasseanalyse met de EVT.

Testspecificatie met de testgraaf-techniek

Het Testdossiertool levert niet een getekende graaf op. Maar wel kan je met het Testdossiertool de logica van een testgraaf invoeren. Op basis hiervan kan een macro voor een door jou te kiezen testmaat de logische testgevallen afleiden. Met de path coverage die dit realiseert biedt dit qua invalshoek een interessante tegenpool ten opzichte van de EVT. Net als voor de EVT is ook de testgraaf niet gebonden aan twee uitkomsten per conditie. Eigenlijk verenigt het Testdossiertool dus ook de techniek van equivalentieklasseanalyse met de testgraaf. Bijzonder aan het gebruikte algoritme is ook dat er de mogelijkheid is om de testmaat voor een specifieke subset van condities hoger te laten zijn dan voor de rest van de graaf. ➔


Test specificatie Testgraaf

Situaties Bepalen | Copieer uit EVT | LTGs Bepalen

Soort	Variabele	Equivalentieklassen					
Exclusief	browsers	IE7	IE8	IE9	Firefox	Chrome	
Exclusief	schermen	Inlogscherm	Startscherm	Bestelscherm	Bevestigscherm	Instellingen scherm	Support scherm

C1		S1	S2	S3	S4	S5
C1.1	IE7	Y	N	N	N	N
C1.2	IE8	N	Y	N	N	N
C1.3	IE9	N	N	Y	N	N
C1.4	Firefox	N	N	N	Y	N
C1.5	Chrome	N	N	N	N	Y

C2		S6	S7	S8	S9	S10	S11
C2.1	Inlogscherm	Y	N	N	N	N	N
C2.2	Startscherm	N	Y	N	N	N	N
C2.3	Bestelscherm	N	N	Y	N	N	N
C2.4	Bevestigscherm	N	N	N	Y	N	N
C2.5	Instellingen scherm	N	N	N	N	Y	N
C2.6	Support scherm	N	N	N	N	N	Y

Flow	Van	Naar
1	C1	C2

Testmaat	Van	Naar	Negeer flow
1			

Actie	Omschrijving	
A1	Pagina bevat de nieuwe Footer	S1,S2,S3,S4,S5

		S1S6	S1S7	S1S8	S1S9	S1S10	S1S11
		LTG 1	LTG 2	LTG 3	LTG 4	LTG 5	LTG 6
C1.1	IE7	Y	Y	Y	Y	Y	Y
C1.2	IE8	N	N	N	N	N	N
C1.3	IE9	N	N	N	N	N	N
C1.4	Firefox	N	N	N	N	N	N

Figuur 3 Impressie van testgraaf-specificatie

Van logisch naar fysiek testgeval

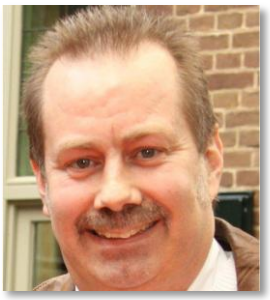
Het uitschrijven van fysieke testgevallen kan met behulp van vier extra kolommen die aan de logische beslistabel worden toegevoegd. Per regel in de beslistabel zijn er nu velden waarin je kunt beschrijven wat er moet gebeuren om een Y- of N-uitkomst van de regel te realiseren. Ook zijn er velden waarin je kunt invullen wat de verwachting is bij de betreffende Y- of N-uitkomst. Een macro schrijft uiteindelijk naar aanleiding van de ingevulde beslistabel de fysieke testgevallen weg in een apart tabblad. Een voordeel van deze methode is dat je dezelfde teksten voor verschillende fysieke testgevallen maar eenmalig hoeft in te voeren. Daarnaast zijn de teksten van de fysieke testgevallen op deze manier gemakkelijker te onderhouden. Diverse macro's helpen met het beheren van de beslistabel. Deze zijn er bijvoorbeeld voor het aanpassen van volgordes, voor sortering, en voor het toevoegen van precondities.

Conclusie

Al met al heeft het Testdossiertool een aantal handige functies die je als tester begeleiden bij de testspecificatie. De tool maakt het de tester gemakkelijker om aanpassingen door te voeren in de testdocumentatie, wat best prettig kan zijn in de stormachtige wereld van Agile, waar we leren om veranderingen te omarmen. Daarbij zorgen de beschikbare testtechnieken dat je redelijk snel een efficiënte set testgevallen met goede testdekking kunt realiseren. De verschillende abstractielagen die het testdossier hanteert kunnen helpen met de communicatie in een ontwikkelteam. Het afstemmen van pseudocode, equivalentieklassen of afgeleide logische testgevallen met andere teamleden kan nuttig zijn om de neuzen dezelfde kant op te krijgen. Al met al een tool waar ik zelf in het dagelijkse testen veel aan heb. Misschien jij ook? Ik zou zeggen probeer het uit! De tool is als open source beschikbaar gesteld en kan gedownload worden op het volgende adres:
https://github.com/RalphSmeenk/Testdossiertool ←

TESTAUTOMATISERING: WAARDEVOL OF VERKWISTING VAN 'SILVER BULLETS'?

Door Heini Veneberg ● h.veneberg@cimsolutions.nl



In alle dagelijkse activiteiten spelen de termen waardevol en verkwisting een belangrijke rol. Dat geldt ook voor testautomatisering. Hoe zorg je ervoor dat testautomatisering als waardevol wordt gezien?

Subjectief

Thuis zeg ik wel eens tegen mijn dochter: 'Draai die kraan dicht. Je verkwist schoon water'. Toen ik afgelopen zomer geruime tijd de sproeier op de tuin had staan, confronteerde zij me met de vraag: 'Is dit geen verkwisting van water?'. 'Nee', was mijn antwoord, 'want als ik de tuin niet sproei gaan de planten dood'. Waardevol of verkwisting? Wat waardevol is en wat verkwisting kan door mensen heel verschillend ervaren worden.

Wat betekenen de termen waardevol en verkwisting eigenlijk?

- Waardevol: kostbaar, belangrijk, bruikbaar, degelijk, nuttig;
- Verkwisting: verspilling, iets wat nutteloos is (Engels: waste), vergooien (van iets waardevols).

Testautomatisering, het ontstaan van barstjes in het imago

In het algemeen durf ik te stellen dat iedereen begint met te zeggen dat testautomatisering waardevol is. Toch loopt testautomatisering nog regelmatig 'vast'. Soms al voordat er ook maar iets getest is. Waar gaat het dan mis? Testautomatisering is natuurlijk in beginsel niets anders dan het geautomatiseerd controleren van een te testen product (het testobject). Voordeel is dat zo'n traject een onbeperkt aantal keer aangeroepen kan worden. U vraagt wij draaien!

Bij dat draaien begint het probleem, het loopt ergens stuk of het geeft op onverwachte momenten een `fout` aan. Dit moet onderzocht worden door leden van het testteam en soms moeten daarbij meerdere disciplines aanschuiven om de fouten te analyseren. Als er fouten uitkomen, die terug te leiden zijn naar het testobject, dan wordt het als waardevol gezien. Helaas is een test automatisering bouwwerk zelf ook een product. Kenmerk van een product is, dat het fouten kan bevatten. Komt uit de analyse van het stuklopen, dat de automaat zelf fout zit, dan ontstaan er klachten: 'We worden telkens benaderd voor een analyse van een probleem en dan blijkt de fout in het testbouwwerk te zitten'. Dan beginnen er barstjes in het beeld waardevol te komen. Na een aantal incidenten wordt testautomatisering als verkwisting ervaren. Herkenbaar? ➔

Testautomatisering, tips voor positief verwachtingsmanagement

Een belangrijk gegeven is dat binnen een organisatie al snel geroepen wordt: 'Dat automatiseer je toch even!?'. Met andere woorden, men begrijpt niet dat het creëren van een testautomatiseringbouwwerk ook een ontwikkeltraject is.

Mijn ervaring heeft me geleerd de verwachtingen rondom testautomatisering goed te managen. Zo kun je testautomatisering als waardevol laten ervaren. Bij dat managen moet je rekening houden met wat je moet testen en hoe je dat geautomatiseerd kunt controleren.

Voordat je geautomatiseerd gaat testen is het goed om een fase van handmatig testen in te bouwen. Tijdens het handmatig testen wordt een hele belangrijke factor gebruikt. Dit is namelijk de mens `Tester`. De tester is een heel kostbaar instrument, één die heel veel waarde kan toevoegen aan een testtraject. Waarom? Heel simpel: de tester voegt met zijn intelligentie en zintuigen veel toe in de voorbereiding en uitvoering van een testtraject. Een automaat doet wat je vraagt en kijkt niet links/rechts en ziet zeker niets vanuit zijn ooghoeken.

Dit geeft al aan dat handmatig testen een hele belangrijke fase is. Deze fase biedt belangrijke input aan het opzetten van een automatisch testtraject. In deze fase kun je onderzoeken hoe stabiel het ontwikkeltraject gaat verlopen. Zijn de requirements stabiel genoeg? Of in het geval van User Interfaces: is het ontwerp daarvan stabiel, zijn er 'onder water' vaste elementen/identifiers opgenomen die bij het automatisch testen gebruikt kunnen worden om op te controleren? Te vaak zie je dat het ontwikkeltraject al gestart is en dat er te veel belangrijke parameters nog in beweging zijn, laat staan dat er bij het ontwikkelen rekening wordt gehouden met de testbaarheid van het product door een testautomaat. Het project moet een bepaalde volwassenheidsgraad krijgen. Als je in deze `puberale` fase begint met automatiseren, neemt de kans op verkwisting toe. Men ziet het team alleen maar bezig met reageren op de verandering en men mist een geautomatiseerd testresultaat.

Tijdens het opzetten hoor je dan ook vaak `hoe kunnen we deze input/output in de automaat controleren dan wel aansturen?' Meestal ligt het antwoord 'kijk hoe er handmatig gecontroleerd wordt', dan heb je al grotendeels de oplossing voor de automaat te pakken. Dan hoeft testautomatisering alleen nog maar gerealiseerd te worden. Mijn advies is begin met een pilot als er nog niets is en bouw dit uit terwijl je de pilot vertraagd laat meelopen met het handmatig testen. Wil je dit meetbaar en volgbaar hebben, dan is een projectaanpak voor automatisch testen wenselijk. Behandel het als een project, dan kun je voortgang en status rapporteren. In de 'Business Case' voor automatisch testen moet duidelijk zijn waarom testautomatisering 'waardevol' is en geen 'verkwisting'. Dit geeft mogelijkheid om verwachtingen te managen. Evalueer tijdens het project regelmatig de geldigheid van de Business Case.

Silver bullet

Als afsluiting van dit verhaal zou ik er nog één ding aan willen toevoegen en dat is Public Relation (PR). Met andere woorden, laat regelmatig zien wat de automaat wel en niet kan. Welke handmatige testen overgenomen zijn door de automaat en welke niet. Middels deze PR kun je ook aangeven wanneer je de automaat laat draaien en dus een nuttige inzet van het machinepark is. Mijn opinie is dat testautomatisering een waardevolle silver bullet is, maar door de omgeving beleefd kan worden als een verkwisting. Het gevoel van verkwisting kun je voorkomen door testautomatisering als een serieus subproject te behandelen. ←

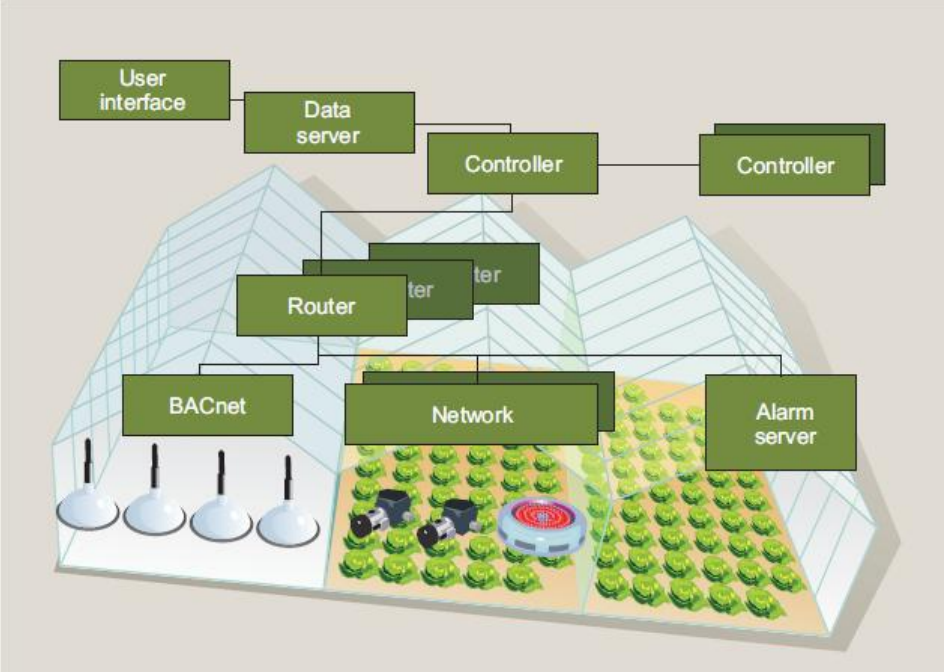
GROWING PAINS

Door Simon de Boer ● Simon.de.Boer@priva.nl en Robin Mackaij

Bij Priva Horticulture testen Simon de Boer en Robin Mackaij klimaat-beheersingssystemen voor de glastuinbouw. Omdat de marktwaarde van een kas vol planten in de miljoenen euro's kan lopen, zal het falen van zo'n systeem grote schade veroorzaken. Maar kun je dat grondig testen? De uitdaging is gelegen in het scheppen van realistisch extern gedrag: zon, wind en regen. Die heb je niet in de hand, dus om te testen zul je je toevlucht moeten nemen tot simulatie.

In hun artikel Growing Pains in de Professional Tester van december 2012 doen Simon en Robin een boeiend verslag van hun ervaringen. Hun boodschap: *A closed control loop requires a closed testing loop.*

http://www.professionaltester.com/magazine/backissue/PT018/ProfessionalTester-December2012-deBoer-and-Mackaij.pdf ←



Componenten van een klimaatbeheersingssysteem

WAT KUN JE LEREN VAN EEN STEDENTRIP NAAR FLORENCE OVER MBT?

Door Bert Zuurke ● bert.zuurke@cgi.com

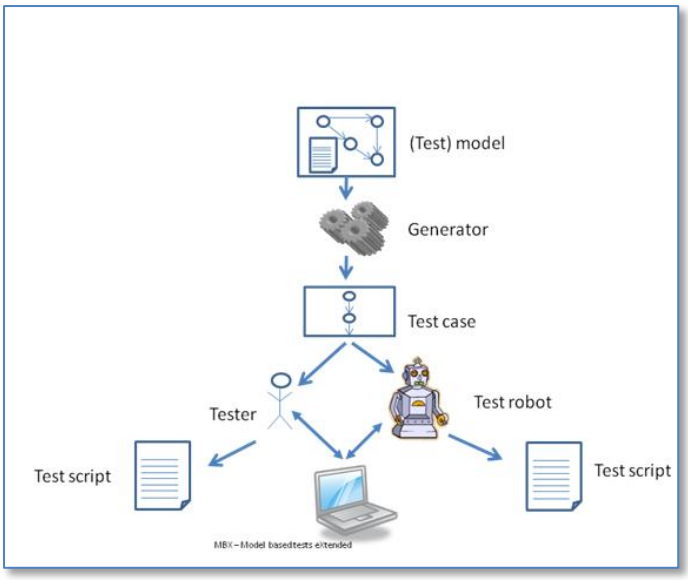
Hoe kun je een stedentrip naar Florence gebruiken als een metafoor voor het maken van een toestandsdiagram? Zo begrijp je alles van toestanden overgangen, guards, toestandsvariabelen, hoofd- en submodellen.

Een metafoor of analogie is een hulpmiddel, dat vaak tijdens brainstormsessies wordt gebruikt om het creatieve proces te ondersteunen. Door de metafoor ontstaat een andere kijk op het brainstorm domein, waardoor nieuwe ideeën en inzichten ontstaan. Ik heb gemerkt dat een stedentrip als metafoor kan helpen bij het opzetten en doorgronden van toestandsdiagrammen. Deze toestandsdiagrammen gebruik ik voor het modelleren van applicaties. En deze modellen gebruik ik voor genereren van testscripts. Bij model based testen kun je met zo'n metafoor stap voor stap de analogie met toestandsdiagrammen tonen.

Model based testen

Model based testen(MBT) is een testspecificatiemethode waarbij de testgevallen uit modellen worden afgeleid. Deze modellen kunnen zijn opgesteld door de ontwerpers – bijvoorbeeld in UML – maar het kunnen ook modellen zijn die speciaal voor de test zijn opgesteld door testers.

Er zijn verschillende uitgangspunten voor het maken van modellen. Het model kan het gedrag van het te testen systeem (System Under Test, SUT) weergeven, maar het model kan ook het gedrag van de 'tester' modelleren of het business proces waar de applicatie in wordt gebruikt.



Het afleiden van de testgevallen uit de modellen kan met de hand worden uitgevoerd, maar bij MBT gaat het meestal over het automatisch genereren van testgevallen. Sommige tools genereren alleen logische testgevallen, andere tools kunnen ook fysieke testgevallen, dus met testdata, genereren.

Het voordeel van het genereren van de testgevallen is dat het werk bespaart van de testanalist (tot wel 75 procent). Bij wijzigingen in het model hoeven niet alle testgevallen handmatig te worden aangepast.

Tot slot wordt vaak ook het eerder ontdekken van fouten als voordeel van MBT genoemd, omdat men al in een vroeg stadium van het ontwikkelproces met testmodellering moet beginnen en zo ontwerpfouten al vroeg opspoort. Ondanks dat het al jaren wordt genoemd als veelbelovende testtechniek, is het in de praktijk nog steeds niet uitgemaakt of MBT een 'Silver bullet of testing' is of niet. ➔

Het proces voor MBT bestaat uit de volgende stappen:

- Maak een model;
- Genereer testgevallen;
- Voer de test handmatig uit
  - Online;
  - Offline;
- Run de test automatisch.

Onder offline wordt verstaan dat de testgevallen na het genereren worden uitgevoerd. Onder online wordt verstaan dat de testgevallen tijdens het genereren direct worden uitgevoerd. Automatisch wil zeggen dat de testen door een testrobot worden uitgevoerd.

Een goed model maken dat het gedrag van het testobject goed voorspeld, is dus essentieel en dat wordt vaak als lastig ervaren. In het volgende laat ik zien dat het ook anders kan: leuker en makkelijker. Daarvoor gebruik ik de stedentrip naar Florence als hulpmiddel. Voor degenen die Florence niet kennen, geef ik eerst even een kleine rondleiding. Als je Florence al kent, kun je dit overslaan en verder lezen bij: 'Ik ga nu ...'.

Stedentrip als metafoor

Ga je mee naar Florence? Wanneer ik naar Florence ga – of een willekeurige andere reis – dan maak ik meestal een lijstje van de bezienswaardigheden die ik wil bezoeken. Mijn lijstje voor Florence zag er de eerste keer zo uit:

		Top 10 Florence
1	☐	Duomo
2	☐	Galleria degli Uffizi
3	☐	Piazza della Signoria
4	☐	Galleria dell'Accademia
5	☐	Ponte Vecchio
6	☐	Santa Croce
7	☐	Palazzo Vecchio
8	☐	Giardino di Boboli
9	☐	Battistero
10	☐	Museo Nationale del Bargello

Vervolgens zoek ik op de kaart op waar deze attracties zich bevinden en bedenk ik hoe ik van de één naar de andere kan gaan. Bijvoorbeeld:

We beginnen met de beroemde kathedraal de Duomo op het Piazza del Duomo. Als het mooi weer is, wil ik de koepel beklimmen van de Duomo, gemaakt door Brunelleschi, en genieten van het prachtige uitzicht over de stad. Voor de Duomo staat het Battistero en dat wil ik ook meteen bekijken. Op dit plein is ook het Museo Dell'opera del Duomo. Maar dat bewaar ik voor een andere keer. Vanaf dit plein kan ik naar de beroemde Ponte Vecchio gaan lopen. Er zijn natuurlijk verschillende routes mogelijk. ➔

Ik kies voor de route over de Via Del Calzaiuoli. Ik kom dan eerst op het Piazza Della Signoria, waar links het beroemde beeld van 'David' staat, gemaakt door de even beroemde Michelangelo (dit is slechts een kopie, het echte staat in de Galleria dell'Accademia, waar ik later nog naar toe kan gaan). Rechts achteraan op het plein bevindt zich het Palazzo Vecchio. Dit is het oude stadhuis van Florence met een imponerende ontvangsthal. In deze hal zijn metershoge fresco's te zien van Vasari (deze fresco's waren kortgeleden nog in het nieuws omdat men vermoedt dat erachter nog beschilderingen van Leonardo da Vinci te zien zouden zijn). Rechts naast het Palazzo Vecchio bevindt zich het Degli Uffizi, het stadskantoor. Hierin is nu een groot schilderijenmuseum gevestigd. Ik kan nu via de Piazale degli Uffizi tussen de twee vleugels van het Uffizi doorlopen naar de rivier Arno en dan zie ik rechts voor mij de Ponte Vecchio over de Arno. Deze brug is beroemd omdat het nog steeds bebouwd is met huisjes. In deze huisjes zijn voornamelijk juweliers en goudsmeden gevestigd en daar kun je de alleraardigste dingen kopen. Over deze huisjes loopt een galerij, die vanaf het Uffizi museum over de Ponte Vecchio naar het Palazzo Pitti aan de overkant van de Arno loopt. Deze galerij heet de Vasari Corridor, naar de bouwer ervan. Het was gemaakt voor de heersers van Florence – de Medici – om ongehinderd van het ene paleis naar het andere te kunnen gaan. Deze galerij kun je bezoeken als je wilt. Je moet dan wel een speciale rondleiding boeken, die begint in het Uffizi museum en loopt naar Palazzo Pitti aan de overzijde van de rivier. Tot zover deze kleine toeristische wandeling door Florence.

Ik ga nu in een paar stappen van de toeristische route naar een model van diezelfde route:

- Ik begin met het tekenen van de route op een kaart;
- Vervolgens benadruk ik de interessante plaatsen waar ik een tijdje blijf om rond te kijken en de verbindingswegen;
- En tot slotte teken ik de interessante plaatsen als blokjes en de verbindingswegen als pijlen.

Wat dan overblijft, is een aantal blokjes, die verbonden zijn met pijlen. Elke pijl heeft een begin en een einde.

Een andere manier is door een lijst te maken en deze dan te tekenen met blokjes voor de plaatsen die we willen bezoeken en met pijlen, die aangeven hoe je van de ene plaats naar de andere wilt gaan:

- Start;
- Ga naar het Piazza Del Duomo in Florence;
- Bekijk het Battistero;
- Ga dan naar de Duomo (Santa Maria del Fiore);
- Bekijk de kathedraal of beklim de koepel;
- Loop via de Via Del Calzaiuoli naar het Piazza della Signoria;
- Loop vanaf het Piazza della Signoria tussen de Uffizi vleugels door naar de Arno en sla rechts af richting de Ponte Vecchio;
- Geniet van de leuke winkeltjes op de brug.

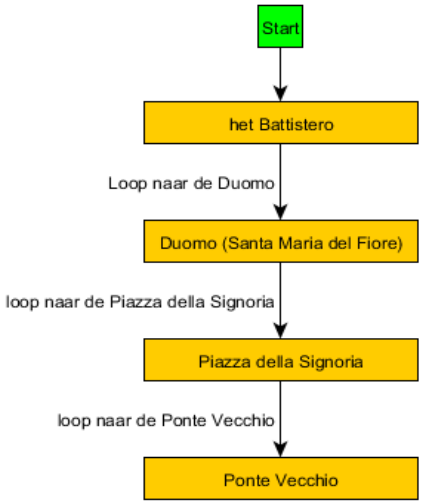
De route kan ook als volgt worden weergegeven:

- Ga van de Duomo door de Via Del Calzaiuoli naar het Piazza della Signoria en dan door naar de Ponte Vecchio.

In deze laatste vorm is het verschil tussen route en interessante plaatsen die je moet hebben gezien niet erg duidelijk. Op grond van andere informatie zul je die keuze moeten maken. Bij het modelleren voor een applicatie kom je ook dit veel tegen. Je moet kiezen wat je als weg ziet en wat als interessante plaats. ➔

Het is nu nog maar een stap naar een toestandsmodel. We hoeven de plaatsen die we bezoeken alleen nog maar toestanden te noemen: plaatsen waar je verblijft. De looproutes tussen die plaatsen noemen we overgangen of transities. We hebben nu de elementen voor een toestandsdiagram en zijn bijna ongemerkt in een toestandsdiagram terechtgekomen (State Transition Diagram).

Voor toestandsdiagrammen is het gebruikelijk om de toestanden weer te geven met een blokje, en de overgangen met een pijl. Het spreekt vanzelf dat we de terugweg ook kunnen aangeven. Ik gebruik daarvoor bij voorkeur extra pijlen. Uit het vervolg zal duidelijk worden waarom dat beter is. Verder is het gewenst om het model een startpunt te geven. Ik modelleer dat altijd met een Startblokje. De route naar de Ponte Vecchio teken je dan op deze manier als toestandsdiagram.



Het is onze keuze of we het plein dat we passeren van de Duomo naar de Ponte Vecchio beschouwen als een toestand of dat we het ook zien als onderdeel van de transitie en niet als aparte toestand opnemen. De reden waarom ik het als toestand opneem, is dat er vanaf deze locatie verschillende interessante plaatsen te bezoeken zijn en natuurlijk ook omdat rondom het plein een heleboel restaurants ons uitnodigen iets te nuttigen. Dit illustreert ook dat we als modelontwerpers zelf kunnen kiezen of we iets een toestand noemen of als onderdeel van een transitie beschouwen of helemaal niet opnemen. Het illustreert ook dat we als ontwerpers details later kunnen toevoegen of weer verwijderen. Of anders gezegd, we kunnen transities gewoon onderbreken met een extra toestand, of andersom. Daarmee kunnen we dus meer accenten toevoegen of ons juist beperken tot de belangrijkste toestanden. Hetzelfde deden we ook al toen we van de kaart met veel details overgingen op een blokjes- en pijltjesschema.

Misschien een wat lange inleiding maar als je het eenmaal door hebt volgen de andere uitbreidingen als vanzelf.

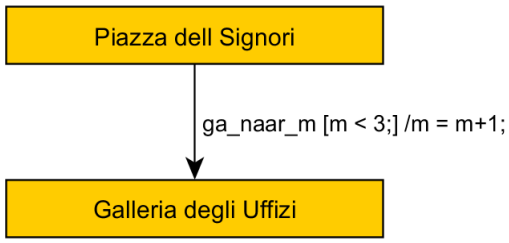
Ik mag graag bijhouden wat ik gedaan heb. Aan het begin gaf ik een lijstje van interessante plaatsen, die ik zeker gezien moet hebben. Als ik dat gedaan hebt vind ik het leuk om dat dan ook in de lijst aan te strepen. ➔

		Top 10 Florence
1	☑	Duomo
2	☐	Galleria degli Uffizi
3	☑	Piazza della Signoria
4	☐	Galleria dell'Accademia
5	☑	Ponte Vecchio
6	☐	Santa Croce
7	☐	Palazzo Vecchio
8	☐	Giardino di Boboli
9	☑	Battistero
10	☐	Museo Nationale del Bargello

In een model kunnen we hetzelfde doen: dit noemen we dan dat de toestand verandert. En die verandering maken we in het model zichtbaar met zogenoemde toestandsvariabelen. Even terug naar mijn lijstje: elk onderdeel geef ik een letter. Met '•' duid ik aan dat ik het nog wil zien. Met een '☐' geef ik aan dat ik er geweest ben. In een model kunnen we hetzelfde doen. Maak voor elke toestand een letter. Geef alle letters in het begin een '•'. Verander dat in een '☐' als de toestand is opgetreden. Wanneer ik in plaats van '-' en '+' getallen 0, 1, 2, 3, … ga gebruiken kan ik ook nog tellen hoe vaak de toestand is bezocht. Het definiëren van een toestandsvariabele gaat technisch als volgt: na de naam van een overgang neem je een actie op.
De actietekst voor een nieuwe toestandsvariabele M ziet er zo uit:

```
/m = m + 1;
```

De '/' geeft aan dat het een actie is. De ';' geeft het einde van de actie definitie weer.



Alweer naar het museum. Ik vind het prachtig, maar mijn kinderen willen ook wel eens ergens anders naar toe. In het model wil ik ook graag kunnen bepalen wat er gebeurt, afhankelijk van wat er al gedaan is. Ik heb uitgelegd dat de transitie bepaalt waar je naar toe gaat. Om te voorkomen dat je dezelfde weg te vaak loopt, benoemd ik een poortwachter. Deze telt het aantal keren dat je langs komt en zegt: 'ho!', als het te vaak is. In het model noemen we deze controle ook een poortwachter (Engels: Guard). Technisch geven we deze poortwachter regel weer op de volgende manier:

```
[m<3]
```

Je mag verder naar het museum als het aantal bezoeken minder dan 3 is. En we hadden 'M' als toestandsvariabele aangemaakt om het aantal museum bezoeken te tellen. Als we in het model het museum bezoeken maken we de teller één hoger:

```
/m = m + 1;
```

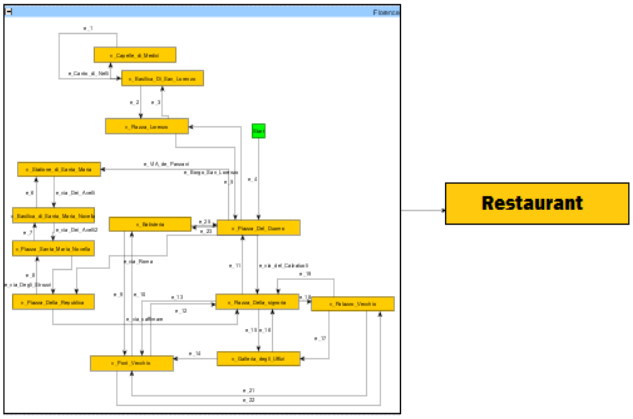
Het label bij de transitie naar het museum, die zowel de bewaking als het tellen regelt wordt nu:

```
Ga_naar_m [m<3] /m = m + 1;
```

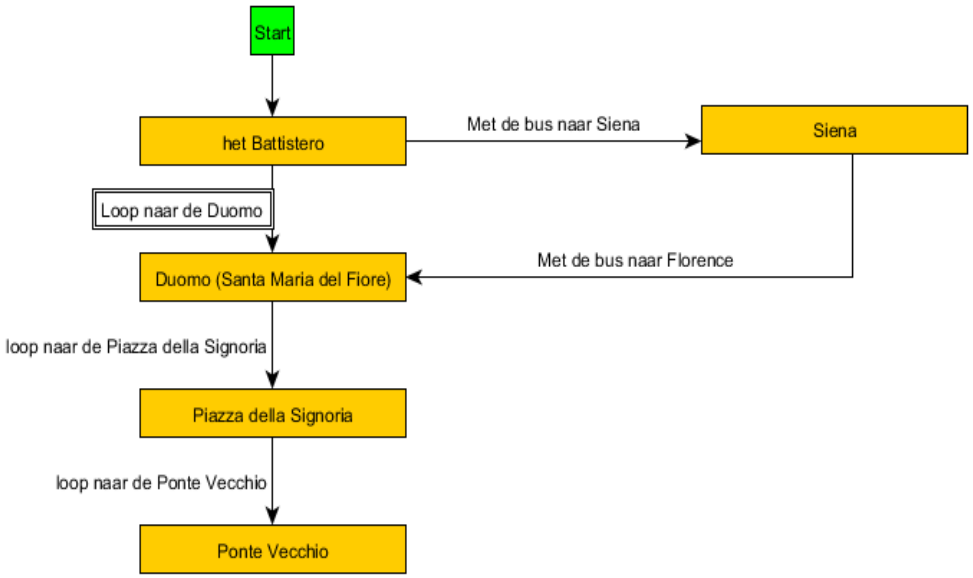
Moet het model compleet zijn? Deze vraag wordt vaak gesteld. Net als bij het plannen van een reis, kun je ook in MBT klein beginnen en later uitbreiden. Laat ik een voorbeeld geven. Tot nu toe ben ik in Florence gebleven. Maar in de buurt van Florence zijn nog veel meer interessante plaatsen. Bijvoorbeeld Siena. Je kunt met de trein of met de bus naar Siena. Deze reis duurt ongeveer een uur. In Siena kun je weer verschillende dingen gaan bekijken. ➔

Of alleen maar gezellig wijn drinken op het bekende Piazza del Campo. In een eerste stap maak ik een transitie naar Siena en één toestand Siena. En natuurlijk een transitie terug naar Florence. Later kan ik plaatsen voor Siena toevoegen. Ik kan ook eerst afzonderlijk een stedentrip voor Siena alleen maken. Deze testen en dan later toevoegen aan de Florence tour.

Als ik een tijd door Florence gewandeld heb begint mijn inwendige mens om aandacht te vragen. Is het nu tijd voor een hapje en een drankje? Tenslotte ben ik in Italië. Dus ik onderbreek mijn ontdekkingstocht en zoek een leuk restaurantje. Na wat gegeten en gedronken te hebben vervolg ik mijn tocht. In het model is deze overgang niet te modelleren als een transitie vanuit een bepaalde toestand. Dit is eerder een transitie die wordt veroorzaakt door een 'gebeurtenis'. Het maakt niet uit waar ik ben, of anders gezegd: het is een transitie die uit alle toestanden kan beginnen en altijd in het restaurant eindigt. In de modeltheorie voor toestandsdiagrammen wordt dit aangegeven door alle toestanden en transities in een groep op te nemen en dan een transitie vanuit die groep te maken.



Als laatste wil ik nog laten zien wat er gebeurt als je niet alleen reist maar met een groepje. Zolang de groep bij elkaar blijft, verandert er niets. Het wordt anders als de groep zich splitst. De ene helft gaat naar Fiesole om de Mont Ceceri te beklimmen waar Leonardo da Vinci een eerste poging deed om te vliegen, terwijl de anderen nog een keer naar het museum willen gaan. Als we dit vertalen naar een model, ontstaan er feitelijk twee afzonderlijke modellen. Beide met hun eigen dynamiek en toestanden. Wanneer beide groepen weer samen komen is het weer één groep. Deze vorm van modelleren kom je vaak tegen bij het testen van communicatie protocollen, waarbij de onderdelen onafhankelijk van elkaar werken. ➔



Samenvatting

Door de stedentrip naar Florence te gebruiken als metafoor voor een toestandsmodel kreeg ik snel inzicht in de werking van zo'n toestandsmodel. Daarnaast heeft het me geholpen om een testgenerator te bouwen voor MBT. Ik hoop dat je na het lezen van deze uiteenzetting ook de smaak te pakken hebt en toestandsmodellen maken leuker bent gaan vinden. MBT kan voorgoed van 'veelbelovend' gepromoveerd worden tot werkbaar en zal het niet verdwijnen als een 'Silver bullet' in de kast met test curiositeiten in een Florentijns museum. ←

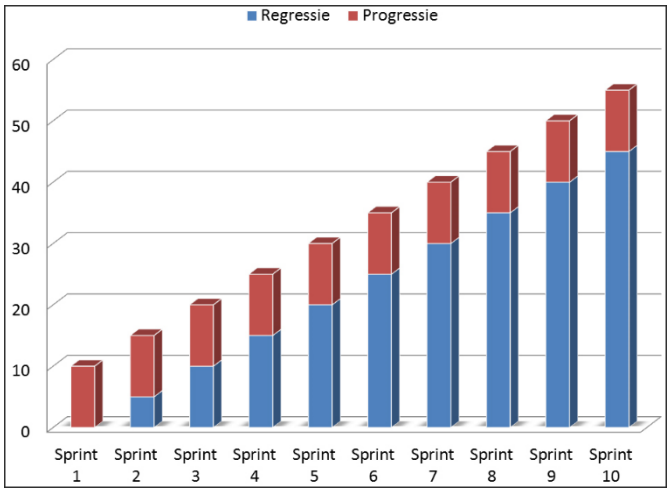
GEAUTOMATISEERD TESTEN IN AGILE, EEN STAPPENPLAN

Door Tom Heintzberger ● tom.heintzberger@preagus.nl ● @tcnh

De traditionele succesfactoren voor testautomatisering worden in Agile omgevingen steeds vaker ingehaald door de realiteit. Waar we als test-professionals voorheen vooral waakzaam moesten zijn op factoren als: 'Is het systeem voldoende stabiel om testen te kunnen automatiseren?', 'Is ons testproces voldoende volwassen om geautomatiseerd testen rendabel te kunnen maken?' of 'Ga ik mijn testen wel voldoende vaak herhalen om de investering van geautomatiseerd testen terug te verdienen?', is testautomatisering in Agile softwareontwikkeling vanaf het allereerste begin noodzaak.

Direct automatiseren

Vanaf het begin van het project wordt immers al werkende software opgeleverd en de functionaliteit wordt met elke user story uitgebreid. Elke user story heeft daarmee het potentieel om functionaliteit uit vorige user stories te breken. Dit terwijl vooraf niet alle requirements of specificaties voldoende zijn uitgewerkt om een testset over de hele breedte van je applicatie te kunnen plannen. Het belang van een complete regressietestset die op elk gewenst moment, of beter nog, bij elke build kan worden uitgevoerd is levensgroot.



Begin je in een dergelijk ontwikkeltraject pas met automatiseren als de software het predicaat 'stabiel' heeft gekregen of als het testproces aan alle kanten is afgekaderd, dan kijk je tegen een achterstand (technical debt) aan die in de beschikbare tijd niet meer in te halen valt. Bovendien loop je dan het risico te maken te krijgen met een product waarvan je de kwaliteit maar moeilijk kunt aantonen.

Direct beginnen met automatiseren dus. Maar hoe te beginnen? De eerdergenoemde succesfactoren voor testautomatisering (zoals: stabiele software, volwassen proces, return on investment) zijn niet uit de lucht gegrepen. Deze factoren hebben in de praktijk hun waarde ruimschoots bewezen. Hiervan zijn legio voorbeelden in de vorm van mislukte testautomatiseringsprojecten. Hoe zorg je er nu voor dat jouw geautomatiseerde testset onderhoudbaar wordt, bruikbaar blijft en vooral waarde levert aan het project?

Stappenplan

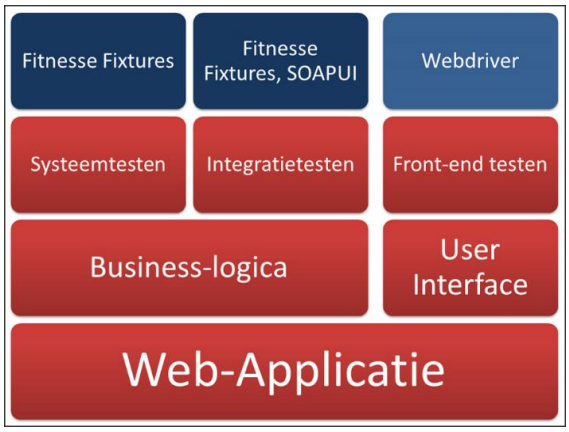
Om te helpen met het bereiken van bovenstaande volgt hieronder een stappenplan uit de praktijk.

1. Denk vooraf na over wat je waar, wanneer en op welke manier (hoe) gaat testen

Dat betekent niet dat je in elke sprint een systeemtest, integratietest, functionele acceptatietest en gebruikersacceptatietest moet inplannen. Het betekent wel dat je vooraf moet bedenken op welke manier je welk type requirement of risico wilt raken in je testen. Voor de testen die uiteindelijk geautomatiseerd moeten worden is het belangrijk om de juiste tool(s) te kiezen voor de testen die moeten worden uitgevoerd. ➔

In veel projecten wordt helaas nog altijd naar testtools gekeken als oplossing voor één afgebakend probleem: namelijk testen automatiseren. Helaas zijn er maar weinig testtools geschikt voor alle testen die je in je project wil automatiseren. Een testtool die gebruikersgedrag simuleert op de frontend van je applicatie is minder geschikt voor het controleren van business logica of de verwerking van berichtenverkeer. Een testtool bedoeld voor webservices is niet geschikt voor frontend testen. Vaak bestaat het werk van de tester uit meer dan één van de genoemde voorbeelden. Bedenk daarom vooraf welke testen je wanneer wilt automatiseren en stem in overleg met het team de keuze voor de testtools daarop af. Houd rekening met beleid binnen de organisatie, het budget, maar vooral ook met aanwezige kennis en kunde. Kies geen testtool waar je alle testen in een programmeertaal moet uitschrijven als je daar als team niet mee uit de voeten kunt.

2. Zet je testtools in de steigers



Je project gaat van start, je zit in de planning sessie van sprint 1. Typische stories in de eerste sprint: Inrichten ontwikkelomgeving, inrichten test- en acceptatieomgeving, Stubs en Drivers. Aan alle kanten worden voorbereidingen getroffen voor een geslaagd stuk software. Het neerzetten van kaders voor geautomatiseerd testen hoort daar ook bij. Maak er dan ook een user story van en schat de complexiteit van deze user story met het hele team in. In zo'n user story kun je denken aan taken als het bepalen van gebruikte tools, afstemmen van naamgevingsconventies die voor geautomatiseerd testen van belang zijn, of het bouwen van een proof-of-concept adapter die werkt met je geselecteerde test tool of -framework.
Het is geen schande om als tester hierbij de hulp en expertise van ontwikkelaars in te roepen. Sterker nog, de meeste ontwikkelaars zullen blij zijn dat de tester in het team aangeeft wat hij/zij nodig heeft om goed geautomatiseerd te kunnen testen. Overleg de mogelijkheden om geautomatiseerde systeem- en integratietesten op gezette momenten te gaan laten uitvoeren. Dit zorgt ervoor dat je geautomatiseerde testen een plek krijgen in het realisatieproces. Daarmee worden geslaagde testen een teamverantwoordelijkheid, in plaats van het 'probleem' van de tester. Gebruikt je team een systeem voor versiebeheer, dan horen ook de testgevallen in dat systeem beheerd te worden.

3. Begin klein (en vaak handmatig)

Alle voorbereidingen en goede intenties ten spijt, testen blijft gewoon mensenwerk. Meer nog: testen is een specialistisch vak en om geautomatiseerd te kunnen testen moet je weten hoe je applicatie werkt en wat je er van mag verwachten. Maar ook zal je vanuit je testexpertise moeten bepalen waar je begint met testen (prioriteit), waarom (risico), hoe veel (dekking) en hoe diep (diepgang). Dit is mensenwerk, hier heb je geen tools voor.
Zorg dat er, voordat die eerste user story richting testomgeving gaat, iets te testen valt. Zorg voor een eerste geautomatiseerde test die, idealiter, kan fungeren als smoke-test voor een omgeving of sanity-check voor een stukje applicatielogica. Probeer in de eerste sprint niet meteen alles te automatiseren. Gebruik je handmatige testsessies of testgevallen om behalve vast te stellen dat het gerealiseerde ook het juiste is, ook om te leren hoe de applicatie werkt. Doe dit tevens om een gevoel te krijgen voor de onhebbelijkheden van de applicatie of de omgeving en vast te stellen welke testsituaties veel zullen voorkomen of in een regressieset terecht moeten komen. ➔

Automatiseer die gevallen dan ook direct. Andere gevallen die je ook zoveel mogelijk direct automatiseert, zijn die gevallen die een bug hebben blootgelegd.

4. Verdeel je testset in logische blokken

Na de ervaring met een paar eerste stories binnen de applicatie heb je een kleine, maar doordachte set geautomatiseerde testen weten te bewerkstelligen. Afhankelijk van het soort applicatie misschien zelfs meer dan één set. Breid je testsets in elke story uit, maar kijk tegelijkertijd naar de samenhang van je testset en let op onderlinge overeenkomsten of afhankelijkheden tussen testgevallen. Muteert testgeval 2 een record dat in testgeval 1 is aangemaakt? Houd er dan rekening mee dat testgeval 2 niet kan worden uitgevoerd als testgeval 1 mislukt.

Vooral wanneer je test op de gebruikersinterface is het van groot belang om nooit op meerdere plaatsen dezelfde handelingen te automatiseren. Maak van een veel voorkomende handeling een herbruikbaar component (modulaire opbouw) en groepeer die scenario's op een logische manier. Wijzigt er iets aan de applicatie op een veelvoorkomende plek, dan wil je die wijziging het liefst slechts op één plaats hoeven door te voeren. Het is een goed idee om je componenten op dezelfde manier te structureren als de ontwikkelaar de applicatielogica structureert.

5. Laat je testset voor je werken (Continu Testen)

Je geautomatiseerde testen zijn nu goed bruikbaar en leveren waarde aan het project. Regressie issues worden snel gevonden. En omdat veel van de gescripte testen geautomatiseerd worden uitgevoerd, heb je meer tijd om de applicatie, vanuit je expertise als tester, handmatig aan de tand te voelen. Dit komt de gebruiksvriendelijkheid en robuustheid van de uiteindelijke applicatie ten goede.

Nu is het tijd om de geautomatiseerde testen dan ook echt voor het team te gaan laten werken. Richt samen met je team de build- en testomgevingen zodanig in dat je testsets meedraaien in het opleverproces. Je regressietest en eventuele systeemtesten voor nieuwe requirements zijn nu al uitgevoerd vóórdat de software op de testomgeving terecht komt.

Test gefaald? Eerst bevindingen oplossen, dan weer opnieuw proberen. Naast het voordeel van vroeg uitvoeren van testen, is dit dé manier om de zichtbaarheid van je testen te waarborgen. Laat dit ook zien in productdemo's! Als de product-owner en de ontwikkelaars keer op keer de testen zien werken en het juiste inzicht krijgen in de kwaliteit en de staat van het opgeleverde product, zal het vertrouwen in het product en de kwaliteit van het eigen werk toenemen. Dit zie je uiteindelijk terug in een meer gemotiveerd ontwikkelteam en een hogere velocity.

6. Blijf bij en weeg af

In de praktijk is het bijna nooit haalbaar (of wenselijk) om al je testen te automatiseren. Er wordt, mede dankzij je betrouwbare testset(s), in een steeds hoger tempo functionaliteit gerealiseerd en het team verwacht van de tester dat de testgevallen continue zijn bijgewerkt.

Houd in de planning sessies rekening met de inspanningen voor test. Een story die weinig complex is om te realiseren, kan een grote impact hebben op het onderhoud van je regressietestset of lastig automatiseerbaar zijn vanwege benodigde data of systeemcondities. Weeg steeds af welke testgevallen voor automatisering in aanmerking komen en welke niet en weeg steeds af welk testgeval je waar en wanneer automatiseert. ➔

Tenslotte

Is testen in een Agile team nu zo anders als in een waterval-traject? Het antwoord is wat mij betreft 'nee'. Testen is ook in een Agile team 'Een verzameling activiteiten die uitgevoerd wordt om een of meer kenmerken van een product, proces of dienst vast te stellen volgens een gespecificeerde procedure.' (ISO/IEC 1991).
De procedure is alleen een beetje veranderd; we hebben als testers niet meer de 'luxe' (en de stress) van een eigen project-in-een-project, waarin we ons eerst bezighouden met reviewen, dan met specificeren en vervolgens met het uitvoeren van testen. We hebben niet meer vooraf een complete set van specificaties van het hele systeem, maar wel steeds van een klein gedeelte daarvan; de user story. De truc is om als tester om te kunnen gaan met het werken aan hele kleine onderdelen, waarbij de uitdaging is om het grote plaatje nooit uit het oog te verliezen.

Grotere rol tester

Er wordt weleens beweerd dat uiteindelijk geautomatiseerde testen de tester overbodig zullen maken, maar de rol van de tester is in de praktijk juist groter aan het worden. Door de continue betrokkenheid en de nauwere samenwerking met zowel de ontwikkelaars als de product-owner wordt de tester als vanzelf de bruggenbouwer tussen techniek, business en exploitatie. De tester levert aan alle kanten meerwaarde door te bewaken dat de specificaties de wens van de klant correct en eenduidig weergeven en vervolgens vast te stellen of de kenmerken van het product overeenkomen met de wens van de klant.
Ondertussen helpt de tester het proces te versnellen door de juiste testen te automatiseren en kwalitatief goede bevindingen op te leveren. Uiteindelijk vertrouwt het beheer- en exploitatieteam op het advies van de tester als het gaat om het in productie en beheer nemen van het opgeleverde product. Dat vertrouwen moet het team verdienen, maar in het uitdragen daarvan is de grootste rol voor de tester weggelegd.
Is de benodigde skillset dan anders? Er wordt weleens gezegd dat een Agile tester een techneut moet zijn. Dat is in mijns inziens niet noodzakelijk. We hebben immers al ontwikkelaars aan boord voor de technische kennis. Wel is het belangrijk dat de tester affiniteit heeft met dat wat zijn of haar team realiseert. Affiniteit met de techniek, maar net zo belangrijk: affiniteit met de gecreëerde waarde. Idealiter minimaal een beetje van beide. Maar is dat nieuw? Of zijn dat gewoon competenties die elke goede tester bezit? ←

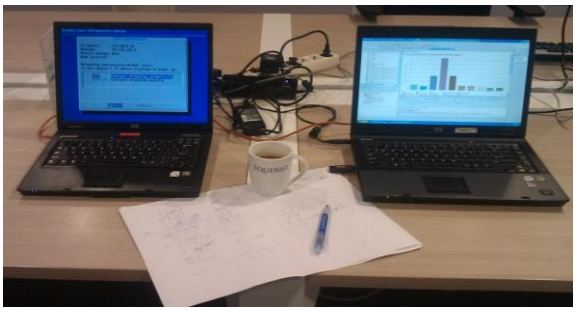
KALIBRATIE VAN PERFORMANCE TESTTOOLS

Door Roland van Leusden ● Roland.van.Leusden@squerist.nl

We hebben met performancemetingen regelmatig discussie met leveranciers van de software en infrastructuur die andere meetinstrumenten gebruiken dan wij om de performance te meten. Vaak meten zij alleen delen van de keten zoals netwerk en/of server performance en niet de gehele keten. Ook de nauwkeurigheid van de resultaten uit de gebruikte testtools wordt regelmatig ter discussie gesteld.

Onlangs kwamen we bij een opdrachtgever waar men zelf een performancetest had gedaan met vijftig virtuele gebruikers, in productie bleek met vijftig 'echte' gebruikers dat de performance onvoldoende was. Ons onderzoek toonde aan dat het gebruikte tool met de default instellingen anders met de applicatie omging dan de 'echte' gebruikers en een veel lagere belasting genereerde dan verwacht.
Daarom hebben we een eigen onderzoek gestart om de oorzaken van deze verschillen te lokaliseren. Hierbij staan de volgende vragen centraal:

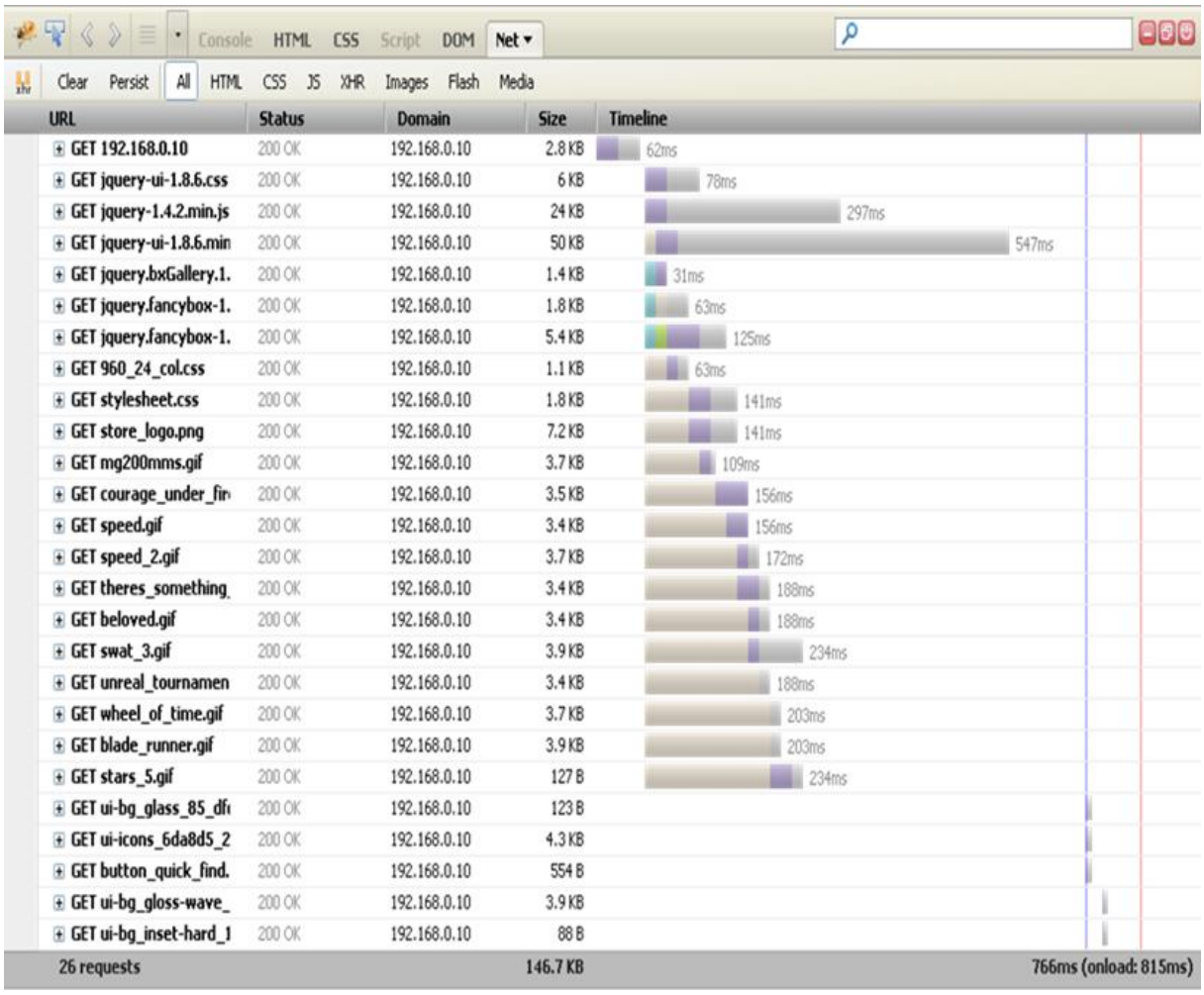
- Welke invloed heeft de performancetool op de gemeten performance?
- Vertoont de tool hetzelfde gedrag op client-, netwerk- en serverniveau als een gebruiker?
- Kunnen we de tool kalibreren zodat wel een vergelijkbaar gedrag ontstaat?

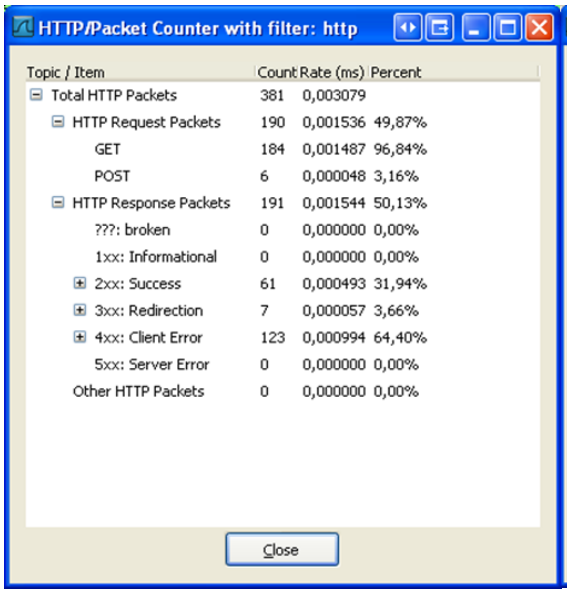


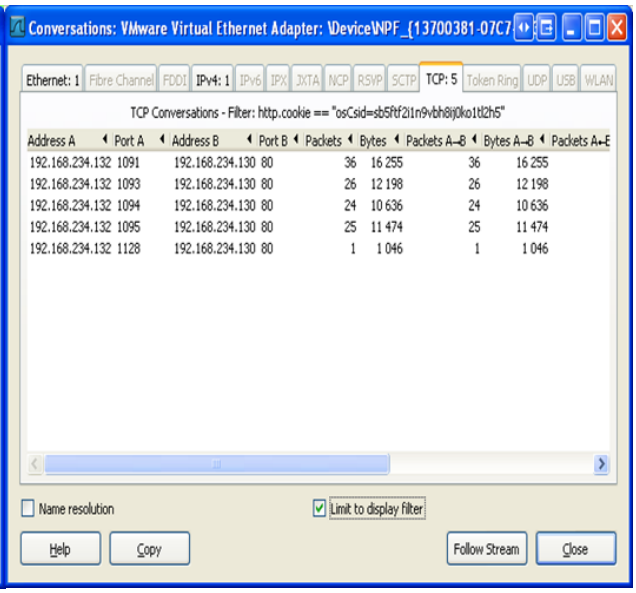
Om de verschillen inzichtelijk te krijgen is het noodzakelijk om voor iedere tool dezelfde uitgangssituatie te creëren en externe factoren zoveel mogelijk te elimineren. Op basis hiervan is er gekozen voor de volgende setup: twee laptops, verbonden met elkaar via een cross kabel om invloed van het netwerk uit te sluiten. Eén laptop werkt als server vanaf een bootable Linux DVD. Zie hiervoor de link: http://www.turnkeylinux.org/oscommerce. Aangezien deze server geheel in het geheugen draait hebben we hiermee de garantie dat de uitgangssituatie na elke reboot gelijk is. Op de andere laptop hebben we vier VMware XP images met op elke image een tool geïnstalleerd. Hiermee verkrijgen we voor iedere tool dezelfde uitgangssituatie. Om het gedrag binnen de browser in kaart te kunnen brengen word Firebug gebruikt. Het netwerkgedrag is inzichtelijk gemaakt met Wireshark. Een simpel scenario is eerst handmatig opgenomen en vervolgens met iedere tool.
Scenario:

- Voeg de DVD 'Speed' toe aan de bestelling;
- Zoek voor een DVD met 'Mary' in de titel;
- Voeg de gevonden DVD 'There is something about Mary' toe aan de bestelling;
- Ga naar afrekenen;
- Maak een nieuw account aan;
- Maak de bestelling en verzending af;
- Terug naar de hoofdpagina en log uit. ➔

Het handmatig uitgevoerde scenario is de referentie. Alle pagina's van het scenario zijn zowel vastgelegd met Firebug als met WireShark. Hieronder de resultaten voor de startpagina. ➔

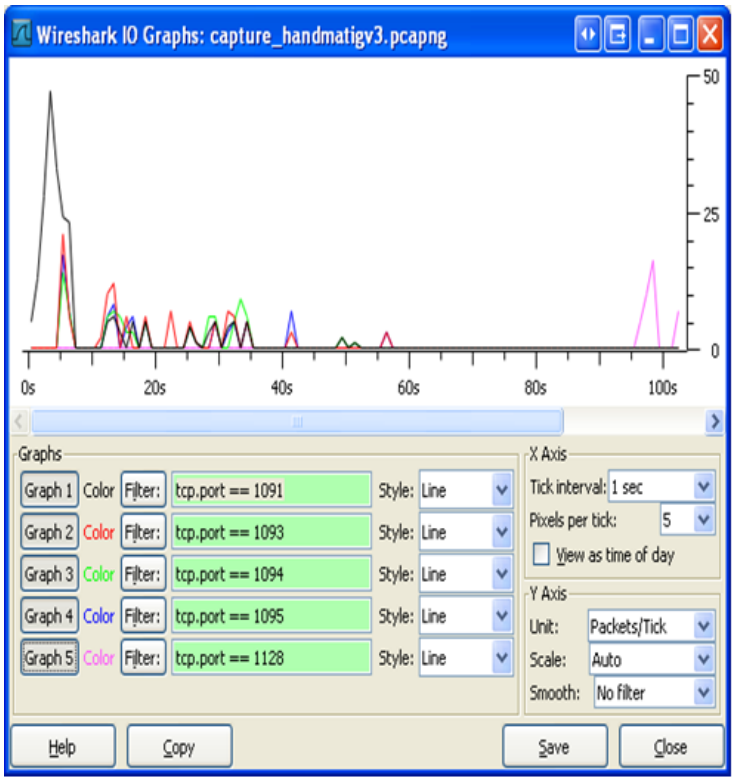




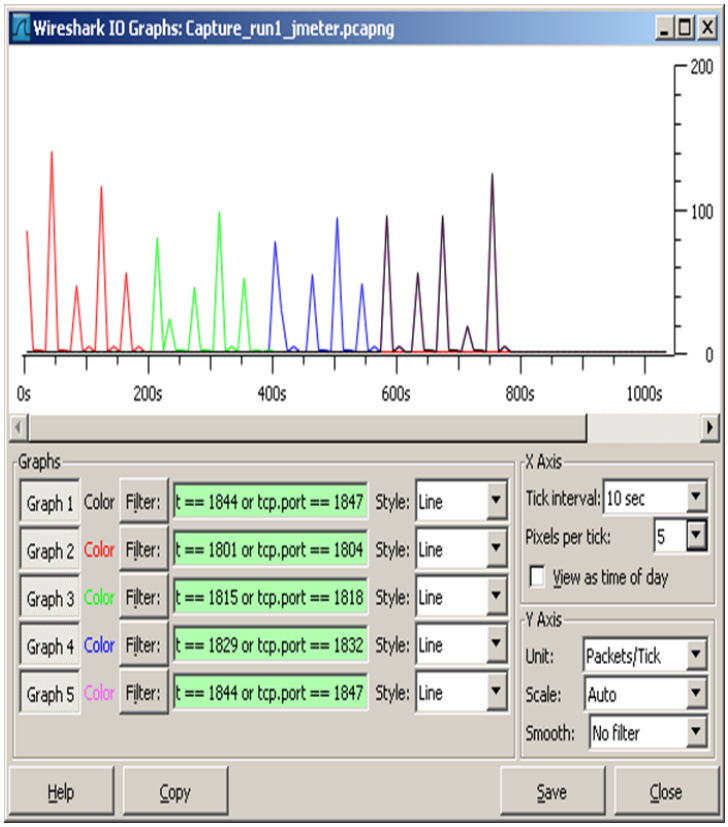


Met iedere tool is dit scenario ook opgenomen en vervolgens één keer met één gebruiker uitgevoerd. Hierbij kwamen de volgende verschillen naar boven:

- De tools emuleren de interne werking van de browser, zoals de uitvoering van javascript niet en compenseren er met de default instellingen ook niet voor;
- Het totaal aantal requests vanuit de browser en vanuit de tool is verschillend;
- De tools openen op netwerk niveau meer sessies met de server dan de browser doet;
- Waar de browser content parallel opvraagt van de server doen sommige tools het serieel. ➔



Browser



Tool

Uit deze resultaten kunnen we de conclusie trekken dat het noodzakelijk is om voor ieder scenario het gedrag van de gebruiker op client-, netwerk- en serverniveau te analyseren. Deze analyse dient vervolgens als referentie om het tool zo in te stellen dat zoveel mogelijk hetzelfde gedrag vertoond wordt. Met andere woorden, kalibratie dus. Hiermee kunnen we richting opdrachtgever en leverancier inzichtelijk maken dat het gebruikte tool een vergelijkbaar gedrag heeft als een 'echte' gebruiker en de testresultaten een afspiegeling zijn van de te verwachten resultaten. Het door de kalibratie verkregen inzicht in de werking van de client-kant helpt bij performance issues die niet gerelateerd zijn aan netwerk-/server-performance. ←

MOBIELE REVOLUTIE: EEN BOOST AAN TESTAUTOMATISERING

Door Thomas Veltman ● thomas.veltman@sogeti.nl

Op het eerste gezicht hebben de mobiele revolutie en testautomatisering weinig met elkaar te maken. Toch zorgt de enorme revolutie in het gebruik van smartphones en tablets voor een extra boost voor testautomatisering in mobile development projecten. Om uit te leggen hoe dit zit, gaan we eerst terug naar het begin van de mobiele revolutie…

Op 9 januari 2007 kondigde Steve Jobs voor een uitzinnig publiek aan dat Apple een toestel had ontwikkeld dat zowel een telefoon, muziekspeler als een revolutionair internettoestel was. Weinigen beseften toen welke invloed smartphones, zoals de iPhone, zouden gaan hebben op het dagelijks leven. Tegenwoordig gebruiken meeste smartphonegebruikers hun telefoon bijna continu. Vanaf het moment dat de wekker op de telefoon afgaat tot nog even voor het slapen het weer van morgen controleren. Soms zelfs nog vaker. Bijvoorbeeld apps die je helpen bij het monitoren van je slaapcyclus.

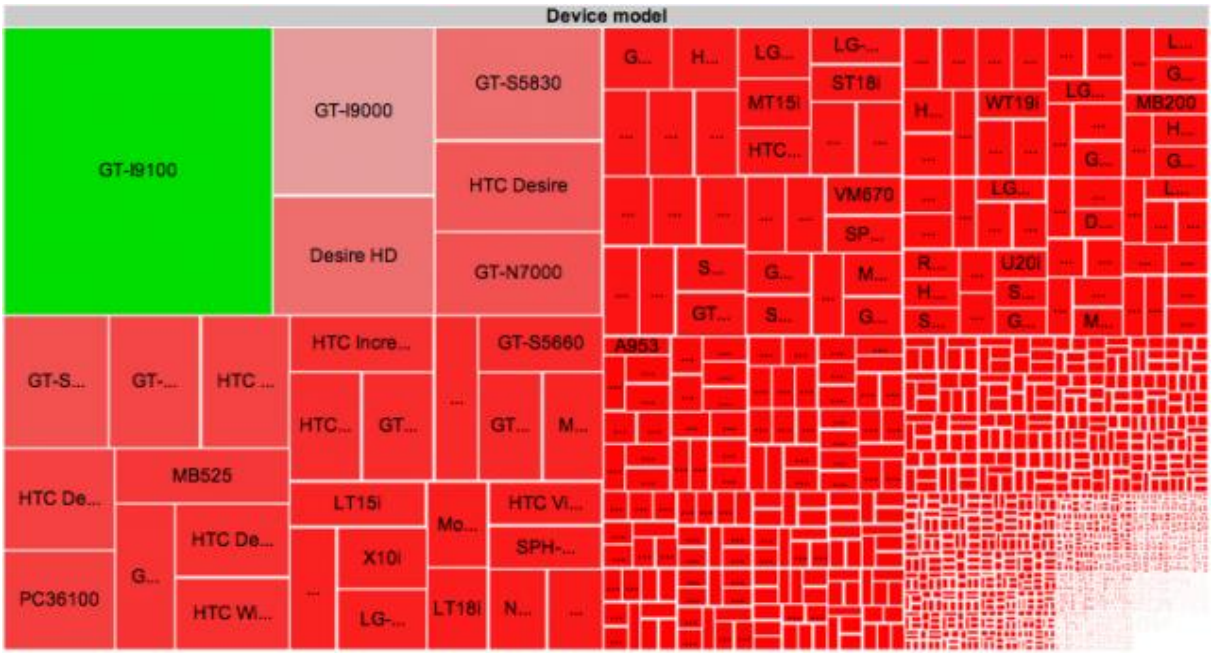
Een van de redenen van het succes van mobiele toestellen is dat het mensen in staat stelt om op andere momenten software te gebruiken. Momenten waarop ze dit voor de mobiele revolutie nog niet deden. Bijvoorbeeld onderweg om te controleren of je trein op tijd of om je saldo checken. Of op de bank marktplaats.nl en funda.nl afzoeken. In de vergaderstoel op het werk om aantekeningen maken en om informatie snel beschikbaar te hebben.

Feit is dat niet alleen mensen die aan het werk zijn achter hun bureau apps gebruiken. Maar ook in situaties waarin ze maar beperkte tijd en aandacht kunnen besteden aan het bedienen van de app. Dit stelt enorm hoge eisen aan kwaliteit van een app. Om succesvol te zijn moet de performance, usability en de functionaliteit van een app de perfectie zeer dicht benaderen. Deze eisen worden nog eens verhoogd omdat gebruikers via de App Stores een podium hebben om hun grieven met alle andere potentiële gebruikers te delen. Mindere kwaliteit leidt snel tot een mindere beoordeling (minder sterren). Dat zal het succes van de app en het imago van de ontwikkelende organisatie schaden.

Specifieke uitdagingen

Hoge kwaliteit is daarom noodzaak. Goed nieuws voor testers dus! Het testen van apps kent echter een aantal specifieke uitdagingen. Een belangrijke uitdaging is dat een app moet werken op allerlei verschillende toestellen. Deze toestellen zijn weer uitgerust met verschillende resoluties en besturingssysteemversies. Vooral voor het Android besturingssysteem bestaan ongelofelijk veel toestellen.

Op de volgende bladzijde zie je een grafische weergave van de 3997 toestellen die in mei vorig jaar op de markt waren. ➔

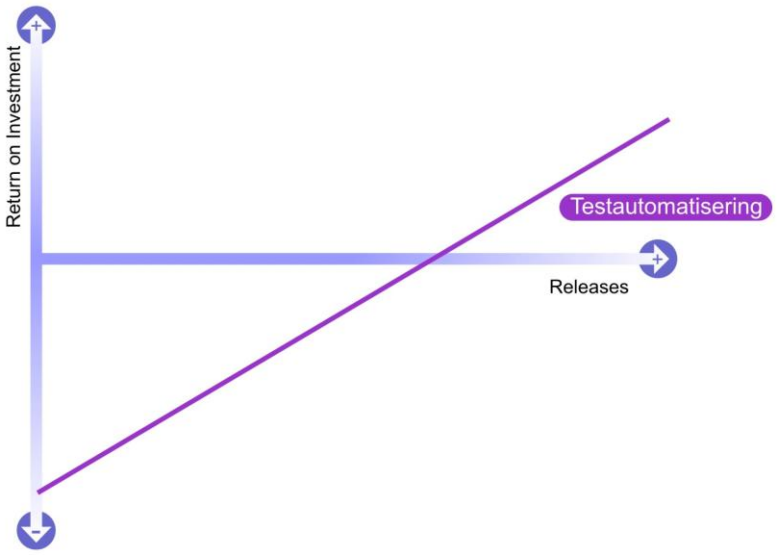


Figuur 1: Verschillende mobile devices en hun marktaandeel.

Uit: opensignalmaps.com study, May 2012

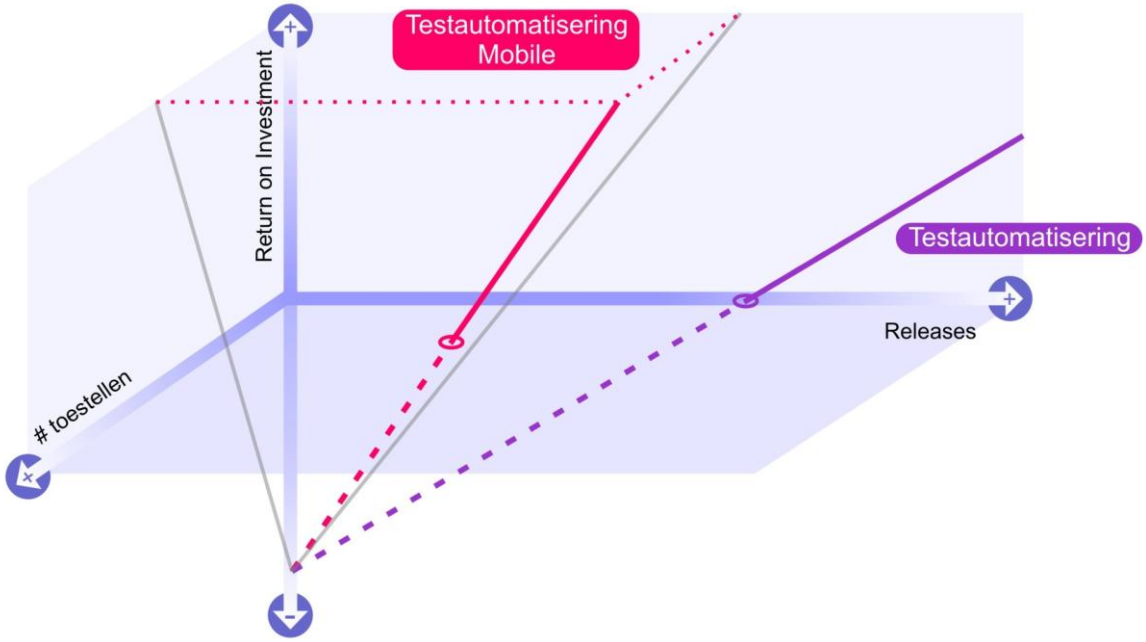
De complexiteit wordt nog vergroot doordat meerdere versies van de besturingssysteem op een toestel kunnen staan. Voor iOS geldt ook een behoorlijke complexiteit met meerdere toestellen zoals de Ipad en de iPad mini.
Een tester komt al snel tot de ontdekking dat handmatig testen van al die toestellen niet de meest attractieve oplossing is. Natuurlijk kan de tester wel een intelligentie selectie maken van toestellen. Maar dan nog kost het te veel tijd om alles handmatig te testen. Daarbij zijn de kosten nog niet eens van het grootste belang. De doorlooptijd van testen wel. Binnen mobile development volgen releases elkaar namelijk enorm snel op. Naast de lang doorlooptijd is het ook nog eens dodelijk saai om steeds dezelfde functionaliteit te testen op verschillende toestellen.

Dat is het moment dat testautomatisering om de hoek komt kijken. Dit stelt de tester in staat om meer van de simpele controles in de beperkte tijd te kunnen doen. Daardoor kunnen testers focussen op meer ingewikkelde testcases. Dit ziet men ook terug als men kijkt naar de business case voor mobile testtooling ten opzichte van die van die voor desktop applicaties. Op desktopapplicaties is de businesscase voor testautomatisering als volgt: men moet eerst investeren in licenties voor de tool en in de kennis die nodig is om de test te automatiseren. Vervolgens bouwt men een geautomatiseerde regressietestset op. Deze investering moet zich in meerdere releases terugverdienen. ➔



Figuur 2: Terugverdienmodel testautomatisering

Bij het investeren in het geautomatiseerd testen van mobile moet men een extra as aan de grafiek toevoegen. Hierdoor ontstaat een 3D-grafiek. Omdat men de geautomatiseerde testset niet alleen per release opnieuw gebruikt, maar ook per toestel is de investering sneller terugverdiend.



Figuur 3: Terugverdienmodel mobile testautomatisering

Kortom, door de mobiele revolutie is het kwaliteitsbewustzijn van gebruikers gestegen. Testers zullen alle mogelijke hulpmiddelen moeten aangrijpen om zoveel mogelijk te controleren in de korte tijd beschikbaar is. Testautomatisering heeft daardoor een nog belangrijkere plaats gekregen in het testproces! ←

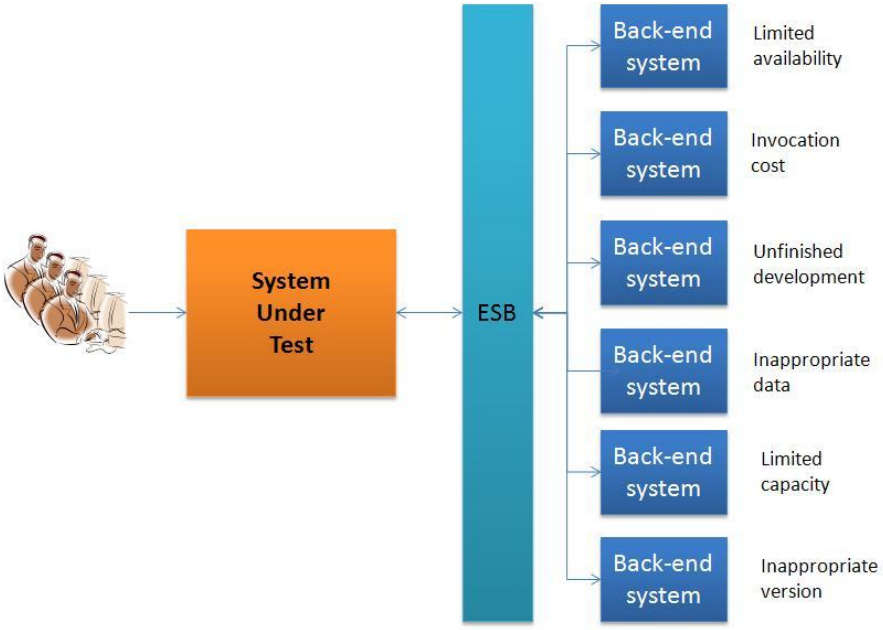
AUTOMATED DEPLOYMENT OF VIRTUALIZED SERVICES, BASED ON FUNCTIONAL, PERFORMANCE AND DATA ASPECTS

Door Faris Nizamic ● faris.nizamic@gmail.com ● @nizzamich
en Rix Groenboom ● rix.groenboom@parasoft.nl ● @rix_groenboom

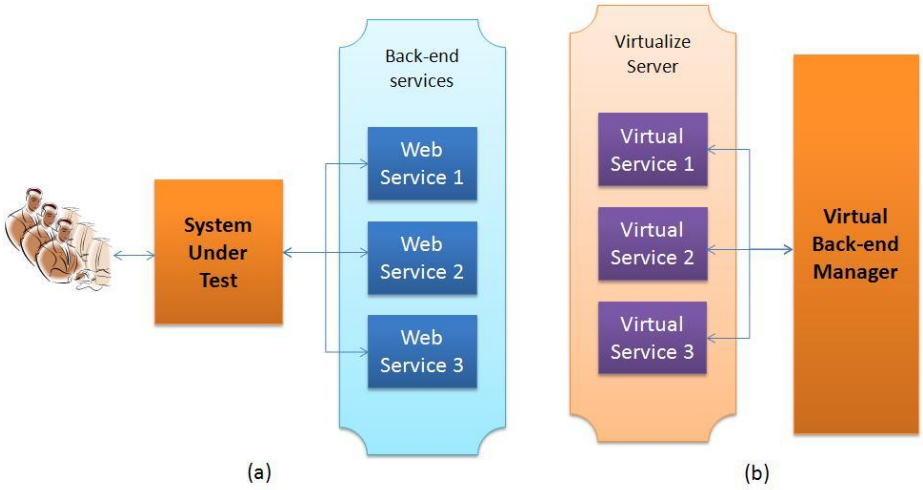
Modern software development methods (Agile, Continuous Delivery) require flexible test environments, whilst applications (like SOA) become more complex and have more external dependencies. How can these constraints be resolved and taken away?

In 2012, a survey asking software developers, testers, and performance test engineers a series of questions about their access to test environments. Key findings showed that testing today's composite, distributed applications requires interacting with a number of dependent (connected) applications, which are difficult to access. Respondents reported needing access to an average of eight dependent applications, but having consistent access to only three of them. An overwhelming majority of respondents (76 percent) reported having restricted access to the test environments required for completing their development and testing tasks. The time available to access test environments is extremely limited and 30 percent of that limited time was consumed by configuration/setup tasks. Finally, testers had time to execute only 50 percent of the available test plan. These results indicate that development and testing teams lack the resources required to complete the expected level of testing.

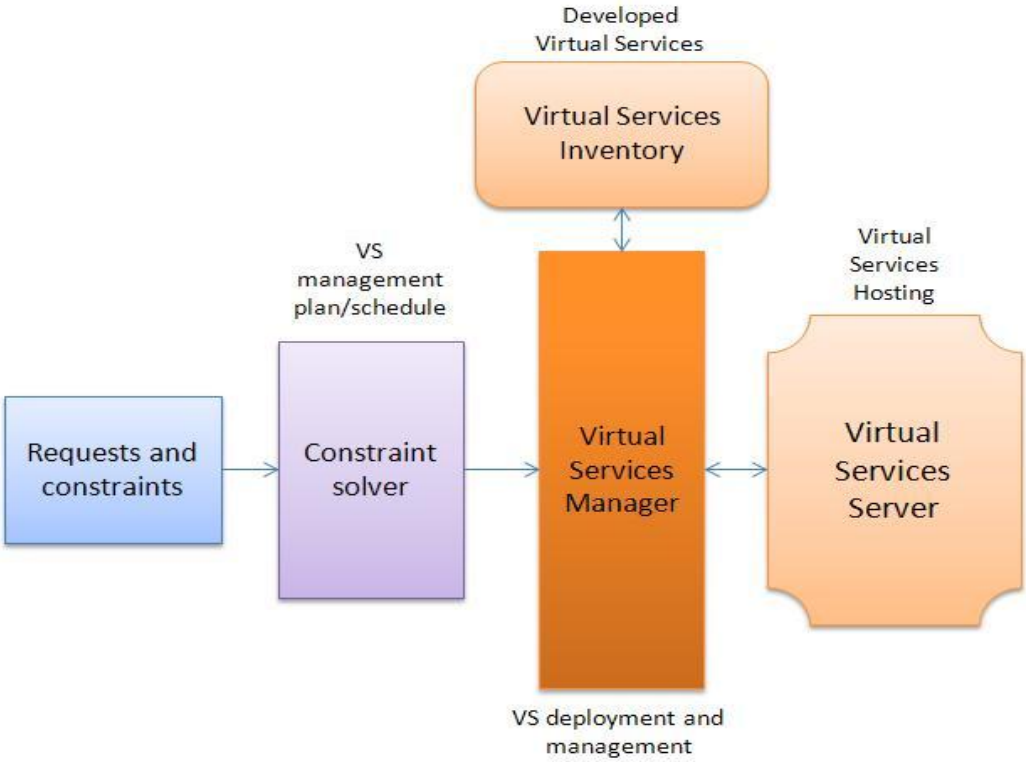
To tackle this complexity of testing today's composite applications, we will link the world of Agile development with the operational aspects of providing flexible test environment (DevOps) as building blocks for Continuous Delivery. Today, development and testing teams have limited control over the (sub) systems that are crucial for reaching the expected quality level of the services. There are a number of constraints that limit the output of dev/test teams. In particular in distributed architectures based on SOA and using an ESB. Our focus is mostly based on external constraints such as back-end systems that our system under test depends on. Those constraints include limited availability, unfinished development, limited capacity (to support load tests for example) or inappropriate (test) data. ➔



The core technology to remove these constraints is Service Virtualization. Service Virtualization provides testers unconstrained access to dependent applications via test environments that are easily configured for distinct testing requirements. Rather than to attempt to replace the complete functionality of the dependent applications, Service Virtualization focuses on simulating only the portion of the dependent applications behaviour that is necessary for completion of development and testing tasks. So, for each (web) service used by the system under test, we have a virtual (simulated) service available that can be used for testing.



We will demonstrate how scheduled manipulation with back-end data, performance and functional parameters can be implemented. Manual work needs to be done only a single time, to model the behaviour of a service and to define the performance profiles relevant for the tests. Then, once the instances of Virtualized Services are prepared (functionality) and are defined how they should behave (performance), deployment and provisioning can be scheduled, and therefore included as part of automated process. ➔



These methods have a significant value for the continuous integration of service-oriented systems. To show that extended functionality can be obtained, an open-source tool is used which implements scheduled automatic deployment of virtualized services by building dynamic infrastructure through SOAP interface. Additionally, through the web service API, we demonstrate how dynamic changes of back-end data, performance and functionality of each service can be easily done.

It is certain that high degree of automation is needed for obtaining consistent and reliable staged environments required by today's agile testing. We are convinced that this automation approach can move organizations from reactive or proactive to managed or even optimized level of maturity. ←

LESSONS LEARNED BIJ IMPLEMENTATIE TESTAUTOMATISERING ING

Door Rogier van der Burg ● Rogier.van.der.Burg@ing.nl en Andréas Prins ● Andreas.J.Prins@ing.nl

Binnen de ING staat testautomatisering hoog op de agenda. Door de transitie van waterval naar Scrum / Continuous Delivery ontstaat hier de noodzaak voor testautomatisering. Daarnaast zijn de technologische mogelijkheden van de tools de afgelopen jaren enorm toegenomen. Deze zaken bij elkaar maakte het voor de ING mogelijk om met geautomatiseerd regressietesten een flinke stap voorwaarts te maken!

Welke uitdagingen hebben we ondervonden bij het implementeren van testautomatisering? Tegen welke misvattingen liepen we aan en hoe hebben we die opgelost?

De transitie binnen ING en de keuzes voor testautomatisering

Sinds enige jaren is bij ING de transitie in gang gezet om de manier van software ontwikkelen te wijzigen. De transitie gaat van waterval en releasematig werken naar Agile-Scrum en Continuous Delivery (CD). Testautomatisering speelt hierin een belangrijke rol als een van de pilaren om succesvol te zijn. We hebben allereerst geleerd dat we testautomatisering op verschillende manieren kunnen implementeren en dat met verschillende soorten tooling kan worden gewerkt. Het kiezen van het tool en de aanpak is elke keer weer de keuze die we moesten maken bij de start van nieuwe projecten.

De tools die we gebruiken voor geautomatiseerd testen

Binnen ING is de HP ALM suite operationeel en daarmee is het Business Process Testing (BPT) Framework van ALM, in combinatie met QTP een hele logische stap. Vanuit de historie maken we binnen ING al vele jaren gebruik van de voorgangers van de huidige ALM suite. Een keuze voor QTP als testautomatiseringstool is zo een kleine stap.

Naast deze ALM-QTP stroming is er binnen ING een tweede stroming die vooral gebruikmaakt van tools als Selenium in combinatie met Fitnesse. Wat we hebben geleerd is dat binnen een groot bedrijf als ING, met veel omgevingen, succesvol twee stromingen naast elkaar kunnen bestaan. Elk met een eigen aanpak, aandachtsgebied en toepassing. ➔

Doordat er steeds meer kortcyclisch wordt gewerkt binnen Scrum en continuous delivery, is regressietesten steeds vaker nodig. De noodzaak groeide hierdoor om dit te automatiseren en vanwege de herhaalbaarheid is regressietesten voor de ING dé testvorm om te automatiseren. Om die reden zijn meerdere ART-trajecten opgestart (Automated Regression Testing). Hierbij is testautomatisering geïmplementeerd op basis van het genoemde BPT Framework. Dit zijn kleinere trajecten van een paar weken, tot grotere trajecten die ruim een half jaar in beslag nemen.

De factoren die de verschillende trajecten van elkaar onderscheiden

In de praktijk is gebleken dat de trajecten vooral door de volgende factoren van elkaar verschillen:

- Het al dan niet beschikbaar zijn van een regressietestset;
- De omvang van de regressietestset;
- De complexiteit van het systeem (webinterface is veelal eenvoudiger dan JAVA of Mainframe; het automatiseren van datawarehouse testen vraagt weer een andere oplossing);
- De beschikbaarheid van de domein experts van de te testen applicatie(s);
- De beschikbaarheid van test (automatiserings)kennis binnen de teams.

Dit heeft tot gevolg dat we niet zomaar overal dezelfde aanpak konden toepassen. Zodat we nu per project een intake doen om deze en andere zaken goed te verkennen.

De geleerde lessen bij het implementeren van testautomatisering

Bij het implementeren van geautomatiseerd regressietesten kwamen we regelmatig misvattingen tegen die een succesvolle implementatie van ART hinderden. De belangrijkste lessons learned zijn:

1. Geef de tester een tool en hij gaat automatiseren

We hebben geleerd dat hier grofweg drie problemen achter zitten. De kennis ontbreekt voor het op de juiste wijze installeren van de tool. Daarnaast ontbreekt de kennis voor het configureren en het opzetten van het framework voor een specifiek project. Tenslotte is het maken en onderhouden van de testgevallen ook makkelijker gezegd dan gedaan. Het hebben van het tool zorgde dus niet direct voor veel geautomatiseerd testen.
Daarom bieden we nu vanuit een centraal team trainingen aan om kennis op te doen van het tool en is er hulp aanwezig voor de installatie. Daarnaast kunnen teams gebruikmaken van een opstartservice waarmee we voor een applicatie het framework inrichten en de eerste bouwblokken samenstellen.

2. De bestaande regressietestset of een paar testgevallen automatiseren is voldoende

Hierbij was de oplossing niet zo eenvoudig als het op het eerste gezicht lijkt. Er is gebleken dat een goede set afhangt van een aantal factoren. Ten eerste wat is het doel van de geautomatiseerde set? Moet het bepaalde componenten afdekken? Welke business waarde moet het afdekken? Daarnaast is het afhankelijk van wat er voorhanden is en welke testen nu al voldoende zijn uitgewerkt om dit op te pakken. Uiteindelijk moeten die dingen worden gedaan die de meeste waarde toevoegen en dan vooral gericht op de waarde voor de business.
We maken daarom nu standaard gebruik van een walk-around waarbij zowel de techniek, de functionele wensen alsook de business aspecten in kaart worden gebracht. Hieruit volgt de aanpak voor de te automatiseren testgevallen. ➔

Overigens is het wel goed om met één of een beperkt aantal testgevallen te starten, om aan te tonen dat ART mogelijk is. Tevens kan de omgeving dan 'wennen' aan het feit dat testen ook geautomatiseerd kan worden. Daarnaast is gebleken dat we vanaf het begin van de automatisering het beheeraspect moeten meenemen. Training en begeleiding zijn hierbij van wezenlijk belang.

3. We hebben geen testers meer nodig als alle testen zijn geautomatiseerd
De tester blijft een rol hebben in het geheel, alleen zal de tester niet meer handmatig vaak herhalende regressietesten hoeven uitvoeren. De tester kan juist een rol spelen in het SMART krijgen van user stories, vaststellen welke testgevallen geautomatiseerd moeten worden en beoordelen wat de kwaliteit is van de geautomatiseerde testset. Bovendien zal je elk nieuw testgeval in de testset handmatig moeten opstellen en uitvoeren om te achterhalen wat en hoe je moet automatiseren.

4. Eén tool, één aanpak en iedereen kan aan de slag
Testautomatisering is juist specifiek geënt op de onderliggende applicatie die getest wordt. Je zoekt dus altijd naar de best passende oplossing en tool. Wel is het goed om kennis zoveel mogelijk te delen om niet elke keer weer opnieuw het wiel uit te vinden. Daarnaast zijn algemene principes vaak bij elke applicatie van toepassing. Dit is binnen de ING vastgelegd in een framework en de kennis hierover wordt uitgewisseld in de test automation community op ING's eigen social media platform genaamd Buzz.

5. Testautomatisering levert altijd direct wat op
Testen automatiseren vraagt op de korte termijn een investering. Testen die je altijd handmatig deed, ga je in een tool zetten. Pas als je meerdere handmatige runs vervangt door de geautomatiseerde runs, verdien je deze investering terug. Bij de transitie naar continuous delivery, is het voordeel dat deze runs vaak voorkomen en daarmee wordt de investering sneller terugverdiend.

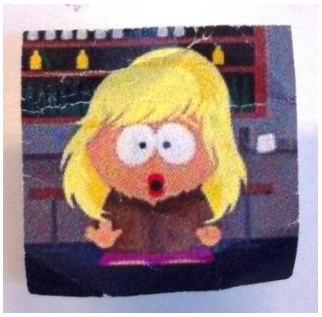
De vervolgstappen voor testautomatisering binnen ING

En hoe gaan we nu verder? Op dit moment zijn we druk bezig standaard building blocks te maken. Hiermee kunnen we vaak voorkomende functies snel automatiseren. Binnen de Selenium community zijn we hier verder mee dan binnen de QTP community, zodat we hierin ook van elkaar leren.
Vervolgens is er de stap om het meer concreet inzichtelijk maken van wat ART ons oplevert. Naast het genoemde feit dat het noodzakelijk is in een kortcyclische omgeving, hebben we de verwachting dat het ons ook een kortere time-2-market geeft, kwaliteitsverbetering oplevert door een vergroting van de test coverage en een afname van incidenten op zal leveren. En dat samen zal de kosten verlagen.
Kortom, elke project dat we uitvoeren geeft ons naast voldoende lessen om het volgende project nog beter aan te pakken, ook het vertrouwen dat we ING hier daadwerkelijk mee verbeteren! Klaar zijn we nog lang niet en ook het hele stuk voorbereiding kan ook wat automatisering gebruiken. Dus wordt vervolgd! ←

YVONNE: A TALE ON MOBILE TEST AUTOMATION

Door Samuël Maljaars ● samuel.maljaars@capgemini.com ● @smaljaars #yvonne

Every day when I come to the office Yvonne has already finished the regression testing for the day. The report she delivered is, as always, on time and ready to be analyzed.

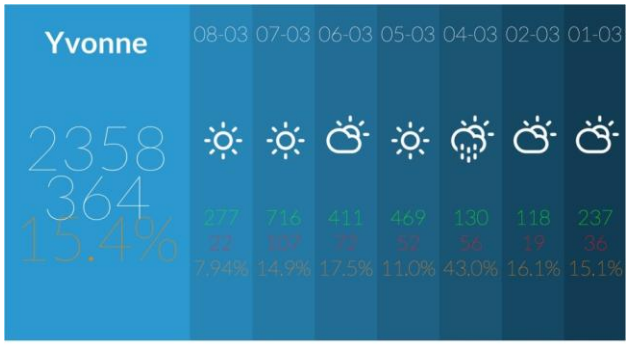


Yvonne is the code name for the continuous integration set-up for User Interface Test Automation (UIA) on the iOS mobile platform at ING Bank NV. Like a real person Yvonne is part of the ING Mobile Banking iOS Scrum team. She even is part of the daily stand up. Well, to be honest she is represented by the team member who has analyzed the logging outputs. Yvonne keeps the team members up-to-date about test execution via messaging in a chat application in real time. Her main purpose is continuously applying quality control.

In this article you'll get to know Yvonne a little bit better. Firstly I'll elaborate on the technical components of the set-up and the code structure of our test automation code. Furthermore I will discuss the benefits of having Yvonne around and show how the set-up of mobile test automation can effectively be a team effort.

Technical components Yvonne

Yvonne consists of a unique integration of several technical resources that enable the continuous quality control in our mobile iOS App development. The input for Yvonne is a set of test scripts written in JavaScript. These are then executed in the Instruments Automation tool which is running on a Jenkins build server. This JavaScript test code is part of the source code repository on a git server that is accessible by all team members. The Jenkins build server checks out the source code (from the master or feature branch) and then compiles and builds the Apps on the devices. The test scripts are then executed on multiple devices simultaneously using a self modified battery charger. The devices, iPhones and iPads running different iOS versions, are plugged into the battery charger which is connected to the Jenkins build server. It is technically sold as a battery charger but we modified it to a docking station to which not only power but also data can be send.

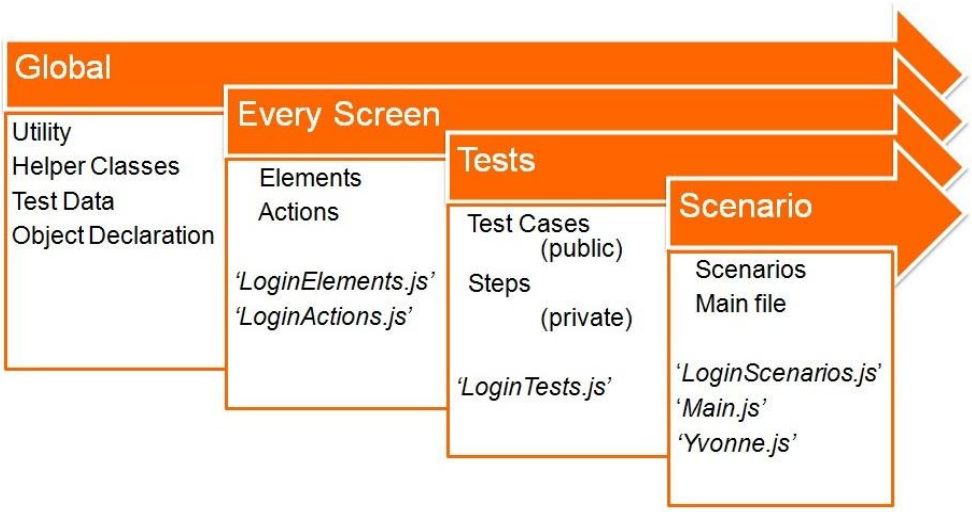


Once the apps are beamed to all devices, the Jenkins build server activates a shell script which kicks off the Instrument Automation tooling to start test execution. The automated test execution logging messages are continuously saved ➔

to disk which is real time accessible from an intranet webpage or pushed via a chat application for smart phones. On a dashboard in the team room everyone can see the general statistics of the latest regression tests that were executed by Yvonne.

Code structure for UIA test scripts

The JavaScript code structure we have established for our test scripts is a general concept that can be applied anywhere when writing test automation code for mobile app development. The basic principle we have established is that the UI automation should be done per screen. It is a bit like the page object pattern that is for example used in the selenium browser automation framework. This principle structures the code and makes it well maintainable and accessible for everyone. It makes the code largely self explanatory. UI automation done per screen basically means that each screen in the app is initially seen as an isolated part. From this screen we generate two files: an Elements file, and an Actions file. These files together form the first layer of test script files. The elements file returns all the elements on the screen. The action file solely performs actions on its corresponding elements file. For example, for a login screen you should make a LoginElements.js file and a LoginActions.js file. A login action could be to tap on a Login button. In the elements file the login button is returned, and in the action file the login button is called and tapped upon.



The second layer of test script files consists of the so-called Test files. To continue the example with login, this would create the LoginTests.js file. In this file multiple actions are called on a screen to create a test case for entering a pin code and tapping a login button. Another test case could be to check the lay-out of the Login screen. Or any exception that may occur after tapping the login button.

In the third layer of test script files, called the Scenario files, the test cases are listed and grouped together in scenarios that cover different functional flows. In the login example one scenario could be a simple login for different types of users with different pins. Another scenario could be a stress test for logging in and logging out multiple times after each other.

The final layer of all the test script files consist a set of supporting files: a `main.js` file, an object declaration file, Utility and Helper files, and test data files. The main.js file is the file that imports all Scenario files and is loaded into Instruments Automation. From there all scripts can be started with 1 click on a button. The object declaration file declares global objects for the Actions and Test files so they can be accessed in respectively any Test or Scenario file. A special feature of the way our test suite is set-up is that the test cases are test data driven. ➔

So one test case in a Test file can be called several times in a Scenario file but with different test data input each time the test case is called.

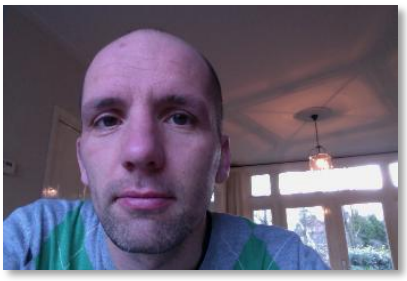
Benefits

Yvonne enables completely automated daily UIA on multiple iPhones/iPads on different iOS versions, against different networks (stub/non-stub, high/low network quality) including exporting and storage of all logging. The basis for the test execution is a set of well maintainable JavaScript test scripts, which runs in the native Apple tooling Xcode Instruments Automation. It is an extensive regression set tool, saving time on manual regression testing, and enabling automated early bug detection already in development phase of our fast mobile development life cycle. Another great benefit of this approach is the relatively low cost of the set-up. Apple's Instruments Automation tool is default part of Xcode and hence comes for free once the iOS development environment is in place.

The key to the complete solution is a whole team approach iOS mobile test automation in the context of Scrum is a team effort, and not only a testing job. The main reason is that test automation in Instruments Automation is a technical programming job, which was a challenge for us at first as test consultants. We have made major improvements in the maintainability of automation testing code because developers have taken the effort to review existing code and to pair up with us to improve it. The complete Yvonne solution has come into existence thanks to the whole team of developers and testers who actively participate in coming up with creative ideas and solutions. So for continuous integration of UIA counts: it's a team effort. Also to continuously improve by incorporating new technologies and by extending the test coverage. It is a great thing to get the best out of available technical resources for a process of continuous quality control. ←

SUCCESSFUL TESTING THE CONTINUOUS DELIVERY PROCESS

Door Huib Schoots ● huib.schoots@codecentric.nl en Pascal Dufour ● pascal.dufour@codecentric.nl



Met behulp van Continuous Delivery bereik je snel je doel: het betrouwbaar en snel releasen van functionaliteit. Meer dan ooit is het van belang dat software optimaal op klanten afgestemd wordt en dat de leveringstijden van (nieuwe) software releases worden verkort. De voordelen van agile, en daarbij specifiek de methoden Scrum en Kanban, zorgen voor een snelle en directe feedback van klanten al vanaf de eerste levering. Het zijn twee belangrijke stappen in de richting van Continuous Delivery. Klanten profiteren snel van nieuwe software of software updates door de korte ontwikkelcycli van twee tot vier weken. Continuous Delivery is als de motor achter het releaseproces: het automatiseert alle stappen tot het in productie brengen van de software: van een handmatig en foutgevoelig proces, naar een geautomatiseerd en reproduceerbaar proces.

Continuous Delivery in het kort

Continuous Delivery versnelt en automatiseert de vele processen die bij het ontwikkelen van software nodig zijn. Dit verkort doorlooptijden, vermindert de error-rate en verhoogt de frequentie van levering. Kortom, het aantal mogelijke releases per tijdseenheid neemt toe. Het resultaat: meer klantgericht en een betere betrouwbaarheid van de geleverde oplossing tegen een gereduceerde prijs per release.

Voordelen:

- Verhoogde waarde van IT door een snelle en veilige levering van veranderingen;
- Betere kwaliteit van een release, door een automatisch en reproduceerbaar proces met minder fouten;
- Lagere ontwikkelingskosten door meer automatisering en versnelde processen;
- Productieve samenwerking bij het testen van deployment op weg naar een naadloos leveringsproces;
- Verhoogde klanttevredenheid door een betrouwbare en probleemloze overgang naar productie door een eenvoudig herhaalbare routineactie.

Alle processen vallen samen als kleine radertjes in het geheel van software ontwikkeling. Continuous Delivery optimaliseert elke stap van de ontwikkeling en het releaseproces. Van het voltooien van de broncode van de software tot het in gebruik nemen door de eindgebruiker. Daarbij maakt het niet uit of het gaat om de integratie van een opgeloste bug, implementatie van kleine veranderingen of het toevoegen van nieuwe functies. Dit vereist een juist samenspel van de verschillende disciplines:

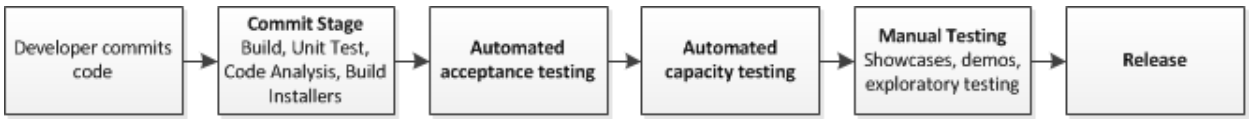
- *Build:* het idee achter Continuous Delivery is dat de huidige stand van de broncode bij elke wijziging gecompileerd wordt en als artifact opgeleverd wordt. Daarbij rekening houdend met afhankelijkheden en via een gestandaardiseerd build proces.
- *Test:* geautomatiseerde unit-, integratie- en acceptatietesten zorgen ervoor dat continu testen bewerkstelligd wordt en dat aan zowel oude als nieuwe, functionele en niet-functionele eisen wordt voldaan.
- *Deploy*: het derde gebied is Continuous Deployment. Daarbij is het mogelijk om nieuwe versies van de software automatisch op de ontwikkeling, integratie en productieomgevingen te deployen. ➔

Continu leveren en met hoge kwaliteit

Om continu te leveren met een constante hoge kwaliteit is het belangrijk dat gewenste kwaliteitsniveau te leveren in elke stap van het ontwikkelproces:

- De integratie van alle code van verschillende teams en ontwikkelaars;
- Het bouwen van een software increment;
- De configuratie van de applicatie voor verschillende omgevingen;
- Kwaliteit en compliancy audits;
- Unit-, integratie- en acceptatietesten;
- Performance en security analyse;
- Database wijzigingen;
- De installatie van de software in een productieomgeving.

Het installeren in de productieomgeving is natuurlijk niet een volledig automatische stap na ontwikkeling, maar wordt handmatig getriggerd om rekening te houden met de klant behoeften en omstandigheden. Het is wel een geautomatiseerd herhaalbaar proces zoals voorgaande deployments.



De bovenstaande processtappen worden in de regel door verschillende personen uitgevoerd waardoor er risico is dat niet alle processen goed op elkaar afgestemd zijn. Daarmee zal de automatische release cycle niet zonder verstoringen uitgevoerd kunnen worden.

De release motor blijft altijd draaien

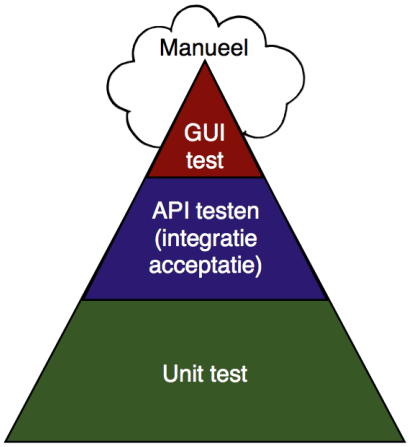
Met Continuous Delivery heeft het team een 'release motor' die continu beschikbaar is, die soepel werkt en transparant is. Elke stap in het softwareontwikkelproces is perfect op elkaar af gestemd. Dit in plaats van afzonderlijke acties waarin development, testen en beheer afzonderlijk van elkaar handelen. Het aantal wijzingen per release is laag waardoor het testen van deze wijzigingen overzichtelijker is.

Van stabiel naar beter

Continuous Deployment forceert geen release naar de klant. Elke eigenaar van een systeem bepaalt zelf wanneer hij de desbetreffende release op zijn systeem wil deployen. Releases worden als het ware naar het systeem getrokken ('pull') in plaats van de gebruikelijke 'push' vanuit development.
Hierdoor heeft de systeemeigenaar controle over welke versie waar en wanneer draait. Als tester vraag je om een release in een bepaalde omgeving en met één druk op de knop kan dit worden gerealiseerd. Met weer een druk op de knop staat het ook op de volgende omgeving. Tot uiteindelijk in één simpele handeling de release live staat op de productieomgeving.
Hierbij kunnen tools helpen, denk hierbij aan bijvoorbeeld een continuous integration server zoals Jenkins met zijn vele plugins. Ook applicaties zoals Puppet zijn erg bruikbaar. Het helpt systeembeheerders bij het beheren van de infrastructuur gedurende de gehele levenscyclus. ➔

Balans in de teststrategie

Hoe zorg jij als tester, uiteraard in samenwerking met je team, ervoor dat er zo goed mogelijk getest wordt? Het is van groot belang dat jij als tester goed weet wat geautomatiseerd afgedekt wordt en wat nog aandacht behoeft. Denk hierbij aan een integraal inzicht in geautomatiseerde unit- en integratietesten. Door een helder inzicht in coverage verminder je de risico's dat de software niet of juist te zwaar getest wordt. Door het inzicht in de test coverage én de software architectuur ben je in staat om de test strategie zo aan te passen dat het past als een maatpak. De teststrategie helpt te bepalen welke tests van toepassing zijn en op welk moment. Hiermee kan feilloos geschakeld worden tussen verschillende testen zoals security- of performancetesten. Deze testen kunnen geautomatiseerd of handmatig worden uitgevoerd. Om een tot een goede wat-test-ik-waar mix te komen zou je gebruik kunnen maken van de zogenoemde 'automation triangle' (zie afbeelding).



Een gouden regel bij het automatiseren is dat je zo veel mogelijk testen doet op een zo laag mogelijk niveau. Alles testen is geen optie door gebrek aan tijd en resources. Ook met geautomatiseerd testen is alles testen nog steeds onmogelijk. Het is daarom van belang om iedere test zo slim mogelijk te doen. Het is raadzaam om met de unittesten de volledige breedte van de functionaliteit test af te dekken. Om de coverage verder te verhogen, worden ook integratietesten ontwikkeld. Als laatste komen de acceptatietesten aan bod. Een goede verdeling is de sleutel tot succes. Een betrouwbare manier om tot een set van testen te komen die later worden opgenomen in de automatische regressietest is vanuit exploratory testen. Eerst worden handmatig testen uitgevoerd en deze worden in een later stadium, indien gewenst, geautomatiseerd. Dit maakt duidelijk dat verschillende testsoorten elkaar niet uitsluiten maar juist aanvullen.

De perfecte mix van testen

Binnen Continuous Delivery en Continuous Deployment zijn veel testen geautomatiseerd. Het voordeel is dat een tool elke keer dezelfde check kan uitvoeren. Een tester voert testen echter elke keer net anders uit. Dat is precies wat je wilt! De verschillende manieren van testen leveren namelijk andere bevindingen op. Een mix van grote hoeveelheden geautomatiseerde testen in combinatie met (exploratory) handmatige testen in het releaseproces zorgen voor een krachtige test strategie. Het maakt gebruik van de sterke punten van zowel handmatig testen als geautomatiseerde checks. Voordat de release live gaat, kan een handmatig smoketest in combinatie met de geautomatiseerde regressietesten een goede strategie zijn. Aangevuld met additionele testen zoals performance als dat nodig is. Uiteindelijk weet je precies welke release draait en met welke kwaliteit. ←

CALL FOR PAPERS NAJAARSEVENEMENT 2013

Het TestNet najaarsevenement 2013 vindt plaats op donderdag 31 oktober 2013, in het voor ons zo vertrouwde NBC te Nieuwegein. Zoals gebruikelijk zal het evenement een opzet kennen met workshops in de ochtend, 3 aansprekende internationale keynotes en parallel-tracks in de middag en avond. We rekenen op 500 tot 600 bezoekers. Het thema voor het najaarsevenement is:

Exploring context-driven testing - a new hype or here to stay?

We haken met dit evenement aan bij de context-driven school of testing, die voor steeds meer testers een inspiratie is bij hun dagelijks werk. Wij roepen je op om jouw visie op en (vooral) ervaringen met context-driven testen te delen met de TestNetters. Voor diegene die nog niet bekend is met context-driven testen staat aan het eind van dit document een beschrijving.

We zijn op zoek naar een breed scala van onderwerpen en nieuwsgierig naar inzendingen vanuit jouw context!

Procedure

- Heb je een interessant, aansprekend of inspirerend verhaal voor een presentatie (45 minuten) of een interactieve workshop (1,5 of 3 uur)? Stuur dan een voorstel in en gebruik hiervoor het template in Word-formaat, dat je van de TestNet site kunt downloaden (https://www.testnet.org/images/stories/call_for_papers_najaarsevenement_2013_template_def.doc).
- De deadline om je voorstel in te sturen is zondag 2 juni 2013.
- TestNet is op zoek naar interessante, niet commerciële verhalen bij voorkeur van haar eigen leden. Commerciële presentaties, toolpromoties en andere sessies waarin een commerciële oplossing wordt aangeprezen, worden niet opgenomen in het programma. We zijn met name op zoek naar ervaringsverhalen uit de praktijk.
- De leden van de evenementencommissie beoordelen elk individueel alle inzendingen. Hun scores worden verzameld en de totaalscore bepaalt of een presentatie wel of niet geselecteerd wordt. In principe selecteren we slechts één inzending per bedrijf om de diversiteit van het programma hoog te houden. Bij onvoldoende kwalitatieve inzendingen van verschillende bedrijven behouden we ons het recht om toch meer sprekers van hetzelfde bedrijf te selecteren. Deze regel geldt apart voor workshops en presentaties, een bedrijf kan dus in ieder geval voor zowel een workshop als presentatie geselecteerd worden.
- Het programma biedt plaats aan presentaties van 45 minuten in parallelle sessies. De helft van de presentaties is 's middags, de andere helft 's avonds. Daarnaast worden er ook weer workshops georganiseerd, zowel in de ochtend (180 minuten) als in middag en avond (90 minuten). Ook hiervoor kun je een voorstel insturen. Geef bij je inzending duidelijk aan of het een presentatie of een workshop betreft. Naast de presentaties en workshops aangemeld via de ingezonden proposals, zorgt de evenementencommissie voor drie keynotes van aansprekende internationale sprekers.
- Stuur je inzending naar evenement@testnet.org en maak gebruik van de template. Zorg ervoor dat alle rubrieken zijn ingevuld.
- Het is mogelijk dat de evenementen commissie vooraf de slides van de presentatie wil inzien of een demo wil zien van de inhoud van de aangeboden sessie.
- Voor eind juni zal de evenementencommissie het programma samenstellen en krijgen alle inzenders bericht of hun presentatie al of niet op het programma staat. ➔

Wat is context-driven testen?

Het uitgangspunt voor een context-driven tester is het feit dat de wereld om hem heen een complexe, veranderlijke en onzekere plek is. Daarin is het dus noodzakelijk dat de tester zich voortdurend aanpast aan de context van dat moment. Testers zullen vaardigheden moeten ontwikkelen om te kunnen omgaan met de complexe, vaak dubbelzinnige en vluchtige, steeds veranderende wereld om hen heen.

> *In een blogpost 'What is context-driven testing?' uit 2009 geven **Cem Kaner** en **James Bach** een definitie: 'Context-driven testers kiezen hun testdoelstellingen, technieken en deliverables (inclusief inbegrip van testdocumentatie) door eerst te kijken naar de details van de specifieke situatie, met inbegrip van de wensen van de stakeholders.'*

De essentie van context driven testing is dat we als testers niet klakkeloos een dogmatische benadering van een ingestampte methodiek volgen. Wij zijn de experts die met een volle rugzak aan tools, technieken en ervaringen de context van de opdracht overzien en bepalen om welke aanpak en middelen de opdracht vraagt. Daarbij staan we open voor alle mogelijke tools en technieken. Uiteindelijk gaat context driven testing over hoe we optimaal kunnen presteren met die middelen die ons gegeven zijn. In plaats van te proberen generieke "best practices" toe te passen, accepteren we dat verschillende omstandigheden vragen om zeer verschillende benaderingen en zelfs verschillende definities van veel voorkomende testtermen.

Het meest opvallende dat geschreven is over context-driven testen zijn de zeven principes:

1. De waarde van elke aanpak hangt af van de context.
2. Er zijn goede aanpakken in een bepaalde context, maar er zijn geen 'best practices'.
3. Mensen die samen werken, zijn het belangrijkste onderdeel van de context van ieder project.
4. Projecten verlopen na verloop van tijd op een manier die vaak niet voorspelbaar is.
5. Het product is een oplossing. Als het probleem niet is opgelost, werkt het product dus niet.
6. Goed software testen is een uitdagend intellectueel proces.
7. Alleen door oordeelsvorming en vaardigheid, coöperatief uitgeoefend gedurende het gehele project, zijn wij in staat om de juiste dingen te doen op het juiste moment om onze producten effectief te testen.

Wil jij je (context-driven) ervaringen met de leden van TestNet delen? Stuur dan **uiterlijk op 2 juni 2013** een voorstel in voor een presentatie of workshop. We kijken uit naar je voorstel! ←